Dankwoord

In de eerste plaats willen we onze begeleiders Jan Dockx, Kristof Mertens en Nele Smeets bedanken voor de vele ideeën die zij ons gaven voor onze thesis en voor de discussies die we met hen konden voeren over de verschillende onderwerpen van onze thesis. Daarnaast bedanken we hen ook voor het nalezen van onze teksten.

Wij willen ook onze promotor, prof. dr. ir. Eric Steegmans bedanken voor het opvolgen van onze thesis, ondanks de zware werklast en omdat hij steeds bereid was onze vragen te beantwoorden.

Tenslotte willen we onze ouders, broers, andere familieleden en vrienden bedanken voor de steun en het vertrouwen die ze ons geboden hebben tijdens onze studies. Daarnaast wil Carolien nog haar vriend Dries bedanken voor het bieden van een luisterend oor.

Onze gedachten gaan ook uit naar de vader van Erwin Hermans die dit academiejaar tot ons verdriet is overleden. Hij heeft zijn kinderen altijd gesteund en we vinden het jammer dat hij dit niet meer mag meemaken.

Abstract

In deze verkennende studie van XML en XSLT hebben we naast deze onderwerpen ook XML Schema, XSL-FO, Cocoon, JDOM en nog enkele andere Application Programming Interfaces zoals DOM, SAX en JAXP bestudeerd. We geven een uitgebreid overzicht van de mogelijkheden en gebruikswijzen van al deze onderwerpen. We hebben de onderwerpen van onze studie ook in de praktijk gebracht aan de hand van een aantal toepassingen. Op het einde van deze thesis wordt ook een vergelijking gemaakt tussen Java en XML/XSLT.

Onze thesis steunt op twee pijlers: aan de ene kant een theoretische uiteenzetting over XML, XSLT, XSL-FO, JDOM en Cocoon, aan de andere kant een illustratie van de mogelijkheden die deze technieken bieden. Wij hebben als concrete toepassing van de theorie onze eigen thesistekst in XML gemaakt, daarvoor een XML Schema ontwikkeld en bijhorende stylesheets geschreven. Wij hebben ook aanpassingen en uitbreidingen aan het jnome open source project aangebracht en een bijdrage geleverd aan het documentatieproject van Cocoon 2.

1. Inleiding.....1
1.1. Het doel van onze thesis.....1
1.2. Inhoud van de tekst.....1
1.2.1. Eerste deel van de tekst.....1
1.2.2. Tweede deel van de tekst.....2
1.3. De gebruikte voorbeelden.....2
1.3.1. Omzetten van de OOP website naar XML/XSL.....2
1.3.2. De slideshow van onze presentatie.....3
1.3.3. jnome en JDOM.....3
1.3.4. Onze thesistekst in XML formaat omzetten naar HMTL en PDF.....4
1.4. Bijlagen.....4

2. XML.....6
2.1. Wat is XML?.....6
2.2. De geschiedenis van XML in het kort.....7
2.3. XML Syntax.....7
2.3.1. Elementen.....9
2.3.2. Attributen.....10
2.3.3. Entiteiten.....10
2.3.4. Commentaar.....11
2.3.5. Processing Instructions.....11
2.3.6. CDATA Secties.....12
2.3.7. Document Type Declaraties.....12
2.3.8. Overige onderwerpen.....12
2.3.8.1. Behandeling van witruimte.....13
2.3.8.2. Attribuutwaarde normalisatie.....13
2.3.8.3. Taalidentificatie.....13
2.4. Namespaces in XML.....14
2.4.1. Het declareren van namespaces.....14
2.4.2. Het toepassen van namespaces op elementen en attributen.....16
2.4.3. Bemerking over namespaces.....16
2.5. Een concreet voorbeeld.....17
2.6. XML: Syntax of taal?.....18
2.7. Conclusie.....19

3. XML Schema.....20
3.1. Wat is XML Schema?.....20
3.2. Geschiedenis van XML Schema.....20
3.3. Gebruik van XML Schema.....21
3.3.1. Associatie van XML document met XML Schema.....22
3.3.2. Validatie van een XML document.....23
3.3.3. Opbouw van een XML Schema.....24
3.3.3.1. Het annotation element.....24
3.3.3.2. Het element element.....25

3.3.3.3. Het simpleType element....26
3.3.3.3.1. restriction....27
3.3.3.3.2. union....27
3.3.3.3.3. list....28
3.3.3.4. Het complexType element....29
3.4. Bespreking van een uitgebreid voorbeeld....29
3.4.1. common.xsd....30
3.4.1.1. Volgorde bepalingen voor kindelementen....31
3.4.2. returntypes.xsd....33
3.4.3. void.xsd....36
3.5. Bespreking van andere mogelijkheden....36
3.5.1. Namespaces....37
3.5.2. import versus include....37
3.5.3. group....39
3.5.4. ref....41
3.5.5. use....41
3.5.6. anyType....42
3.5.7. witruimte....42
3.5.8. Element waarvan de inhoud ofwel string ofwel integer is....43
3.5.9. Gemengde inhoud....44
3.5.10. Globale versus lokale declaraties....44
3.5.11. commentaar en processing instructions....45
3.6. Omzetting van een DTD naar XML Schema....45
3.6.1. De DTD....45
3.6.2. DTD omzetten naar XML Schema....48
3.6.2.1. Conflicten....49
3.6.3. Bedenkingen en aanpassingen aan XML Schema....55
3.6.4. Automatische omzetting....59
3.7. Vergelijking tussen DTD en XML Schema....63
3.8. Conclusie....64
4. XML Schema van onze tekst....65
4.1. Keuze voor XML....65
4.2. Het schema van onze tekst....65
4.2.1. Enkele algemene definities....65
4.2.2. Algemene structuur....67
4.2.3. Algemene structuur van een hoofdstuk....68
4.2.3.1. Index/referentie elementen....71
4.2.3.2. URI gerelateerde elementen....71
4.2.3.3. Lijst gerelateerde elementen....72
4.2.3.4. Elementen IMAGE en QUOTE....73
4.2.3.5. TeX/LaTeX gerelateerde elementen....74
4.2.3.6. Andere elementen....75
4.2.4. Ontwikkeld XML formaat voor XML/HTML....75
4.2.5. Ontwikkeld XML formaat voor DTD....78

4.2.6. Ontwikkeld XML formaat voor Java....81
4.2.7. Ontwikkeld XML formaat voor litteratuurstudie....91
4.2.8. Ontwikkeld XML formaat voor bibliografie....92
4.2.9. Bedenking....93
4.3. Problemen: speciale tekens....93
5. XSL(T)....95
5.1. Wat is XSL?....95
5.2. Korte geschiedenis van XSL....95
5.2.1. Cascading Style Sheets (CSS)....95
5.2.2. XML Query Language....96
5.2.3. eXtensible Stylesheet for Transformation....96
5.2.4. XPath....96
5.3. XSL Transformaties....96
5.3.1. XSLT syntax....98
5.3.1.1. XSLT stylesheet....99
5.3.1.2. Templates....100
5.3.1.2.1. Matchpatronen....100
5.3.1.2.1.1. Matchen van het rootknooppunt....101
5.3.1.2.1.2. Matchen van elementnamen....101
5.3.1.2.1.3. Matchen met behulp van wild cards....101
5.3.1.2.1.4. Matchen van kindknooppunten....102
5.3.1.2.1.5. Matchen van nakomelingen....102
5.3.1.2.1.6. Matchen op ID....103
5.3.1.2.1.7. Matchen van attributen....103
5.3.1.2.1.8. Matchen van processing instructions....104
5.3.1.2.1.9. De of-operator....105
5.3.1.2.1.10. Predikaten....105
5.3.1.2.1.11. Matchen van commentaar....106
5.3.1.2.1.12. Matchen van tekstknooppunten....107
5.3.1.2.2. Het xsl:apply-templates element....107
5.3.1.2.2.1. Het select attribuut....108
5.3.1.2.3. Imperatief of declaratief?....110
5.3.1.3. Het berekenen van waardes....111
5.3.1.4. De default template regels....112
5.3.1.4.1. Regels voor elementen en het rootknooppunt....112
5.3.1.4.2. Regels voor tekstknooppunten en attributen....113
5.3.1.4.3. Regels voor processing instructions en commentaren....113
5.3.1.5. Toevoegen van markup aan de uitvoer....114
5.3.1.5.1. Elementen toevoegen....114
5.3.1.5.2. Attributen toevoegen....114
5.3.1.5.3. Definiëren van attribuutsets....115
5.3.1.5.4. Processing instructions toevoegen....116
5.3.1.5.5. Attribuutwaarde templates....117
5.3.1.5.6. Commentaar toevoegen....118

5.3.1.5.7. Tekst toevoegen....119
5.3.1.6. Modes....119
5.3.1.7. Keuzes maken....120
5.3.1.7.1. Het xsl:if element....120
5.3.1.7.2. Het xsl:choose, het xsl:when en het xsl:otherwise element....121
5.3.1.8. Kopiëren van knooppunten....122
5.3.1.9. Tellen van knooppunten....123
5.3.1.9.1. Defaultwaardes en hoe ze aan te passen....124
5.3.1.9.1.1. Het level attribuut....124
5.3.1.9.1.2. Het count attribuut....125
5.3.1.9.1.3. Het from attribuut....126
5.3.1.9.2. Omzetting van getal naar string....126
5.3.1.9.2.1. Het format attribuut....126
5.3.1.9.2.2. Het letter-value attribuut....128
5.3.1.9.2.3. Het grouping-separator en het grouping-size attribuut....128
5.3.1.10. Sorteren van uitvoerelementen....128
5.3.1.11. Variabelen, templates met een naam en parameters....129
5.3.1.11.1. Variabelen....130
5.3.1.11.2. Templates met een naam....131
5.3.1.11.3. Doorgeven van parameters....133
5.3.1.12. Sleutels....134
5.3.1.13. Witruimte....136
5.3.1.14. Meerdere stylesheets samenvoegen....137
5.3.1.14.1. Het xsl:import element....137
5.3.1.14.2. Het xsl:include element....138
5.3.1.14.3. Voorrangsregels....138
5.3.1.15. Uitvoermethoden....139
5.3.1.15.1. Het xsl:output element....140
5.3.1.15.1.1. Aanpassen van de XML declaratie....140
5.3.1.15.1.2. Indentatie....141
5.3.1.15.1.3. CDATA secties....141
5.3.2. Een uitgewerkt voorbeeld....142
5.4. XPath....146
5.4.1. Wat is XPath?....146
5.4.2. De context....147
5.4.3. Locatiepaden....147
5.4.3.1. Locatiestappen....148
5.4.3.1.1. Axes....149
5.4.3.1.2. Knooppunttesten....151
5.4.3.1.3. Predikaten....151
5.4.3.1.4. Verkorte syntax....152
5.4.4. XPath Expressies....153
5.4.4.1. Sets van knooppunten....153
5.4.4.2. Booleans....155
5.4.4.3. Getallen....157

5.4.4.4. Strings....158
5.4.4.5. Resulterende boomfragmenten....159
5.5. Conclusie....159
6. Gebruik van XSLT in onze tekst....161
6.1. Navigatie in HMTL....161
6.1.1. Opbouw initiële pagina....162
6.1.2. Verwerken van URL parameters....163
6.1.3. Tonen van de juiste sectie....165
6.1.4. Besluit....166
6.2. Referenties in de tekst....166
7. XSL-FO....170
7.1. Wat is XSL-FO?....170
7.2. Korte geschiedenis van XSL-FO....171
7.3. Een transformatieproces in twee stappen....171
7.3.1. Transformatie....173
7.3.2. Vormgeving....174
7.3.3. Enkele belangrijke concepten van vormgeving....175
7.4. XSL-FO namespace....177
7.5. XSL vormgevingsobjecten en hun eigenschappen....177
7.5.1. Vormgevingseigenschappen....183
7.5.2. Transformatie naar vormgevingsobjecten....188
7.6. De lay-out van een pagina....190
7.6.1. Het rootelement....190
7.6.2. Het fo:simple-page-master element....190
7.6.2.1. Regio kindelementen....192
7.6.2.2. Eigenschappen....193
7.6.3. Het fo:page-sequence element....193
7.6.3.1. Het fo:flow element....194
7.6.3.2. Het fo:static-content element....195
7.6.3.3. Paginanummering....196
7.6.4. Het fo:page-sequence-master element....198
7.6.4.1. Het fo:single-page-master-reference element....199
7.6.4.2. Het fo:repeatable-page-master-reference element....200
7.6.4.3. Het fo:repeatable-page-master-alternatives element....200
7.7. De inhoud....201
7.7.1. Vormgevingsobjecten van blokniveau....202
7.7.2. In-regel vormgevingsobjecten....202
7.7.3. Tabelvormgevingsobjecten....203
7.7.4. Lijstvormgevingsobjecten....203
7.7.5. Uit-regel vormgevingsobjecten....203
7.8. Lijnen en leaders....204
7.9. Grafische objecten....205
7.9.1. Het fo:external-graphic element....205

7.9.2. Het fo:instream-foreign-object element....206
7.9.3. Grafische eigenschappen....207
7.9.3.1. Het content-type attribuut....207
7.9.3.2. Grootte....208
7.9.3.3. Scalering....208
7.10. Lijsten....209
7.11. Tabellen....211
7.12. In-regel objecten....214
7.13. Voetnoten....215
7.14. Floats....215
7.15. Vormgevingseigenschappen....216
7.15.1. De id eigenschap....216
7.15.2. De taal eigenschap....217
7.15.3. Blokeigenschappen....217
7.15.3.1. Brekingseigenschappen....217
7.15.3.2. Eigenschappen voor het gebruik van koppeltekens....219
7.15.3.3. Indentatie-eigenschappen....219
7.15.4. Eigenschappen van tekens....220
7.15.4.1. Het color attribuut....220
7.15.4.2. Lettertype-eigenschappen....221
7.15.4.3. Teksteigenschappen....221
7.15.5. Eigenschappen van zinnen....222
7.15.5.1. De letter-spacing eigenschap....222
7.15.5.2. De word-spacing eigenschap....223
7.15.5.3. Eigenschappen voor tussenregelruimtes....223
7.15.5.4. Eigenschappen voor het aligneren van tekst....224
7.15.5.5. White space eigenschappen....224
7.15.6. Eigenschappen van gebieden....225
7.15.6.1. Achtergrondeigenschappen....225
7.15.6.2. Kadereigenschappen....225
7.15.6.3. Paddingeigenschappen....226
7.15.6.4. Kantlijneigenschappen....226
7.15.6.4.1. Kantlijneigenschappen voor blokken....226
7.15.6.4.2. Kantlijneigenschappen voor in-regel objecten....228
7.15.6.5. Grootte-eigenschappen....228
7.15.6.6. De overflow eigenschap....229
7.15.6.7. De reference-orientation eigenschap....229
7.15.6.8. De writing-mode eigenschap....230
7.15.6.9. Weduwen en wezen....230
7.16. Conclusie....230
8. Gebruik van XSL-FO in onze tekst....232
8.1. Genereren van de inhoudstafel....232
8.2. Gebruik van page-sequences....234
8.3. Tekortkomingen van FOP....236

8.4. Besluit....237
9. Cocoon XML publishing framework....239
9.1. Cocoon....239
9.1.1. Wat is Cocoon?....239
9.1.2. Werken met Cocoon....239
9.1.2.1. Cocoon installeren....239
9.1.2.2. Werking van Cocoon nader toegelicht....240
9.1.2.2.1. Uitleg over de processing instruction cocoon-format....241
9.1.2.2.1.1. Locatie van de verwerkingsinstructies....242
9.1.2.3. Geavanceerde mogelijkheden....243
9.1.2.3.1. Output op basis van een browser agent....243
9.1.2.3.2. Dynamisch gegenereerde inhoud....244
9.1.2.3.3. Andere mogelijkheden....245
9.1.3. Conclusie....246
9.2. Cocoon 2....247
9.2.1. Inleiding....247
9.2.2. Installatie van Cocoon 2....248
9.2.3. Werking van Cocoon 2....248
9.2.3.1. De filosofie van Cocoon 2....249
9.2.3.2. Cocoon 2 intern....250
9.2.3.2.1. Pipeline....250
9.2.3.2.2. Componenten....250
9.2.3.2.3. Sitemap....252
9.2.3.2.3.1. Componenten: algemeen....252
9.2.3.2.3.2. Componenten: generators....253
9.2.3.2.3.3. Componenten: readers....254
9.2.3.2.3.4. Componenten: transformers....254
9.2.3.2.3.5. Componenten: actions....255
9.2.3.2.3.6. Componenten: serializers....255
9.2.3.2.3.7. Componenten: matchers....255
9.2.3.2.3.8. Componenten: selectors....256
9.2.3.2.3.9. Componenten: definitie van de gebruikte componenten....256
9.2.3.2.3.10. Pipelines....257
9.2.3.2.3.10.1. XInclude in de praktijk....260
9.2.3.2.3.11. Subsitemaps....265
9.2.3.2.3.12. Meerdere pipeline elementen....267
9.2.4. Vergelijking met Cocoon 1....268
9.2.5. Conclusie....270
10. Gebruik van Cocoon 2....271
10.1. De sitemap....271
10.2. Valkuilen....273
10.3. Besluit....274

11. JDOM.....275
11.1. Wat is JDOM?.....275
11.2. Waarom JDOM en een vergelijking met bestaande API's.....275
11.2.1. DOM versus JDOM.....275
11.2.2. SAX versus JDOM.....276
11.2.3. JDOM kort samengevat.....277
11.3. JDOM syntax.....277
11.3.1. XML lezen van een bestaande bron.....277
11.3.2. Opbouwen van een JDOM document.....278
11.3.2.1. Creatie van een document.....278
11.3.2.2. Elementen en sub-Elementen.....278
11.3.2.3. Een attribuut toevoegen.....279
11.3.2.4. Commentaar toevoegen.....279
11.3.2.5. Processing instructions toevoegen.....280
11.3.2.6. Volledig uitgewerkt voorbeeld.....280
11.3.3. Manipuleren van een JDOM document.....281
11.3.3.1. Selecteren van kinderen.....281
11.3.3.2. Verwijderen van kinderen.....282
11.3.4. Het genereren van uitvoer via JDOM.....282
11.4. Conclusie.....283

12. XML API's.....284
12.1. DOM.....284
12.1.1. Voorbeeld: DOM boomstructuur.....285
12.2. SAX.....288
12.2.1. Implementatie van een event handler.....289
12.2.2. Creatie van een SAXParser.....291
12.2.3. Associatie tussen SAXParser en event handler.....292
12.2.4. Parsen van een document.....293
12.2.5. Slotbeschouwing.....294
12.3. JAXP.....294
12.3.1. Aanmaken van parser via JAXP.....295
12.3.1.1. Aanmaken van een SAXParser.....295
12.3.1.2. Aanmaken van een DOMParser.....296

13. jnome.....298
13.1. jnome: kort overzicht.....298
13.1.1. Korte beschrijving van het metamodel voor Java.....298
13.1.1.1. Aandachtspunten.....298
13.1.1.2. Het model.....298
13.1.1.2.1. Pakketten.....299
13.1.1.2.2. Compilation units.....300
13.1.1.3. Variabelen/Methodes/Documentatie commentaar.....301
13.1.1.4. Unresolved elementen.....301

13.1.1.5. Een voorbeeld van een objectstructuur....303
13.1.2. Werking van jnomedoc....303
13.1.2.1. De drie fases....303
13.1.2.1.1. Eerste fase: lexen en parsen....303
13.1.2.1.2. Tweede fase: opbouwen objectmodel....304
13.1.2.1.3. Derde fase: uitvoer genereren....305
13.2. jnome in onze thesis....306
13.2.1. Uitvoer....306
13.2.2. Omzetting van de DTD's....310
13.2.2.1. Het vertrekpunt: de bestaande DTD's....310
13.2.2.2. Omzetting naar XML Schema....310
13.2.3. jnome koppelen aan Cocoon 2....311
13.2.4. De toekomst....316
14. Vergelijking tussen Java en XML/XSLT....318
14.1. De parallellen....318
14.1.1. Algemene overeenkomsten....318
14.1.2. XPath en Java....318
14.1.3. Overerving....318
14.1.3.1. Overerving en XML Schema....318
14.1.3.1.1. xsd:include en xsd:redefine....319
14.1.3.1.2. xsd:import....319
14.1.3.1.3. implements....320
14.1.3.1.4. abstract....320
14.1.3.2. Overerving en XSLT....320
14.1.3.2.1. xsl:include....320
14.1.3.2.2. xsl:import....321
14.1.4. Polymorfisme en dynamische binding....321
14.1.4.1. De concepten in de context van XML Schema....321
14.1.4.2. De concepten in de context van XSL(T)....322
14.1.5. XSLT variabelen en Java....322
14.2. Besluit....323
15. Toekomst....324
15.1. jnome en jnomedoc....324
15.2. XMLdoc....324
15.3. Java en XML....325
15.4. Koppeling XSLT en XML Schema....325
15.5. XSL-FO....325
15.6. Besluit....326
16. Besluit....327
16.1. Bestudeerde onderwerpen....327
16.1.1. XML en XML Schema....328
16.1.2. XML en XSLT....328

16.1.3. XML en XSL-FO.....328
16.1.4. Cocoon 2.....328
16.1.5. jnome en jnomedoc.....329
16.1.6. XML en Java.....329

Appendix A. Documentformaten.....331
A.1. Een bespreking van DocBook.....331
A.1.1. Wat is DocBook?.....331
A.1.2. Hoe DocBook gebruiken?.....331
A.1.3. Ervaringen met DocBook.....332
A.2. TeX en LaTeX.....333
A.2.1. Oorsprong.....333
A.2.2. Hoe TeX gebruiken?.....334
A.2.3. Ervaringen met TeX en LaTeX.....335
A.3. Onze afwegingen.....335
A.4. Ons zelfgedefinieerd documentformaat.....336
Appendix B. XSLT stylesheets en browsers.....348
B.1. XSLT stylesheets en Internet Explorer.....348
Appendix C. HOW-TO: Cocoon 1 installeren.....350
C.1. Vereiste packages.....350
C.2. Vereiste pakketten installeren - JServ 1.1.2.....350
C.2.1. Vereiste pakketten installeren - JSDK.....350
C.3. Vereiste packages installeren - JServ 1.1.2 - vervolg.....351
C.3.1. Configuratie-opties (voor DSO).....351
C.3.2. Installeren van Apache JServ 1.1.2.....352
C.4. Cocoon 1.8.2 installeren.....353
Appendix D. Installatie van Cocoon 2.....357
D.1. Machinegegevens.....357
D.2. Vereiste downloads.....357
D.3. Installeren van Tomcat.....357
D.4. Installeren van Cocoon 2.....358
D.5. Jakarta Tomcat afstemmen op Cocoon 2.....358
D.6. Besluit.....359
Appendix E. How to write your own Generator for Cocoon 2.....360
E.1. Preface.....360
E.2. Planning.....361
E.3. Our planning, step by step.....361
E.3.1. Classes and interfaces to be extended/implemented.....361
E.3.1.1. Classes.....361
E.3.1.1.1. AbstractGenerator.....362
E.3.1.1.2. ComposerGenerator.....365
E.3.1.1.3. ServletGenerator.....366
E.3.1.1.4. AbstractServerPage.....366
E.3.1.2. Interface(s).....366

E.3.1.2.1. Generator....366
E.3.2. Writing a test generator....367
E.3.2.1. The code of our first generator....367
E.3.2.2. Deploying MyGenerator....370
E.3.2.3. Considerations afterwards....372
E.3.3. Going the distance....373
E.3.3.1. Setting up a RMI server....373
E.3.3.2. Setting up a RMI client....375
E.3.3.3. Testing the RMI components....376
E.3.3.4. Putting the pieces together....378
E.3.3.5. The final step: deployment....380
E.4. Future plans....381

Litteratuurstudie 382
Bibliografie 395

1. Inleiding

1.1. Het doel van onze thesis

Het doel van deze thesis is een verkennende studie uitvoeren over de mogelijkheden van XML, XSL (bestaande uit XSLT, XSL-FO en XPath) en aanverwante onderwerpen zoals XML Schema, Cocoon 1 en 2, JDOM, DOM, SAX en JAXP. Wij hebben in de daarvoor voorziene tijdsspanne geprobeerd een zo breed mogelijke kijk op de mogelijkheden van XML en XSL te krijgen. Aan de hand van een viertal case-studies hebben wij de door ons bestudeerde materie verwerkt en in de praktijk toegepast.

1.2. Inhoud van de tekst

Onze thesistekst is opgesplitst in twee delen. Het eerste deel omvat de volgende hoofdstukken:

- XML [Hoofdstuk 2.]
- XML Schema [Hoofdstuk 3.]
- XML Schema van onze tekst [Hoofdstuk 4.]
- XSL(T) [Hoofdstuk 5.]
- Gebruik van XSLT in onze tekst [Hoofdstuk 6.]
- XSL-FO [Hoofdstuk 7.]
- Gebruik van XSL-FO in onze tekst [Hoofdstuk 8.]

Het tweede deel bestaat uit de volgende hoofdstukken:

- Cocoon XML publishing framework [Hoofdstuk 9.]
- Gebruik van Cocoon 2 [Hoofdstuk 10.]
- JDOM [Hoofdstuk 11.]
- XML API's [Hoofdstuk 12.]
- jnome [Hoofdstuk 13.]
- Vergelijking tussen Java en XML/XSLT [Hoofdstuk 14.]
- Toekomst [Hoofdstuk 15.]
- Besluit [Hoofdstuk 16.]

Het tweede deel bouwt verder op de theorie over XML, XML Schema, en XSLT die wij in het eerste deel van onze thesistekst uiteengezet hebben en illustreert enkele toepassingen die van deze theorie gebruik maken.

Wij hebben ervoor gekozen telkens een vrij theoretisch hoofdstuk te laten afwisselen met een praktisch hoofdstuk. In de meer theoretische hoofdstukken komen de verschillende mogelijkheden en technieken van een bepaald onderwerp aan bod. In de praktische hoofstukken illustreren wij hoe wij deze theorie in praktijk toegepast hebben

1.2.1. Eerste deel van de tekst

We beginnen met een kort hoofdstuk over **XML** [TRXML]. In dit hoofdstuk worden de basisideeën van XML uitgelegd. Dit hoofdstuk wordt gevolgd door een hoofdstuk over **XML Schema** [SQHTSCHEMA], waarin uitgebreid aan bod komt hoe een XML Schema opgesteld wordt. Er wordt ook een vergelijking gemaakt tussen XML Schema en DTD's [RDW3S]. Na deze twee eerder theoretische hoofdstukken, wordt in het hoofdstuk over het XML Schema van onze tekst uitgelegd hoe wij onze tekst aan de hand van XML gemodelleerd hebben. Wij

hebben bijvoorbeeld een XML formaat ontwikkeld om XML documenten zelf te beschrijven. Het volledige XML formaat van onze tekst hebben wij vastgelegd met behulp van XML Schema, zodat het mogelijk is toekomstige uitbreidingen aan onze tekst te valideren met dit schema.

Vervolgens is er een hoofdstuk over **XSLT** [TRXSLT] dat de mogelijkheden en de gebruikte syntax in detail beschrijft. Ook dit hoofdstuk wordt weer gevolgd door een meer praktisch hoodfstuk waarin wij illustreren hoe wij van XSLT gebruik gemaakt hebben om onze tekst te transformeren naar een HTML formaat [TRHTML].

Na de hoofdstukken over XSLT bespreken wij **XSL-FO** [TRXSL], een XML woordenschat die het mogelijk maakt vormgevingsinformatie aan XML documenten te verbinden. Via een bijkomende omzetting uitgevoerd door een Formatting Objects Processor kunnen XSL-FO documenten omgevormd worden naar een formaat geschikt voor een bepaald medium. In het hoofstuk over het gebruik van XSL-FO in onze tekst, illustreren wij hoe wij XSL-FO gebruikt hebben om onze tekst in XSL formaat om te zetten naar PDF [ZPDF].

1.2.2. Tweede deel van de tekst

Het tweede deel begint met een uiteenzetting over het XML publishing framework **Cocoon**. Wij hebben dit framework gebruikt om onze XML documenten naar andere formaten om te zetten. Dit hoofdstuk beschrijft zowel **Cocoon 1** [COCOON1] als **Cocoon2** [COCOON2], omdat wij beide frameworks gebruikt hebben. In het aansluitende hoofdstuk over het gebruik van Cocoon 2, gaan wij dieper in op hoe wij dit framework concreet aangewend hebben. We bespreken ook de voor- en nadelen die wij ondervonden hebben bij het gebruik van Cocoon 2.

Na de hoofdstukken over Cocoon volgen twee kortere hoofdstukken. Het eerste hoofdstuk heeft als onderwerp **JDOM** [JDOM], een Application Programming Interface, speciaal ontworpen om XML gegevens te gebruiken en te manipuleren vanuit Java code [JAVA]. In het daarop volgende hoofstuk bespreken we enkele andere veelgebruikte API's: **DOM** [TRDOM], **SAX** [SAX] en **JAXP** [JAXP]. Van deze drie API's besteden we veruit het meeste aandacht aan SAX omdat deze API een centrale rol speelt in Cocoon 2. Het volgende hoofdstuk behandelt **jnome** [JNOME], een open source project dat een metamodel biedt voor Java. In dit hoofdstuk wordt behandeld welke aanpassingen en uitbreidingen wij gedaan hebben aan **jnomedoc**, een onderdeel van het jnome project dat XML documentatie genereert voor Java bronbestanden.

We wijden ook nog een hoofdstuk aan het bespreken van de parallellen die er volgens ons tussen Java en XML/XSLT bestaan. We sluiten onze thesis af met enkele formuleringen over mogelijke toekomstige uitbreidingen aan deze thesis en ronden het geheel af met een algemeen besluit.

1.3. De gebruikte voorbeelden

In de loop van deze tekst gebruiken wij vaak voorbeelden die afkomstig zijn uit de verschillende projecten waaraan wij in de loop van het jaar gewerkt hebben. In deze sectie behandelen wij kort de verschillende case-studies zodat wij hieraan kunnen refereren in het vervolg van de tekst.

1.3.1. Omzetten van de OOP website naar XML/XSL

Ons eerste project bestond uit het (gedeeltelijk) omzetten van de website voor de cursus

Object Oriented Programming. Deze site bestaat eigenlijk uit twee, licht verschillende sites: een site voor de Masters of Artficial Intelligence opleiding[1] en een site voor de business cursus van OOP[2].

Het omzetten van de hele site naar XML/XSL, bleek een quasi onmogelijke opgave. Daarom besloten wij ons te concentreren op een onderdeel van de OOP site dat klein genoeg was om het overzicht niet kwijt te geraken, maar toch genoeg mogelijkheden bood om de verschillende aspecten van XML en XSL te bestuderen. Onze keuze viel op de menustructuur die gebruikt wordt om door de verschillende onderdelen van de site te navigeren. Het XML formaat en de stylesheets die wij voor deze menubalken ontwikkeld hebben, worden als voorbeeld gebruikt in de hoofdstukken over XML [Sectie 2.5., Voorbeeld 2.11.] en XSL [Sectie 5.3.2., Voorbeeld 5.92.] .

Voor het omzetten van deze website van XML naar HTML hebben wij geëxperimenteerd met zowel Cocoon1 als Cocoon2. Onze bevindingen op dit vlak vind je terug in de hoofdstukken over Cocoon1 en Cocoon2. Onze conclusie is dat Cocoon2 een aantal tekortkomingen van Cocoon1 heeft opgelost. De grootste verwezenlijking van Cocoon2 is volgens ons de **sitemap**. Deze maakt het mogelijk om een XML document te koppelen aan een welbepaalde XSL stylesheet voor de omzetting naar HTML, PDF of een ander formaat. In Cocoon1 is dit niet mogelijk en moet in het XML document zelf aangegeven worden welke stylesheet gebruikt wordt voor de verwerking.

1.3.2. De slideshow van onze presentatie

Ons tweede project is de thesispresentatie die wij in december hebben gegeven voor de leden van de SOM onderzoeksgroep. Het doel van deze presentatie was een degelijke inleiding over XML, XSL en JDOM geven. Deze presentatie had als doelpubliek mensen die nog niet vertrouwd waren met deze begrippen. Als belangrijkste illustratie hebben wij onze eigen slideshow in XML formaat en de bijhorende stylesheets gebruikt.

Wij hebben XSLT stylesheets geschreven om de slideshow om te zetten naar HTML en XSL-FO stylesheets om ze om te zetten naar PDF. Uiteindelijk hebben wij de PDF versie van de slides gebruikt om de presentatie te geven en de HTML stylesheets als illustratie van de belangrijkste begrippen. Deze HTML stylesheets van de slideshow worden gebruikt als voorbeeld in het hoofdstuk over XSLT [Sectie 5.3.1.3., Voorbeeld 5.32.] .

1.3.3. jnome en JDOM

Het derde project bouwde voort op een thesis van vorig jaar. In deze thesis werd een metamodel opgebouwd van Java code [JAVA] en de bijhorende documentatie. Het oorspronkelijke project heette SoFT, maar nog niet zo lang geleden is het herdoopt naar jnome [JNOME]. jnomedoc (de concrete toepassing van jnome) begint in de eerste fase met het verwerken (lexen en parsen) van de Java code, vervolgens wordt het metamodel opgebouwd en de derde stap is het genereren van de documentatie voor de Java code. Wij hebben ons vooral geconcentreerd op die derde stap.

De documentatie die door jnomedoc gegenereerd wordt staat in XML formaat. Wat wij gedaan hebben is de oorspronkelijke **println**'s in de tekst vervangen door JDOM functies. Dit zorgt ervoor dat de XML documenten voortaan met JDOM opgebouwd worden en de programmeur zich geen zorgen moet maken over het genereren van goedgevormde XML.

[1] *http://www.cs.kuleuven.ac.be/cwis/research/som/education/OOP/*

[2] *http://www.cs.kuleuven.ac.be/cwis/research/som/services/courses/InProgress/CAPCO-20001120-24/*

JDOM neemt dit voor zijn rekening. Voor de voorbeelden in het hoofdstuk over JDOM hebben wij ons dan ook voornamelijk gebaseerd op de code die wij voor het jnome project geschreven hebben. Wij hebben ook de DTD's waaraan de gegenereerde bestanden moesten voldoen, omgezet naar XML Schema. Tenslotte hebben we ook een eigen generator geschreven voor Cocoon 2 zodat de gegenereerde XML documentatie rechtstreeks aan Cocoon 2 kon afgeleverd worden [Sectie 13.2.3.] . Op die manier hebben we een tussenstap, met name het bewaren van de bestanden op schijf, geëlimineerd. Wij komen in het hoofdstuk over jnome [Hoofdstuk 13.] nog uitgebreider op deze materie terug.

1.3.4. Onze thesistekst in XML formaat omzetten naar HMTL en PDF

Onze laatste en grootste case-study is in feite onze thesistekst zelf. Wij hebben hiervoor een eigen XML formaat ontwikkeld en dit formaat in XML Schema vastgelegd. Onze tekst wordt geserveerd met Cocoon2 en kan omgezet worden naar zowel HTML als PDF. Wij hebben hier twee soorten stylesheets voor geschreven. Zuivere XSLT stylesheets om de omzetting naar HTML te doen en XSL-FO stylesheets om de omzetting naar PDF te doen.

Het formaat van onze tekst is geleidelijk aan gegroeid doordat we tijdens het schrijven nood hadden aan extra elementen. Zo hebben we niet alleen een documentformaat ontwikkeld dat in grote lijnen uit hoofdstukken (**CHAPTER**), secties (**SECTION**) en paragrafen (**PARA**) bestaat, maar ook een formaat om XML en XSL zelf voor te stellen. We hebben ook een XML formaat ontwikkeld voor de Javacode die wij als voorbeeld gebruiken in onze hoofdstuk over JDOM en het hoofstuk over de andere API's zoals DOM, SAX en JAXP. Wij bespreken het formaat van onze thesistekst in een afzonderlijk hoofdstuk [Hoofdstuk 4.] .

Het XML formaat van onze tekst wordt heel vaak als voorbeeld gebruikt omdat de stylesheets die wij hiervoor geschreven hebben vrij omvangrijk zijn en het dus niet moeilijk was daaruit een geschikt voorbeeld te halen om een bepaald aspect te illustreren. Je vind deze voorbeelden terug onder andere in het hoofdstuk over XML, XSL [Sectie 5.3.1.6., Voorbeeld 5.51.] en XSL-FO [Sectie 7.6.3.3., Voorbeeld 7.8.] . Vooral voor het hoofdstuk XSL-FO zijn deze voorbeelden belangrijk, omdat de omzetting van onze tekst naar PDF ons enige experiment met XSL vormgevingsobjecten is. Wij hebben over het gebruik van XSLT en XSL-FO in onze tekst nog twee afzondelijke hoofdstukken geschreven, dit vooral omdat deze stylesheets zo'n belangrijk onderdeel van onze studie vormen. [Hoofdstuk 6.] , [Hoofdstuk 7.]

Wij hebben van Cocoon 2 gebruik gemaakt om de XML documenten van onze tekst naar HTML of PDF om te zetten. Daarom hebben we nog een extra hoofdstuk geschreven dat zich voornamelijk concentreert op de aanpassingen die wij in de sitemap hebben aangebracht om deze omzetting te realiseren. [Hoofdstuk 10.]

Tijdens het lezen van de thesis zal het u misschien opvallen dat de opmaak op sommige gebieden nog niet helemaal correct is. We hebben geprobeerd dit zo goed mogelijk op te lossen, maar de slotsom was dat een aantal eigenschappen nog niet door FOP ondersteund werden. FOP is de component die door Cocoon 2 gebruikt wordt om PDF documenten te genereren.

1.4. Bijlagen

Bij deze thesistekst hoort een lijst van gelezen artikels en een CDROM.

Achteraan de thesistekst hebben wij een opsomming toegevoegd van (een deel van) de artikels die wij gelezen hebben. Deze opsomming bevat voor elk artikel een korte

beschrijving van de inhoud, een bespreking en een cijfer op tien dat een waarde-oordeel over dit artikel uitdrukt. Wij denken dat deze lijst met artikels zeer nuttig kan zijn voor mensen die zich willen verdiepen in XML/XSLT. Het is niet altijd evident om uit de massa artikels die op het Internet te vinden zijn, de meest relevante te filteren. Deze opsomming wil dit proces vereenvoudigen.

De CDROM die bij deze thesistekst gevoegd is bevat:

- de HTML en PDF versie van de tekst
- de originele XML en XSLT/XSL-FO bestanden aan de hand waarvan de HTML en PDF bestanden gegenereerd werden
- het herwerkte jnome project
- de volledige inhoud van de CVS met al onze projecten

2. XML

In dit hoofdstuk geven we een beknopt overzicht van XML en hoe XML te gebruiken. XML is het fundament waarop de rest van deze thesis gebouwd wordt en waarvan al onze praktische toepassingen gebruik maken. We beginnen dit hoofdstuk met uit de doeken te doen wat er nu juist schuil gaat achter dit drieletterwoord, dat de laatste tijd nogal een modewoord geworden is. Verder geven we een kort overzicht van de ontstaansgeschiedenis van XML en illustreren we aan de hand van kleine voorbeeldjes uit de praktijk hoe de syntax in mekaar zit. We sluiten het hoofdstuk af met een groter voorbeeld en enkele algemene conclusies.

Voor dit hoofdstuk baseerden wij ons op [JERXML], [DTXML], [AMMISXML], [RDXML], [SVXML] en [TRXML].

2.1. Wat is XML?

XML staat voor eXtensible Markup Language en is een World Wide Web Consortium [W3C] standaard. XML is een meta-markuptaal (een taal die gebruikt wordt om andere talen te beschrijven).

XML is ontwikkeld om structuur op te leggen aan de informatie vervat in een document. Nemen we als voorbeeld van een document een lijst van werknemers van een bedrijf met hun respectievelijke adressen. Zo een document bevat meestal zowel inhoud (woorden, beelden) als een indicatie over de rol die deze inhoud speelt in het document. Toegepast op het voorbeeld van onze lijst met werknemers, wil dit zeggen dat de rol van bepaalde woorden in het document de naam van de werkgever is, weer andere woorden hebben als rol de straatnaam, het huisnummer, enzovoort.

Om de rol van een bepaald onderdeel van je document aan te geven, maakt XML gebruik van tags. Je kan dit vergelijken met de tags die in HTML [TRHTML] gebruikt worden. Er zijn wel enkele verschillen. HTML tags geven aan hóe een document eruit moet zien (bijvoorbeeld <b> geeft aan dat de tussenliggende tekst vet moet weergegven worden, maar dit zegt niets over de structuur), ze hebben een vaste betekenis en bestaan uit een op voorhand vastgelegde set. In XML daarentegen is het mogelijk om je eigen tags en je eigen documentstructuur te definiëren. De naam die je voor je tags kiest is volkomen willekeurig, maar het verdient aanbeveling een semantische betekenis aan je tags te geven. Om de verschillende hoofdstukken in een boek aan te duiden, neem je bijvoorbeeld de <CHAPTER> tag. XML is dus een eenvoudige manier om structuren te definiëren in je documenten. Als je een bepaalde vaststaande structuur wilt opleggen aan je documenten, dan kan je dit doen aan de hand van een DTD (Document Type Definition) [RDW3S] of een XML Schema, waarover later meer [Hoofdstuk 3.] .

XML documenten worden gewoon opgeslagen in tekstformaat. XML is dus niet afhankelijk van de gebruikte software of hardware, wat een belangrijk voordeel oplevert voor het uitwisselen en delen van gegevens tussen verschillende applicaties. Bijvoorbeeld twee verschillende programma's die voor hun communicatie gebruik maken van een overeengekomen XML formaat (vastgelegd in een XML Schema of een DTD). Of databases die, zonder dat er een omzetting nodig is, elkaars gegevens die in XML opgeslagen zijn, kunnen lezen.

De software module die gebruikt wordt om XML documenten te lezen en die toegang verschaft tot hun inhoud en structuur, wordt een XML processor genoemd. In het algemeen wordt aangenomen dat een XML processor werkt in dienst van een andere module die de

applicatie genoemd wordt.

2.2. De geschiedenis van XML in het kort

Om XML ten volle te appreciëren, is het belangrijk te weten waarom het oorspronkelijk ontworpen is, namelijk om rijkgestructureerde documenten over het web te kunnen gebruiken. De enige twee alternatieven HTML (Hypertext Markup Language) en SGML (Standard Generalized Markup Language, ISO 8879), zijn niet geschikt voor dit doeleinde. HTML, dat rond 1990 in CERN [CERN] is ontworpen als een zeer eenvoudige versie van SGML [RCSGML], is niet geschikt omdat HTML tags een vaststaande semantiek hebben en het dus niet mogelijk is een willekeurige structuur te definiëren. SGML, dat gebruikt wordt sinds de jaren '80, bezit wel de mogelijkheid om een willekeurige structuur te definiëren, maar is dan weer zeer complex en dus moeilijk te implementeren voor een webbrowser. Bovendien zijn de ontwikkelingskosten van SGML documenten zeer hoog en kosten de tools om ermee te werken veel geld. Volledige SGML systemen lossen grote, complexe problemen op, wat de hogere kosten rechtvaardigt. Om gestructureerde documenten over het web te kunnen bekijken is deze kost echter te groot.

In 1996 ontstonden discussies over het ontwerpen van een markup taal met de kracht en uitbreidbaarheid van SGML, maar met de eenvoud van HTML. Het World Wide Web Consortium [W3C] besliste om een groep SGML goeroes waaronder Jon Bosak van Sun, te sponsoren. Wat Bosak en zijn team met SGML deden, is te vergelijken met wat het Java team gedaan heeft met C++. Alle niet-essentiële, ongebruikte en cryptische delen van SGML werden weggesneden. Wat overbleef was een "lean, mean marking up machine": XML . De specificatie van XML (voor het grootste deel geschreven door Tim Bray en C.M. Sperberg-McQueen) bestond uit slechts zesentwintig pagina's in tegenstelling tot de meer dan vijfhonderd pagina's van de SGML specificatie. Het ziet er echter niet naar uit dat XML op termijn SGML volledig zal vervangen. XML is vooral ontworpen om het afleveren van gestructureerde inhoud over het web te vergemakkelijken, dit omdat SGML hiervoor niet zo geschikt is. Voor de creatie en de lange termijn opslag van complexe documenten is SGML wel een goeie keuze. Voor gedetailleerde informatie over de verschilpunten tussen SGML en XML verwijzen wij naar [JCW3].

In de jaren na 1996 evolueerde XML verder. XML kon de vruchten plukken van de bijdragen van zijn sponsors en het werk van ontwikkelaars die met gelijkaardige problemen bezig waren zoals Peter Murray-Rust die werkte aan CML (Chemical Markup Language) en het consortium van mensen die werkten aan MathML. Midden 1997 was het eXtensible Linking Language (XLL) project op komst en in de zomer van 1997 lanceerde Microsoft het Channel Definition Format (CDF), één van de eerste real-world applicaties van XML. In maart 1998 werden de specificaties van het W3C voor XML link-, wijs- en adresseringsmechanismen, op dat moment samengebracht onder de naam "Extensible Linking Language (XLL)" hernoemd naar XLink. (Momenteel staat XLL voor de verzameling van XLink, XPointer en XPath.) [RCXLL] In 1998 werd dan uiteindelijk Versie 1.0 van de XML specificatie door het W3C goedgekeurd. Het begin van een nieuw tijdperk... [SSWD]

2.3. XML Syntax

Vooraleer we ingaan op de exacte XML syntax, vermelden we eerst een aantal algemene regels die je altijd in het achterhoofd moet houden als je met XML bezig bent (begrippen die hierbij gebruikt worden, worden hieronder verklaard):

- Elke starttag van een XML element moet een corresponderende eindtag hebben.
- XML tags zijn hoofdlettergevoelig, dit wil zeggen dat er een onderscheid gemaakt wordt tussen hoofdletters en kleine letters.

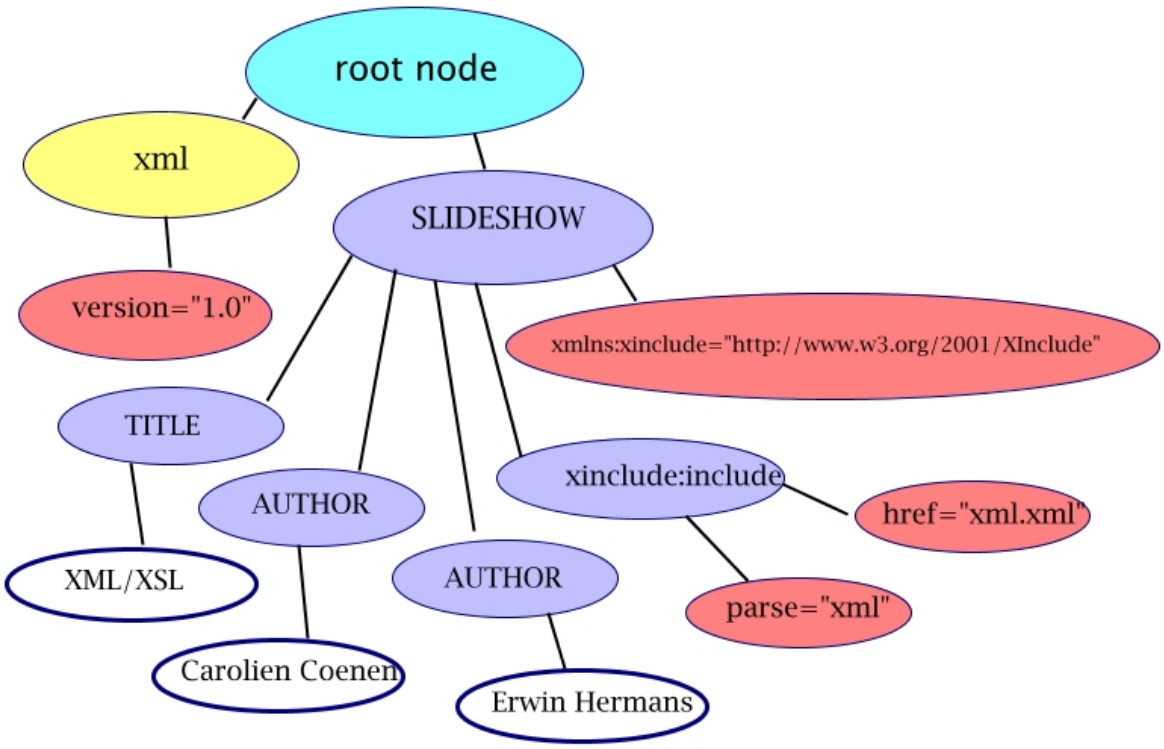
- Alle XML elementen moeten op de juiste manier genest zijn; als je binnen een <a> tag een <b> tag nest, dan moet die <b> tag ook binnen de <a> tag afgesloten worden.
- Alle XML documenten moeten exact één root(=wortel)element hebben. Dit rootelement is een kind van het rootknooppunt. Het rootknooppunt komt overeen met het XML document zelf.
- Attribuutwaardes moeten altijd tussen aanhalingstekens staan.
- Een attribuut mag niet meer dan één keer bij dezelfde starttag staan en nooit bij een eindtag.

Rekening houdend met deze regels, kunnen we drie soorten XML documenten onderscheiden:

- Ongeldige documenten (invalid documents): Documenten die de regels die wij zonet besproken hebben niet volgen of documenten die een DTD of een XML schema hebben en de regels van deze DTD of dit XML Schema niet volgen.
- Goedgevormde documenten (well-formed documents): Documenten die de XML tag regels volgen, maar die geen DTD of XML Schema hebben.
- Geldige documenten (valid documents): Documenten die zowel de XML tag regels, als een DTD of XML schema volgen.

Ieder goedgevormd XML document is een boom. Een boom is een datastructuur die samengesteld is uit verbonden knooppunten (nodes). Het knooppunt van het hoogste niveau is de wortel (root) van de boom, deze is verbonden met zijn kindknooppunten (child nodes), welke op hun beurt weer verbonden zijn met hun eigen kindknooppunten, enzovoorts. Knooppunten die geen kinderen hebben worden bladeren (leaves) genoemd. De belangrijkste eigenschap van een boom is dat elke knoop en zijn kinderen op zichzelf weer een boom vormen. Dus een boom is een hiërarchische structuur van bomen waarin elke boom uit kleinere bomen is opgebouwd.

Hieronder geven we een voorbeeld van zo'n boomstructuur. Het element op het hoogste niveau is het rootknooppunt. Daaronder zitten het rootelement **THESIS** en de XML declaratie. De elementen van het XML document [Sectie 2.3.1.] zijn in het paars aangeduid en de tekstuele inhoud van de elementknooppunten is in het wit weergegeven. De roze knooppunten zijn attributen [Sectie 2.3.2.] . Op de exacte betekenis van al deze begrippen komen we in de onderstaande paragrafen terug.



Figuur 2.1. De boomstructuur van een XML document

Bijna alle XML documenten beginnen met de XML declaratie: `<?xml version="1.0"?>` . Alhoewel deze niet noodzakelijk is, identificeert deze declaratie het document als zijnde een XML document en geeft het aan welke versie van XML gebruikt wordt.

Er zijn zeven soorten markup die kunnen voorkomen in een XML document: elementen, attributen, entiteitsreferenties, commentaar, processing instructions, gemarkeerde secties (CDATA secties) en document type declarations (DTD). In de volgende paragrafen zullen we elk van deze markup concepten kort bespreken.

2.3.1. Elementen

Elementen zijn de meest voorkomende vorm van markup. Niet-lege elementen bestaan uit een begin- en eindtag. Tags worden omsloten door kleiner dan- en groter dan-tekens, denk aan de reeds eerder vermelde <b> tag. De gebruikte tags geven meestal een aanduiding over de inhoud van het element. De naam van een element moet beginnen met een letter, een dubbele punt ":" of een underscore "_". Het vervolg van de elementnaam mag letters, cijfers, koppeltekens, underscores, dubbele punten of punten bevatten. Elementnamen die beginnen met de string **xml** zijn gereserveerd voor standaardisatie. Opgepast echter, een elementnaam mag géén witruimte bevatten. Voor een formele beschrijving van de voorwaarden waaraan de naam moet voldoen verwijzen wij naar [TRXMLCSC].

Een klein voorbeeldje: `<TITLE> Dit is een titel! </TITLE>` . De <TITLE> tag geeft de semantiek van de inhoud van het element weer.

Sommige elementen kunnen leeg zijn, dit wil zeggen dat ze geen inhoud hebben. Lege elementen hebben een aangepaste syntax en worden als volgt weergegeven: <br/>. De /> op het einde geeft door aan het verwerkende programma dat het om een leeg element gaat en er

bijgevolg dus niet naar een bijpassende eindtag gezocht moet worden. Er is een subtiel verschil tussen elementen die EMPTY (leeg) zijn en elementen die gewoon geen inhoud hebben, maar in XML zijn deze twee verschillende syntaxvormen onderling uitwisselbaar. Met een leeg element bedoelen we hier ook echt leeg, dit wil zeggen geen attributen en geen inhoud. Een element zonder inhoud kan wel nog attributen hebben.

Elementen kunnen andere elementen als inhoud hebben. Ze kunnen ook gewone tekst bevatten (zoals in het bovenstaande voorbeeld), gemengde inhoud (zowel tekst als elementen) of helemaal geen inhoud.

2.3.2. Attributen

Attributen zijn naam-waarde paren die geplaatst worden binnen de begintag van een element, na de naam van het element. Attributes leveren extra informatie over elementen. Vaak is deze extra informatie geen onderdeel van de gegevens zelf. Een voorbeeld van informatie die eigenlijk niet relevant is voor de inhoud zelf, is een identifier die door middel van een attribuut aan een element wordt verbonden. Een voorbeeldje: `<CHAPTER ID="t5"/>` . Dit **CHAPTER** element heeft een attribuut **ID** met als waarde **t5**. Attribuutnamen voldoen aan dezelfde beperkingen als elementnamen en mogen net als elementnamen geen witruimte bevatten. Bemerk ook dat in XML alle attribuutwaardes tussen aanhalingstekens geplaatst moeten worden. Of dit enkelvoudige of dubbele aanhalingstekens zijn, maakt geen verschil.

Alhoewel het perfect mogelijk is dat attributen gewone gegevens bevatten, draagt dit niet onze voorkeur weg. Wij vinden dat het beter is attributen zo veel mogelijk te vermijden en geven dan ook de voorkeur aan het gebruik van elementen om gegevens te bevatten. Een document waar attributen veel gegevens bevatten, is immers moeilijk te lezen en moeilijk te onderhouden. Als gouden regel geldt dat attributen best alleen gebruikt worden om ze metadata (gegevens over de gegevens) te laten bevatten.

2.3.3. Entiteiten

Om markup te kunnen aanbrengen in een document, zijn er een aantal tekens gereserveerd. Het kleiner dan-teken duidt bijvoorbeeld het begin van de start-tag of de eind-tag van een element aan. Als je nu deze speciale karakters als inhoud in je document wil gebruiken (iets waarmee je veelvuldig te maken krijgt, als je een tekst over XML/XSL schrijft), dan moet er een alternatieve manier zijn om deze tekens weer te geven. In XML worden entiteiten (entities) gebruikt om deze speciale tekens voor te stellen. Entiteiten worden ook gebruikt om te verwijzen naar vaak herhaalde of variërende tekst of om de inhoud van een extern bestand in het document te verwerken.

Elke entiteit moet een unieke naam hebben. Je kan je eigen entiteiten definiëren met behulp van een entiteitsdeclaratie. Een entiteitsdeclaratie is een onderdeel van een DTD. Hieronder geven we een voorbeeld van twee entiteitsdeclaraties. In de eerste declaratie wordt de interne entiteit "copy" gedeclareerd. Een interne entiteit associeert een naam met een bepaalde string, in dit geval met de string "Copyright Erwin Hermans en Carolien Coenen". In de tweede declaratie wordt de externe entiteit "charEntity" gedeclareerd. Een externe entiteit associeert een naam met de inhoud van een ander bestand, in dit geval met het bestand ../characters.ent. In dit bestand worden al de speciale leestekens, zoals ë, ï, ó, ø en é, die in onze thesistekst gebruikt worden, gedeclareerd. Er bestaat nog een derde soort entiteit, namelijk een parameterentiteit. Deze laatste soort zullen wij niet bespreken, aangezien wij die nergens gebruikt hebben.

```
<!ENTITY % copy "Copyright Erwin Hermans en Carolien Coenen">
```

```
<!ENTITY % charEntity SYSTEM "../characters.ent">
```

Om een entiteit te gebruiken, verwijs je er gewoon naar via zijn naam. Een entiteitsreferentie (entity reference) begint met een ampersand en eindigt met een puntkomma. Een voorbeeldje: de lt entiteit plaatst een kleiner dan-teken in je document. Dus als je een tag letterlijk wilt weergeven, schrijf je <element>. gt is de entiteit om het groter dan teken weer te geven. De entiteitsreferenties die gebruikt worden om te verwijzen naar de speciale leestekens die we in vorige paragraaf hebben opgesomd, zien er als volgt uit: ë, ï, ó, ø en é. Er bestaat ook nog zoiets als een tekenreferentie (character reference), maar dat zullen we hier niet bespreken, aangezien dit voor ons van weinig belang is. Belangrijk om te weten is dat er vijf voorgedefinieerde entiteiten in XML zijn, met name:

Tabel 2.1. Voorgedefinieerde entiteiten

Entiteit	**Voorstellingswijze**
kleiner-dan <	<
groter dan >	>
aanhalingsteken "	"
afkappingsteken '	'
ampersand &	&

2.3.4. Commentaar

Commentaar begint met <!-- en eindigt met -->. Het kan alle data bevatten, behalve de string "--". Commentaar kan om het even waar in je document geplaatst worden. Het is geen onderdeel van de tekstuele inhoud van een XML document, dit wil zeggen dat het niet nodig is dat een XML processor de commentaar doorgeeft aan de applicatie.

2.3.5. Processing Instructions

Processing instructions dienen om informatie door te geven aan een applicatie. Net zoals commentaar maken ze geen deel uit van het tekstgedeelte van het XML document. De XML processor moet ze gewoon doorspelen aan de applicatie. Processing instructions zijn van de vorm

Voorbeeld 2.1.

```
<?naam attribuut="bijhorende waarde"?>
```

De naam, die het doel (target) is van de processing instruction, identificeert de processing instruction voor de applicatie. Applicaties verwerken alleen die doelen (targets) die ze herkennen, alle andere processing instructions worden genegeerd. De gegevens die volgen op de processing instruction zijn optioneel, zij zijn bedoeld voor de applicatie die het doel herkend heeft. Namen van processing instructions die beginnen met **xml** zijn gereserveerd voor XML standaardisatie. Enkele voorbeelden van processing instructions:

Voorbeeld 2.2. Processing instructions

```
<?xml version="1.0"?>
<?xml-stylesheet href="stylesheet.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
```

Wij zijn van mening dat het beter is processing instructions zoals de twee laatste in het voorbeeld zoveel mogelijk te vermijden in je XML document. Het toevoegen van deze instructies is in tegenspraak is met het principe van het scheiden van inhoud en de manier waarop deze inhoud verwerkt moet worden. In Cocoon1 is het nodig de **xml-stylesheet** instructie toe te voegen aan elke XML document om aan te geven met welke XSL stylesheet dit document verwerkt moet worden. Ook processing instructions die eruit zien zoals het laatste voorbeeld, zijn niet te vermijden in Cocoon1. In Cocoon2 is dit niet meer nodig door het gebruik van de sitemap, wat een serieuze verbetering inhoudt. Voor een gedetailleerde uitleg over de betekenis van processing instructions in Cocoon1 verwijzen wij naar de sectie over Cocoon1 [Sectie 9.1.] . Voor een gedetailleerde uitleg over de sitemap verwijzen we naar de sectie over Cocoon2 [Sectie 9.2.] .

2.3.6. CDATA Secties

Een CDATA sectie instrueert de parser om de meeste markup tekens (kleiner dan, groter dan, aanhalingsteken, afkappingsteken en ampersand) te negeren. Het enige markup teken dat nog herkend wordt is **]]>**, dit duidt het einde van de CDATA sectie aan. Als je een gedeelte broncode in je XML document wilt opnemen, is het heel waarschijnlijk dat daar kleiner dan- of groter dan-tekens in voorkomen. Om te voorkomen dat de XML parser dit (verkeerdelijk) interpreteert als markup, kan je een CDATA sectie gebruiken. CDATA staat voor **c**haracter **data**. Een CDATA sectie ziet er als volgt uit: <![**CDATA**[b= (i < 3);]]> . De inhoud van de CDATA sectie wordt dan letterlijk naar de uitvoer gekopieerd. Voor het voorbeeldje wil dat zeggen dat b= (i < 3); letterlijk in de uitvoerstroom wordt opgenomen.

2.3.7. Document Type Declaraties

Een van de sterkste punten van XML is dat het je toelaat je eigen tags te creëren. Maar als je een applicatie hebt, is het niet erg zinvol om die tags zomaar in een willekeurige volgorde door elkaar te klutsen. Zo krijg je documenten die syntactisch wel correct zijn, maar die geen enkele zinnige betekenis hebben. Als je een betekenis aan je document wilt geven, dan moeten er restricties zijn op de opeenvolging en de nesting van de tags. DTDs dienen om deze restricties weer te geven.

Declaraties zorgen ervoor dat een document meta-informatie over zijn inhoud kan doorgeven aan de parser. Die meta-informatie bevat onder andere:

- de toegelaten opeenvolging en nesting van de tags
- attribuutwaardes en hun types eventueel samen met de defaultwaardes die de attributen aannemen
- de entiteiten die je kan ontmoeten

Er bestaan vier types van XML declaraties: declaraties van element types (element type declarations), declaraties van attribuutlijsten (attribute list declarations), declaraties van entiteiten (entity declarations) en declaraties van notaties (notation declarations). Wij gaan in dit hoofdstuk niet dieper in op de verschillende soorten van declaraties, omdat wij in onze thesis zelf maar een minimaal gebruik maken van DTD's. Wel hebben wij de XML variant van DTD, namelijk XML Schema, wel uitvoerig bestudeerd en daar een apart hoofdstuk aan gewijdt. Als je meer over de syntax van DTD's wilt lezen, verwijzen wij graag naar externe bronnen zoals [DTDW], [RDW3S] enzovoort.

2.3.8. Overige onderwerpen

Tenslotte behandelen we nog kort hoe om te gaan met witruimte, attribuutwaarde normalisatie en de taal waarin een document geschreven is.

2.3.8.1. Behandeling van witruimte

Verwerking van witruimte in een XML document is vaak een subtiele zaak. In veel gevallen, zoals in het onderstaande voorbeeld staat er tussen twee opeenvolgende elementen witruimte (tabulatoren en/of een nieuwe lijn) en heeft de witruimte tussen de elementen geen betekenis. Onthoud dat volgens de XML specificatie een nieuwe lijn door de XML processor moet doorgegeven worden als een LF (Line Feed). [TRXMLEOL]

Voorbeeld 2.3. Verwerking van witruimte

```
<TITLE> Titel van een hoofdstuk </TITLE>
<PARA> Paragraaf van een hoofdstuk </PARA>
```

Hoe weet je nu of witruimte betekenisvol is of niet? Je kan dit slechts te weten komen als je het inhoudsmodel van de elementen kent. Kort samengevat is witruimte betekenisvol bij gemengde inhoud en is het niet betekenisvol als enkel de inhoud van een element van belang is. Een element heeft een gemengde inhoud als het uit zowel tekstuele inhoud als andere elementen bestaat.

De regel voor XML processoren is dat ze alle tekens die geen markup zijn moeten doorgeven aan de applicatie. Wat er met de markup moet gebeuren, hangt van de XML processor zelf af. Als de processor een validerende XML processor is (een processor die nagaat of het document voldoet aan een DTD of XML Schema), dan moet die ook doorgeven aan de applicatie welke witruimte tekens betekenisvol zijn.

Er bestaat een speciaal attribuut **xml:space** dat gebruikt kan worden om expliciet weer te geven dat witruimte betekenisvol is. Bij elk element dat de attribuutspecificatie **xml:space="preserve"** bevat, is alle witruimte binnen dat element en binnen alle subelementen van dit element betekenisvol, tenzij dit attribuut overschreven wordt in één van de kindelementen. De enige waardes die het attribuut **xml:space** kan aannemen zijn **preserve** en **default**. De default waarde geeft aan dat de default verwerking die door de applicatie gehanteerd wordt, gewenst is.

2.3.8.2. Attribuutwaarde normalisatie

De XML processor voert attribuutwaarde normalisatie uit op attribuutwaardes:

- Referenties naar een teken worden vervangen door het teken in kwestie.
- Entiteitreferenties worden recursief vervangen.
- Witruimte wordt genormaliseerd.

Voor meer details over de exacte normalisatieprocedure verwijzen wij naar [TRXMLAVN].

2.3.8.3. Taalidentificatie

Veel applicaties voor documentverwerking kunnen voordeel hebben bij informatie over de natuurlijke taal waarin het document is geschreven. XML definieert een attribuut **xml:lang** om aan te duiden om welke taal het gaat (Engels, Frans, Nederlands enzovoort). Aangezien het doel van dit attribuut is om de informatie tussen verschillende applicaties te

standaardiseren, beschrijft de XML specificatie ook hoe talen geïdentificeerd moeten worden, bijvoorbeeld welke speciale tekens in een bepaalde taal voorgedefinieerd zijn. Het **xml:lang** attribuut bestaat uit een combinatie van een ISO 693 taalcode [ISO693] en een (optionele) ISO 3166 landcode [IS03166], gescheiden door een liggend streepje, zoals beschreven staat in RFC 1766 [RFC1766]. Default staat het **xml:lang** attribuut op Engels: **en**. Een concreet voorbeeld:

Voorbeeld 2.4. Taalidentificatie

```
<p xml:lang="en"> The quick brown fox jumps over the lazy dog. </p>
<p xml:lang="en-GB"> What colour is it? </p>
<p xml:lang="en-US"> What color is it? </p>
```

2.4. Namespaces in XML

In een XML document kunnen elementen en attributen voorkomen die oorspronkelijk gedefinieerd waren voor verschillende applicaties die gebruik maken van verschilende "markup woordenschatten" (een verzameling elementen en attributen gedefinieerd voor een bepaalde toepassing). Doordat een document gebruik kan maken van verschillende "markup woordenschatten", is er dus geen garantie dat de element- en attribuutnamen altijd uniek zijn. Dit kan de softwaretoepassing soms voor problemen plaatsen, omdat niet altijd duidelijk is welke interpretatie aan een bepaald element of attribuut moet gegeven worden. Door aan elementen en attributen van een bepaalde markup woordenschat universele namen toe te kennen, kunnen we ze ook buiten het document waarin ze oorspronkelijk werden gedefinieerd, gebruiken. Om dit te realiseren maken we gebruik van XML namespaces. Voor onze uiteenzetting over XML namespaces baseren wij ons op de W3C standaard voor XML Namespaces [TRXMLNA].

2.4.1. Het declareren van namespaces

Een namespace wordt gedeclareerd met behulp van gereserveerde attributen. De naam van zo een attribuut moet ofwel **xmlns** ofwel **xmlns:<voorvoegsel>** zijn. Het voorvoegsel is een, binnen de scope, unieke naam om te refereren aan een welbepaalde namespace. De scope behandelen we in de volgende paragraaf. De waarde van dit attribuut (een URI referentie) is de naam van de te identificeren namespace. De enige voorwaarde die aan de URI gesteld wordt, is dat deze de namespace een unieke naam moet geven, zodat ook de elementen en attributen die met deze namespace geassocieerd worden, unieke namen krijgen. De URI moet dus niet gebruikt kunnen worden om bijvoorbeeld een XML Schema op te zoeken, alhoewel de URI vaak wel een bestaande link is naar een webpagina met meer uitleg over de namespace, of wel degelijk een verwijzing is naar het XML Schema zelf.

Een namespace declaratie geldt alleen voor het element waarin de declaratie gespecificeerd is en voor alle elementen en attributen in de inhoud van dat element, tenzij de namespace declaratie overschreven wordt door een andere namespace declaratie met hetzelfde voorvoegsel. Elementen of attributen die niet behoren tot de inhoud van het element waarin de namespace gedeclareerd werd, kunnen nooit tot de daarin gedeclareerde namespace behoren. De elementen en attributen die gebruik kunnen maken van de gedeclareerde namespace behoren tot de "scope" van die namespace.

XML namespaces voorzien dus in een eenvoudige methode om element- en attribuutnamen die gebruikt worden in XML documenten te karakteriseren. Elke element- of attribuutnaam kan je namelijk associëren met een welbepaalde namespace. Zo'n associatie ziet er als volgt

uit: **<voorvoegsel>:elementnaam**. Twee namespaces zijn identiek als hun URI referenties identiek zijn, dit wil zeggen dat alle tekens waaruit ze bestaan exact hetzelfde moeten zijn. Binnen een XML namespace moeten natuurlijk alle gebruikte element- en attribuutnamen uniek zijn, anders kunnen er nog steeds conflicten ontstaan.

Er zijn twee soorten namespaces. Default namespaces en namespaces die geassocieerd worden met een voorvoegsel. Een default namespace is van toepassing op het element waarin de namespace gedeclareerd wordt (als dat element zelf tenminste geen voorvoegsel heeft) en op alle elementen zonder voorvoegsel in de inhoud van dat element. Als de URI referentie in een default namespace declaratie leeg is, dan worden de elementen zonder voorvoegsel die zich binnen dat element bevinden, beschouwd als behorende tot geen enkele namespace. Opgepast, default namespaces zijn niet direct van toepassing op attributen. Deze opmerking komt rechtstreeks uit de technische aanbeveling van het W3C en lijkt op het eerste zicht wat vreemd. Verderop in de aanbeveling staat dan dat een attribuut behoort tot de namespace van het element waar hij bij staat. Onze redenering is nu als volgt: als een element tot de default namespace behoordt, dan behoort zijn attribuut indirect ook tot de default namespace. [TRXMLNAND]

De declaratie van een default namespace gebeurt met behulp van het **xmlns** attribuut:

Voorbeeld 2.5. Declaratie van een default namespace

```
<THESIS xmlns="http://carolien.ulyssis.org/thesis/">
   <CHAPTER>
      <TITLE> Titel </TITLE>
      <SECTION> Sectie van de tekst </SECTION>
   </CHAPTER>
</THESIS>
```

Het namespace voorvoegsel (behalve als het **xml** of **xmlns** is) moet steeds gedeclareerd worden met behulp van een namespace declaratie attribuut in ofwel de starttag van het element waarin het voorvoegsel wordt gebruikt, ofwel in een ouder van dat element (een element waarvan in de inhoud markup voorkomt voorafgegaan door het voorvoegsel dat de namespace aanduidt). In het volgende voorbeeld wordt de **xinclude** namespace gedeclareerd met behulp van het **xmlns:xinclude** attribuut. Het voorvoegsel **xinclude** wordt hierdoor geassocieerd met de namespace *http://www.w3.org/2001/XInclude*

Voorbeeld 2.6. Namespace declaratie

```
<THESIS xmlns:xinclude="http://www.w3.org/2001/XInclude">
   <xinclude:include parse="xml" href="xml.xml"/>
</THESIS>
```

Het is perfect mogelijk om meerdere namespaces te declareren in éénzelfde element. In het onderstaande voorbeeld wordt een default namespace en een namespace die hoort bij een voorvoegsel gedeclareerd:

Voorbeeld 2.7. Declaratie van meerdere namespaces binnen eenzelfde element

```
<THESIS xmlns="http://carolien.ulyssis.org/thesis/"
xmlns:xinclude="http://www.w3.org/2001/XInclude">
   <CHAPTER>
      <!--Inhoud van het CHAPTER element-->
   </CHAPTER>
```

```
</THESIS>
```

De technische aanbeveling vermeldt niet wat er gebeurt indien meerdere keren hetzelfde voorvoegsel binnen éénzelfde element wordt gedeclareerd. We hebben dit dan zelf uitgetest en dit gaf ons een foutmelding. Proefondervindelijk zijn we dus tot de conclusie gekomen dat dit niet mag. Deze conclusie lag in de lijn van onze verwachtingen.

Er is één beperking op de keuze van het voorvoegsel om de namespace aan te duiden. Voorvoegels die beginnen met de drie-letter sequentie x, m, l, in eender welke hoofdletter combinatie, zijn gereserveerd voor gebruik door XML en XML-gerelateerde specificaties. Het voorvoegsel **xml** is per definitie gebonden aan de namespace *http://www.w3.org/XML/1998/namespace*. Het voorvoegsel **xmlns** wordt alleen gebruikt om namespaces te verbinden met een voorvoegsel en is op zichzelf niet verbonden met een namespace.

2.4.2. Het toepassen van namespaces op elementen en attributen

Element- en attribuutnamen die tot een XML namespace behoren verschijnen in het document als gekwalificeerde namen. Een gekwalificeerde naam bestaat uit een namespace voorvoegsel en een lokaal gedeelte, gescheiden door een dubbele punt. Het voorvoegsel (dat geassocieerd is met een bepaalde URI referentie) selecteert een bepaalde namespace. De combinatie van een globaal beheerde URI namespace en de eigen namespace van het document levert identifiers op voor de gebruikte namen die universeel uniek zijn. Omdat URI referenties karakters bevatten die niet toegestaan zijn in namen, kunnen ze niet direct gebruikt worden als namespace voorvoegsels. Daarom moet er een namespace voorvoegsel gedefinieerd worden die als een soort proxy voor een URI referentie dient.

In het volgende voorbeeldje wordt het **xsl** voorvoegsel verbonden met de *http://www.w3.org/1999/XSL/Transform* namespace. Alle elementen met het voorvoegsel **xsl**, behoren tot de **xsl** namespace.

Voorbeeld 2.8. Gebruik van de xsl namespace

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="THESIS">
      <html>
         <head>
            <title> Thesis Erwin Hermans en Carolien Coenen </title>
         </head>
         <body bgcolor="#FFFFFF" text="#000000">
            <xsl:apply-templates select="CHAPTER" mode="TOC"/>
            <xsl:apply-templates select="APPENDIX" mode="TOC"/>
            <xsl:apply-templates select="CHAPTER"/>
            <xsl:apply-templates select="APPENDIX"/>
            <xsl:apply-templates select="BIBLIOGRAPHY"/>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

2.4.3. Bemerking over namespaces

De hierboven beschreven manier van namespaces declaren, kan op het eerste gezicht een

beetje vreemd lijken. Wij hebben dan ook overwogen of er geen andere, meer logische manieren zouden zijn om een namespace te declareren.

Je zou bijvoorbeeld een namespace kunnen declareren door een **namespace** element met een attribuut **href** rond de elementen die tot een bepaalde namespace moeten behoren, te zetten:

Voorbeeld 2.9. Alternatieve manier om namespaces te declareren (gedachtenexperiment 1)

```
<namespace href="1">
   <ELEMENT1>
      <namespace href="2">
         <ELEMENT2 attribuut="attribuutwaarde"> inhoud van het element2
         </ELEMENT2>
      </namespace>
      <ELEMENT3> inhoud van het element3 </ELEMENT3>
   </ELEMENT1>
</namespace>
```

Bij deze manier van werken is het helaas onmogelijk om binnen een bepaald element, bijvoorbeeld **ELEMENT2** een attribuut van een ander namespace te gebruiken. Deze oplossing is dus niet werkbaar.

Een andere denkpiste die wij gevolgd hebben, bestaat uit het declareren van een namespace als een soort macro. Je zou dit kunnen doen aan de hand van bijvoorbeeld een **xml:namespace** element dat zich rond het element bevindt waarbinnen je de namespace wilt gebruiken. In dit geval wordt dan weer de namespace met een voorvoegsel geassocieerd en alle elementen die voorafgegaan worden door dit voorvoegsel (maar ook alleen maar die elementen) behoren tot de bewuste namespace. Deze manier van werken heeft het voordeel dat in tegenstelling tot voorgaand voorbeeld wel attributen een andere namespace kunnen hebben dan het element waartoe ze behoren.

De declaratie en het gebruik van een namespace zou er dan als volgt uitzien:

Voorbeeld 2.10. Alternatieve manier om namespaces te declareren (gedachtenexperiment 2)

```
<xml:namespace name="test" href="URI_die_namespace_indentificeert">
   <ELEMENT1 test:style="comment">
      <ELEMENT2> inhoud van het element2 </ELEMENT2>
      <ELEMENT3> inhoud van het element3 </ELEMENT3>
   </ELEMENT1>
</xml:namespace>
```

Het **style** attribuut behoort tot de namespace **test**, de overige elementen behoren tot de defaultnamespace van het document.

Deze manier van werken is verwarrend, omdat je zou verwachten dat alle elementen die zich binnen het **xml:namespace** element bevinden tot dezelfde namespace behoren. In dit opzicht lijkt de vorige denkpiste dan weer logischer.

Onze uiteindelijke conclusie is dan ook dat de gestandaardiseerde werkwijze voor namespaces, het duidelijkst en het gemakkelijkst is. Wij weten echter niet hoe men tot deze werkwijze gekomen is en ook niet welke afwegingen er gemaakt zijn en welke alternatieven het W3C daarvoor overwogen heeft.

2.5. Een concreet voorbeeld

In voorgaande paragrafen hebben wij de syntax van XML besproken aan de hand van kleine voorbeeldjes. Om een duidelijker beeld te krijgen van hoe een echt XML document eruit ziet, geven we nu een groter voorbeeld ter illustratie van de eerder besproken concepten. Het XML document dat dient als voorbeeld hebben wij opgesteld in het kader van het omzetten van een SHTML-gebaseerde site naar een XML-gebaseerde site en stelt het gemeenschappelijk onderdeel van de inhoudstabel (table of contents) van de site voor. Het is niet zo belangrijk dat dit voorbeeld tot in de details begrepen wordt, het is slechts bedoeld om een algemeen idee te geven over hoe een XML document er kan uitzien.

Voorbeeld 2.11. toc.common.xml

```
<?xml version="1.0"?>
 <GENERAL>
    <MENUITEM>
       <NAME> Syllabus </NAME>
       <URL> </URL>
       <FILE> syllabus.html </FILE>
       <TARGET> siteContent </TARGET>
    </MENUITEM>
    <MENUITEM>
       <NAME> Toc </NAME>
       <URL> common/ </URL>
       <FILE> coursetoc.html </FILE>
       <TARGET> siteContent </TARGET>
    </MENUITEM>
    <MENUITEM>
       <NAME> References </NAME>
       <URL> common/ </URL>
       <FILE> references.html </FILE>
       <TARGET> siteContent </TARGET>
    </MENUITEM>
    <MENUITEM>
       <NAME> Teachers </NAME>
       <URL> </URL>
       <FILE> teachers.html </FILE>
       <TARGET> siteContent </TARGET>
    </MENUITEM>
    <MENUITEM>
       <NAME> Copyright </NAME>
       <URL> common/ </URL>
       <FILE> copyright.html </FILE>
       <TARGET> siteContent </TARGET>
    </MENUITEM>
 </GENERAL>
```

2.6. XML: Syntax of taal?

In deze sectie willen wij even kort ingaan op de verwarring die er bestaat over het feit of XML een taal dan wel een syntax is. In de meeste documenten die wij gelezen hebben, wordt XML als een taal aanzien, wat vrij logisch is omdat XML staat voor eXtensible Markup Language. Daarom hebben wij er ook voor gekozen dit parlando te volgen. Dit is echter niet helemaal correct. XML is meer syntax dan taal. Een taal beschrijft wat toegelaten is, dus ook welke tags je mag gebruiken en het is net op dit punt waar je de vrijheid hebt bij XML (dit in tegenstelling tot HTML, wat wel een taal is). XML is dus een manier om je eigen markup taal te definiëren, met andere woorden, het is een taal met een zelf te definiëren woordenschat. Dit maakt dat XML strikt genomen een syntax is om talen te ontwerpen, dit wordt ook zo vermeld in de W3C Recommendation [TRXML].

2.7. Conclusie

Alhoewel XML een modewoord is en misschien hier en daar een beetje overroepen wordt, geloven wij toch in het potentieel van deze meta-markup taal. Nu al ontdekken meer en meer bedrijven de mogelijkheden van XML en worden er reeds vele toepassingen voor talloze domeinen ontwikkeld. Voorbeelden van toepassingen zijn B2B (Business To Business) uitwisseling van informatie, WAP en WML [RCWAP], databases die XML gebruiken [SWAG], het toegankelijk maken van gegevens voor blinden [BGJMSU], enzovoort. Wij geloven dan ook dat in de toekomst niemand meer XML zal kunnen negeren. Het feit dat zo veel bedrijven en organisaties op deze standaard gesprongen zijn, illustreert dit ten overvloede.

Een grapje om af te sluiten, met dank aan [RDXML]:

Vraag: *"When should I use XML?"*

Antwoord: *"When you need a buzzword in your resume"*

3. XML Schema

In dit hoofdstuk zullen wij een overzicht trachten te geven van de geschiedenis van XML Schema, en ook bespreken wat de mogelijkheden zijn van XML Schema, de eventuele beperkingen, de voor- en nadelen ten opzichte van DTD's, dit aan de hand van enkele voorbeelden. Op het einde van dit hoofdstuk geven we ook nog een voorbeeld van een omzetting van een DTD naar een schema, waar we ook nog enkele zaken bespreken in verband met de verhouding tussen deze twee technieken.

Wij baseren ons op de W3C XML Schema Language [TRXMLSCH0], [TRXMLSCH1] en [TRXMLSCH2] voor deze bespreking, maar we zijn er ons ten volle van bewust dat er nog andere schema talen zijn. Dit hoofdstuk is ook deels gebaseerd op [ERHSCHEMAS].

3.1. Wat is XML Schema?

XML Schema is een taal die het mogelijk maakt om, in XML, een specificatie te schrijven voor een hele klasse van XML documenten. Je zou kunnen zeggen dat XML Schema is voor XML wat een DTD is voor HTML. Deze laatste uitspraak is niet helemaal correct, maar ze dient om XML Schema te situeren in de wereld van XML/XSLT. XML Schema is gegroeid uit de idee dat indien het concept achter XML toch zo krachtig is, het mogelijk moet zijn een XML document te beschrijven met XML syntax, uiteraard gebruik makend van een gedefinieerde woordenschat en welbepaalde regels. Uit dit idee is XML Schema ontstaan, dat heel wat meer mogelijkheden biedt dan een DTD, zoals datatypering, maar in tegenstelling tot bij een DTD is het niet mogelijk om een entiteit te definiëren met XML Schema.

Vermits schema zo een generieke term is, zijn er meerdere schema talen voor XML ontwikkeld vanuit verschillende hoeken. Enkele van deze talen zijn: Relax [RELAX], TREX - Tree Regular Expressions for XML [TREX] en Schematron [SCHEMATRON]. Dit zijn slechts enkele van de beschikbare schema talen. In de loop der tijd zijn er ook schema talen ontwikkeld die door hun makers reeds verbannen zijn. Daaronder bevinden zich onder andere Schema for Object-Oriented XML (SOX) [NOTESOX] en Document Content Description (DCD) [NOTEDCD]. Dit om aan te geven dat de W3C XML Schema Language niet de enige taal is met het doel XML te beschrijven en dit in een gedefinieerd XML formaat.

3.2. Geschiedenis van XML Schema

Nadat de XML 1.0 specificatie in februari 1998 voltooid was, waren er nog een aantal zaken waar men in het W3C verder aan wilde werken. Hiervoor werd de XML Schema Groep opgericht binnen het W3C. De belangrijkste werkzaamheden hielden onder meer in om een primitieve datatypering toe te voegen aan XML, overervingsmechanismen te voorzien voor elementtypes en het definiëren en gebruiken van namespaces integreren in deze nieuwe taal. Dit zijn enkele doelstellingen bovenop de doelstellingen van DTD. De taal, XML Schema, die men wilde ontwikkelen moest daarenboven uitgedrukt worden in XML, gemakkelijk te gebruiken zijn en eenvoudig genoeg zijn zodat programma's die er gebruik van willen maken heel gemakkelijk te ontwerpen zijn en weinig eisen opleggen aan de computers.

Er volgen veel bijdragen vanuit verschillende hoeken en in mei 1999 werd er een working draft van XML Schema publiek gemaakt. Deze was voornamelijk gebaseerd op bijdragen uit vier voorstellen:

- XML-Data [NOTEXMLD]
- Document Content Description (DCD) [NOTEDCD]
- Schema for Object-Oriented XML (SOX) [NOTESOX]

- Document Definition Markup Language [NOTEDDML]

Dit maar om te zeggen dat XML Schema niet zo maar uit de lucht is komen vallen, maar mede gebaseerd is op de inbreng van zowel onderzoeksgroepen als op de inbreng van commerciële bedrijven. In dit hoofdstuk bespreken we XML Schema 1.0, maar op dit moment wordt er gepeild naar de wensen voor XML Schema 1.1, die ook dient om foutjes die eventueel in versie 1.0 zitten, te verhelpen.

3.3. Gebruik van XML Schema

XML Schema is een taal die er op gericht is om de inhoud en de structuur van XML documenten te kunnen beschrijven. Het doel is hetzelfde als dat van de DTD (Document Type Definition) taal, maar de structuur en de beperkingen kunnen op een krachtigere en gemakkelijkere manier beschreven worden. Een schema (XML Schema document) is met andere woorden een specificatie van een hele klasse van XML documenten, die kunnen beschouwd worden als instanties van de specificatie. Het is mogelijk een XML document te associëren met een schema dat de structuur en de inhoud van dat document beschrijft. Aan de hand van het schema kan het document dan gevalideerd worden. Dit kan ook met een DTD, maar in tegenstelling tot XML Schema, heeft DTD geen XML syntax. Bestaande programma's voor de verwerking van XML documenten kunnen, indien we DTD's zouden gebruiken, dan niet herbruikt worden om ook de beschrijving van het document in te lezen. Maken we echter gebruik van XML Schema hiervoor, dan gaat dit wel.

We zullen dit illustreren met een kort voorbeeldje. Eerst geven we een XML document, vervolgens een mogelijke DTD voor dit document en ten slotte een schema voor dit document. Bij dit voorbeeldje geven we geen uitleg, maar na het lezen van dit hoofdstuk zou dit voorbeeldje gemakkelijk te begrijpen moeten zijn. Maar we durven zelfs zeggen dat het schema zelfs begrepen kan worden, zonder verdere uitleg over XML Schema gelezen te hebben.

Voorbeeld 3.1. book.xml

```
<?xml version="1.0"?>
 <BOOK>
    <ISBN> 0-112-11111-1 </ISBN>
    <TITLE> Lord Of The Rings: The Fellowship Of The Ring </TITLE>
    <AUTHOR> J.R.R. Tolkien </AUTHOR>
    <YEAR> 1954 </YEAR>
    <PRICE> 17.95 </PRICE>
 </BOOK>
```

DTD 3.1. DTD - book.dtd

```
<!ELEMENT BOOK (ISBN, TITLE, AUTHOR+, YEAR?, PRICE)>
<!ELEMENT ISBN #PCDATA>
<!ELEMENT TITLE #PCDATA>
<!ELEMENT AUTHOR #PCDATA>
<!ELEMENT YEAR #PCDATA>
<!ELEMENT PRICE #PCDATA>
```

Voorbeeld 3.2. XML Schema - book.xsd

```
<?xml version="1.0"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
    <xsd:element name="BOOK" type="BookType"/>
    <xsd:complexType name="BookType">
      <xsd:sequence>
        <xsd:element name="ISBN" type="ISBNType"/>
        <xsd:element name="TITLE" type="xsd:string"/>
        <xsd:element name="AUTHOR" type="xsd:string"
        maxOccurs="unbounded"/>
        <xsd:element name="YEAR" type="YearType" minOccurs="0"/>
        <xsd:element name="PRICE" type="PriceType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="ISBNType">
      <xsd:restriction base="xsd:string">
        <xsd:pattern
        value="[0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9][0-9]-[0-9A-Z]"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="YearType">
      <xsd:restriction base="xsd:gYear">
        <xsd:minInclusive value="1540"/>
        <xsd:maxInclusive value="2002"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="PriceType">
      <xsd:restriction base="xsd:decimal">
        <xsd:minInclusive value="0.00"/>
        <xsd:fractionDigits value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
```

Een schema is een gewoon XML document, maar heeft een specifieke namespace en de elementen die gebruikt kunnen worden, zijn hierdoor beperkt tot een gedefinieerde set. Het 'geraamte' van een schema ziet er dan als volgt uit:

Voorbeeld 3.3.

```
 <?xml version="1.0"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="elementNaam" type="xsd:elementType"/>
  </xsd:schema>
```

3.3.1. Associatie van XML document met XML Schema

Een XML document associëren we op de volgende manier met een DTD:

Voorbeeld 3.4.

```
<!DOCTYPE BOOK PUBLIC "book.dtd">
  <BOOK>
    <!--rest van het document volgt hier-->
    ...

  </BOOK>
```

Dit is een declaratie die aan het begin van het document moet voorkomen.

We kunnen ook een schema associëren met een XML document. Hiertoe gebruiken we het

xsi:noNamespaceSchemaLocation attribuut dat we toevoegen aan het hoofdelement. In plaats van aan het hoofdelement mag dit ook toegevoegd worden aan het eerste element waarop het aangeduide schema betrekking heeft. Het voorvoegsel **xsi** gebruiken we om de namespace *http://www.w3.org/2001/XMLSchema-instance* aan te duiden. Het schema dat geassocieerd is met deze namespace voorziet in een aantal attributen voor onmiddellijk gebruik in eender welk XML document. Waarom men hiervoor een aparte namespace heeft gekozen is onduidelijk, maar het is nu eenmaal zo gedaan en dan moeten we er op die manier ook maar mee leren werken. Deze associatie ziet er dan als volgt uit:

Voorbeeld 3.5.

```
<BOOK xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="book.xsd">
   <!--rest van het document volgt hier-->
   ...

</BOOK>
```

Het schema dat overeenkomt met de namespace *http://www.w3.org/2001/XMLSchema-instance* voorziet voor dit geval ook nog een attribuut **schemaLocation**. Er is een miniem verschil tussen het gebruik van deze twee attributen. Indien je **xsi:noNamespaceSchemaLocation** gebruikt, dan geef je aan dat er geen target namespace is gedefinieerd in het schema. Indien je **xsi:schemaLocation** gebruikt, geef je aan dat er wel een target namespace is gedefinieerd. Wij vinden dit slechts een miniem onderscheid, en eigenlijk zo miniem dat we ons afvragen waarom men hier twee attributen voor heeft voorzien.

3.3.2. Validatie van een XML document

We geven hier een voorbeeld van hoe Xerces2 [XERCES2] gebruikt kan worden om een XML document, geassocieerd met een XML Schema, te valideren. De code die we gebruiken om te valideren ziet er als volgt uit:

Voorbeeldcode 3.1

```
...

import org.apache.xerces.parsers.SAXParser;
import org.xml.sax.SAXException;

...

SAXParser parser = new SAXParser();
// instellen van de parser
try {
    parser.setFeature("http://apache.org/xml/features/validation/schema",true);
    parser.setFeature("http://xml.org/sax/features/validation",true);
} catch (Exception e) {
    System.out.println("Instellen van parser mislukt: " + e.toString());
    System.exit(1);
}
// parse en valideer document
try {
    parser.parse(file);
    System.out.println("Document succesvol geparst!");
} catch (Exception e) {
    System.out.println("Parsen mislukt: " + e.toString());
    System.exit(1);
}
...
```

Na het importeren van de benodigde klassen, maken we een SAXParser object aan. Vervolgens moeten we de parser configureren zodat het te parsen XML document wel degelijk gevalideerd wordt tegenover het geassocieerde XML Schema. De eerste eigenschap, **http://xml.org/sax/features/validation**, geeft aan dat het document gevalideerd moet worden en de fouten gerapporteerd moeten worden. Met de tweede eigenschap, **http://apache.org/xml/features/validation/schema**, geven we aan dat de validatie tegen een schema moet gebeuren. Zonder deze derde eigenschap op **true** te zetten zou de validatie tegenover een DTD gebeuren. Uiteindelijk geven we een bestandsnaam mee aan de parse methode en tijdens het parsen zal het document gevalideerd worden.

Indien het document aan het XML Schema voldoet, dan krijgen we alleen de melding *"Document succesvol geparst!"* op het scherm. Staat er een fout in, dan krijgen we vanzelfsprekend een foutmelding op het scherm te zien. Passen we dit toe op het voorbeeld **book.xml** [Sectie 3.3., Voorbeeld 3.1.] , waar we de associatie leggen naar het XML Schema **book.xsd** [Sectie 3.3., Voorbeeld 3.2.] . Dit zal geen problemen opleveren bij de validatie. Indien we nu van het ISBN element een attribuut maken bij het BOOK element dan zal dit de volgende uitvoer opleveren bij het valideren:

```
[Error] book.xml:4:23: cvc-complex-type.3.2.2: Attribute 'ISBN' is not valid
respect to the attribute wildcard of Elment 'BOOK'.
[Error] book.xml:5:9: cvc-complex-type.2.4.a: Invalid content starting with element
'TITLE'. The content must match
'(((((":ISBN),(":TITLE)),(":AUTHOR){1-UNBOUNDED}),(":YEAR){0-1}),(":PRICE))'.
```

Je moet dan wel al goed weten waar het over gaat om deze foutmeldingen goed te kunnen begrijpen. Volgens ons zou de uitvoer wel iets duidelijker mogen zijn.

3.3.3. Opbouw van een XML Schema

Zoals uit bovenstaande voorbeeldjes blijkt, wordt een XML Schema document (schema) opgebouwd zoals eender welk XML document, maar met een andere namespace en als root element van het document het element **<schema> </schema>** , al dan niet voorafgegaan door een namespace-voorvoegsel. Dit element heeft als voornaamste kinderen de elementen **annotation**, **element**, **complexType** en **simpleType**. Deze elementen verdienen nadere uitleg.

3.3.3.1. Het annotation element

Zoals de naam van dit element het al laat vermoeden, dient dit element om aanmerkingen (annotations) te maken in een schema. Dit element bevat zelf niet de inhoud (de eigenlijke aanmerkingen), maar kan de subelementen **documentation** en **appInfo** bevatten. Het **documentation** element bevat de documentatie die bedoeld is om door mensen gelezen te worden. Er kunnen meerdere **documentation** elementen kind zijn van hetzelfde **annotation** element. Het **appInfo** element daarentegen dient om informatie te verschaffen voor programma's die de schema's misschien op een heel andere manier verwerken. Het **appInfo** element bevat dus informatie, gericht op programma's die schema's automatisch kunnen verwerken. Net zoals met Javadoc zal het dus belangrijk zijn om voor een bepaalde klasse of per klasse programma's een woordenschat te definiëren. Met deze elementen is het mogelijk om schema's te documenteren; dit is te vergelijken met de specificatiecommentaar bij IPS (Informatie- en ProgrammaStructuren) [SDB99]. De commentaar die omsloten wordt door <-- en --> is dan op te vatten als implementatiecommentaar.

Ons insziens is het dus duidelijk de bedoeling dat er ook programma's komen die op basis van

deze elementen documentatie kunnen genereren van schema's. Volgens de specificatie is het zo dat het annotation element in het begin kan staan van de volgende elementen (deze lijst is niet noodzakelijk volledig): **schema**, **element**, **complexType**, **simpleType**, **attribute**. Volgens onze interpretatie van de specificatie wil dit zeggen dat **annotation** elementen het eerste subelement (*"... may also appear at the **beginning** of other schema constructions ..."* - [TRXS0-26]) van de genoemde elementen moeten zijn. Let wel, het mogen ook meerdere **annotation** elementen zijn die aan het begin verschijnen.

Het **annotation** element beschrijft dus zijn ouderelement. Op die manier is het mogelijk automatisch te bepalen bij welk element een **annotation** element hoort, aangezien dit met deze definitie een éénduidige relatie definieert.

We zullen dit illustreren aan de hand van een simpel voorbeeldje:

Voorbeeld 3.6.

```
<?xml version="1.0"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xml="http://www.w3.org/XML/1998/namespace">
     <xsd:element name="BOOK" type="BookType">
        <xsd:annotation>
           <xsd:documentation xml:lang="en"> The top level element, used to
           describe a book </xsd:documentation>
           <xsd:documentation xml:lang="nl"> Het hoofdelement, gebruikt om
           een boek te beschrijven </xsd:documentation>
           <xsd:appInfo>
              <bind xmlns="http://www.voorbeeld.org/Binding">
                 <class value="org.voorbeeld.Binding.Book"/>
              </bind>
           </xsd:appInfo>
        </xsd:annotation>
     </xsd:element>
     ...

  </xsd:schema>
```

We hebben hier gebruik gemaakt van het **annotation** element om het element **BOOK** te beschrijven. De definitie van **BookType** hebben we hier niet meer herhaald. Het **annotation** element heeft drie kindelementen, twee **documentation** elementen en één **appInfo** element. De twee **documentation** elementen bevatten documentatie die bedoeld is voor de menselijke lezers van dit document. Bij deze elementen vinden we ook het **xml:lang** attribuut terug en we geven hiermee aan in welke taal de inhoud van het element is. Het **appInfo** element daarentegen is bedoeld voor programma's die met schema's werken. In dit geval heeft het **appInfo** het element **bind** dat met een andere namespace wordt geassocieerd. Het **bind** element heeft een **class** element als kind en een attribuut **value** om aan te geven om welke (Java-)klasse het gaat. Blijkbaar is de bedoeling om aan te geven dat het element **BOOK** kan gebonden worden aan de klasse **org.voorbeeld.Binding.Book**. In dit voorbeeld wordt ook de namespace xml gedeclareerd, maar dit is niet nodig zoals we in de bespreking van namespaces al aangehaald hebben [Sectie 2.4.] .

3.3.3.2. Het element element

Op een bepaalde manier moeten we in een XML Schema kunnen aangeven dat er op een bepaalde plaats in het XML document bepaalde elementen verwacht worden. Dat is de taak van het `<element/>` element. Dit element heeft 1 verplicht en 1 optioneel attribuut (indien het optionele attribuut niet gespecificeerd wordt, dan moet dit met een inner element gebeuren). Het verplichte attribuut dient om de naam van het element te bepalen, en heeft de

volgende syntax: **name**="<elementnaam>". Elementnaam is eender welke geldige elementnaam. Het optionele attribuut verdient wat meer uitleg.

Het optionele attribuut geeft aan welk type data het element mag bevatten en heeft de volgende syntax: **type**="<elementtype>". Elementtype is één van de voorgedefinieerde types, maar het kan ook een zelfgedefineerd type (waarover later meer) zijn. Enkele voorbeelden van voorgedefinieerde types zijn:

- string
- anyURI
- duration
- dateTime
- nonPositiveInteger
- byte

Voor de volledige lijst verwijzen wij naar [TRXS2-3]. Naast de mogelijkheid om het datatype aan te duiden met behulp van een attribuut, kan het type ook in-line aan de element-definitie aangegeven worden. Met in-line bedoelen wij hier: als een kindelement van de element-definitie waar dit type bijhoort. Zo een definitie ziet er dan als volgt uit:

Voorbeeld 3.7.

```
<?xml version="1.0"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="NAME">
       <xsd:complexType>
          <xsd:sequence>
             <xsd:element name="GIVEN" type="xsd:string"/>
             <xsd:element name="FAMILY" type="xsd:string"/>
          </xsd:sequence>
       </xsd:complexType>
    </xsd:element>
 </xsd:schema>
```

Dit voorbeeld illustreert hoe een type in-line kan gedefinieerd worden. We hebben hier ook al een aantal elementen gebruikt, die we nog niet hebben uitgelegd. Deze uitleg volgt later in dit hoofdstuk. Wat opvallend is aan de in-line types is dat we deze types geen naam geven, het zijn anonieme types.

In het vorige voorbeeld is het duidelijk te zien dat je ook een zelfgedefinieerd type kan opgeven. Dit vereist wel dat het type ergens gedefinieerd is en in de scope valt van het element waar het bij hoort. Deze definitie kan voorzien zijn van een naam, maar dat is niet nodig als de definitie in-line is aan het element waar het bij hoort. We komen later nog terug op de zelfgedefinieerde types.

3.3.3.3. Het simpleType element

Met dit element kan je een bepaald type definiëren. Maar het gebruik van dit element is, zoals de naam van dit element (**simpleType**) misschien al laat vermoeden, aan een aantal beperkingen onderworpen. Met dit element geef je aan dat het type dat je definieert uit de lijst van ingebouwde simpele datatypes komt of dat het een afleiding van één van deze types is. Anders gezegd: een simpel type mag alleen van een ander simpel type afgeleid worden. Een andere beperking is dat je voor eenvoudige types geen attributen of elementen als inhoud kan definiëren. Wil je wel attributen en/of elementen als inhoud mogelijk maken voor het gedefinieerde element, dan moet je het type definiëren met behulp van het **complexType** element. De lijst met ingebouwde datatypes is te vinden op [TRXS2-3].

Naast gewoon het gebruik van de ingebouwde datatypes is het ook mogelijk om types van de ingebouwde eenvoudige types af te leiden. Dit kan op drie manieren gebeuren:

- **restriction**: selectie van een subset van waarden toegelaten door het basistype
- **union**: meerdere types combineren
- **list**: opgeven van een lijst van mogelijke waarden, die behoren tot een bestaand eenvoudig type

3.3.3.3.1. restriction

Met het **restriction** element geef je aan dat je een nieuw type wil definiëren door beperkingen op te leggen aan een bestaand type, dit kan zowel een complex als een eenvoudig type zijn. Dit element heeft één attribuut, met name **base**, waarvan de waarde aangeeft van welk basistype het nieuwe type is afgeleid. We geven nu een voorbeeldje om dit te verduidelijken.

Voorbeeld 3.8.

```
<xsd:simpleType name="birthYear">
   <xsd:restriction base="xsd:gYear">
      <xsd:minInclusive value="1970"/>
   </xsd:restriction>
</xsd:simpleType>
```

In dit voorbeeldje definiëren we een nieuw type, genaamd **birthYear**. Dit type is afgeleid van het ingebouwde type **gYear** (Gregorian Year), en legt, buiten de beperking van een geldig jaar, een verdere beperking op, met name dat een birthYear in dit geval 1970 of later moet zijn. 1969 en 1920 zijn geen geldige birthYears. Het element minInclusive geeft dus aan dat alle instanties een waarde moeten hebben die groter is dan of gelijk is aan 1970.

Naast minInclusive zijn er nog een aantal andere elementen waarmee beperkingen kunnen opgelegd worden. We zullen hier een kort overzicht geven van deze elementen, maar voor hun volledige betekenis verwijzen we naar [TRXS2-43].

- **xsd:minInclusive**: de waarde van alle instanties moet groter zijn dan of gelijk aan
- **xsd:minExclusive**: de waarde van alle instanties moet groter zijn dan
- **xsd:maxInclusive**: de waarde van alle instanties moet kleiner zijn dan of gelijk aan
- **xsd:maxExclusive**: de waarde van alle instanties moet kleiner zijn dan
- **xsd:enumeration**: een lijst van toegelaten waarden
- **xsd:whiteSpace**: hoe witruimte wordt behandeld binnen het element
- **xsd:pattern**: een reguliere expressie waarmee de instantie wordt vergeleken
- **xsd:length**: exacte aantal karakters toegestaan in het element
- **xsd:minLength**: minimum aantal karakters toegestaan in het element
- **xsd:maxLength**: maximum aantal karakters toegestaan in het element
- **xsd:totalDigits**: maximum aantal cijfers toegestaan in het element
- **xsd:fractionDigits**: maximum aantal cijfers toegestaan in het komma-gedeelte van het element

Het is vanzelfsprekend dat niet al deze elementen van toepassing zijn op alle types. Een overzicht hiervan wordt gegeven op [TRXS0-B].

3.3.3.3.2. union

Een tweede manier om een nieuw eenvoudig type te creëren is de combinatie van verschillende types door middel van een **union** element. Zo kan er aangegeven worden dat

een geldige instantie van een element tot één van de opgegeven types moet behoren. Stel dat we een type willen definiëren dat volledig gelijk is aan het type **Integer**, maar dat we niet willen dat 0 een geldige waarde is voor het nieuwe type. Dit kan bijvoorbeeld gerealiseerd worden door het nieuwe type te definiëren als een unie van twee bestaande types, namelijk **negativeInteger** en **positiveInteger**. In dit geval kan er op twee manieren (met behulp van een unie) aan deze definitie gestalte gegeven worden:

Voorbeeld 3.9.

```
<xsd:element name="nonZeroInteger">
   <xsd:union memberTypes="negativeInteger positiveInteger"/>
</xsd:element>
```

Voorbeeld 3.10.

```
<xsd:element name="nonZeroInteger">
   <xsd:union>
      <xsd:simpleType>
         <xsd:restriction base="xsd:negativeInteger"/>
      </xsd:simpleType>
      <xsd:simpleType>
         <xsd:restriction base="xsd:positiveInteger"/>
      </xsd:simpleType>
   </xsd:union>
</xsd:element>
```

In dit geval zou de eerste manier de voorkeur verdienen vanwege de beknopte, maar toch duidelijke, beschrijving van het element. Maar er zijn ook gevallen waarin de unie niet op zo een beknopte manier kan beschreven worden en dat de uitgebreide beschrijving moet genomen worden.

3.3.3.3.3. list

Een derde manier om een nieuw type af te leiden van een bestaand type is door aan te geven dat een attribuut of een element een lijst bevat van elementen van een bepaald eenvoudig type. Een voorbeeld van zo'n definitie is:

Voorbeeld 3.11.

```
<xsd:simpleType name="yearList">
   <xsd:list itemType="xsd:gYear"/>
</xsd:simpleType>
```

Een element van het type **yearList** ziet er dan als volgt uit:

Voorbeeld 3.12.

```
<YEARS> 1987 1999 1992 2002 </YEARS>
```

Elementen van het type yearList moeten een lijst bevatten van geldige xsd:gYear waarden, waarbij de waarden gescheiden worden door witruimte. Het is ook mogelijk aan te geven, met behulp van het **xsd:length** element als kind van het xsd:list element, hoeveel items in zo een lijst mogen voorkomen.

We willen toch nog een bedenking formuleren bij het gebruik van lijsttypes. Alhoewel een lijsttype er misschien op het eerste gezicht aantrekkelijker (korter) uitziet dan een lijst van elementen met één waarde, vinden wij het toch beter, vooral bij elementen, voor elke waarde een eigen element te voorzien. Ons inziens zorgt dat voor een betere structuur in het document en je vermijdt er ook een aantal mogelijke problemen door. Bij een lijsttype wordt een spatie beschouwd als een scheidingsteken tussen twee verschillende waarden. Wat indien een van deze waarden zelf ook een spatie bevat? Indien je elk element beperkt tot slechts één waarde en een lijst van waardes verbiedt, dan is dit probleem opgelost.

3.3.3.4. Het complexType element

In tegenstelling tot een eenvoudig type, kan je in een complex type wel elementen en attributen definiëren. Dit gebeurt met het **complexType** element. Volgens de aanbeveling [TRXS0-22] is de declaratie niet op zichzelf een type, maar eerder een associatie tussen een bepaalde naam en de beperkingen die worden opgelegd aan het voorkomen van die naam, in documenten die geassocieerd worden met het gedefinieerde schema. De definitie van een complex type kan er dan als volgt uitzien:

Voorbeeld 3.13.

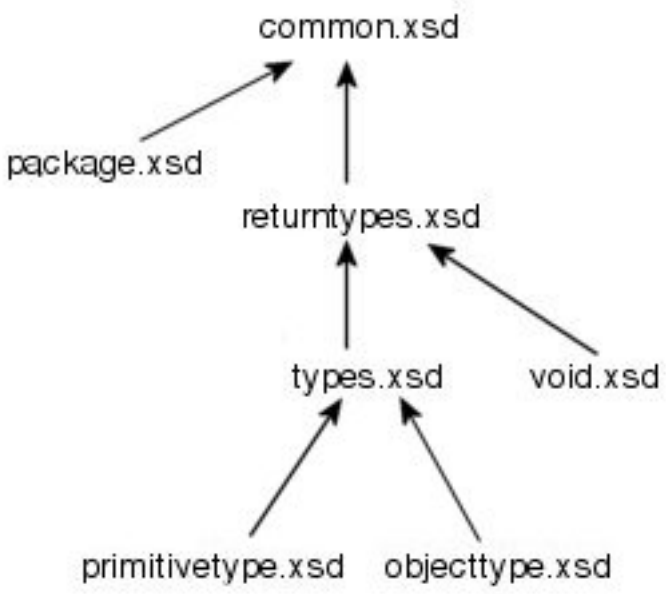
```
<xsd:complexType name="Address">
   <xsd:sequence>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="number" type="xsd:string"/>
      <xsd:element name="zipcode" type="xsd:integer"/>
   </xsd:sequence>
</xsd:complexType>
```

We hebben hier nu niet alle elementen besproken, langs de ene kant omdat dat ons te ver zou leiden, en langs de andere kant omdat dit niet echt nodig is om de basisprincipes van XML Schema te begrijpen. Met de hier aangereikte elementen kan al een mooi XML Schema opgebouwd worden, maar er moet in het achterhoofd gehouden worden dat XML Schema nog vele andere krachtige mogelijkheden biedt om een bepaald niveau van abstractie te verkrijgen in de definities. Verderop in dit hoofdstuk geven wij een uitgebreid voorbeeld met de bijhorende uitleg over die krachtige mogelijkheden en de abstractie. We zijn niet onmiddellijk met dit voorbeeld begonnen omdat we vonden dat we eerst een behoorlijke basis moesten aanreiken vooraleer met de deur in huis te vallen.

3.4. Bespreking van een uitgebreid voorbeeld

De voorbeelden die we hier gaan bespreken, zijn schema's die we verkregen hebben door bestaande DTD's om te zetten naar een equivalent XML Schema. De DTD's die we gebruikt hebben zijn geen willekeurige DTD's, maar zij maken deel uit van het Jnome project. We hebben een apart hoofdstuk gewijd aan deze toepassing en we verwijzen naar [Hoofdstuk 13.] voor een uitgebreide beschrijving van het project en het gebruik van de DTD's in dat project [Sectie 13.2.2.] . Aandachtige lezers zullen opmerken dat deze DTD's Java beschrijven, maar dat er ook delen inzitten die niet voorkomen in Java. Wij hebben hier niets aan veranderd aangezien ons eerste doel was om te zien hoe we dit konden omzetten naar XML Schema.

De schema's die we op die manier verkregen hadden heel wat gemeenschappelijk. We hebben daar gebruik van gemaakt om de gemeenschappelijke gedeelten onder te brengen in aparte schema's. Uiteindelijk leverde ons dit de volgende hiërarchie op, voor de betekenis (in feite van de DTD's) verwijzen we naar [Sectie 13.2.2.] :



Figuur 3.1. XML Schema hiërarchie

Van deze schema's hebben we de onderstaande voorbeelden gekozen om aan bod te laten komen in de volgende secties:

- **common.xsd**: bevat definities van types die gemeenschappelijk zijn aan de XML Schema's
- **returntypes.xsd**: bevat definities van returntypes, gemeenschappelijk aan de XML Schema's voor primitivetype, types en void
- **void.xsd**: bevat definities van de elementen en de types die specifiek zijn voor de XML bestanden die betrekking hebben op het type **void**

We hebben ons beperkt tot deze drie, omdat we met deze drie voorbeelden een heel goed idee kunnen geven van wat met XML Schema mogelijk is. Deze drie voorbeelden zullen we ook in de volgorde zoals hoger aangegeven behandelen. Merk op dat we deze schema's de extensie **xsd** hebben gegeven, maar dat dit niet noodzakelijk is. Dit is louter een indicatie voor ons, programma's houden geen rekening met de extensie.

3.4.1. common.xsd

Voorbeeld 3.14. common.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="emptyType"> </xsd:complexType>
    <xsd:complexType name="possibilitiesType">
       <xsd:sequence>
          <xsd:element name="returnTypeName" type="returnTypeNameType"
          minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="returnTypeNameType">
       <xsd:sequence>
          <xsd:element name="package" type="xsd:string" minOccurs="0"
          maxOccurs="unbounded"/>
```

```
            <xsd:element name="name" type="xsd:string"/>
            <xsd:choice minOccurs="0" maxOccurs="1">
               <xsd:element name="possibilities" type="possibilitiesType"/>
               <xsd:element name="isResolved" type="emptyType"/>
            </xsd:choice>
            <xsd:element name="bracket" type="emptyType" minOccurs="0"
            maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:schema>
```

Dit schema begint met een XML hoofding, een zogenaamde verwerkingsinstructie, om aan te geven dat het hier om een XML document gaat waarbij de aangegeven versie van XML en een specifieke codeerwijze wordt gebruikt.

Het hoofdelement van dit bestand is het **schema** dat thuishoort in de namespace, die geïdentificeerd wordt aan de hand van *http://www.w3.org/2001/XMLSchema* en hier geassocieerd wordt met het voorvoegsel xsd. Dit wil zeggen dat indien we elementen uit deze namespace willen gebruiken, we het voorvoegsel **xsd** gebruiken. We kunnen de default namespace natuurlijk ook veranderen, maar op deze manier is het vrij snel duidelijk dat dit document een schema voorstelt. Dit omdat **xsd** en **xs** de meest gebruikte voorvoegsels zijn om dit aan te geven. We zijn er ons ten volle van bewust dat dit geen goede reden is (het voorvoegsel is door iedereen vrij te kiezen, zolang de URI waarnaar de voorvoegsel verwijzen maar dezelfde is), maar het gebruik van deze voorvoegsels is zo ingeburgerd dat er redelijkerwijze mag vanuitgegaan worden dat indien men deze voorvoegsels tegenkomt men met een XML Schema te maken heeft. Voorzichtigheid omtrent deze veronderstellingen blijft echter te allen tijde geboden.

Als eerste definitie van een nieuw, complex type komen we het type **emptyType** tegen. Wat hier vooral opvalt aan deze definitie is dat er voor dit type alleen een naam wordt opgegeven. Er worden geen attributen of kindelementen gedefinieerd. Dit type kan gebruikt worden om een leeg element voor te stellen. De elementen die dit type hebben, kunnen dan op één van de volgende twee manieren geschreven worden:

- `<element/>`
- `<element> </element>`

Als dit toch een leeg type is, waarom is dit dan een complex en geen eenvoudig type? De reden hiervoor is dat het mogelijk moest zijn om een type te definiëren zonder inhoud, maar met attributen. Een type dat attributen heeft is een complex type. Wil je een type hebben zonder attributen en zonder inhoud, dan kan dit door dezelfde werkwijze te volgen die je zou gebruiken indien je een type zou definiëren zonder inhoud en met attributen, maar in dit geval definieer je ook geen attributen.

De volgende definitie van een complex type is **possibilitiesType**. Deze definitie bevat één kindelement, met name **xsd:sequence**. Wanneer een type kindelementen kan bevatten, moet er met één van de drie elementen **all**, **choice**, **sequence** gespecificeerd worden aan welke volgordevereisten deze kindelementen moeten voldoen. Over het verschil tussen deze drie geven we in een latere sectie meer uitleg. Bij dit type eisen we dat het kindelement de naam **returnTypeName** heeft. Dit element kan nul of meer keer voorkomen en het heeft als type returnTypeNameType. Dit type wordt verderop in common.xsd gedefinieerd.

3.4.1.1. Volgorde bepalingen voor kindelementen

Zoals reeds eerder aangegeven, gaan we wat meer uitleg geven over het opleggen van een bepaalde volgorde aan de kindelementen. Een hoofdelement heeft een bepaald type, en via de definitie van dat type kunnen we afleiden welke elementen toegestaan zijn onder het hoofdelement. Aan deze kindelementen kunnen we een bepaalde volgorde opleggen; dit dient altijd te gebeuren, zelfs indien er slechts één kindelement is. In een XML bestand zijn het dus de kindelementen van het hoofdelement waar het over gaat, maar deze worden in het XML schema bij het type van het hoofdelement gedefinieerd. Het aangeven van de volgorde dient met één van de volgende drie elementen te gebeuren:

- all
- choice
- sequence

Het **all**-element legt aan elk element (dat binnen het all-element gedefinieerd is) op dat dit element hoogstens éénmaal mag voorkomen als kind van het ouderelement, maar dat de volgorde niet belangrijk is. Door het gebruik van dit element worden aan bepaalde attributen, zoals **maxOccurs** en **minOccurs**, beperkingen opgelegd wat betreft de waarden die ze mogen aannemen (in dit geval zijn de toegelaten waarden voor **maxOccurs** en **minOccurs** nul of één). Daarenboven mag het all element geen choice- of sequence-elementen als kinderen bevatten en mag het all-element ook niet voorkomen als kind van een choice- of sequence-element. Ook groepen (zie [Sectie 3.5.3.]) zijn niet toegelaten als kindelementen van het all-element. Dit zijn enkele beperkingen van de W3C Schema Language.

Met het **choice**-element geef je aan dat slechts één van zijn kindelementen mag voorkomen in een documentinstantie, maar dit moet dan ook voorkomen. Bij dit element kunnen ook de attributen **minOccurs** en **maxOccurs** gebruikt worden. Met de waarden van deze attributen kunnen we aangeven dat we willen dat tussen **min** en **max** elementen voorkomen. Deze elementen moeten dan gekozen worden uit de kindelementen van het choice-element. De volgorde heeft dan geen belang. Ook bij het choice-element gelden ook de standaardwaarden van de attributen **minOccurs** en **maxOccurs**. De standaardwaarde voor deze attributen ligt vast op **1**. Het is een fout indien de waarde van het **minOccurs** attribuut groter is dan de waarde van het **maxOccurs** attribuut. Ook de kindelementen van het choice element kunnen de attributen **minOccurs** en **maxOccurs** dragen. De waarde die deze attributen hebben bij het choice element moeten overeenkomstig aangepast worden.

Voorbeeld 3.15.

```
...
 <xsd:choice minOccurs="6" maxOccurs="10">
   <xsd:element name="name" type="xsd:string" minOccurs="4"
   maxOccurs="6"/>
   <xsd:element name="title" type="xsd:string" minOccurs="2"
   maxOccurs="4"/>
 </xsd:choice>
...
```

In dit voorbeeldje demonstreren we het gebruik van de attributen minOccurs en maxOccurs samen met het choice element. Het is een kunstmatig voorbeeldje om de complexiteit aan te tonen van het samenspel van de waarden van deze attributen. Het choice element levert ons normaal gezien een keuze op tussen de elementen die als kindelementen ervan zijn gespecificeerd. Maar wanneer we gaan spelen met de waarden van de attributen minOccurs en maxOccurs wordt het ingewikkelder. We hebben hier twee kindelementen waarbij we deze

twee attributen een waarde hebben gegeven. Het element **name** heeft als waarde van minOccurs vier waar dit bij het element **type** twee is. Dit wil zeggen dat er een minimum van **zes** elementen uit het choice blok zal komen en dat is dan ook de waarde van het minOccurs attribuut bij choice. Een analoge redenering kan gemaakt worden voor het maxOccurs attribuut.

We willen hier wel de nadruk op leggen dat we dit niet in de uitgebreide Technical Recommendation hebben teruggevonden, maar dat we tot deze conclusie zijn gekomen na het maken van een dergelijke constructie in een schema en te controleren met de Schema Quality Checker van IBM [IBMSQC]. We hebben dit geprobeerd omdat we met de vraag bleven zitten of dit een toegestane of verboden constructie was. Blijkbaar is dit geldig maar we hebben hier toch onze twijfels bij, aangezien we het niet in de Technical Recommendation hebben teruggevonden.

Als laatste is er dan nog het **sequence**-element, waarmee we eisen dat alle elementen van de groep in die specifieke volgorde voorkomen. De attributen minOccurs en maxOccurs hebben hier een andere betekenis: ze geven aan hoe vaak de sequentie moet/mag herhaald worden. Binnen elk van deze sequenties moeten de elementen ook in de opgegeven volgorde voorkomen.

Het laatste type dat in common.xsd gedefinieerd wordt is het type **returnTypeNameType**. De elementen die deel uitmaken van dit type moeten in volgorde voorkomen zoals ze hier gedefinieerd worden. Het eerste element draagt de naam package, is optioneel (minOccurs=0), kan een onbeperkt aantal keer voorkomen en de inhoud is van het type string. Het tweede element, name, is een verplicht element van het type string. Het volgende element dat moet voorkomen is ofwel possibilities, met als type possibilitiesType ofwel het element isResolved, met als type emptyType. Dit wil zeggen dat deze twee elementen hier niet samen kunnen voorkomen, maar dat slechts één van deze twee elementen mag voorkomen, maar het is niet verplicht dat één van deze elementen voorkomt. Het laatste element, bracket, is ook een leeg element, en mag nul of meer keer voorkomen.

3.4.2. returntypes.xsd

In deze sectie gaan we returntypes.xsd bespreken. Dit schema-document bevat type-definities die gemeenschappelijk zijn voor primitiveType, type en void. Omdat deze drie slechts in een heel kleine mate verschillen is dit document het langste van de drie. We geven het hier volledig en gaan dan de nieuwe elementen stap voor stap bespreken.

Voorbeeld 3.16.

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:include schemaLocation="common.xsd"/>
    <xsd:complexType name="documentationBlockType">
       <xsd:sequence>
          <xsd:element name="descriptionBlock" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
          <xsd:element name="tagBlock" type="tagBlockType" minOccurs="0"
          maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="tagBlockType">
       <xsd:sequence>
          <xsd:element name="tag" type="xsd:string"/>
          <xsd:element name="informal" type="xsd:string" minOccurs="0"
```

```
        maxOccurs="1"/>
        <xsd:element name="formal" type="formalType" minOccurs="0"
        maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="formalType">
      <xsd:sequence>
        <xsd:element name="formalSentence" type="xsd:string"
        minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="typeType">
      <xsd:sequence>
        <xsd:element name="returnTypeName" type="returnTypeNameType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="parentType">
      <xsd:sequence>
        <xsd:element name="returnTypeName" type="returnTypeNameType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="formalParametersType">
      <xsd:sequence>
        <xsd:element name="parameter" type="parameterType" minOccurs="0"
        maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="parameterType">
      <xsd:sequence>
        <xsd:element name="final" type="emptyType"/>
        <xsd:element name="type" type="typeType"/>
        <xsd:element name="name" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="throwsClauseType">
      <xsd:sequence>
        <xsd:element name="exception" type="exceptionType" minOccurs="0"
        maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="exceptionType">
      <xsd:sequence>
        <xsd:element name="returnTypeName" type="returnTypeNameType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="modifierBlockMethodType">
      <xsd:sequence>
        <xsd:element name="accessibility" type="accessibilityType"/>
        <xsd:element name="abstract" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="static" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="final" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="synchronized" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="native" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="strictfp" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="accessibilityType">
      <xsd:attribute name="value">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="public"/>
            <xsd:enumeration value="private"/>
            <xsd:enumeration value="protected"/>
```

```
            <xsd:enumeration value="package"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="methodsReturningThisTypeType">
      <xsd:sequence>
        <xsd:element name="method" type="methodType" minOccurs="0"
        maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="methodType">
      <xsd:sequence>
        <xsd:element name="documentationBlock"
        type="documentationBlockType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="modifierBlockMethod"
        type="modifierBlockMethodType"/>
        <xsd:element name="type" type="typeType"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="parent" type="parentType"/>
        <xsd:element name="formalParameters"
        type="formalParametersType"/>
        <xsd:element name="throwsClause" type="throwsClauseType"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
```

Het eerste wat opvalt, is dat het eerste kindelement van het schema-element een **include** element is. Dit element heeft hier één attribuut, met name **schemaLocation**. Dit attribuut heeft als waarde **common.xsd**. Het is vrij duidelijk dat we hiermee willen zeggen dat de definities die zich in het schemadocument **common.xsd** bevinden, in dit document moeten ingevoegd worden. Voor een uitgebreide bespreking van de werking van het **include** element verwijzen we naar [Sectie 3.5.2.] .

Indien we nu, zoals hier, een schema hebben dat in meerdere schema bestanden beschreven wordt, dan moeten we in een XML document, conform dat schema, alleen verwijzen naar het meest specifieke schema document. Dit wil zeggen dat wanneer we de onderlinge samenhang van de schema documenten uittekenen als een boomstructuur met de wortel van boven, naar het laagste document waarmee we het XML document kunnen beschrijven. Indien er in dat schema verwezen wordt naar andere schema's is het de taak van de verwerker om die schema's te localiseren en de definities daarin te verzamelen. Je zou dit kunnen vergelijken met de definitie van Java klassen. Stel dat je een publieke klasse definieert die overerft van een abstracte klasse. Het is nu de taak van de compiler en/of de JVM om bij het gebruik van die publieke klasse de definitie van de abstracte klasse en zijn ouderklassen en/of gebruikte interfaces te verzamelen.

Wat bij de volgende definities opvallend is, is het feit dat we op de meeste plaatsen expliciet de waarde één toekennen aan het attribuut maxOccurs. We doen dit enkel voor de duidelijkheid, maar in principe is dit niet nodig vermits de default waarde van maxOccurs, net zoals voor minOccurs één is.

De volgende definitie die we bespreken is die van **accessibilityType**. Deze definitie valt op omdat dit de eerste definitie is waar een attribuut (**xsd:attribute**) wordt gedefinieerd. Elk element met als type accessibilityType is leeg en heeft één attribuut, genaamd **value**. Dit attribuut heeft een anoniem type, dat in-line wordt gedefinieerd. Dit anoniem type is een eenvoudig type dat wordt afgeleid, met beperking, van het basistype string. Met elk **enumeration** element geven we een toegelaten waarde op, deze waarde wordt aangegeven

met de waarde van het attribuut **value** van het element **enumeration**. Per toegelaten waarde is er één **enumeration** element vereist. In dit geval zijn de toegelaten waarden public, private, protected en package.

3.4.3. void.xsd

Voorbeeld 3.17.

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:include schemaLocation="returntypes.xsd"/>
    <xsd:element name="returnType" type="returnTypeType"/>
    <xsd:complexType name="returnTypeType">
       <xsd:sequence>
          <xsd:element name="returnTypeName" type="returnTypeNameType"/>
          <xsd:element name="methodsReturningThisType"
          type="methodsReturningThisTypeType"/>
       </xsd:sequence>
    </xsd:complexType>
 </xsd:schema>
```

Buiten de elementen include en complexType, zoals in de andere twee schema's, is er als kindelement van het schema-element nog een derde element, genaamd element. Hiermee wordt aangegeven dat de XML documenten die conform zijn met dit schema als hoofdelement het element returnType hebben en dat het type van dit element, in dit geval, het type returnTypeType is.

De elementen die op dit niveau worden gedefinieerd worden globale elementen genoemd. Deze elementen kunnen als hoofdelementen voorkomen van een document instantie, die dit schema gebruiken.

3.5. Bespreking van andere mogelijkheden

In de vorige secties hebben we uitgebreid besproken hoe een basisschema kan geconstrueerd worden. Nu gaan we nog enkele mogelijkheden bespreken die toelaten meer abstractie te gebruiken in een schema. Hierbij bespreken we nog enkele, nog niet besproken elementen. We zullen echter op het einde van dit hoofdstuk lang niet volledig XML Schema besproken hebben. Daarvoor is bijvoorbeeld de Technical Recommendation al veel te uitgebreid. De elementen die we nog gaan bespreken zijn:

- namespaces
- import versus include
- group
- ref
- use
- anyType
- witruimte
- element waarvan de inhoud ofwel string ofwel integer is
- gemengde inhoud
- globale versus lokale declaraties
- commentaar en processing instructions

3.5.1. Namespaces

Hoe kan je aangeven dat je een bepaalde namespace wil gebruiken in XML documenten die moeten voldoen aan het schema? Dit gebeurt met het **targetNamespace** attribuut dat hoort bij het **xsd:schema** element. Een voorbeeldje:

Voorbeeld 3.18.

```
<?xml version="1.0"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bs="http://www.voorbeeld.org/BookSchema"
  targetNamespace="http://www.voorbeeld.org/BookSchema">
     <xsd:element name="BOOK" type="bs:BookType"/>
     <xsd:complexType name="BookType">
        <xsd:sequence>
           <xsd:element name="TITLE" type="xsd:string"/>
           <xsd:element name="AUTHOR" type="xsd:string"
           maxOccurs="unbounded"/>
           <xsd:element name="YEAR" type="bs:YearType"/>
           <xsd:element name="PRICE" type="bs:PriceType"/>
        </xsd:sequence>
     </xsd:complexType>
     <xsd:simpleType name="YearType">
        <xsd:restriction base="xsd:gYear">
           <xsd:minInclusive value="1540"/>
           <xsd:maxInclusive value="2002"/>
        </xsd:restriction>
     </xsd:simpleType>
     <xsd:simpleType name="PriceType">
        <xsd:restriction base="xsd:decimal">
           <xsd:minInclusive value="0.00"/>
           <xsd:fractionDigits value="2"/>
        </xsd:restriction>
     </xsd:simpleType>
  </xsd:schema>
```

Indien we dit schema van voor naar achter bekijken, komen we eerst een elementdeclaratie tegen van het element **BOOK**. Het gevolg van deze definitie is dat het element **BOOK** toegevoegd wordt aan de namespace *http://www.voorbeeld.org/BookSchema*. Merk op dat we nu bij de verwijzing naar een type met een voorvoegsel werken. Dit voorvoegsel **bs** verwijst ook naar de namespace *http://www.voorbeeld.org/BookSchema* en een programma dat dit schema verwerkt weet, omdat beide namespaces gelijk zijn, dat er in dit schema moet gezocht worden naar de definitie van dat type. Na de elementdefinitie hebben we nog drie typedefinities die ook alledrie tot de namespace *http://www.voorbeeld.org/BookSchema* behoren.

3.5.2. import versus include

Zoals reeds eerder aangehaald bij de voorbeelden, is het mogelijk met behulp van **include** de definities van een bestaand schema in te voegen in een ander schema. Het schema dat het include element bevat noemen we het **invoegende document** en het schema waarnaar verwezen wordt met behulp van het include element noemen we het **ingevoegde document**. Het ingevoegde document is dus dat document dat gespecificeerd wordt met behulp van het **schemaLocation** element. Het invoegen van de definities kan je nog het best beschouwen alsof alle definities uit het ingevoegde document deel uitmaken van het invoegende document, dat ze als het ware deel uitmaken van het invoegende document en aan de targetNamespace van het invoegende document worden toegevoegd.

Wel moet je bij het gebruik van het **include** element oppassen. Indien er een

targetNamespace is gedefinieerd, dan moet de namespace van het ingevoegde document gelijk zijn aan de namespace van het invoegende document. Het is ook correct om zowel in het ingevoegde document en het invoegende document geen targetNamespace te specifiëren. Het is een fout wanneer beide een **targetNamespace** definiëren maar deze verschillend is. Wel toegestaan is het declareren van een targetNamespace door het invoegende document, terwijl het ingevoegde document geen targetNamespace declareert. Omgekeerd is echter niet toegestaan. Deze redenering kan gemakkelijk uitgebreid worden naar een hele hiërarchie van documenten, zeker omdat er meerdere include elementen toegestaan zijn in een schema.

Hoe conflictresolutie in geval van dubbele definities zou moeten gebeuren, hebben we na lang zoeken in [TRXMLSCH0], [TRXMLSCH1], [TRXMLSCH2] niet teruggevonden. Het enige relevante dat we vonden was: *"2 Each of the {type definitions}, {element declarations}, {attribute group definitions}, {model group definitions} and {notation declarations} must not contain two or more schema components with the same {name} and {target namespace}."*. We concluderen hieruit dat dubbele definities niet zijn toegestaan, maar hoe conflicten in dat geval zouden aangepakt moeten worden hebben we niet teruggevonden.

Er is ook nog een element waarvan de werking en het gebruik volgens dezelfde regels verloopt als bij het **include** element, maar dit element laat toe om types (zowel eenvoudig als complex), groepen en attribuutgroepen [Sectie 3.5.3.] te herdefiniëren. Hiervoor wordt he **redefine** element gebruikt. De wijzigingen worden binnen het redefine element worden. Een voorbeeldje:

Voorbeeld 3.19.

```
<?xml version="1.0"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bs="http://www.voorbeeld.org/BookSchema"
  targetNamespace="http://www.voorbeeld.org/BookSchema">
     <xsd:redefine schemaLocation="gemeenschappelijk.xsd">
       <xsd:complexType name="BookType">
         <xsd:complexContent>
           <xsd:extension base="bs:BookType">
             <xsd:sequence>
               <xsd:element name="SUMMARY" type="xsd:string"/>
             </xsd:sequence>
           </xsd:extension>
         </xsd:complexContent>
       </xsd:complexType>
     </xsd:redefine>
  </xsd:schema>
```

In dit voorbeeldje gebruiken we de syntax van het **extension** element om het type **BookType** uit te breiden met het element **SUMMARY**. Het extension element is hier een kind van het **complexContent** element om aan te geven dat het hier om complexe inhoud, i.e. attributen en kindelement mogelijk, gaat. Wat wel eigenaardig overkomt is dat we ook gebruik maken van **BookType** als basistype. Mochten we dezelfde definitie van **BookType** willen gebruiken buiten het **redefine** element dan zal dit resulteren in een foutief schema. Juist omdat deze definitie van een complex type met dezelfde naam en in dezelfde namespace als het basistype binnen het **redefine** element staat zal dit niet in een fout resulteren. Deze uitgebreide definitie van **BookType** zal in dit schema ook de enige gekende definitie van **BookType** zijn.

Indien je toch definities uit een bepaald schema wilt gebruiken en dat schema een andere targetNamespace definieert, dan moet je het **import** element gebruiken. Het import element heeft een verplicht attribuut **namespace** dat dient om aan te geven van welke namespace we de definities willen importeren. Dit element heeft ook een optioneel attribuut **schemaLocation** waarvan de waarde dient om aan te geven waar een schema document voor

de namespace, opgegeven bij hetzelfde import element, kan teruggevonden worden. Elke geïmporteerde namespace moet geassocieerd worden met een uniek voorvoegsel, dit op de typische manier van een namespacedeclaratie bij het **schema** element. Een andere vereiste is dat de **import** elementen als de eerste kinderen van het **schema** element moeten voorkomen. We komen het geval van meerdere import elementen tegen wanneer we definities van meerdere schema's wensen te gebruiken. Wel kunnen alleen de globale definities gebruikt worden na het importeren van een bepaalde namespace.

We zullen nu een kort voorbeeldje geven waarin het **import** element gebruikt wordt. In dit voorbeeldje gaan we gebruik maken van een nieuwe namespace *http://www.voorbeeld.org/BookSchema2* en we gaan daarvoor de componenten behorende tot de namespace *http://www.voorbeeld.org/BookSchema* importeren.

Voorbeeld 3.20.

```
<?xml version="1.0"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bs="http://www.voorbeeld.org/BookSchema"
  targetNamespace="http://www.voorbeeld.org/BookSchema2">
     <xsd:import namespace="http://www.voorbeeld.org/BookSchema"/>
     <xsd:element name="BOOK" type="BookType"/>
     <xsd:complexType name="BookType">
        <xsd:complexContent>
           <xsd:extension base="bs:BookType">
              <xsd:sequence>
                 <xsd:element name="SUMMARY" type="xsd:string"/>
              </xsd:sequence>
           </xsd:extension>
        </xsd:complexContent>
     </xsd:complexType>
  </xsd:schema>
```

Het **import** element heeft ook een optioneel **schemaLocation** attribuut dat dient om de schema's, geassocieerd met namespace bij het import element, te localiseren.

Eén van de zaken waarvoor de import elementen nuttig kunnen gebruikt worden is het aanmaken van een typebibliotheek. Zodra XML Schema's meer verspreid geraken zullen auteurs van schema's een aantal basisschema's willen maken die simpele en complexe typedefinities bevatten. Een bepaald type, bijvoorbeeld een geldtype, kan gedefinieerd zijn als een keuze uit alle mogelijke valuta en een auteur zou wel eens een targetNamespace willen associëren met het schema dat dat geldtype definieert, zodat ook andere auteurs van zijn inspanningen kunnen profiteren. Om nu dat type te hergebruiken in een ander schema, met een andere targetNamespace, kunnen we het import element gebruiken.

3.5.3. group

Voor deze bespreking gebruikten wij [TRXS0-27] en [TRXS0-28] als bron.

XML Schema voorziet ook in de mogelijkheid om groepen van elementen of van attributen te definiëren en aan die groep een naam te geven. Deze groepen kunnen dan gebruikt worden om de inhoud van complexe types zeer eenvoudig te modelleren. Dit is handig wanneer de verschillende inhoudsmodellen van sommige elementen zeer grote gelijkenissen vertonen maar toch niet identiek zijn. Het gedeelte dat identiek is kan als een groep gemodelleerd worden en wanneer nodig kan naar die groep verwezen worden. Aan deze groepen en aan de elementen die deel uitmaken van deze groep kunnen vanzelfsprekend ook beperkingen opgelegd worden.

We zullen eerst een voorbeeld geven van de definitie en het gebruik van een groep van elementen, waarna we dit zullen herhalen voor een groep van attributen.

Voorbeeld 3.21. Definitie van een groep van elementen

```
<xsd:group name="elementGroep">
   <xsd:sequence>
      <xsd:element name="elementEen" type="xsd:integer"/>
      <xsd:element name="elementTwee" type="xsd:string"/>
   </xsd:sequence>
</xsd:group>
```

Voorbeeld 3.22. Gebruik van een groep van elementen

```
<xsd:element name="elementDrie">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="locatie" type="xsd:anyURI"/>
         <xsd:group ref="elementGroup"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

In de typedefinitie van dit element wordt verwezen naar de groep van elementen die we de naam **elementGroep** hebben gegeven. We kunnen ook bij andere typedefinities verwijzingen naar die groep gebruiken. Hetgeen in het laatste voorbeeldje staat is qua definitie identiek aan de definitie in het volgende voorbeeldje.

Voorbeeld 3.23. Definitie zonder verwijzing naar groep

```
<xsd:element name="elementDrie">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="locatie" type="xsd:anyURI"/>
         <xsd:element name="elementEen" type="xsd:integer"/>
         <xsd:element name="elementTwee" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Maar het grote verschil hier is dat we de expressieve kracht van de groep kwijt zijn.

We zullen nu dan een voorbeeldje geven waarin een groep van attributen wordt gedeclareerd en vervolgens in een definitie wordt gebruikt, maar we zullen dit voorbeeldje, omwille van de eenvoud ervan, niet verder bespreken.

Voorbeeld 3.24. Gebruik en definitie van attribuutgroepen

```
<xsd:attributeGroup name="commonAttributes">
   <xsd:attribute name="name" type="xsd:string"/>
   <xsd:attribute name="type" type="xsd:string"/>
</xsd:attributeGroup>
<xsd:element name="element">
   <xsd:complexType>
      <xsd:attributeGroup ref="commonAttributes"/>
   </xsd:complexType>
</xsd:element>
<xsd:element name="image">
   <xsd:complexType>
```

```
      <xsd:attributeGroup ref="commonAttributes"/>
      <xsd:attribute name="location" type="xsd:anyURI"/>
   </xsd:complexType>
</xsd:element>
```

3.5.4. ref

Neem nu het volgende voorbeeld:

Voorbeeld 3.25.

```
<xsd:element ref="commentaar" minOccurs="0"/>
```

Hiermee wordt aangegeven dat we een bestaand element gaan gebruiken in plaats van een nieuw element te definiëren. Deze werkwijze is soms te verkiezen. We refereren hier aan een ander element, met name commentaar, dat elders in het schema gedeclareerd is. De waarde van het ref-attribuut moet refereren aan een globaal element, niet aan een element dat deel uitmaakt van een complexe type-definitie. Door het gebruik van het ref-attribuut wordt aangegeven dat op die plaats in een documentinstantie het element commentaar mag voorkomen, consistent met het type van dat element.

Het ref-attribuut kan ook gebruikt worden om te refereren aan groepen van elementen (i.e. naar groepen gedefinieerd door het element **xsd:group**), maar ook om te refereren aan groepen van attributen (i.e. naar attribuutgroepen gedefinieerd door het element **xsd:attributeGroup**), of om te refereren aan een attribuut (i.e. naar attributen gedefinieerd door het element **xsd:attribute**). Indien het ref-attribuut gebruikt wordt **moet** er gerefereerd worden aan een globale declaratie.Het is mogelijk om op die manier lussen te creëren, maar lussen zijn niet toegestaan. Wanneer je bijvoorbeeld gebruik maakt van het **xsd:group** element mag binnen die groep niet aan die groep gerefereerd worden, op welke diepte dan ook. Het volgende voorbeeld is dan ook geen geldige constructie:

Voorbeeld 3.26. Ongeldige constructie

```
<xsd:group name="elementenGroep">
   <xsd:sequence>
      <xsd:element name="elementenEen" type="integer"/>
      <xsd:group ref="elementenGroep"/>
   </xsd:sequence>
</xsd:group>
```

Binnen de definitie van de groep wordt, met behulp van het **ref** attribuut verwezen naar de groep zelf. De diepte waarop dit gebeurt is van geen belang, het is gewoonweg een foute constructie.

3.5.5. use

Attributen mogen nul of éénmaal voorkomen, maar niet een ander bepaald aantal keer. Het use-attribuut dient daarom om aan te geven of het gebruik van een attribuut required (vereist), optional (optioneel) of prohibited (verboden) is. De waarde prohibited zou je kunnen gebruiken wanneer je een element herdefinieert en het attribuut in die context, waar je het voor wilt gebruiken, niet mag voorkomen. Het volgende voorbeeld eist dat het attribuut orderDate gebruikt wordt in het bijhorende element.

Voorbeeld 3.27.

```
<xsd:attribute name="orderDate" type="xsd:date" use="required"/>
```

3.5.6. anyType

Het **anyType** type vertegenwoordigt een abstract type. Dit abstracte type is het basistype van welk alle eenvoudige en complexe types zijn afgeleid. Het **anyType** type legt totaal geen beperkingen op aan de inhoud van het element. De mogelijkheid bestaat om **anyType** type te gebruiken zoals eender welk type. Een voorbeeldje:

Voorbeeld 3.28.

```
<xsd:element name="containerElement" type="anyType"/>
```

De inhoud van dit element is onbeperkt. De inhoud kan zowel **10,25** als **De Euro is ingevoerd op 1 januari 2002.** zijn, maar de inhoud kan evengoed bestaan uit een aantal elementen of zelfs een mengeling van tekens en elementen. Het **anyType** type is het standaardtype wanneer er bij de declaratie van een element geen type wordt opgegeven. Het element in het vorige voorbeeldje hadden we dus evengoed als volgt kunnen definiëren:

Voorbeeld 3.29.

```
<xsd:element name="containerElement"/>
```

3.5.7. witruimte

Voor de bespreking van witruimte hebben we [TRXS2-436] als bron gebruikt.

De schema auteur kan definiëren hoe de witruimte in XML documenten behandeld moet worden, maar dit alleen voor types die afgeleid zijn van het type **string**. De volgende constructie, hier geïllustreerd met een voorbeeldje, geeft aan hoe dit kan gespecificeerd worden:

Voorbeeld 3.30. Witruimte

```
...
 <xsd:simpleType name="myString">
    <xsd:restriction base="string">
       <xsd:whiteSpace value="collapse"/>
    </xsd:restriction>
 </xsd:simpleType>
...
```

In dit voorbeeld definiëren we een type **myString** dat afgeleid wordt van het type string. Met de waarde van het **value** attribuut van het **whiteSpace** element geven we aan hoe de witruimte behandeld moet worden bij elementen/attributen die dit type hebben. Het **value** attribuut kan in deze context drie verschillende waarden aannemen:

- **preserve**: er vindt geen normalisatie van witruimte plaats, de inhoud van het

element/waarde van het attribuut wordt niet veranderd
- **replace**: alle voorkomens van tab, line feed en carriage return worden vervangen door een spatie
- **collapse**: na het verwerken, zoals bij **replace**, worden alle sequenties van spaties herleid tot één enkele spatie, waarna ook de inhoud wordt getrimd (zodat er geen spatie is die het eerste of laatste teken is van de inhoud)

Het element **whiteSpace** is alleen van toepassing op atomaire [TRXS2-2511] en lijst [Sectie 3.3.3.3.3.] datatypes. Voor alle atomaire datatypes, buiten string of de types afgeleid van string, moet witruimte behandeld worden op de manier voorgeschreven door **collapse** en dit kan niet overschreven worden door de schema auteur. Voor het type string is de standaard waarde **preserve** en voor elke type afgeleid van string is één van de mogelijke drie waarden toegestaan. Voor alle datatypes, afgeleid via list [Sectie 3.3.3.3.3.] , wordt witruimte behandeld via **collapse** en ook dit kan niet veranderd worden. Bij union datatypes [Sectie 3.3.3.3.2.] wordt het type bepaald op basis van de samenstellende types, zoals besproken in [Sectie 3.5.8.] . De normalisatie van de witruimte wordt in dit geval bepaald door de voorgeschreven normalisatie van het type waartegen het union datatype gevalideerd is.

3.5.8. Element waarvan de inhoud ofwel string ofwel integer is

Het kan zijn dat je wil dat de inhoud van een element niet beperkt is tot één type, maar dat je de mogelijkheid wilt bieden dat voor de inhoud van een element gekozen kan worden uit meerdere types, afhankelijk van de context. **Let wel**: de types die je kan gebruiken als keuzemogelijkheden zijn beperkt tot de atomaire types en de lijsttypes. Stel dat we een element **grootte** willen definiëren en we de mogelijkheid willen bieden om dit in te vullen met een **integer** of met een **string**. Dit definiëren we als volgt:

Voorbeeld 3.31.

```
<xsd:element name="grootte">
   <xsd:simpleType>
      <xsd:union>
         <xsd:simpleType>
            <xsd:restriction base="xsd:integer"/>
         </xsd:simpleType>
         <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
         </xsd:simpleType>
      </xsd:union>
   </xsd:simpleType>
</xsd:element>
```

We maken hier gebruik van het **union** element om de mogelijke types te verenigen binnen één **xsd:simpleType** element. De volgorde waarin de mogelijke types worden aangegeven in de typedefinitie is van belang wanneer we een XML document gaan valideren [Sectie 3.3.2.] . Een validator zal de inhoud van het **grootte** element proberen te valideren met de mogelijke types in de volgorde dat we ze hebben opgegeven. We zullen hier een voorbeeldje van geven:

Voorbeeld 3.32.

```
<grootte> 5 </grootte>
<grootte> klein </grootte>
<grootte xsi:type="xsd:string"> 5 </grootte>
```

We hebben hier driemaal het element **grootte** met een bepaalde inhoud. In het eerste geval zal

de inhoud als een **integer** gevalideerd worden, in het tweede en derde geval wordt de inhoud als een **string** gevalideerd. In het tweede geval is dit vrij logisch, in het derde geval is dit omdat we de evaluatievolgorde overschrijven door gebruik te maken van het **xsi:type** attribuut om dit aan te geven, waarmee we expliciet het type van het element aangeven. Met het voorvoegsel **xsi** verwijzen we hier naar de namespace **http://www.w3.org/2001/XMLSchema-instance**. Deze namespace definieert een aantal attributen die rechtstreeks in XML documenten gebruikt kunnen worden, op voorwaarde dat ze met de juiste namespace geassocieerd worden.

3.5.9. Gemengde inhoud

Het is ook mogelijk om aan te geven dat een element zowel tekens (character data) als kindelement kan bevatten [TRXS0-252]. Dit wordt aangegeven door door bij een typedefinitie het attribuut **mixed** de waarde **true** te geven. Een voorbeeldje:

Voorbeeld 3.33.

```
<xs:complexType name="TitleType" mixed="true">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="INDEX" type="xs:string"/>
      <xs:element name="EMPH" type="xs:string"/>
   </xs:choice>
</xs:complexType>
```

Hier geven we aan dat een element van het type **TitleType** gewone tekst kan bevatten, maar daarnaast ook nog een willekeurig aantal kindelementen kan hebben met de naam **EMPH** en/of **INDEX**. De inhoud van een element met dit type (in het volgende voorbeeld TITLE) kan er dan als volgt uitzien:

Voorbeeld 3.34.

```
<TITLE>
   <EMPH> Hoofdstuk </EMPH>
over
   <INDEX> XML Schema </INDEX>
</TITLE>
```

3.5.10. Globale versus lokale declaraties

Een globale declaratie van een element (of van een attribuut) is een declaratie waarvan het ouderelement het **xsd:schema** element is. Een lokale declaratie hebben we indien het ouderelement van de declaratie **niet** het **xsd:schema** element is. Globale declaraties van elementen zijn onderhevig aan een aantal beperkingen, omdat deze elementen als hoofdelement van een XML document, conform het schema, mogen voorkomen. Globale attribuutdeclaraties kunnen ook voorkomen en er kan naar verwezen worden, maar ze zijn ook aan beperkingen onderhevig.

Een eerste beperking is dat globale declaraties geen referenties kunnen bevatten, zij moeten eenvoudige en complexe types rechtstreeks identificeren. Concreet wil dit zeggen dat het **ref** attribuut niet mag gebruikt worden, maar dat het **type** attribuut moet gebruikt worden, of er moet gewerkt worden met een anonieme type-definitie.

Ook kunnen er geen cardinaliteitsbeperkingen opgelegd worden aan deze elementen, vermits in XML documenten slechts één hoofdelement is toegestaan. Cardinaliteitsbeperkingen kunnen wel opgelegd worden aan de declaraties van lokale elementen en attributen, zelfs indien de lokale declaraties refereren aan globale declaraties. Dit wil zeggen dat globale declaraties van elementen de volgende attributen niet kunnen bevatten: **minOccurs** en **maxOccurs**.

Bij globale attribuutdefinities mag het attribuut **use** niet gebruikt worden. Dit wil zeggen dat we niet kunnen afdwingen dat het gedefinieerde attribuut bijvoorbeeld gebruikt **moet** worden. Dit is logisch omdat een attribuut dat globaal gedefinieerd is niet geassocieerd is met een element. We kunnen alleen opleggen dat een attribuut gebruikt **moet** worden indien het attribuut geassocieerd is met een element. Het is wel mogelijk om bij lokale attribuutdeclaraties met behulp van het **ref** attribuut te verwijzen naar globale attribuutdeclaraties. Omdat er bij een lokale attribuutdeclaratie wel een associatie is met een element, mag het attribuut **use** dan wel gebruikt worden. Wel kunnen attributen die globaal zijn gedefinieerd geïmporteerd worden om te gebruiken in documenten die bijvoorbeeld niet het volledige schema willen gebruiken. Als voorbeeld hiervan kunnen we denken aan de attributen uit de **http://www.w3.org/2001/XMLSchema-instance** namespace.

3.5.11. commentaar en processing instructions

Zoals besproken in [Sectie 2.3.4.] maakt commentaar geen deel uit van de tekstuele inhoud van het XML document. In [TRXS1-314] wordt vermeld dat commentaar en processing instructions genegeerd worden tijdens het valideren. Dit kan als verklaring beschouwd worden waarom er voor deze constructies geen voorzieningen zijn getroffen in de W3C XML Schema Language.

3.6. Omzetting van een DTD naar XML Schema

In deze sectie gaan we stap voor stap een voorbeeld bespreken van een omzetting van een DTD naar een XML Schema. Er bestaan ook programma's die deze omzetting automatisch doen. Wat je dan krijgt is wel een correct schema, maar qua leesbaarheid en aanpasbaarheid is zo een schema nihil. Stel dat je op verschillende plaatsen in je DTD gebruik maakt van een bepaald element, dan zal dat element niet op één plaats worden gedeclareerd en dan hergebruikt worden. Ook verwachten wij bij sommige van deze programma's dat indien er een hele hiërarchie van elementen is die op een aantal plaatsen wordt gebruikt, deze hiërarchie dan ook op die plaatsen telkens zal voorkomen in het resulterende schema. Wordt bijvoorbeeld een bepaalde hiërarchie van elementen vijfmaal gebruikt in de DTD, dan zal deze hiërarchie ook vijfmaal in het schema voorkomen, in plaats van deze op één plaats te definiëren en er op die vijf plaatsen naar te verwijzen. We zouden kunnen zeggen dat daar geen abstractie van wordt gemaakt.

We gaan nu een DTD geven en die gaan we dan stap voor stap omzetten naar een XML Schema. Om het verschil aan te tonen met een programma dat deze omzetting automatisch doet, zullen we bij elke stap ook het overeenkomstige gedeelte van het gegenereerde schema geven. Op deze manier willen we laten zien dat deze programma's wel een correcte omzetting leveren, maar dat in het resulterende schema het potentieel van de XML Schema taal niet benut wordt (we kunnen hier alleen spreken over [DTD2XS] omdat wij alleen dit programma bekeken hebben. Het kan heel goed zijn dat er commerciële of gratis producten zijn die al een stap verder staan, maar die hebben wij niet getest).

3.6.1. De DTD

Voor de uitleg over DTD's is een beroep gedaan op [ZVONDTDTUT].

We geven in deze sectie de DTD die we gaan gebruiken om de omzetting naar XML Schema aan te geven. Omdat niet bij iedereen de voorkennis over DTD's even groot is, zullen we ook kort deze DTD bespreken. Deze bespreking houdt onder andere in hoe elementen, attributen en entiteiten kunnen gedefinieerd worden.

DTD 3.2.

```
<!ELEMENT returnType (returnTypeName, methodsReturningThisType,
variablesOfThisType)>
<!ELEMENT returnTypeName (package*, name, (isResolved|possibilities),
bracket*)>
<!ELEMENT isResolved EMPTY>
<!ELEMENT package #PCDATA>
<!ELEMENT name #PCDATA>
<!ELEMENT possibilities (returnTypeName*)>
<!ELEMENT bracket EMPTY>
<!ELEMENT methodsReturningThisType (method*)>
<!ELEMENT method (documentationBlock?, modifierBlockMethod, type, name,
parent, formalParameters, throwsClause)>
<!ELEMENT documentationBlock (descriptionBlock?, tagBlock*)>
<!ELEMENT descriptionBlock #PCDATA>
<!ELEMENT tagBlock (tag, informal?, formal?)>
<!ELEMENT tag #PCDATA>
<!ELEMENT informal #PCDATA>
<!ELEMENT formal (formalSentence+)>
<!ELEMENT formalSentence #PCDATA>
<!ELEMENT modifierBlockMethod (accessibility, abstract?, static?, final?,
synchronized?, native?, strictfp?)>
<!ATTLIST accessibility value (public|private|protected|package)
#REQUIRED>
<!ELEMENT accessibility EMPTY>
<!ELEMENT abstract EMPTY>
<!ELEMENT static EMPTY>
<!ELEMENT final EMPTY>
<!ELEMENT synchronized EMPTY>
<!ELEMENT native EMPTY>
<!ELEMENT strictfp EMPTY>
<!ELEMENT type (returnTypeName)>
<!ELEMENT parent (returnTypeName)>
<!ELEMENT formalParameters (parameter*)>
<!ELEMENT parameter (final?, type, name)>
<!ELEMENT throwsClause (exception*)>
<!ELEMENT exception (returnTypeName)>
<!ELEMENT variablesOfThisType (variable*)>
<!ELEMENT variable (documentationBlock?, modifierBlockVariable, type,
name, parent)>
<!ELEMENT modifierBlockVariable (accessibility, static?, final?,
transient?, volatile?)>
<!ELEMENT transient EMPTY>
<!ELEMENT volatile EMPTY>
```

Deze DTD definieert een structuur voor documenten die aan deze DTD willen voldoen.

DTD 3.3.

```
<!ELEMENT returnType (returnTypeName, methodsReturningThisType,
variablesOfThisType)>
```

Het eerste element dat wordt gedefinieerd is het element returnType. Vervolgens staan er drie

elementnamen tussen haakjes. Hiermee wordt aangegeven dat deze elementen precies éénmaal moeten voorkomen en in de aangegeven volgorde als kind van het element returnType. Buiten deze drie elementen mag het element returnType niks bevatten.

DTD 3.4.

```
<!ELEMENT returnTypeName (package*, name, (isResolved|possibilities),
bracket*)>
<!ELEMENT isResolved EMPTY>
<!ELEMENT package #PCDATA>
<!ELEMENT name #PCDATA>
<!ELEMENT possibilities (returnTypeName*)>
<!ELEMENT bracket EMPTY>
```

Vervolgens definieert deze DTD waaraan het element returnTypeName moet voldoen. In deze volgorde moeten voorkomen als kindelementen: nul, één of meerdere keren (*) het element package, precies éénmaal het element name, vervolgens één van de elementen isResolved of possibilities (isResolved|possibilities), niet beide, en dit nul of éénmaal (?). Tenslotte kan het element bracket nog nul, één of meerdere keren volgen. Vervolgens worden de hier geïntroduceerde elementen gedefinieerd. De elementen isResolved en bracket worden als lege elementen gedefinieerd. De elementen package en name daarentegen kunnen zelf (tekstuele) inhoud (#PCDATA) bevatten, maar geen kindelementen. Het element possibilities tenslotte bevat nul, één of meerdere malen het element returnTypeName, waarmee we deze paragraaf begonnen zijn.

DTD 3.5.

```
<!ELEMENT methodsReturningThisType (method*)>
<!ELEMENT method (documentationBlock?, modifierBlockMethod, type, name,
parent, formalParameters, throwsClause)>
<!ELEMENT documentationBlock (descriptionBlock?, tagBlock*)>
<!ELEMENT descriptionBlock #PCDATA>
<!ELEMENT tagBlock (tag, informal?, formal?)>
<!ELEMENT tagBlock #PCDATA>
<!ELEMENT informal #PCDATA>
<!ELEMENT formal (formalSentence+)>
<!ELEMENT formalSentence #PCDATA>
```

Het element methodsReturningThisType bevat nul, één of meerdere keren het element method. De inhoud van method is als volgt opgebouwd: nul of éénmaal het element documentationBlock, de elementen modifierBlockMethod, type, name, parent, formalParameters en throwsClause komen telkens éénmaal voor in de voornoemde volgorde. Een documentationBlock bestaat uit een optioneel (nul of één) descriptionBlock dat gewoon tekst bevat. Daarnaast bevat een documentationBlock ook nog nul, één of meerdere keren een tagBlock. Dit tagBlock bevat welgeteld één tag-element, naast de optionele elementen informal en formal. Een informal-element bevat gewoon tekst, terwijl een formal-element één of meer formalSentence-elementen bevat, die op hun beurt gewoon tekst bevatten.

DTD 3.6.

```
<!ELEMENT modifierBlockMethod (accessibility, abstract?, static?, final?,
synchronized?, native?, strictfp?)>
<!ATTLIST    accessibility    value    (public|private|protected|package)
#REQUIRED>
<!ELEMENT accessibility EMPTY>
```

```
<!ELEMENT abstract EMPTY>
<!ELEMENT static EMPTY>
<!ELEMENT final EMPTY>
<!ELEMENT synchronized EMPTY>
<!ELEMENT native EMPTY>
<!ELEMENT strictfp EMPTY>
```

Het element modifierBlockMethod bevat naast één verplicht element accessibility ook nog de optionele elementen abstract, static, final, synchronized, native en strictfp. Dit zijn allemaal lege elementen, maar het verplichte element accessibility heeft één vereist attribuut, met name value, dat de opgesomde waarden, public, private, protected en package, mag aannemen. Andere waarden zijn niet toegestaan.

DTD 3.7.

```
<!ELEMENT type (returnTypeName)>
<!ELEMENT parent (returnTypeName)>
<!ELEMENT formalParameters (parameter*)>
<!ELEMENT parameter (final?, type, name)>
<!ELEMENT throwsClause (exception*)>
<!ELEMENT exception (returnTypeName)>
```

De elementen type en parent hebben elk als kindelement een returnTypeName-element. Het formalParameters element bevat nul, één of meer parameter-elementen, die op hun beurt zijn opgebouwd uit een optioneel final-element en verplichte (exact één) type- en name-elementen. Tenslotte bevat het throwsClause element nul, één of meer exception-elementen, die op hun beurt telkens één returnTypeName element bevatten.

DTD 3.8.

```
<!ELEMENT variablesOfThisType (variable*)>
<!ELEMENT variable (documentationBlock?, modifierBlockVariable, type,
name, parent)>
<!ELEMENT modifierBlockVariable (accessibility, static?, final?,
transient?, volatile?)>
<!ELEMENT transient EMPTY>
<!ELEMENT volatile EMPTY>
```

Tenslotte, definiëren we het element variablesOfThisType, dat nul, één of meerdere malen het element variable bevat. Een variable-element is opgebouwd uit een optioneel documentationBlock, en de verplichte elementen modifierBlockVariable, type, name en parent. Alleen het element modifierBlockVariable heeft nog geen definitie, en dat wordt daarom op deze plaats gedefinieerd en wel als volgt: het bevat één verplicht element accessibility, naast de optionele elementen static, final, transient en volatile. We moeten nu alleen nog transient en volatile definiëren en deze worden gedefinieerd als lege elementen.

Hiermee besluiten wij de bespreking van de DTD die wij gaan gebruiken om te tonen hoe wij deze omzetting naar een schema hebben aangepakt.

3.6.2. DTD omzetten naar XML Schema

We gaan nu stap voor stap de besproken DTD omzetten naar een schema. We zullen dit doen door eerst het gedeelte van de DTD te geven dat we zullen omzetten naar een overeenkomstig schema, en dat dan in het kort bespreken. Daarna kunnen we dan alle gedeelten samenvoegen

en bekomen we een volledig schema, waaraan we dan nog enkele aanpassingen zullen aanbrengen. Uiteindelijk vergelijken we ons resultaat met een schema bekomen met een programma dat DTD's kan omzetten naar schema's. We merken op dat we bij de verschillende stappen niet alles gaan vermelden. De lezer moet er rekening mee houden dat voor sommige declaraties in het schema enkele stappen verder of terug gekeken moet worden, indien er onduidelijkheden zouden zijn. Op het einde van deze sectie geven we ook nog eens het volledige schema zoals we dat opgebouwd hebben, evenwel niet vooraleer nog enkele bedenkingen geformuleerd te hebben.

DTD 3.9. DTD

```
<!ELEMENT    returnType    (returnTypeName,    methodsReturningThisType,
variablesOfThisType)>
```

Omgezet geeft dit:

Voorbeeld 3.35. XML Schema

```
<xsd:element name="returnType" type="returnTypeType"/>
<xsd:complexType name="returnTypeType">
   <xsd:sequence>
      <xsd:element name="returnTypeName" type="returnTypeNameType"/>
      <xsd:element name="methodsReturningThisType"
      type="methodsReturningThisTypeType"/>
      <xsd:element name="variablesOfThisType"
      type="variablesOfThisTypeType"/>
   </xsd:sequence>
</xsd:complexType>
```

Eerst definiëren we een element met de naam returnType en met dit element associëren wij het type returnTypeType. Dit type wordt gedefinieerd als een complex type, omdat er nog kindelementen kunnen zijn. We eisen voor dit type een strikte volgorde van de elementen (xsd:sequence), en als volgorde leggen we op: returnTypeName, methodsReturningThisType, variablesOfThisType. Deze elementen associëren we elk met hun eigen type (de definities van deze types komen later nog). Wij hebben ervoor gekozen om niet met anonieme type-definities, maar wel met benoemde type-definities te werken. De naamconventie die we voor de types hebben gevolgd is de naam van het element gevolgd door **Type**.

3.6.2.1. Conflicten

Wat indien er door de naamconventie voor types conflicten ontstaan tussen een typenaam en een elementnaam? We bedoelen hiermee het geval dat je een type en een element declareert die beide dezelfde naam hebben. Volgens de standaard [TRXS0-223] is er in dit geval geen sprake van een conflict. Dit valt vrij eenvoudig in te zien. Stel dat we de situatie hebben waarin een type en een element dezelfde naam hebben. Wanneer dit type aan een element toegekend wordt, dan zal er, bijvoorbeeld wanneer we de geldigheid controleren van dit schema, alleen gekeken worden indien er een type met die naam bestaat. Of er een element met die naam bestaat maakt niets uit, want er zal alleen naar de type-definities gekeken worden.

Wel zijn er nog een aantal andere gevallen waarbij conflicten kunnen voorkomen:

- er is sprake van een conflict wanneer de ontwikkelaar van een schema twee types definieert met dezelfde naam
- indien we twee elementdeclaraties hebben die beiden een element met dezelfde naam

definiëren, maar deze twee declaraties voorkomen binnen verschillende typedefinities, dan is er geen conflict. Hieruit volgt dat indien er twee elementdefinities met dezelfde naam binnen één typedefinitie voorkomen er sprake is van een conflict.

- indien je bijvoorbeeld een type **string** zelf zou definiëren, dan geeft dit geen conflict met het ingebouwde type **string** van XML Schema. Dit lijkt tegenstrijdig, maar deze types behoren tot een verschillende namespace en daarom is dat geen oorzaak van een conflict.

DTD 3.10. DTD

```
<!ELEMENT returnTypeName (package*, name, (isResolved|possibilities),
bracket*)>
<!ELEMENT isResolved EMPTY>
<!ELEMENT package #PCDATA>
<!ELEMENT name #PCDATA>
<!ELEMENT possibilities (returnTypeName+)>
<!ELEMENT bracket EMPTY>
```

Voorbeeld 3.36. XML Schema

```
<xsd:element name="returnTypeName" type="returnTypeNameType"/>
<xsd:element name="isResolved" type="emptyType"/>
<xsd:element name="package" type="xsd:string"/>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="possibilities" type="possibilitiesType"/>
<xsd:element name="bracket" type="emptyType"/>
<xsd:complexType name="returnTypeNameType">
  <xsd:sequence>
    <xsd:element ref="package" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="name"/>
    <xsd:choice minOccurs="0" maxOccurs="1">
      <xsd:element ref="possibilities"/>
      <xsd:element ref="isResolved"/>
    </xsd:choice>
    <xsd:element ref="bracket" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="emptyType"/>
<xsd:complexType name="possibilitiesType">
  <xsd:sequence>
    <xsd:element ref="returnTypeName" minOccurs="1"
    maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

We volgen hier weer dezelfde strategie. Zoals de attributen bij bijvoorbeeld package aangeven kan het package element nul of meerdere keren voorkomen, en dit element heeft als type het ingebouwde eenvoudige type string. We zien hier dat #PCDATA (ongetypeerd) vertaald wordt naar een type string (getypeerd). Er is ook de keuze tussen het isResolved en het possibilities element. Deze elementen moeten niet voorkomen, maar als ze voorkomen, is hun aantal beperkt tot één en mag slechts één van deze twee elementen voorkomen. Merk op dat we bij het xsd:choice element expliciet een waarde voor maxOccurs hebben opgegeven. Dit is niet nodig, vermits de standaard waarde van dit attribuut 1 is. Deze elementen hebben hun eigen type, maar het type van isResolved is wel bijzonder. Het type dat we hier gebruiken is om aan te geven dat het hier om een leeg element gaat. De mogelijkheid bestaat ook in XML Schema om dit te specifiëren, en dit wordt gedaan door een leeg type, een type dat geen vermelding maakt van attributen of elementen, te definiëren, wat wij geheel intuïtief

emptyType hebben genoemd. We moeten dit definiëren als een complex type omdat een leeg element bij het W3C wordt opgevat als een element zonder inhoud, maar mogelijk met attributen. Ook dienen we op te merken dat bij possibilitiesType, alhoewel er maar één kindelement wordt gedefinieerd, we toch moeten aangeven in welke volgorde dit moet voorkomen (xsd:sequence).

We merken hier op dat we eerst de elementen hebben gedeclareerd en waar we ze later nodig hebben, hebben we ernaar verwezen met het **ref** attribuut. Op die manier moeten we geen elementen tweemaal definiëren. Het reeds gedeclareerde element moet wel binnen de scope van de nieuwe definitie vallen. Tot de scope behoort alles wat op een hoger niveau gedefinieerd is, maar ook de globale declaraties. Niet tot de scope behoren definities die op een lager niveau, i.e. in de kindelementen voorkomen.

DTD 3.11. DTD

```
<!ELEMENT methodsReturningThisType (method*)>
<!ELEMENT method (documentationBlock?, modifierBlockMethod, type, name,
parent, formalParameters, throwsClause)>
<!ELEMENT documentationBlock (descriptionBlock?, tagBlock*)>
<!ELEMENT descriptionBlock #PCDATA>
<!ELEMENT tagBlock (tag, informal?, formal?)>
<!ELEMENT tag #PCDATA>
<!ELEMENT informal #PCDATA>
<!ELEMENT formal (formalSentence+)>
<!ELEMENT formalSentence #PCDATA>
```

Voorbeeld 3.37. XML Schema

```
<xsd:element name="methodsReturningThisType"
type="methodsReturningThisTypeType"/>
<xsd:element name="method" type="methodType"/>
<xsd:element name="documentationBlock" type="documentationBlockType"/>
<xsd:element name="descriptionBlock" type="xsd:string"/>
<xsd:element name="tagBlock" type="tagBlockType"/>
<xsd:element name="tag" type="xsd:string"/>
<xsd:element name="informal" type="xsd:string"/>
<xsd:element name="formal" type="formalType"/>
<xsd:element name="formalSentence" type="xsd:string"/>
<xsd:complexType name="methodsReturningThisTypeType">
   <xsd:sequence>
      <xsd:element ref="method" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="methodType">
   <xsd:sequence>
      <xsd:element ref="documentationBlock" minOccurs="0"
      maxOccurs="1"/>
      <xsd:element ref="modifierBlockMethod"/>
      <xsd:element ref="type"/>
      <xsd:element ref="name"/>
      <xsd:element ref="parent"/>
      <xsd:element ref="formalParameters"/>
      <xsd:element ref="throwsClause"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="documentationBlockType">
   <xsd:sequence>
      <xsd:element ref="descriptionBlock" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="tagBlock" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="tagBlockType">
   <xsd:sequence>
      <xsd:element ref="tag"/>
      <xsd:element ref="informal" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="formal" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="formalType">
   <xsd:sequence>
      <xsd:element ref="formalSentence" minOccurs="1"
      maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

Ook in deze stap worden de elementen omgezet in hun XML Schema equivalent. Het is duidelijk dat we bij het omzetten veelvuldig gebruik maken van de volgende regels:

- #PCDATA wordt xsd:string (in een eerste fase, in een tweede fase kan dit verstrengd worden en we zullen dit ook doen waar mogelijk)
- ? verkrijgen we door minOccurs op nul te zetten en maxOccurs op één (of maxOccurs geen waarde te geven aangezien die standaard op één staat)
- * verkrijgen we door minOccurs op nul te zetten en maxOccurs op unbounded
- + verkrijgen we door minOccurs op één te zetten en maxOccurs op unbounded
- bijna alle elementen krijgen hun eigen type, en de naam van dat type is de elementnaam gevolgd door **Type**

In dit gedeelte vinden we ook referenties terug naar elementen die we nog niet gedefinieerd hebben. Deze elementen zullen in een latere stap gedefinieerd worden.

DTD 3.12. DTD

```
<!ELEMENT modifierBlockMethod (accessibility, abstract?, static?, final?,
synchronized?, native?, strictfp?)>
<!ATTLIST    accessibility    value    (public|private|protected|package)
#REQUIRED>
<!ELEMENT accessibility EMPTY>
<!ELEMENT abstract EMPTY>
<!ELEMENT static EMPTY>
<!ELEMENT final EMPTY>
<!ELEMENT synchronized EMPTY>
<!ELEMENT native EMPTY>
<!ELEMENT strictfp EMPTY>
```

Voorbeeld 3.38. XML Schema

```
<xsd:element name="modifierBlockMethod"
type="modifierBlockMethodType"/>
<xsd:element name="accessibility" type="accessibilityType"/>
<xsd:element name="abstract" type="emptyType"/>
<xsd:element name="static" type="emptyType"/>
<xsd:element name="final" type="emptyType"/>
<xsd:element name="synchronized" type="emptyType"/>
<xsd:element name="native" type="emptyType"/>
<xsd:element name="strictfp" type="emptyType"/>
<xsd:complexType name="modifierBlockMethodType">
   <xsd:sequence>
      <xsd:element ref="accessibility"/>
      <xsd:element ref="abstract" minOccurs="0" maxOccurs="1"/>
```

```
      <xsd:element ref="static" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="final" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="synchronized" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="native" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="strictfp" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="accessibilityType">
   <xsd:attribute name="value" use="required">
      <xsd:simpleType>
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="public"/>
            <xsd:enumeration value="private"/>
            <xsd:enumeration value="protected"/>
            <xsd:enumeration value="package"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:attribute>
</xsd:complexType>
```

We hebben hier opnieuw te maken met een aantal elementen die vrij eenvoudig om te zetten zijn naar hun schema-equivalent. Hier hebben we ervoor gekozen om ook bij de (optionele) elementen maxOccurs de waarde één te geven. Dit is enkel voor de duidelijkheid, een andere reden hiervoor is er niet. Wat we hier wel nog hebben is de definitie van een vereist attribuut bij het element accessibility. Het type accessilibityType definieert het attribuut value met een anoniem simpel type. Het type wordt gedeclareerd als een beperking op het basistype xsd:string. En met xsd:enumaration geven we de waarden op die dit attribuut mag aannemen. Per mogelijke waarde moeten we één xsd:enumeration element opgeven. Merk ook het gebruik op van het attribuut use bij het element xsd:attribute. Het vereist zijn wordt aangegeven door de waarde required te geven aan het use attribuut van het xsd:attribute element. Bij een attribuut kan geen gebruik gemaakt worden van minOccurs en maxOccurs omdat een attribuut ofwel voorkomt ofwel niet. Het is niet toegestaan dat een attribuut bijvoorbeeld driemaal voorkomt.

DTD 3.13. DTD

```
<!ELEMENT type (returnTypeName)>
<!ELEMENT parent (returnTypeName)>
<!ELEMENT formalParameters (parameter*)>
<!ELEMENT parameter (final?, type, name)>
<!ELEMENT throwsClause (exception*)>
<!ELEMENT exception (returnTypeName)>
```

Voorbeeld 3.39. XML Schema

```
<xsd:element name="type" type="typeType"/>
<xsd:element name="parent" type="parentType"/>
<xsd:element name="formalParameters" type="formalParametersType"/>
<xsd:element name="parameter" type="parameterType"/>
<xsd:element name="throwsClause" type="throwsClauseType"/>
<xsd:element name="exception" type="exceptionType"/>
<xsd:complexType name="typeType">
   <xsd:sequence>
      <xsd:element ref="returnTypeName"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="parentType">
   <xsd:sequence>
      <xsd:element ref="returnTypeName"/>
```

```
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="formalParametersType">
    <xsd:sequence>
      <xsd:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="parameterType">
    <xsd:sequence>
      <xsd:element ref="final"/>
      <xsd:element ref="type"/>
      <xsd:element ref="name"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="throwsClauseType">
    <xsd:sequence>
      <xsd:element ref="exception" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="exceptionType">
    <xsd:sequence>
      <xsd:element ref="returnTypeName"/>
    </xsd:sequence>
  </xsd:complexType>
```

Deze omzettingen van deze stap en van de volgende stap verlopen volledig analoog aan de andere stappen die we nu reeds besproken hebben. Daarom gaan we deze stappen hier wel geven, maar niet verder uitgebreid bespreken.

DTD 3.14. DTD

```
<!ELEMENT variablesOfThisType (variable*)>
<!ELEMENT variable (documentationBlock?, modifierBlockVariable, type,
name, parent)>
<!ELEMENT modifierBlockVariable (accessibility, static?, final?,
transient?, volatile?)>
<!ELEMENT transient EMPTY>
<!ELEMENT volatile EMPTY>
```

Voorbeeld 3.40. XML Schema

```
  <xsd:element name="variablesOfThisType"
  type="variablesOfThisTypeType"/>
  <xsd:element name="variable" type="variableType"/>
  <xsd:element name="modifierBlockVariable"
  type="modifierBlockVariableType"/>
  <xsd:element name="transient" type="emptyType"/>
  <xsd:element name="volatile" type="emptyType"/>
  <xsd:complexType name="variablesOfThisTypeType">
    <xsd:sequence>
      <xsd:element ref="variable" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="variableType">
    <xsd:sequence>
      <xsd:element ref="documentationBlock" minOccurs="0"
      maxOccurs="1"/>
      <xsd:element ref="modifierBlockVariable"/>
      <xsd:element ref="type"/>
      <xsd:element ref="name"/>
      <xsd:element ref="parent"/>
    </xsd:sequence>
```

```
</xsd:complexType>
<xsd:complexType name="modifierBlockVariableType">
   <xsd:sequence>
      <xsd:element ref="accessibility"/>
      <xsd:element ref="static" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="final" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="transient" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="volatile" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>
```

Het combineren van de verschillende resultaten levert ons dan een volledig omgezet schema op.

3.6.3. Bedenkingen en aanpassingen aan XML Schema

In het vorige schema valt het op dat er heel wat elementen rechtstreeks onder het **xsd:schema** element gedeclareerd zijn. Dit wil zeggen dat in een XML document, conform met dit schema, deze elementen als hoofdelement kunnen en mogen voorkomen. Overeenkomstig de DTD is dit natuurlijk juist, maar eigenlijk willen we kunnen aangeven dat alleen het element **returnType** als hoofdelement kan en mag voorkomen. We kunnen een aantal van deze globale declaraties lokaal maken door op te merken dat er slechts één referentie is aan een aantal globaal gedeclareerde elementen. Door nu deze globale declaraties lokaal te maken is al een deel van ons probleem opgelost. We illustreren dit met een klein voorbeeldje.

Voorbeeld 3.41.

```
<xsd:element name="documentationBlock" type="documentationBlockType"/>
<xsd:element name="descriptionBlock" type="xsd:string"/>
<xsd:element name="tagBlock" type="tagBlockType"/>
<xsd:element name="tag" type="xsd:string"/>
<xsd:element name="informal" type="xsd:string"/>
<xsd:element name="formal" type="formalType"/>
<xsd:element name="formalSentence" type="xsd:string"/>
<xsd:complexType name="documentationBlockType">
   <xsd:sequence>
      <xsd:element ref="descriptionBlock" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="tagBlock" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="tagBlockType">
   <xsd:sequence>
      <xsd:element ref="tag"/>
      <xsd:element ref="informal" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="formal" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="formalType">
   <xsd:sequence>
      <xsd:element ref="formalSentence" minOccurs="1"
      maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

We hebben hier een aantal elementen, zoals **descriptionBlock**, **tagBlock**, waarnaar éénmaal verwezen wordt. Om deze globale declaraties lokaal te maken, kunnen we de lokale referenties naar de globale declaraties (gebruik van het **ref** attribuut) vervangen door lokale declaraties en de globale declaraties verwijderen. Toegepast op het vorige voorbeeld levert ons dit het volgende resultaat op:

Voorbeeld 3.42.

```
<xsd:element name="documentationBlock" type="documentationBlockType"/>
<xsd:complexType name="documentationBlockType">
   <xsd:sequence>
      <xsd:element name="descriptionBlock" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
      <xsd:element name="tagBlock" type="tagBlockType" minOccurs="0"
      maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="tagBlockType">
   <xsd:sequence>
      <xsd:element name="tag" type="xsd:string"/>
      <xsd:element name="informal" type="xsd:string" minOccurs="0"
      maxOccurs="1"/>
      <xsd:element name="formal" type="formalType" minOccurs="0"
      maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="formalType">
   <xsd:sequence>
      <xsd:element name="formalSentence" type="xsd:string" minOccurs="1"
      maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

Op deze manier wordt het aantal globale declaraties van elementen al serieus verminderd, zodat we onder het **xsd:schema** element uiteindelijk nog maar enkele elementdeclaraties overhouden.

We hebben dan nog een aantal elementdefinities waar meerdere malen naar verwezen wordt. Neem bijvoorbeeld het element **returnTypeName** waar vijfmaal naar verwezen wordt. We kunnen dezelfde methode toepassen die we eerder gebruikt hebben bij elementen waar éénmaal aan gerefereerd werd. Op het gebied van de typedefinitie levert dit geen probleem op. Ook brengt dit het voordeel met zich mee dat de elementen lokaal een andere naam kunnen hebben, maar toch gelijk gestructureerd moeten zijn omdat ze naar hetzelfde type verwijzen. Indien je wil dat alle elementen van hetzelfde type dezelfde naam hebben, kan je dit ook als een nadeel beschouwen. In onze ogen is het juist een voordeel dat je de elementen lokaal een andere naam kan geven, maar dat ze toch van hetzelfde type kunnen zijn. Wij hebben ervoor gekozen om met lokale definities te werken, enerzijds omdat we dan de keuzevrijheid van de elementnaam behouden en anderzijds omdat we geen andere mogelijkheid vonden om af te dwingen welk element het enige mogelijke root element mocht zijn.

Naar het element **name** vinden we een viertal verwijzingen terug. De keuze om deze referenties om te zetten naar lokale declaraties is vanzelfsprekender. De reden daarvoor is dat we het type van dit element nog gaan verstrengen en dit niet **xsd:string** zal blijven en we het hier toch over verschillende klassen van naamgeving hebben, met name **returnTypeName**, **method**, **parameter** en **variable**. We gaan beperkingen opleggen waaraan de inhoud van het **name** element moet voldoen indien de inhoud een geldige naam wil voorstellen.

Bekijken we dan alle elementen die we een type **xsd:string** hebben gegeven, dan vinden we dat we alleen nog een verstrenging kunnen opleggen aan het type van het element **package**. De inhoud van dit element moet namelijk een geldige Java-pakketnaam weerspiegelen. Het verstrengen van het type is een mooi voorbeeld van het gebruik van anonieme types en van het **xsd:restriction** element. Indien we de vorige opmerkingen bekijken en de voorgestelde oplossingen toepassen op ons volledige schema geeft ons dit het volgende resulterende

schema.

Voorbeeld 3.43.

```
<?xml version="1.0"?>
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="returnType" type="returnTypeType"/>
    <xsd:simpleType name="identifierType">
       <xsd:restriction base="xsd:string">
          <xsd:pattern value="[A-Za-z_$][a-zA-Z0-9_$]*"/>
       </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="returnTypeType">
       <xsd:sequence>
          <xsd:element name="returnTypeName" type="returnTypeNameType"/>
          <xsd:element name="methodsReturningThisType"
          type="methodsReturningThisTypeType"/>
          <xsd:element name="variablesOfThisType"
          type="variablesOfThisTypeType"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="returnTypeNameType">
       <xsd:sequence>
          <xsd:element name="package" type="identifierType" minOccurs="0"
          maxOccurs="unbounded"/>
          <xsd:element name="name" type="identifierType"/>
          <xsd:choice minOccurs="0" maxOccurs="1">
             <xsd:element name="possibilities" type="possibilitiesType"/>
             <xsd:element name="isResolved" type="emptyType"/>
          </xsd:choice>
          <xsd:element name="bracket" type="emptyType" minOccurs="0"
          maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="emptyType"/>
    <xsd:complexType name="possibilitiesType">
       <xsd:sequence>
          <xsd:element name="returnTypeName" type="returnTypeNameType"
          minOccurs="1" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="methodsReturningThisTypeType">
       <xsd:sequence>
          <xsd:element name="method" type="methodType" minOccurs="0"
          maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="methodType">
       <xsd:sequence>
          <xsd:element name="documentationBlock"
          type="documentationBlockType" minOccurs="0" maxOccurs="1"/>
          <xsd:element name="modifierBlockMethod"
          type="modifierBlockMethodType"/>
          <xsd:element name="type" type="typeType"/>
          <xsd:element name="name" type="identifierType"/>
          <xsd:element name="parent" type="parentType"/>
          <xsd:element name="formalParameters"
          type="formalParametersType"/>
          <xsd:element name="throwsClause" type="throwsClauseType"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="documentationBlockType">
       <xsd:sequence>
          <xsd:element name="descriptionBlock" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
          <xsd:element name="tagBlock" type="tagBlockType" minOccurs="0"
          maxOccurs="unbounded"/>
```

```
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="tagBlockType">
      <xsd:sequence>
        <xsd:element name="tag" type="xsd:string"/>
        <xsd:element name="informal" type="xsd:string" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="formal" type="formalType" minOccurs="0"
        maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="formalType">
      <xsd:sequence>
        <xsd:element name="formalSentence" type="xsd:string"
        minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="modifierBlockMethodType">
      <xsd:sequence>
        <xsd:element name="accessibility" type="accessibilityType"/>
        <xsd:element name="abstract" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="static" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="final" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="synchronized" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="native" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="strictfp" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="accessibilityType">
      <xsd:attribute name="value" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="public"/>
            <xsd:enumeration value="private"/>
            <xsd:enumeration value="protected"/>
            <xsd:enumeration value="package"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="typeType">
      <xsd:sequence>
        <xsd:element name="returnTypeName" type="returnTypeNameType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="parentType">
      <xsd:sequence>
        <xsd:element name="returnTypeName" type="returnTypeNameType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="formalParametersType">
      <xsd:sequence>
        <xsd:element name="parameter" type="parameterType" minOccurs="0"
        maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="parameterType">
      <xsd:sequence>
        <xsd:element name="final" type="emptyType"/>
        <xsd:element name="type" type="typeType"/>
        <xsd:element name="name" type="identifierType"/>
      </xsd:sequence>
    </xsd:complexType>
```

```
    <xsd:complexType name="throwsClauseType">
      <xsd:sequence>
        <xsd:element name="exception" type="exceptionType" minOccurs="0"
        maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="exceptionType">
      <xsd:sequence>
        <xsd:element name="returnTypeName" type="returnTypeNameType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="variablesOfThisTypeType">
      <xsd:sequence>
        <xsd:element name="variable" type="variableType" minOccurs="0"
        maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="variableType">
      <xsd:sequence>
        <xsd:element name="documentationBlock"
        type="documentationBlockType" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="modifierBlockVariable"
        type="modifierBlockVariableType"/>
        <xsd:element name="type" type="typeType"/>
        <xsd:element name="name" type="identifierType"/>
        <xsd:element name="parent" type="parentType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="modifierBlockVariableType">
      <xsd:sequence>
        <xsd:element name="accessibility" type="accessibilityType"/>
        <xsd:element name="static" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="final" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="transient" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
        <xsd:element name="volatile" type="emptyType" minOccurs="0"
        maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
```

3.6.4. Automatische omzetting

Om een voorbeeld te geven van hoe een programma een DTD kan omzetten naar een XML Schema document hebben we dtd2xs [DTD2XS] gebruikt. Het is een gratis programma en we geven hier de uitvoer van dit programma bij de omzetting van onze voorbeeld DTD.

Voorbeeld 3.44.

```
<?xml version="1.0"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="returnType">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="returnTypeName"/>
          <xs:element ref="methodsReturningThisType"/>
          <xs:element ref="variablesOfThisType"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

```
<xs:element name="returnTypeName">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" maxOccurs="unbounded"
         ref="package"/>
         <xs:element ref="name"/>
         <xs:choice minOccurs="0">
            <xs:element ref="isResolved"/>
            <xs:element ref="possibilities"/>
         </xs:choice>
         <xs:element minOccurs="0" maxOccurs="unbounded"
         ref="bracket"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="isResolved">
   <xs:complexType/>
</xs:element>
<xs:element name="package" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="possibilities">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" maxOccurs="unbounded"
         ref="returnTypeName"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="bracket">
   <xs:complexType/>
</xs:element>
<xs:element name="methodsReturningThisType">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" maxOccurs="unbounded"
         ref="method"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="method">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" ref="documentationBlock"/>
         <xs:element ref="modifierBlockMethod"/>
         <xs:element ref="type"/>
         <xs:element ref="name"/>
         <xs:element ref="parent"/>
         <xs:element ref="formalParameters"/>
         <xs:element ref="throwsClause"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="documentationBlock">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" ref="descriptionBlock"/>
         <xs:element minOccurs="0" maxOccurs="unbounded"
         ref="tagBlock"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="descriptionBlock" type="xs:string"/>
<xs:element name="tagBlock">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="tag"/>
         <xs:element minOccurs="0" ref="informal"/>
         <xs:element minOccurs="0" ref="formal"/>
      </xs:sequence>
```

```
      </xs:complexType>
   </xs:element>
   <xs:element name="tag" type="xs:string"/>
   <xs:element name="informal" type="xs:string"/>
   <xs:element name="formal">
      <xs:complexType>
         <xs:sequence>
            <xs:element maxOccurs="unbounded" ref="formalSentence"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="formalSentence" type="xs:string"/>
   <xs:element name="modifierBlockMethod">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="accessibility"/>
            <xs:element minOccurs="0" ref="abstract"/>
            <xs:element minOccurs="0" ref="static"/>
            <xs:element minOccurs="0" ref="final"/>
            <xs:element minOccurs="0" ref="synchronized"/>
            <xs:element minOccurs="0" ref="native"/>
            <xs:element minOccurs="0" ref="strictfp"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="accessibility">
      <xs:complexType>
         <xs:attribute name="value" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="public"/>
                  <xs:enumeration value="private"/>
                  <xs:enumeration value="protected"/>
                  <xs:enumeration value="package"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <xs:element name="abstract">
      <xs:complexType/>
   </xs:element>
   <xs:element name="static">
      <xs:complexType/>
   </xs:element>
   <xs:element name="final">
      <xs:complexType/>
   </xs:element>
   <xs:element name="synchronized">
      <xs:complexType/>
   </xs:element>
   <xs:element name="native">
      <xs:complexType/>
   </xs:element>
   <xs:element name="strictfp">
      <xs:complexType/>
   </xs:element>
   <xs:element name="type">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="returnTypeName"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="parent">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="returnTypeName"/>
         </xs:sequence>
```

```
      </xs:complexType>
   </xs:element>
   <xs:element name="formalParameters">
      <xs:complexType>
         <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded"
            ref="parameter"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="parameter">
      <xs:complexType>
         <xs:sequence>
            <xs:element minOccurs="0" ref="final"/>
            <xs:element ref="type"/>
            <xs:element ref="name"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="throwsClause">
      <xs:complexType>
         <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded"
            ref="exception"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="exception">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="returnTypeName"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="variablesOfThisType">
      <xs:complexType>
         <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded"
            ref="variable"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="variable">
      <xs:complexType>
         <xs:sequence>
            <xs:element minOccurs="0" ref="documentationBlock"/>
            <xs:element ref="modifierBlockVariable"/>
            <xs:element ref="type"/>
            <xs:element ref="name"/>
            <xs:element ref="parent"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="modifierBlockVariable">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="accessibility"/>
            <xs:element minOccurs="0" ref="static"/>
            <xs:element minOccurs="0" ref="final"/>
            <xs:element minOccurs="0" ref="transient"/>
            <xs:element minOccurs="0" ref="volatile"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="transient">
      <xs:complexType/>
   </xs:element>
   <xs:element name="volatile">
      <xs:complexType/>
```

```
    </xs:element>
</xs:schema>
```

Het programma dtd2xs levert ons een correcte letterlijke vertaling van het schema op, maar het resultaat verschilt toch veel van het schema dat wij hebben opgebouwd. In dit schema zien we bijvoorbeeld alleen globale elementdefinities, maar geen enkele globale, benoemde typedefinitie. Alle typedefinities in dit schema zijn anoniem. Dit heeft het nadeel dat bepaalde definities van types nodeloos herhaald worden. In ons voorbeeld hebben we te maken met tien lege elementen en bij elk van deze elementdefinities wordt de definitie van hun type herhaald. Dit kan eleganter opgelost worden door deze definitie éénmaal op het hoogste niveau te maken en er dan bij de elementdefinities naar te verwijzen.

Zoals reeds opgemerkt hebben we in dit schema alleen globale elementdefinities. Dit wil zeggen dat deze elementen in een XML document, conform dit schema, kunnen voorkomen als hoofdelement van het document. Op dat vlak is dit schema evenwaardig met de DTD, maar de mogelijkheid is geboden in XML Schema om dit te verstrengen. Maar het is ook niet meer dan normaal dat een programma deze beperkingen niet automatisch kan opleggen. Er is immers geen invoer geweest aan het programma om dit duidelijk te maken. Wij hebben wel de kennis opgedaan over deze documenten en daarom wisten wij dat we het schema op die manier mochten verstrengen.

Als laatste punt willen we aanhalen dat in onze ogen, het door ons opgebouwd schema leesbaarder en dus ook iets begrijpelijker is dan het automatisch gegenereerde schema. Wel kan het automatisch gegenereerde schema als een goede basis dienen om nog door iemand bewerkt te worden en de eisen opgelegd aan de XML documenten te verstrengen. Op die manier kan de kracht van XML Schema ten volle benut worden. We willen besluiten met een laatste bedenking over programma's die deze omzettingen automatisch kunnen doen. Deze programma's leveren goed werk, maar ze zullen nooit het niveau kunnen halen van een handmatige omzetting. Een eerste reden is dat het hier gaat om een omzetting naar een krachtigere en expressievere taal en als tweede reden voor deze bedenking halen wij het feit aan dat datatypering niet mogelijk is in een DTD, maar dat daarvoor wel uitgebreide mogelijkheden zijn voorzien in XML Schema.

3.7. Vergelijking tussen DTD en XML Schema

Het is duidelijk dat er een aantal verschillen zijn tussen de DTD's en XML Schema. We zullen hier kort spreken over de verschillen betreffende datatypering, beperkingen op aantal voorkomens en het gebruik van opsommingen. Buiten deze drie verschilpunten zijn er nog een aantal verschillen die we hier niet gaan bespreken.

Wat de typering betreft van de inhoud van de elementen en van de attributen, heb je in een DTD eigenlijk alleen de keus uit respectievelijk PCDATA en CDATA. XML Schema daarentegen voorziet in een lange lijst van aanwezige datatypes waaronder string, int, float, unsignedLong, uriReference, nonNegativeInteger om er maar enkele te noemen. XML Schema voorziet ook in de mogelijkheid om je eigen types te definiëren.

Zowel in XML Schema als in een DTD is het mogelijk om de beperkingen op het aantal voorkomens op te geven. Alleen is dit in XML Schema iets korter te doen, dan in DTD's. Wil je bijvoorbeeld in XML Schema opgeven dat een element minimum vijf en maximum negen keer mag voorkomen, dan gaat dit heel gemakkelijk door het gebruik van de attributen minOccurs en maxOccurs. In een DTD moet je dan negen keer dat element opgeven waarbij

je vier keer door middel van een vraagteken moet aangeven dat die instanties van dat element optioneel zijn.

Zowel DTD's als XML Schema voorzien in de mogelijkheid om de mogelijke waarden van een attribuut aan te geven met behulp van een opsomming. Maar in XML Schema kan die opsomming ook gebruikt worden om beperkingen op te leggen aan de inhoud van een element, een mogelijkheid die niet bestaat in DTD's.

Alhoewel het door deze drie punten lijkt alsof XML Schema veel beter is dan SGML DTD's, heeft XML Schema ook nog enkele beperkingen. Zo is het bijvoorbeeld niet mogelijk om entiteiten te definiëren. Voor de validatie van documenten zijn er nog geen heel volwassen programma's voor XML Schema, terwijl die er natuurlijk voor DTD's al veel langer zijn. Het voordeel van XML Schema is wel dat een XML syntax gebruikt wordt, terwijl de DTD's een heel eigen syntax gebruiken.

3.8. Conclusie

XML Schema biedt de mogelijkheid om met behulp van XML een XML document te beschrijven. Het doet dit met behulp van een niet al te moeilijke syntax, maar door de uitgebreidheid is het niet makkelijk om de eerste stappen in die richting te zetten. Maar eenmaal vertrouwd met XML Schema ontdek je nieuwe dingen in de standaarden die juist hetgene doen wat je wou doen. Toch heeft XML Schema nog enkele beperkingen, waaronder de onmogelijkheid om entiteiten te definiëren, en daarvoor moeten nog altijd DTD's gebruikt worden (of de entiteiten moeten in het XML document zelf gedeclareerd worden). Vooraleer XML Schema wijdverspreid gaat worden zullen er echter programma's moeten komen die de hele standaard ondersteunen en die kunnen instaan voor de validatie van de documenten. Gelukkig beginnen deze programma's, zij het met mondjesmaat, toch beschikbaar te worden. Om de omschakeling van DTD's naar XML Schema te kunnen maken is het ook belangrijk dat er programma's zijn die deze omzetting automatisch kunnen doen. Deze programma's zijn er, en alhoewel ze nog geen perfecte XML Schema's afleveren, leveren ze toch een goede basis om mee te beginnen. XML Schema is een veelbelovende standaard, maar er moet wel een uitgebreide ondersteuning komen vooraleer vele bedrijven de overstap gaan durven wagen van DTD's naar XML Schema.

4. XML Schema van onze tekst

Onze eigen tekst is volledig in XML geschreven. We hebben wel andere formaten bestudeerd, maar hebben uiteindelijk toch besloten om een eigen XML formaat te definiëren [Appendix A.] . Voor ons eigen XML formaat hebben we ook een XML Schema ontwikkeld. Het is dit schema dat we in dit hoofdstuk gaan bespreken. De bespreking zal geen uitleg inhouden over XML Schema [Hoofdstuk 3.] , maar zal uitleggen waarom we bepaalde keuzes gemaakt hebben.

We gaan het schema bespreken door telkens eerst een stukje van het schema te geven en dan de keuzes te bespreken en uit te leggen waarom we precies daarvoor gekozen hebben.

4.1. Keuze voor XML

Naast XML hebben we nog andere formaten bekeken om onze tekst in te schrijven. We zochten een formaat waarin er een duidelijke scheiding is tussen (de structuur van) de tekst en de lay-out. Het gebruik van klassieke tekstverwerkers hebben we bewust niet overwogen omdat de lay-out nogal eens wil verschillen wanneer deze tekstverwerkers gebruikt worden op verschillende configuraties. Een tekst die je op die manier maakt, bevat veel lay-out instructies, maar hiermee beschrijf je niet de structuur van je tekst. De scheiding tussen lay-out en tekst was hier volgens ons niet aanwezig. Zoals in [Sectie 2.1.] beschreven is, is het juist het doel van XML om de structuur te beschrijven. De afwegingen die we gemaakt hebben, zijn te vinden in [Appendix sectie A.3.] . XML voldeed duidelijk aan onze noden.

Uiteindelijk hebben we besloten om ons eigen XML formaat te definiëren. Enerzijds omdat onze thesis daarover ging en we dus al doende ervaringen konden opdoen. Anderzijds moesten we ons niet aan een vooropgelegde structuur houden, maar konden we onze eigen structuur kiezen. We waren er ons wel ten volle van bewust dat we niet zomaar iets in elkaar konden flansen, maar dat we op de proppen moesten komen met een weldoordacht formaat dat makkelijk uitbreidbaar zou zijn. Omdat we op dat moment nog geen notie hadden van XML Schema [Hoofdstuk 3.] beschreven we de structuur van onze tekst in een simpel tekstbestandje waar de indentatie de relatie aangaf tussen de verschillende elementen. Ondertussen is dit geëvolueerd naar een volwaardig XML Schema. Het is dit schema dat we in dit hoofdstuk gaan bespreken. Zoals eerder vermeld gaan we hier eerder onze keuzes uitleggen dan nog eens uit te leggen wat de XML Schema elementen betekenen.

Misschien is het ook belangrijk om te vermelden dat alleen de algemene structuur van de tekst (hoofdstukken, secties en paragrafen) op voorhand is vastgelegd. De meeste elementen zijn pas gedefinieerd op het moment dat we tot de vaststelling kwamen dat we ze nodig hadden. Ons formaat is dus in principe heel dynamisch, maar het onderhouden van het schema van onze tekst was daardoor niet altijd even gemakkelijk. Dit komt doordat nieuw ingevoerde elementen soms op meerdere plaatsen moesten kunnen voorkomen en de beschrijving van deze nieuwe elementen kort na hun introductie in ons formaat nogal eens van beschrijving durfden veranderen.

4.2. Het schema van onze tekst

Voor de duidelijkheid bespreken we het ontwikkelde schema in kleinere stukken. Het volledige schema wordt gegeven in [Appendix sectie A.4.] .

4.2.1. Enkele algemene definities

Voorbeeld 4.1. Algemene definities

```
<xs:attribute name="accessibility">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:enumeration value="public"/>
         <xs:enumeration value="private"/>
         <xs:enumeration value="protected"/>
      </xs:restriction>
   </xs:simpleType>
</xs:attribute>
<xs:attribute name="occurs">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:enumeration value="many"/>
         <xs:enumeration value="more"/>
         <xs:enumeration value="optional"/>
      </xs:restriction>
   </xs:simpleType>
</xs:attribute>
<xs:attribute name="id">
   <xs:simpleType>
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
</xs:attribute>
<xs:group name="TitleGroup">
   <xs:choice>
      <xs:element name="INDEX" type="xs:string"/>
      <xs:element name="EMPH" type="xs:string"/>
   </xs:choice>
</xs:group>
<xs:complexType name="EmptyType"/>
<xsd:simpleType name="identifierType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-zA-Z_$][a-zA-Z0-9_$]*"/>
   </xsd:restriction>
</xsd:simpleType>
<xs:simpleType name="composedIdentifierType">
   <xs:restriction base="xs:string">
      <xs:pattern
      value="[a-zA-Z_$][a-zA-Z0-9_$]*(\.[a-zA-Z_$][a-zA-Z0-9_$])*"/>
   </xs:restriction>
</xs:simpleType>
```

We definiëren hier drie attributen, drie types en één groep. Het attribuut **accessibility** gebruiken we om bij een aantal elementen, waarmee we een gedeelte van Java proberen te beschrijven in XML [Sectie 4.2.6.] , de toegankelijkheid uit te drukken.

Het **occurs** attribuut wordt gebruikt om bij de beschrijving van DTD's in ons XML formaat [Sectie 4.2.5.] aan te geven hoe vaak een element in een bepaald inhoudsmodel mag voorkomen. **many** komt overeen met *****, **more** komt overeen met **+** en **optional** komt overeen met **?**. De keuze van optional als waarde lijkt logisch, **more** hebben we gekozen als korte notatie voor **one or more occurrences** en **many** staat voor **zero, one or many occurrences**.

Het attribuut **id** gebruiken we om een unieke, zelfgedefinieerde identifier te associëren met bepaalde elementen. Op die manier kunnen we intern referenties leggen tussen de verschillende delen van onze tekst. Voor meer informatie over interne referenties verwijzen we naar .

We definiëren ook een groep **TitleGroup** die we gebruiken bij de typedefinities van de verschillende types voor titels.

Tenslotte hebben we nog het type **EmptyType** dat het type is om lege elementen voor te stellen en het type **identifierType** dat een geldige identifier in Java voorstelt. Het type **composedIdentifierType** stelt een samenstelling van Java identifiers voor. De samenstellende onderdelen worden gescheiden door een punt (**.**). We hebben geprobeerd om een bepaalde relatie te definiëren tussen de types **composedIdentifierType** en **identifierType**, maar bij het controleren van het schema met behulp van SQC [IBMSQC] resulteerde dit in foutmeldingen.

4.2.2. Algemene structuur

Het volgende fragment van ons schema dient om de algemene structuur van onze tekst te beschrijven:

Voorbeeld 4.2. Algemene structuur

```
<xs:element name="THESIS" type="ThesisType"/>
<xs:complexType name="ThesisType">
   <xs:sequence>
      <xs:element name="TITLE" minOccurs="1" maxOccurs="2"
      type="TitleLangType"/>
      <xs:element name="AUTHOR" minOccurs="1" maxOccurs="unbounded"
      type="xs:string"/>
      <xs:element name="PROMOTOR" minOccurs="1" maxOccurs="unbounded"
      type="xs:string"/>
      <xs:element name="COACH" minOccurs="1" maxOccurs="unbounded"
      type="xs:string"/>
      <xs:element name="READER" minOccurs="1" maxOccurs="unbounded"
      type="xs:string"/>
      <xs:element name="ABSTRACT" minOccurs="1" maxOccurs="1"
      type="SummaryType"/>
      <xs:element name="ACKNOWLEDGEMENT" minOccurs="1" maxOccurs="1"
      type="SummaryType"/>
      <xs:element name="CHAPTER" minOccurs="1" maxOccurs="unbounded"
      type="ChapterType"/>
      <xs:element name="APPENDIX" minOccurs="0" maxOccurs="unbounded"
      type="ChapterType"/>
      <xs:element name="READINGS" minOccurs="0" type="ReadingsType"/>
      <xs:element name="BIBLIOGRAPHY" type="BibliographyType"/>
   </xs:sequence>
</xs:complexType>
```

Het root element van onze tekst is **THESIS**. Onze thesis heeft in dit geval minstens één **TITLE** en maximum twee titels. We hebben dit zo gekozen om het mogelijk te maken dat er een titel is in de taal van de thesis en een titel in een andere taal. Dit is het enige element waar dit voorzien is. De typedefinitie van **TitleLangType** ziet er als volgt uit:

Voorbeeld 4.3. TitleLangType

```
<xs:complexType name="TitleLangType" mixed="true">
   <xs:sequence>
      <xs:group ref="TitleGroup" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="language" type="xs:language" use="optional"/>
</xs:complexType>
```

Het gaat hier om een element met gemengde inhoud [Sectie 3.5.9.] . De elementen **EMPH** en **INDEX**, die we later nog zullen bespreken, kunnen als kindelement voorkomen. Dit volgt uit

het refereren naar de groep **TitleGroup**. Ook is er een optioneel attribuut **language** om de gebruikte taal aan te geven, dit volgens de gedefinieerde landcodes [TRXS2-333].

Wat we in dit geval echt hadden willen doen was het eisen van één TITLE element zonder attribuut. We wilden wel voorzien dat er nog een ander TITLE element mocht volgen, maar met een verplicht attribuut language. We hebben geen manier gevonden om dit af te dwingen. Twee elementen met dezelfde naam in één typedefinitie mag niet [Sectie 3.6.2.1.] . We vonden het ook niet mooi om het tweede TITLE element een andere naam te geven, vandaar dat we tot dit compromis gekomen zijn. Regels om af te dwingen dat, indien er meerdere voorkomens zijn van een element met dezelfde naam, één element geen attribuut mag hebben maar de andere element een bepaald attribuut moeten hebben, zijn er blijkbaar niet.

We hebben daarnaast elementen voorzien voor de auteurs (**AUTHOR**), de promotor (**PROMOTOR**), de begeleiders (**COACH**) en de lezers (**READER**). Deze elementen zijn van het type **xs:string** omdat we het niet nodig vonden dit verder op te splitsen in bijvoorbeeld een achternaam, een voornaam en een titel. We hebben deze keuze gemaakt omdat dit de meest gangbare praktijk is en we deze elementen niet nodeloos wilden compliceren. We vonden dat, indien we deze opsplitsing zouden maken, we ook rekening moesten houden met initialen, meerdere voornamen, meerdere achternamen (e.g. zoals de praktijk is in Spanje), geen achternaam, enzovoort... Het is niet het doel van onze thesis om dit te bestuderen en daarom hebben we voor de eenvoudigste oplossing gekozen.

Vervolgens hebben we elementen voorzien voor een abstract (**ABSTRACT**) en voor een dankwoord (**ACKNOWLEDGEMENT**). Deze twee elementen zijn van het type **SummaryType**, waar we in [Sectie 4.2.3.] verder op terug komen.

De hoofdmoot van onze thesis is vervat in de verschillende hoofdstukken (**CHAPTER**) en appendices (**APPENDIX**). Over het type **ChapterType** zullen we het hebben in [Sectie 4.2.3.] .

Daarnaast hebben we ook nog een litteratuurstudie gedaan (**READINGS**). Hoe deze litteratuurstudie beschreven wordt, behandelen we in [Sectie 4.2.7.] .

Tenslotte zis er nog de obligate bibliografie (**BIBLIOGRAPHY**), die we zullen behandelen in [Sectie 4.2.8.] .

4.2.3. Algemene structuur van een hoofdstuk

De algemene vorm voor een hoofdstuk hebben we als volgt bepaald:

Voorbeeld 4.4. Algemene vorm hoofdstuk

```
<xs:complexType name="ChapterType">
   <xs:sequence>
      <xs:element name="TITLE" minOccurs="1" maxOccurs="1"
      type="TitleType"/>
      <xs:element name="SUMMARY" minOccurs="0" type="SummaryType"/>
      <xs:element name="SECTION" maxOccurs="unbounded"
      type="SectionType"/>
   </xs:sequence>
   <xs:attribute ref="id" use="optional"/>
</xs:complexType>
<xs:complexType name="TitleType" mixed="true">
   <xs:sequence>
      <xs:group ref="TitleGroup" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
```

```
</xs:complexType>
```

Een hoofdstuk heeft logischerwijze één titel, een beknopte beschrijving van dat hoofdstuk (**SUMMARY**) en een aantal secties (**SECTION**). Elk element van het type **ChapterType** kan daarnaast ook nog het attribuut **id** bevatten, waarvan de rol in [Sectie 4.2.1.] en besproken wordt.

De titel van een hoofdstuk heeft hier het type **TitleType** dat alleen verschilt van het type **TitleLangType** door het ontbreken van het attribuut **language**. Vandaar dat we ook naar dezelfde groep verwijzen. Op die manier moeten we de definities in die groep niet herhalen. De definitie van het type **TitleType** verhindert het voorkomen van lege titel niet. Het **minOccurs** attribuut bij de referentie naar de gedefinieerde groep wijst er alleen op dat de elementen die in die groep gedefinieerd zijn niet moeten voorkomen. Met behulp van het **mixed** attribuut geven we aan dat elementen van dit type gemengde inhoud kunnen bevatten. We hebben echter geen manier gevonden om dan af te dwingen dat een titel een bepaalde lengte moet hebben. Indien we geen kindelementen zouden toelaten in dit type, dan kunnen we dit type definiëren met behulp van het **xs:simpleType** element als een **restriction** op het basistype **xs:string**. In dat geval kunnen we wel een minimale en een maximale lengte opgeven. Maar we moesten hier afwegen welke voorzieningen we wilden treffen. Met het oog op een mogelijk index hebben we gekozen voor de kindelementen. Dit is de enige reden voor deze keuze. Het **EMPH** element is pas na het nemen van deze beslissing toegevoegd.

De korte samenvatting van een hoofdstuk (**SUMMARY**) heeft het type **SummaryType** dat als volgt gedefinieerd is:

Voorbeeld 4.5. SummaryType

```
<xs:complexType name="SummaryType">
   <xs:sequence>
      <xs:element name="PARA" minOccurs="1" maxOccurs="unbounded"
      type="ParaType"/>
   </xs:sequence>
</xs:complexType>
```

Een **SUMMARY** bevat één of meerdere paragrafen (**PARA**), wat ons de meest logische keuze leek. Zoals zo dadelijk uit de definitie van **SectionType** en **ParaType** zal blijken, zit de eigenlijke inhoud vervat in de **PARA** elementen. Door hier gebruik van te maken, wordt de definitie van het type **ParaType** vrij groot, maar dat zal dan ook het enige type zijn dat zo groot is.

Het type voor een sectie, die bij ons een gedeelte van een hoofdstuk voorstelt, wordt als volgt gedefinieerd:

Voorbeeld 4.6. SectionType

```
<xs:complexType name="SectionType">
   <xs:sequence>
      <xs:element name="TITLE" minOccurs="1" maxOccurs="1"
      type="TitleType"/>
      <xs:element name="SUBTITLE" minOccurs="0" type="TitleType"/>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
         <xs:element name="PARA" type="ParaType"/>
         <xs:element name="SECTION" type="SectionType"/>
```

```
      </xs:choice>
    </xs:sequence>
    <xs:attribute ref="id" use="optional"/>
  </xs:complexType>
```

Een element met dit type heeft een verplichte titel, een optionele ondertitel (**SUBTITLE**) die iets meer uitleg kan geven over de titel, en bestaat uit een aantal **PARA** en/of **SECTION** elementen. Ook voorzien we voor dit type weer het attribuut **id**. Deze keuzes lijken ons allemaal vrij logisch.

Het type voor een paragraaf (**PARA**) **ParaType** is het type dat de meeste kindelementen kan hebben. Dit komt omdat dit type als een "container" dient voor de eigenlijke inhoud. We hebben **ParaType** gedefinieerd op deze manier:

Voorbeeld 4.7. ParaType

```
  <xs:complexType name="ParaType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <!--Elementen voor index/referenties-->
      <xs:element name="INDEX" type="xs:string"/>
      <xs:element name="BIBREFERENCE" type="xs:string"/>
      <xs:element name="REFERENCE" type="xs:string"/>
      <!--URI gerelateerde elementen-->
      <xs:element name="SITE" type="SiteType"/>
      <xs:element name="URL" type="xs:anyURI"/>
      <xs:element name="DIRECTORY" type="xs:anyURI"/>
      <!--Lijst gerelateerde elementen-->
      <xs:element name="LIST" type="ListType"/>
      <xs:element name="EXPLANATIONS" type="ExplanationsType"/>
      <!--XML-gerelateerde elementen-->
      <xs:element name="TAG" type="TagType"/>
      <xs:element name="XML" type="StructureType"/>
      <xs:element name="DOCDECLARATION" type="DocDeclarationType"/>
      <xs:element name="PI" type="PiType"/>
      <xs:element name="ELEMENT" type="ElementType"/>
      <xs:element name="ATTRIBUTE" type="AttributeType"/>
      <xs:element name="COMMENT" type="xs:string"/>
      <xs:element name="SGML" type="StructureType"/>
      <xs:element name="HTML" type="StructureType"/>
      <!--DTD gerelateerde elementen-->
      <xs:element name="DTD" type="DTDType"/>
      <xs:element name="PCDATA" type="EmptyType"/>
      <!--Java gerelateerde elementen-->
      <xs:element name="JAVA" type="JavaType"/>
      <xs:element name="METHOD" type="MethodType"/>
      <xs:element name="VARIABLE" type="VariableType"/>
      <xs:element name="ARGUMENT" type="MethodArgumentType"/>
      <!--Image en Quote-->
      <xs:element name="IMAGE" type="ImageType"/>
      <xs:element name="QUOTE" type="QuoteType"/>
      <!--TeX/LaTeX gerelateerde elementen-->
      <xs:element name="TEXDOCUMENT" type="TexDocumentType"/>
      <xs:element name="TEXMARKUP" type="TexMarkupType"/>
      <!--aantal andere elementen-->
      <xs:element name="EMPH" type="xs:string"/>
      <xs:element name="FORMATTED" type="xs:string"/>
      <xs:element name="CONFIGURATION" type="xs:string"/>
      <xs:element name="CDATA" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
```

Elementen van het type **ParaType** kunnen gemengde inhoud bevatten, zoals het **mixed** attribuut aangeeft. Met behulp van commentaar hebben we de verschillende kindelementen in categorieën proberen in te verdelen. We zullen deze categorieën hier in het kort bespreken. Merk op dat we door middel van het **xs:choice** element aangeven dat de volgorde waarin deze elemten voorkomen niet van belang is. Voor de elementen in verband met XML/HTML [Sectie 4.2.4.] , DTD [Sectie 4.2.5.] en Java [Sectie 4.2.6.] voorzien we een uitgebreidere bespreking.

4.2.3.1. Index/referentie elementen

Het element **INDEX** bevat een string en dit element dient om rond een aantal begrippen te zetten zodat deze begrippen in een index kunnen opgenomen worden. Dit element is voorzien, maar wordt nog niet echt gebruikt.

De elementen **REFERENCE** en **BIBREFERENCE** bevatten referenties. Het element REFERENCE bevat referenties naar delen van onze tekst. De inhoud van dit element moet overeenkomen met de waarde van een **id** attribuut. Dit is een conventie die wij hanteren, maar we hebben hiervoor niets gevonden in de W3C XML Schema Language. Het kan goed zijn dat er wel zoiets bestaat, maar de technical recommendations maken het niet eenvoudig om iets terug te vinden. Het element BIBREFERENCE gebruiken we om te verwijzen naar werken in de bibliografie, meer bepaald naar het REFERENCE kindelement van een **BIBITEM** element (zie [Sectie 4.2.8.]).

4.2.3.2. URI gerelateerde elementen

De elementen **URL** en **DIRECTORY** worden gebruikt om locaties aan te geven die met een URI geïdentificeerd kunnen worden. Het element DIRECTORY is het eerst ingevoerd maar geleidelijk aan is dit vervangen door het element URL, omdat URL algemener is. Omdat het element DIRECTORY nog op sommige plaatsen nuttig gebruikt kan worden, bijvoorbeeld om een fysische directory aan te duiden, hebben we het niet overal vervangen door URL. Deze elementen hebben beide het type **xs:anyURI** en ze kunnen dus beide eender welke geldige URI bevatten.

Het element **SITE** met het type **SiteType** wordt niet meer (zoveel) gebruikt in onze tekst. Dit type hebben we eerst ingevoerd om op een algemene wijze referenties naar websites te kunnen gebruiken. Maar nadat we een eigen syntax hadden gedefinieerd voor onze bibliografie zijn we geleidelijk overgeschakeld naar het gebruik van bibliografiereferenties. Dit element wordt nog gebruikt om te verwijzen naar websites die niet in onze bibliografie zijn opgenomen. Het type **SiteType** is als volgt gedefinieerd:

Voorbeeld 4.8. SiteType

```
<xs:complexType name="SiteType">
   <xs:sequence>
      <xs:element name="TITLE" minOccurs="1" maxOccurs="1"
      type="TitleType"/>
      <xs:element name="URL" minOccurs="1" maxOccurs="unbounded"
      type="xs:anyURI"/>
   </xs:sequence>
</xs:complexType>
```

Een website wordt beschreven door middel van één titel en mogelijk meerdere URL's. De website van jnome heeft namelijk meerdere URL's waaronder: *http://www.jnome.org* en

http://www.cs.kuleuven.ac.be/projecten/jnome/.

4.2.3.3. Lijst gerelateerde elementen

Het element **LIST** dient, zoals de naam al aangeeft, om een lijst te beschrijven. Het type van dit element, **ListType** is als volgt gedefinieerd:

Voorbeeld 4.9. ListType

```
<xs:complexType name="ListType" mixed="true">
   <xs:sequence>
      <xs:element name="LISTITEM" minOccurs="1" maxOccurs="unbounded"
      type="ParaType"/>
   </xs:sequence>
</xs:complexType>
```

Een lijst bestaat uit een aantal **LISTITEM** elementen. Deze elementen hebben het type **ParaType**. We hebben dit zo gekozen omdat we zo van het hele arsenaal aan elementen gebruik kunnen maken in deze elementen. Dit is handig om lijsten in lijsten te kunnen gebruiken, ...

Het **EXPLANATIONS** element wordt gebruikt om een aantal definities of begrippen te kunnen opsommen en aan elk begrip of definitie een uitleg te kunnen verbinden, vergelijkbaar met de elementen **dd** en **dt** in HTML. De verschillende typedefinities die hiervoor gebruikt worden zien er als volgt uit:

Voorbeeld 4.10. ExplanationsType en andere definities

```
<xs:complexType name="ExplanationsType">
   <xs:sequence>
      <xs:element name="TITLE" type="TitleType"/>
      <xs:element name="HEADING" type="HeadingType"/>
      <xs:element name="EXPITEM" maxOccurs="unbounded"
      type="ExpItemType"/>
   </xs:sequence>
   <xs:attribute ref="id" use="optional"/>
</xs:complexType>
<xs:complexType name="HeadingType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:element name="EXPLAINS" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="ExpItemType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:element name="EXPLANATION" type="ExplanationType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="ExplanationType" mixed="true">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="REFERENCE" type="xs:string"/>
      <xs:element name="EMPH" type="xs:string"/>
      <xs:element name="BIBREFERENCE" type="xs:string"/>
   </xs:choice>
</xs:complexType>
```

Een element van het type **ExplanationsType** bevat een titel (**TITLE**) die beschrijft waar de

uitleg over gaat. Daarnaast bevat dit ook nog een hoofding (**HEADING**). Deze hoofding geeft een naam aan de begrippen die uitgelegd gaan worden en geeft ook een korte omschrijving voor de uitleg. Door dit in het XML document zelf te laten specificeren, kan dit zeer algemeen gebruikt worden. Daarnaast bevatten elementen van dit type ook nog minstens één **EXPITEM** element. EXPITEM is een afkorting voor **Explained item**. Een EXPITEM element bevat de naam van het begrip dat uitgelegd gaat worden en de uitleg (**EXPLANATION**). Voor de uitleg is een speciaal type voorzien omdat we in de uitleg ook nog het **EMPH** element willen kunnen gebruiken en referenties willen kunnen opnemen. Omdat we naar sommige van deze elementen van het type **ExplanationsType** willen kunnen verwijzen in de tekst, voorzien we ook hier het **id** attribuut.

4.2.3.4. Elementen IMAGE en QUOTE

Het **IMAGE** element gebruiken we om te verwijzen naar afbeeldingen die we willen opnemen in onze tekst. Het type van dit element is als volgt gedefinieerd:

Voorbeeld 4.11. ImageType, ImgMeasurementType

```
<xs:complexType name="ImageType">
   <xs:sequence>
      <xs:element name="LOCATION" type="xs:anyURI"/>
      <xs:element name="COMMENT" minOccurs="0" type="xs:string"/>
      <xs:element name="COPYRIGHT" minOccurs="0" type="xs:string"/>
      <xs:element name="WIDTH" minOccurs="0" type="ImgMeasurementType"/>
      <xs:element name="HEIGHT" minOccurs="0"
      type="ImgMeasurementType"/>
   </xs:sequence>
   <xs:attribute ref="id" use="optional"/>
</xs:complexType>
<xs:simpleType name="ImgMeasurementType">
   <xs:restriction base="xs:string">
      <xs:pattern value="[1-9][0-9]*px"/>
   </xs:restriction>
</xs:simpleType>
```

Een afbeelding heeft een locatie (**LOCATION**), van het type **xs:anyURI**. De locaties die we gebruiken zijn relatief ten opzichte van de locatie van het hoofddocument. Voor het omzetten naar HTML vormt dit geen probleem, omdat we een URI locatie in de sitemap [Sectie 9.2.3.2.3.] naar de juiste locatie kunnen mappen. Voor het genereren van de PDF stelde ons dit voor problemen omdat de component die zorgde voor de PDF generatie aan andere basisdirectory gebruikte en de relatieve locaties niet meer correct waren. We hebben hiervoor een work-around gebruikt die eruit bestond de relatieve locatie in de stylesheet om te zetten in een absolute locatie.

Bij een afbeelding hoort vaak ook nog wat commentaar (**COMMENT**). Omdat we niet alle afbeeldingen zelf gemaakt hebben, is het hier en daar ook nodig copyright informatie (**COPYRIGHT**) op te nemen. Omdat het renderen van de afbeeldingen in de PDF soms nogal verkeerde uitvoer oplevert (uitgerokken prentjes) vonden we het ook nodig dat we de mogelijkheid schiepen om de breedte (**WIDTH**) en de hoogte (**HEIGHT**) te kunnen opgeven. De beperking die we aan de inhoud van deze laatste twee elementen opleggen is dat het om een getal moet gaan gevolgd door de letters **px**. We vonden dat we deze beperking het makkelijkst konden opleggen door middel van een reguliere expressie. Er was ook de mogelijkheid om de afmetingen af te leiden van bijvoorbeeld het basistype **xs:integer**, maar we vonden dan nergens de mogelijkheid om op te leggen dat het opgegeven getal gevolgd moest worden door de letters **px**. Het enige spijtige hieraan is dat we toch in bepaalde mate lay-out informatie gaan opnemen in ons XML bestand, wat niet de bedoeling was. Gelukkig

blijft dit al bij al nog vrij beperkt omdat het nog ook mogelijk is deze opgegeven waarden gewoon te negeren.

Het element **QUOTE** dient om citaten aan te duiden in onze tekst. Dit element voldoet aan de volgende typedefinitie:

Voorbeeld 4.12. QuoteType

```
<xs:complexType name="QuoteType" mixed="true">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="PARA" type="ParaType"/>
      <xs:element name="EMPH" type="xs:string"/>
   </xs:choice>
</xs:complexType>
```

Naast gewone tekens kan dit element ook het **EMPH** element bevatten om nadruk te leggen op bepaalde delen in het citaat. Ook is het mogelijk om **PARA** elementen op te nemen als kindelementen. Dit gebruiken we voor grote citaten, die uit één of meerdere paragrafen bestaan.

4.2.3.5. TeX/LaTeX gerelateerde elementen

De elementen **TEXDOCUMENT** en **TEXMARKUP** gebruiken we om bepaalde aspecten van TeX/LaTeX te beschrijven in ons XML formaat. Dit hebben we gebruikt in [Appendix sectie A.2.] . De typedefinities van deze elementen zien er als volgt uit:

Voorbeeld 4.13. TeX/LaTeX typedefinities

```
<xs:complexType name="TexDocumentType" mixed="true">
   <xs:sequence>
      <xs:element name="TEXMARKUP" type="TexMarkupType"
      maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="TexMarkupType">
   <xs:choice>
      <xs:element name="MACRO" type="xs:string"/>
      <xs:sequence>
         <xs:element name="NAME" minOccurs="0" type="xs:string"/>
         <xs:element name="ARGUMENTS" minOccurs="0"
         type="TexArgumentsType"/>
         <xs:element name="TEXCONTENT" type="xs:string"/>
      </xs:sequence>
   </xs:choice>
</xs:complexType>
<xs:complexType name="TexArgumentsType">
   <xs:sequence>
      <xs:element name="ARGUMENT" maxOccurs="unbounded"
      type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Een TeXdocument kan gemengde inhoud bevatten. Naast gewone tekens kan een TeXdocument ook nog een aantal, maar minstens één, **TEXMARKUP** elementen van het type **TexMarkupType** bevatten. Een element van het type TeXMarkupType bevat ofwel een macro-oproep (**MACRO**) ofwel bestaat dat uit een naam (**NAME**), nul of meerdere argumenten (**ARGUMENTS**, waar per argument één **ARGUMENT** kindelement gebruikt

moet worden) en tenslotte nog wat inhoud (**TEXCONTENT**).

Opmerking: het was niet de bedoeling om een volledige omschrijving van TeX/LaTeX in XML mogelijk te maken. We hebben ons beperkt tot wat we nodig hadden.

4.2.3.6. Andere elementen

Onder andere elementen hebben we de elementen **EMPH**, **FORMATTED**, **CONFIGURATION** en **CDATA** gegroepeerd. Het element EMPH gebruiken we om de nadruk (emphasis) te leggen op bepaalde gedeelten van de tekst. Het CDATA element bevat inhoud die in de uitvoer moet voorgesteld worden als opgenomen in een CDATA sectie. Dit element hebben we bijvoorbeeld in [Sectie 2.3.6.] gebruikt.

De elementen FORMATTED en CONFIGURATION bevatten inhoud waar de witruimte belangrijk is. Deze elementen hebben we ingevoerd als syntactische suiker. Op die manier konden we het gebruik van het attribuut **xml:space** [Sectie 2.3.8.1.] vermijden. Het verschil tussen deze twee elementen is dat CONFIGURATION bedoeld is om (delen van) configuratiebestanden te bevatten. FORMATTED daarentegen is ingevoerd om andere inhoud te bevatten waar de witruimte van belang is. We hebben dit onderscheid gemaakt omdat het om twee verschillende concepten gaat en het mogelijk te maken om de inhoud van deze elementen op een andere manier te tonen.

4.2.4. Ontwikkeld XML formaat voor XML/HTML

We hernemen hier eerst de elementen die we als XML gerelateerd hebben bestempeld, maar in een licht gewijzigde volgorde:

Voorbeeld 4.14. XML gerelateerde elementen

```
<xs:element name="TAG" type="TagType"/>
<xs:element name="XML" type="StructureType"/>
<xs:element name="SGML" type="StructureType"/>
<xs:element name="HTML" type="StructureType"/>
<xs:element name="DOCDECLARATION" type="DocDeclarationType"/>
<xs:element name="PI" type="PiType"/>
<xs:element name="ELEMENT" type="ElementType"/>
<xs:element name="ATTRIBUTE" type="AttributeType"/>
<xs:element name="COMMENT" type="xs:string"/>
```

We beginnen met het element **TAG** dat we gebruiken om bepaalde tags te kunnen opnemen in de tekst. Met tag bedoelen we hier de begintag van een element, zonder attributen. Het type van dit element heeft dan de volgende eenvoudige definitie:

Voorbeeld 4.15. TagType

```
<xs:complexType name="TagType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Dit element heeft dus als enige kind een **NAME** element dat de naam van de tag bevat. Het is misschien raar dat dit element alleen een kindelement **NAME** kan bevatten dat dan van het type **xs:string** is. Waarom hebben we dan niet gewoon aan het element **TAG** het type

xs:string? Oorspronkelijk was het de bedoeling dat element van dit type ook nog **ATTRIBUTE** kind elementen van het type **AttributeType** kon hebben. Omdat we dit uiteindelijk nergens hebben gebruikt is dat hier ook niet gedefinieerd. Maar door dit zo te doen kunnen we probleemloos de definitie van dit type uitbreiden zodat we ook (nul of meer) attributen kunnen opgeven. De definitie kan dan veranderd worden, zonder dat het XML bestand herwerkt moet worden om geldig te blijven. Er mag dan wel niet geëist worden dat er minstens éé of meer attributen voorkomen.

De elementen **XML**, **SGML** en **HTML** dienen om bepaalde structuren te beschrijven. Deze elementen kunnen op dezelfde manier beschreven worden, het is alleen de uitvoer die misschien anders vormgegeven moet worden indien het over kindelementen van het **HTML**, het **SGML** of het **XML** element gaat, maar dit wordt op het niveau van de XSLT transformaties bepaald.

Voorbeeld 4.16. StructureType

```
<xs:complexType name="StructureType">
   <xs:sequence>
      <xs:element name="NAME" minOccurs="0" maxOccurs="1"
      type="xs:string"/>
      <xs:element name="PI" minOccurs="0" maxOccurs="1" type="PiType"/>
      <xs:element name="DOCDECLARATION" minOccurs="0" maxOccurs="1"
      type="DocDeclarationType"/>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
         <xs:element name="DOTS" minOccurs="0" type="EmptyType"/>
         <xs:element name="ELEMENT" minOccurs="0" type="ElementType"/>
         <xs:element name="PI" minOccurs="0" type="PiType"/>
         <xs:element name="COMMENT" minOccurs="0" type="xs:string"/>
      </xs:choice>
   </xs:sequence>
   <xs:attribute ref="id" use="optional"/>
</xs:complexType>
```

Een element van dit type kan een naam (**NAME**) bevatten. Deze naam dient voor een korte beschrijving van wat gaat volgen. Na deze optionele naam kan een optionele processing instruction (**PI**) volgen, gevolgd door een optionele document type declaratie (**DOCDECLARATION**). Daarnaast kunnen in willekeurige volgorde en niet beperkt in aantal de volgende elementen voorkomen: **DOTS**, **ELEMENT**, **PI** en **COMMENT**. De reden waarom we bij deze vier elementen expliciet het attribuut **minOccurs** de waarde **0** hebben gegeven is om aan te geven dat het niet uitmaakt welk van deze vier elementen is dat voorkomt. We eisen immers dat er minstens één van deze elementen voorkomt.

We geven hier een voorbeeldje van hoe bijvoorbeeld een **XML** met meerdere kindelementen met de naam **ELEMENT** er kan uitzien:

Voorbeeld 4.17.

```
<DOTS/>
<ELEMENT>
   <NAME> xsl:template </NAME>
   <ATTRIBUTE>
      <NAME> match </NAME>
      <CONTENT> PARA </CONTENT>
   </ATTRIBUTE>
   <CONTENT> lichaam van de template hier </CONTENT>
</ELEMENT>
<DOTS/>
```

```
<ELEMENT>
   <NAME> xsl:template </NAME>
   <ATTRIBUTE>
      <NAME> match </NAME>
      <CONTENT> XML </CONTENT>
   </ATTRIBUTE>
   <CONTENT> lichaam van de template hier </CONTENT>
</ELEMENT>
<DOTS/>
```

Door de mogelijkheid te voorzien dat meerdere **ELEMENT** elementen kunnen voorkomen als kinderen van element van het type **StructureType** is het mogelijk om delen van XML bestanden te beschrijven zonder steeds het root element van de bestand te moeten beschrijven.

Om een processing instruction in te voegen gebruiken we het element **PI** dat het type **PiType** heeft. Dit type is als volgt gedefinieerd:

Voorbeeld 4.18. PiType, AttributeType

```
<xs:complexType name="PiType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:element name="ATTRIBUTE" minOccurs="0" maxOccurs="unbounded"
      type="AttributeType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="AttributeType">
   <xs:sequence>
      <xs:element name="NAME" minOccurs="1" maxOccurs="1"
      type="xs:string"/>
      <xs:element name="CONTENT" minOccurs="1" maxOccurs="1"
      type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Elementen met dit type hebben een verplichte naam (**NAME**) en nul of meer attributen (**ATTRIBUTE**). Een attribuut is verplicht van een naam (**NAME**) te hebben en heeft altijd een bepaalde waarde (**CONTENT**). We hebben ervoor gekozen om de namen van het type **xs:string** te laten zijn. We weten dat dit niet helemaal correct is, maar indien je de formele beschrijving [TRXMLCSC] bestudeert voor zo een naam, dan zie je al snel in dat een reguliere expressie hiervoor schrijven wel mogelijk is, maar die is dan wel totaal onleesbaar.

Na een eventuele processing instruction kan er een document type declaratie (**DOCDECLARATION**) volgen. Het type van dit element is als volgt gedefinieerd:

Voorbeeld 4.19. DocDeclarationType

```
<xs:complexType name="DocDeclarationType">
   <xs:sequence>
      <xs:element name="ROOTELEMENT" type="xs:string"/>
      <xs:element name="TYPE" type="xs:string"/>
      <xs:element name="URI" type="xs:anyURI" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
```

Een document type declaratie geeft aan welk het root element (**ROOTELEMENT**) is van het beschreven document. Het type (**TYPE**) geeft aan om welk type document type declaratie het gaat. Bijvoorbeeld publiek (PUBLIC) of systeem (SYSTEM). We kunnen dan aan de hand

van het element **URI** ook de locatie van een DTD opgeven. Daarnaast hebben we het volgende ook als een URI beschouwd: **-//OASIS//DTD DocBook V4.1//EN**. Vandaar dat we zeggen dat het voorkomen hiervan onbegrensd is. In de praktijk kan de limiet twee, drie, ... zijn, maar dit is de meest algemene vorm.

Het element **DOTS** wordt gebruikt om aan te geven dat er een gedeelte is weggelaten, met andere woorden dat het niet om de volledige inhoud gaat maar slechts om het relevante deel. We weten dat dit nogal een rare naam is voor dit element, maar onze inspiratie om nog een goede elementnaam te vinden was op. We hebben dan maar het eerste genomen dat ons te binnen schoot.

Het element **ELEMENT** gebruiken we om elementen te beschrijven. Het type **ElementType** wordt als volgt gedefinieerd:

Voorbeeld 4.20. ElementType

```
<xs:complexType name="ElementType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="ATTRIBUTE" type="AttributeType"/>
         <xs:element name="CONTENT" type="xs:string"/>
         <xs:element name="COMMENT" type="xs:string"/>
         <xs:element name="ELEMENT" type="ElementType"/>
         <xs:element name="PI" type="PiType"/>
         <xs:element name="DOTS" type="EmptyType"/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
```

Een element heeft altijd een verplichte naam. Zoals reeds eerder aangehaald vonden we het niet doenbaar om het type van de naam nog te verstrengen. We beschouwen als naam hier zowel een gewone elementnaam als een naam voorafgegaan door een namespacevoorvoegsel. Een element kan een aantal attributen hebben (**ATTRIBUTE**). Daarnaast dient het element **CONTENT** voor de tekstuele inhoud. We kunnen ook commentaar (**COMMENT**) opnemen onder een element, maar ook andere elementen (**ELEMENT**) of processing instructions (**PI**). Met **DOTS** kunnen we ook aangeven dat we een gedeelte hebben weggelaten, dus dat we alleen het relevante gedeelte hebben gegeven.

Dit besluit de bespreking van ons XML formaat om XML/HTML/SGML te beschrijven. We hebben tot nu toe geen XML/HTML(/SGML) constructie gevonden die we hier niet mee konden uitdrukken.

4.2.5. Ontwikkeld XML formaat voor DTD

Ook om DTD's te kunnen modelleren in ons eigen XML formaat hebben we aan aantal elementen en types ingevoerd. We herhalen hier eerst de elementen die rechtstreeks in een paragraaf gebruikt kunnen worden:

Voorbeeld 4.21. DTD gerelateerde elementen

```
<xs:element name="DTD" type="DTDType"/>
<xs:element name="PCDATA" type="EmptyType"/>
```

Het element **PCDATA** dient om de string #PCDATA voor te stellen. Met het element **DTD** stellen we een DTD voor en het type van dit element, **DTDType**, definiëren we als:

Voorbeeld 4.22. DTDType

```
<xs:complexType name="DTDType">
   <xs:sequence>
      <xs:element name="NAME" minOccurs="0" type="xs:string"/>
      <xs:choice maxOccurs="unbounded">
         <xs:element name="ETD" minOccurs="0" maxOccurs="unbounded"
         type="ETDType"/>
         <xs:element name="ADL" minOccurs="0" maxOccurs="unbounded"
         type="ADLType"/>
      </xs:choice>
   </xs:sequence>
   <xs:attribute ref="id" use="optional"/>
</xs:complexType>
```

Een element van het type **DTDType** bevat een optionele naam en daarnaast een aantal elementdefinities (**ETD** - Element Type Definition) en/of attribuutdefinities (**ADL** - Attribute Definition List).

Het element **ETD** heeft als type **ETDType** en is als volgt gedefinieerd:

Voorbeeld 4.23. ETDType

```
<xs:complexType name="ETDType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:choice>
         <xs:element name="EMPTY" type="EmptyType"/>
         <xs:element name="CONTENTMODEL" type="ContentmodelType"/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
```

Een elementdefinitie heeft een verplichte naam (**NAME**). Daarnaast wordt aangegeven of het om een leeg element (**EMPTY**) gaat of om een element met inhoud, waarbij we in dat geval het inhoudsmodel (**CONTENTMODEL**) beschrijven.

Het element **CONTENTMODEL** heeft als type **ContentmodelType** en dit type is als volgt gedefinieerd:

Voorbeeld 4.24. ContentModelType, ContentType

```
<xs:complexType name="ContentmodelType">
   <xs:choice>
      <xs:element name="PCDATA" type="EmptyType"/>
      <xs:sequence>
         <xs:choice maxOccurs="unbounded">
            <xs:element name="CONTENT" minOccurs="0" type="ContentType"/>
            <xs:element name="CHOICE" minOccurs="0">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="CONTENT" maxOccurs="unbounded"
                     type="ContentType"/>
                  </xs:sequence>
                  <xs:attribute ref="occurs" use="optional"/>
               </xs:complexType>
            </xs:element>
         </xs:choice>
```

```
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="ContentType" mixed="true">
    <xs:attribute ref="occurs" use="optional"/>
  </xs:complexType>
```

Dit type gebruiken we om het inhoudsmodel van een elementdefinitie te beschrijven. De inhoud is ofwel #PCDATA (**PCDATA**) ofwel kan de inhoud beschreven worden door een opeenvolging van elementnamen (**CONTENT**) en/of een keuze tussen een aantal elementnamen (**CHOICE**). Het type **ContentType** is gemengd zodat elementen van dit type inhoud kunnen bevatten en we het attribuut **occurs** bij dit type kunnen definiëren.

Het element **ADL** heeft als type **ADLType** en dat type hebben we op de volgende manier gedefinieerd:

Voorbeeld 4.25. ADLType

```
  <xs:complexType name="ADLType">
    <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:element name="ATTRIBUTE">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="NAME" type="xs:string"/>
            <xs:element name="CONTENTMODEL">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="PCDATA" type="EmptyType"/>
                  <xs:element name="CONTENT" type="xs:string"/>
                  <xs:element name="CHOICE">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="CONTENT"
                        maxOccurs="unbounded" type="xs:string"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:choice>
              </xs:complexType>
            </xs:element>
            <xs:element name="REQUIRED" minOccurs="0"
            type="EmptyType"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

Een element van dit type heeft een kindelement dat de naam (**NAME**) van het element aangeeft waar dit attribuut bijhoort. Achteraf bekeken hadden we dit element beter **ELEMENTNAME** genoemd. Het attribuut (**ATTRIBUTE**) heeft zelf ook een naam (**NAME**) en een bepaald inhoudsmodel (**CONTENTMODEL**). We kunnen ook aangeven dat dit attribuut vereist (**REQUIRED**) is. Andere mogelijkheden hebben we niet voorzien wegens niet nodig in onze thesis. Voor het inhoudsmodel kunnen we kiezen tussen #PCDATA (**PCDATA**), een bepaalde inhoud (**CONTENT**) of een keuze (**CHOICE**) tussen een aantal waarden.

We geven hier een klein voorbeeldje van het gebruik van deze elementen:

Voorbeeld 4.26.

```
<ETD>
   <NAME> accessibility </NAME>
   <EMPTY/>
</ETD>
<ADL>
   <NAME> accessibility
      <ATTRIBUTE>
         <CONTENTMODEL>
            <CHOICE>
               <CONTENT> public </CONTENT>
               <CONTENT> private </CONTENT>
               <CONTENT> protected </CONTENT>
               <CONTENT> package </CONTENT>
            </CHOICE>
         </CONTENTMODEL>
         <REQUIRED/>
      </ATTRIBUTE>
   </NAME>
</ADL>
```

Dit zorgt voor de volgende uitvoer:

DTD 4.1.

```
<!ELEMENT accessibility EMPTY>
<!ATTLIST    accessibility    value    (public|private|protected|package)
#REQUIRED>
```

Hiermee besluiten wij de bespreking van het modelleren van DTD's in ons XML formaat. We willen de **nadruk** er op leggen dat deze modellering **niet volledig** is, dat was ook helemaal niet de bedoeling. We hebben datgene gemodelleerd wat wij nodig hadden, niet meer en ook niet minder.

4.2.6. Ontwikkeld XML formaat voor Java

We willen ook hier **benadrukken** dat we van Java alleen gemodelleerd hebben wat we nodig hadden. We hebben zeker niet alles gemodelleerd en sommige zaken zijn ook **semantisch niet volledig correct** gemodelleerd. We zijn hierbij uitgegaan van een ad hoc aanpak: dit hebben we nodig en dat kunnen we op die manier beschrijven. We hebben al zoveel bestudeerd en het zou ons teveel tijd vergen om ook dit nog eens uitgebreid te bestuderen en 100% correct te modelleren. Dit zou een thesis op zich kunnen zijn.

We zullen hier eerst even de element herhalen die we geklasseerd hebben als Java gerelateerd en die we rechtstreeks als kindelementen van een element van het type **ParaType** kunnen gebruiken.

Voorbeeld 4.27. Elementen gerelateerd aan Java

```
<xs:element name="JAVA" type="JavaType"/>
<xs:element name="METHOD" type="MethodType"/>
<xs:element name="VARIABLE" type="VariableType"/>
<xs:element name="ARGUMENT" type="MethodArgumentType"/>
```

We kunnen met deze elementen volledige blokken Java code (**JAVA**) beschrijven, maar ook methodes of methodedefinities (**METHOD**), variabelen en variabeledefinites (**VARIABLE** en parameters of argumenten (**ARGUMENT**). Dit is één van de aspecten waar onze naamgeving conflicteert met de benamingen in Java. Alhoewel een methode-oproep argumenten heeft en een methodedefinitie parameters hebben we deze afzonderlijke concepten dezelfde naam gegeven. We weten dat dit niet correct is maar we gaan dit met het naderen van de deadline niet meer veranderen.

We gaan eerst de eenvoudigste types beschrijven en op die manier toewerken naar de beschrijving van de complexere types. Het element **ARGUMENT** gebruiken we om argumenten van methodes te kunnen beschrijven. De typedefinitie **MethodArgumentType** ziet er als volgt uit:

Voorbeeld 4.28. MethodArgumentType

```
<xs:complexType name="MethodArgumentType">
   <xs:sequence>
      <xs:element name="TYPE" type="identifierType"/>
      <xs:element name="NAME" type="identifierType"/>
   </xs:sequence>
</xs:complexType>
```

Een element van dit type bevat verplicht het type (**TYPE**) en de naam (**NAME**), in dit geval van een argument. Deze twee elementen zijn van het type **identifierType** omdat het geldige Java identifiers moeten zijn. Het type van **TYPE** kan eventueel veranderd worden in **composedIdentifierType**, mocht dit nodig zijn. Dit zou ook kunnen gespecificeerd worden met behulp van het **union** element. We hebben dit niet gedaan omdat het voor ons niet nodig was.

Variabelen (**VARIABLE**) moeten voldoen aan de structuur beschreven door het type **VariableType**. Dit type is als volgt gedefinieerd:

Voorbeeld 4.29. VariableType

```
<xs:complexType name="VariableType">
   <xs:sequence>
      <xs:element name="FINAL" minOccurs="0" maxOccurs="1"
      type="EmptyType"/>
      <xs:element name="STATIC" minOccurs="0" maxOccurs="1"
      type="EmptyType"/>
      <xs:element name="TYPE" type="identifierType"/>
      <xs:element name="NAME" type="identifierType"/>
      <xs:element name="INITIAL" minOccurs="0" type="xs:string"/>
   </xs:sequence>
   <xs:attribute ref="accessibility" use="optional"/>
</xs:complexType>
```

Een variabele kan static (**STATIC**) en/of final (**FINAL**) zijn. Daarnaast is een variabele van een bepaald type (**TYPE**) en heeft een variabele een bepaalde naam (**NAME**). De laatste twee elementen hebben in dit geval weer als type **identifierType**. Net zoals bij **ArgumentType** geldt ook hier de opmerking dat voor **TYPE** we ook **composedIdentifierType** kunnen kiezen. Een variabele kan ook een initiële waarde (**INITIAL**) hebben. We hebben hier gewoon voor het type **xs:string** gekozen omdat we dan eender welke waarde konden gebruiken, inclusief Java expressies. Deze modellering is niet mooi, maar geeft wel veel vrijheid in het kiezen van de waarde. Voor een betere modellering is in onze ogen is daar een grondige studie van de Java taal voor nodig. Tenslotte hebben we ook nog het optionele attribuut **accessibility** waarmee we de toegankelijkheid tot die variabele kunnen specificeren.

Het ontbreken van dit attribuut geeft aan de het om een **package accessible** variabele gaat.

We gaan nu bekijken hoe methodes gemodelleerd worden. We doen dit aan de hand van het type **MethodType**.

Voorbeeld 4.30. MethodType, MethodArgumentsType

```
<xs:complexType name="MethodType">
   <xs:sequence>
      <xs:element name="STATIC" minOccurs="0" type="EmptyType"/>
      <xs:element name="RETURNTYPE" minOccurs="0"
      type="identifierType"/>
      <xs:element name="NAME" type="identifierType"/>
      <xs:element name="ARGUMENTS" type="MethodArgumentsType"/>
      <xs:element name="EXCEPTIONS" minOccurs="0">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="NAME" maxOccurs="unbounded"
               type="identifierType"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="BODY" minOccurs="0" type="BodyType"/>
   </xs:sequence>
   <xs:attribute ref="accessibility" use="optional"/>
</xs:complexType>
<xs:complexType name="MethodArgumentsType">
   <xs:sequence>
      <xs:element name="ARGUMENT" minOccurs="0" maxOccurs="unbounded"
      type="MethodArgumentType"/>
   </xs:sequence>
</xs:complexType>
```

Een element van dit type, typisch een element om een methode te beschrijven, definieert of deze methode al dan niet statisch (**STATIC**) is. Een methode heeft ook een returntype (**RETURNTYPE**). Het element **RETURNTYPE** is optioneel zodat we met deze beschrijving ook constructoren, die geen terugkeerwaarde hebben, kunnen modelleren. Hierdoor is het mogelijk methodes zonder terugkeertype te modelleren, maar we moeten dat kleine beetje zelfdiscipline opbrengen om dat niet te doen. We merken op dat ook **void** voor ons tot de returntypes behoort. Daarnaast heeft de methode ook nog een naam (**NAME**), een aantal argumenten (**ARGUMENTS**) en een lichaam (**BODY**). Verder kan een methode ook nog een aantal uitzonderingen (**EXCEPTIONS**) gooien. Merk ook hier weer het gebruik van het attribuut **accessibility** op. De lijst van argumenten kan nul of meer **ARGUMENT** elementen bevatten die van het reeds behandelde type **MethodArgumentType** zijn.

Het lichaam (**BODY**) van een methode wordt gemodelleerd aan de hand van het type **BodyType** dat we als volgt hebben gedefinieerd:

Voorbeeld 4.31. BodyType

```
<xs:complexType name="BodyType">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="ASSIGNMENT" type="AssignmentType"/>
      <xs:element name="COMMENT" type="CommentType"/>
      <xs:element name="VARIABLE" type="VariableType"/>
      <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
      <xs:element name="CONSTRUCTION" type="ConstructionType"/>
      <xs:element name="TRYBLOCK" type="TryBlockType"/>
      <xs:element name="IFBLOCK" type="IfBlockType"/>
      <xs:element name="FORLOOP" type="ForLoopType"/>
      <xs:element name="RETURNSTATEMENT" type="RetStmtType"/>
      <xs:element name="THROW" type="ThrowType"/>
```

```
      <xs:element name="DOTS" type="EmptyType"/>
   </xs:choice>
</xs:complexType>
```

In het lichaam van een methode kan er vanalles zitten zoals uit deze definitie blijkt. Het gaat hier om verschillende soorten uitdrukkingen, met uitzondering van de elementen **COMMENT** en **DOTS**. We kunnen gebruik maken van toekenningen (**ASSIGNMENT**), commentaar (**COMMENT**) en (locale) variabelen (**VARIABLE**). Ook het toepassen van methodes (**APPLYMETHOD**) of het aanmaken van nieuwe objecten (**CONSTRUCTION**) kan in het lichaam van een methode gebeuren. Voorwaardelijke uitvoering (**IFBLOCK**), lussen (**FORLOOP**) of defensief programmeren (**TRYBLOCK**) zijn mogelijk. Tenslotte kunnen we nog uitzonderingen gooien (**THROW**) of een waarde teruggeven (**RETURNSTATEMENT**), maar we kunnen ook de niet-relevante delen weglaten (**DOTS**).

We geven aan de we een toekenning willen modelleren door gebruik te maken van het **ASSIGNMENT** element. Het type van dit element, met name **AssignmentType** is als volgt gedefinieerd:

Voorbeeld 4.32. AssignmentType

```
<xs:complexType name="AssignmentType">
   <xs:sequence>
      <xs:element name="TO">
         <TYPE minOccurs="0" type="identifierType"/>
         <NAME type="identifierType"/>
      </xs:element>
      <xs:choice>
         <xs:element name="FROM" type="xs:string"/>
         <xs:element name="CONSTRUCTION" type="ConstructionType"/>
         <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
```

In een toekenning willen we een waarde (in de meest ruime zin van het woord) toekennen aan (**TO**) iets. Dit iets beschrijven we als een optioneel type (**TYPE**), maar het moet wel een geldige naam hebben (**NAME**). De waarde die we toekennen moet natuurlijk ergens van komen. Dit kan het resultaat zijn van de uitvoering van een methode (**APPLYMETHOD**) of van het aanmaken van een nieuw object (**CONSTRUCTION**). Tenslotte kunnen we ook gewoon een waarde opgeven (**FROM**). Uitdrukkingen van de vorm **x = 5;** worden door ons als volgt voorgesteld:

Voorbeeld 4.33.

```
<ASSIGNMENT>
   <TO>
      <NAME> x </NAME>
   </TO>
   <FROM> 5 </FROM>
</ASSIGNMENT>
```

Ook kunnen we gebruik maken van commentaar (**COMMENT**) binnen het lichaam van een methode. Dit element heeft als type **CommentType** en dat type voldoet aan de volgende definitie:

Voorbeeld 4.34. CommentType

```
<xs:complexType name="CommentType" mixed="true">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="PARAM" type="CommentParamType"/>
      <xs:element name="RETURN" maxOccurs="1" type="xs:string"/>
   </xs:choice>
   <xs:attribute name="type" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="impl"/>
            <xs:enumeration value="doc"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
<xs:complexType name="CommentParamType">
   <xs:sequence>
      <xs:element name="NAME" type="identifierType"/>
      <xs:element name="DESCRIPTION" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Elementen van dit type hebben een verplicht attribuut **type**. Met de waarde van dit attribuut geef je aan of het om implementatiecommentaar (**impl**) of documentatiecommentaar (**doc**) gaat. We hebben dit type gedefinieerd als een type met gemengde inhoud. Voor het geval we bezig zijn met documentatiecommentaar vinden we het ook wel nuttig om de parameters van een methode (**PARAM**) en het resultaat (**RETURN**) te kunnen beschrijven. Een parameter heeft een bepaalde naam (**NAME**) en we plakken daar een beschrijving (**DESCRIPTION**) aan vast. We zouden echter willen specificeren dat deze elementen alleen mogen gebruikt worden indien het attribuut **type** de waarde **doc** heeft. Dit blijkt helaas niet mogelijk te zijn. We konden ook een opsplitsing maken in twee elementen, maar we vinden het nog altijd logischer dat onderscheid te maken op basis van een attribuut. Uit praktisch oogpunt bekeken zou het ons echter ook teveel werk vragen om onze documenten en stylesheets nog te herwerken. Ook zouden we willen opleggen dat alleen implementatiecommentaar in het lichaam van een methode kan voorkomen. Dit zou ook mogelijk gemaakt kunnen worden door de opsplitsing in twee elementen, maar opnieuw, vanuit praktisch oogpunt bekeken zou dat te veel tijd vragen, tijd die we goed konden gebruiken voor het afwerken van onze teksten.

Het element **VARIABLE** van het type **VariableType** hebben we reeds behandeld.

Met behulp van het **APPLYMETHOD** element modelleren we het toepassen van methodes. De volgende typedefinities zijn hiervoor van belang:

Voorbeeld 4.35. Typedefinities voor APPLYMETHOD

```
<xs:complexType name="ApplyMethodType">
   <xs:sequence>
      <xs:element name="METHODNAME" minOccurs="1" maxOccurs="1"
      type="identifierType"/>
      <xs:element name="APPLYTO" minOccurs="0" maxOccurs="1"
      type="ArgumentType"/>
      <xs:element name="ARGUMENTS" minOccurs="1" maxOccurs="1"
      type="ArgumentsType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="ArgumentsType">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="ARGUMENT" type="ArgumentType"/>
   </xs:choice>
</xs:complexType>
<xs:complexType name="ArgumentType" mixed="true">
   <xs:choice minOccurs="0">
```

```
      <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
      <xs:element name="CONSTRUCTION" type="ConstructionType"/>
    </xs:choice>
  </xs:complexType>
```

Het **ApplyMethodType** modelleert het toepassen van een methode (**METHODNAME**). Deze methode kan toegepast worden op iets. Dit iets kan het resultaat zijn van het toepassen van een methode (**APPLYMETHOD**), het uitvoeren van een constructor (**CONSTRUCTION**) of kan één of ander Java object zijn (waarde van het **mixed** attribuut is **true**). **Merk op**: door het gebruiken van het attribuut mixed verliezen we eenn aantal verstrengingen die we zouden kunnen opleggen. De bedoeling voor de elementen **ARGUMENT** en **APPLYTO** was de volgende: ofwel bevatten ze een Java identifier ofwel hebben ze één van de elementen **APPLYMETHOD** en **CONSTRUCTION** als kindelement. Dit is niet af te dwingen in XML Schema, tenzij we de modellering op een andere manier zouden doen. Deze andere manier zou er dan bestaan uit het weglaten van het **mixed** attribuut en een extra kindelement invoeren dat de Java identifier kan bevatten. Op die manier kan het ook op een modelleringsfoutje wijzen. Indien we met behulp van **xs:union** [Sectie 3.5.8.] de keuze konden aangeven tussen aan atomair type en een complex type stelde dit probleem zich niet. Dit is echter niet toegelaten.

Voor sommige mensen kan de volgorde van de elementen van het type **ApplyMethodType** raar overkomen, maar voor ons leek dit vrij logisch. Meer uitleg kunnen we ook niet geven over deze keuze.

We kunnen ook nieuwe objecten aanmaken (**CONSTRUCTION**), althans we hebben getracht dit te modelleren. Het type van dit element, **ConstructionType**, wordt als volgt gedefinieerd:

Voorbeeld 4.36. ConstructionType

```
  <xs:complexType name="ConstructionType">
    <xs:sequence>
      <xs:element name="CONSTRUCTNAME" minOccurs="0" maxOccurs="1"
      type="identifierType"/>
      <xs:element name="CONSTRUCTCLASS" minOccurs="0" maxOccurs="1"
      type="composedIdentifierType"/>
      <xs:element name="CLASS" minOccurs="1" maxOccurs="1"
      type="composedIdentifierType"/>
      <xs:element name="ARGUMENTS" minOccurs="1" maxOccurs="1"
      type="ArgumentsType"/>
    </xs:sequence>
  </xs:complexType>
```

Het nieuw gecreëerde object heeft een bepaalde naam (**CONSTRUCTNAME**) en is van een bepaald type (**CONSTRUCTCLASS**). Om dit object aan te maken roepen we een bepaalde constructor (**CLASS**) op met de nodige argumenten (**ARGUMENTS**). Dat **CONSTRUCTNAME** voor **CONSTRUCTCLASS** komt is een volgorde die ontstaan is omwille van historische redenen. Het was eerst niet nodig dat het type van het gecreëerde object verschillend was van het type van de constructor. Met het vorderen van de tekst bleek dit wel nodig te zijn. Er is toen gekozen om de namen van de verschillende types achter elkaar en na de naam van het te creëren object te zetten. Alhoewel het daarna logischer bleek om de volgorde van **CONSTRUCTCLASS** en **CONSTRUCTNAME** om te wisselen zijn we hier niet toe gekomen.

Het is ook mogelijk om een try-catch block te modelleren. We hebben daarvoor de volgende types gedefinieerd:

Voorbeeld 4.37. Types voor TRYBLOCK

```
<xs:complexType name="TryBlockType">
   <xs:sequence>
      <xs:element name="BODY" type="BodyType"/>
      <xs:element name="CATCHBLOCK" maxOccurs="unbounded"
      type="CatchBlockType"/>
      <xs:element name="FINALLY" minOccurs="0" type="FinallyType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="CatchBlockType">
   <xs:sequence>
      <xs:element name="EXCEPTION">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="NAME" type="identifierType"/>
               <xs:element name="SHORTNAME" type="identifierType"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="BODY" type="BodyType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="FinallyType">
   <xs:sequence>
      <xs:element name="BODY" type="BodyType"/>
   </xs:sequence>
</xs:complexType>
```

Een element van het type **TryBlockType** heeft altijd een lichaam (**BODY**), waarvan we het type reeds besproken hebben. Met een try statement in Java is ook altijd minstens één catch statement (**CATCHBLOCK**) verbonden en hooguit één finally statement (**FINALLY**). Bij een catch statement geven we aan welke uitzondering we gaan behandelen (**NAME**) en we geven deze een naam (**SHORTNAME**), die we in het lichaam (**BODY**) van het catch statement kunnen gebruiken. Ons finally statement heeft alleen een lichaam (**BODY**) met het reeds besproken type **BodyType**. Merk op dat we een element van het type **TryBlockType** ook laten dienen als container voor de elementen **CATCHBLOCK** en **FINALLY**. Alhoewel een **CATCHBLOCK** bij een **TRYBLOCK** hoort en niet erin in de Java taal, vonden we dit makkelijker in het gebruik. We hadden ook voor de naam **TRYCATCHFINALLYBLOCK** kunnen kiezen, waardoor deze bedenking niet geopperd zou kunnen worden, maar deze naam is te lang om comfortabel mee te werken.

Een if-blok (**IFBLOCK**) hebben we beschreven aan de hand van de volgende types:

Voorbeeld 4.38. Types voor IFBLOCK

```
<xs:complexType name="IfBlockType">
   <xs:sequence>
      <xs:element name="TEST" type="TestType"/>
      <xs:element name="BODY" type="BodyType"/>
      <xs:element name="ELSEIF" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="TEST" type="TestType"/>
               <xs:element name="BODY" type="BodyType"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="ELSE" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="BODY" type="BodyType"/>
```

```
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TestType">
      <xs:sequence>
        <xs:choice>
          <xs:sequence>
            <xs:element name="ARGUMENT" minOccurs="2" maxOccurs="2"
            type="ArgumentType"/>
          </xs:sequence>
          <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="equals"/>
            <xs:enumeration value="notequals"/>
            <xs:enumeration value="equalsmethod"/>
            <xs:enumeration value="applymethod"/>
            <xs:enumeration value="notapplymethod"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
```

Het type voor een if-blok (**IfBlockType**) hebben we gemodelleerd als bestaande uit een test (**TEST**). Afhankelijk van die test wordt al dan niet het lichaam (**BODY**) van het if-blok uitgevoerd. Indien de test mislukt, hebben we ook nog de mogelijkheid voorzien om andere code (**BODY**) uit te voeren op een bepaalde voorwaarde (**ELSEIF**) of dat er gewoon code moet uitgevoerd worden indien alle voorgaande testen mislukken (**ELSE**).

Een element van het type **TestType** heeft een verplicht attribuut **type** dat aangeeft om wat voor een test het gaat. We geven hier een kort lijstje wat de betekenis is van de verschillende waarden van dit attribuut:

- **equals**: de gelijkheidsoperator **==**
- **notequals**: de gelijkheidsoperator **!=**
- **equalsmethod**: de methode die dient om twee objecten op gelijkheid te testen, met name arg1.**equals**(arg2)
- **applymethod**: testen of het resultaat van een booleanse methode waar is of niet
- **notapplymethod**: de logische not operatie toegepast op **applymethod**

We zijn er ons ten volle van bewust dat dit niet de hele lading dekt, maar het volstond voor onze doeleinden. De mogelijkheid om twee elementen met de naam **ARGUMENT** op te geven willen we eigenlijk beperken tot het geval dat het attribuut **type** een van de eerste drie waarden heeft, en het gebruik van **APPLYMETHOD** willen we beperken tot het geval dat het attribuut **type** ofwel de waarde **applymethod** ofwel de waarde **notapplymethod** heeft. Dit gaat volgens ons niet in de W3C XML Schema Language. Ook hier hebben we de opsplitsing in twee afzonderlijke elementen niet overwogen. Indien we dit overal zouden moeten doen dan hebben we een overdaad aan elementen en is het gebruik niet langer intuïtief. We hebben daarom gekozen voor het opleggen van zelfdiscipline in plaats van het opsplitsten van elementen.

Een forloop (**FORLOOP**) heeft een initialisatie (**INITIALIZATION**), een stopconditie (**TERMINATION**) en een iteratiestap (**ITERATION**). In het lichaam (**BODY**) van de forloop kan dan allerhande code gemodelleerd worden.

Voorbeeld 4.39. Types voor FORLOOP

```
<xs:complexType name="ForLoopType">
   <xs:sequence>
      <xs:element name="INITIALIZATION" type="xs:string"/>
      <xs:element name="TERMINATION" type="xs:string"/>
      <xs:element name="ITERATION" type="xs:string"/>
      <xs:element name="BODY" type="BodyType"/>
   </xs:sequence>
</xs:complexType>
```

Het type van het element **RETURNSTATEMENT** is als volgt gedefinieerd:

Voorbeeld 4.40. Types voor RETURNSTATEMENT

```
<xs:complexType name="RetStmtType">
   <xs:sequence>
      <xs:element name="ARGUMENT" type="ArgumentType"/>
   </xs:sequence>
</xs:complexType>
```

Er is één kindelement, namelijk **ARGUMENT**, dat beschrijft wat er teruggeven moet worden bij het verlaten van de methode waar dit statement deel van uitmaakt. Vermits het type **ArgumentType** gemengde inhoud beschrijft is eender welke waarde als terugkeerwaarde mogelijk.

Het type van het element **THROW** is als volgt gedefinieerd:

Voorbeeld 4.41. Types voor THROW

```
<xs:complexType name="ThrowType">
   <xs:sequence>
      <xs:element name="ARGUMENT" type="ArgumentType"/>
   </xs:sequence>
</xs:complexType>
```

Er is één kindelement, namelijk **ARGUMENT**, dat beschrijft welke uitzondering er gegooid moet worden.

Het element **DOTS** hebben we reeds besproken.

Naast alles wat we opgesomd hebben is het ook mogelijk om interfaces en klassen te modelleren. Voor een interface gebruiken we het element **INTERFACE** en het type **InterfaceType** dat we nu zullen bespreken:

Voorbeeld 4.42. Types voor interfaces

```
<xs:complexType name="InterfaceType">
   <xs:sequence>
      <xs:element name="NAME" type="identifierType"/>
      <xs:element name="EXTEND" minOccurs="0" maxOccurs="unbounded"
      type="identifierType"/>
      <xs:element name="BODY" type="InterfaceBodyType"/>
   </xs:sequence>
   <xs:attribute ref="accessibility" use="optional"/>
</xs:complexType>
```

```
<xs:complexType name="InterfaceBodyType">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="VARIABLE" type="VariableType"/>
      <xs:element name="METHOD" type="InterfaceMethodType"/>
      <xs:element name="COMMENT" type="CommentType"/>
      <xs:element name="DOTS" type="EmptyType"/>
   </xs:choice>
</xs:complexType>
<xs:complexType name="InterfaceMethodType">
   <xs:sequence>
      <xs:element name="STATIC" minOccurs="0" type="EmptyType"/>
      <xs:element name="RETURNTYPE" minOccurs="1"
      type="identifierType"/>
      <xs:element name="NAME" type="identifierType"/>
      <xs:element name="ARGUMENTS" type="MethodArgumentsType"/>
      <xs:element name="EXCEPTIONS" minOccurs="0">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="NAME" type="identifierType"/>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
   </xs:sequence>
   <xs:attribute ref="accessibility" use="optional"/>
</xs:complexType>
```

Een interface (**INTERFACE**) kan dus gemodelleerd worden (**InterfaceType**) als een element met een optioneel attribuut **accessibility**. Een interface heeft ook een naam (**NAME**) en kan de uitbreiding (**EXTEND**) zijn van een aantal andere interfaces. Ook heeft een interface een lichaam (**BODY**). In het lichaam van een interface kunnen variabelen (**VARIABLE**), commentaar (**COMMENT**) en methodes (**METHOD**) staan. Het verschil met het reeds behandelde **MethodType** is dat een methode in een interface geen lichaam kan hebben. Merk op dat bij **InterfaceMethodType** inderdaad de mogelijkheid ontbreekt om een lichaam op te geven.

Zoals reeds gezegd kunnen we ook klassen modelleren. Voor een klasse (**CLASS**) hebben we dit gedaan aan de hand van het type **ClassType**:

Voorbeeld 4.43. ClassType. ClassBodyType

```
<xs:complexType name="ClassType">
   <xs:sequence>
      <xs:element name="NAME" type="identifierType"/>
      <xs:element name="EXTEND" minOccurs="0" type="identifierType"/>
      <xs:element name="IMPLEMENT" minOccurs="0" maxOccurs="unbounded"
      type="identifierTYpe"/>
      <xs:element name="BODY" type="ClassBodyType"/>
   </xs:sequence>
   <xs:attribute ref="accessibility" use="optional"/>
</xs:complexType>
<xs:complexType name="ClassBodyType">
   <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="VARIABLE" type="VariableType"/>
      <xs:element name="METHOD" type="MethodType"/>
      <xs:element name="COMMENT" type="CommentType"/>
      <xs:element name="DOTS" type="EmptyType"/>
   </xs:choice>
</xs:complexType>
```

Naast het optionele attribuut **accessibility** heeft een klasse ook een name (**NAME**) en kan een klasse overerven (**EXTEND**) van maximum één andere klasse. Een klasse kan ook een aantal interfaces implementeren (**IMPLEMENT**). Daarnaast hebben we ook nog het lichaam

BODY) van de klasse waarin we variabelen (**VARIABLE**), methodes (**METHOD**) en commentaar (**COMMENT**) kunnen gebruiken.

Al deze Java gerelateerde elementen kunnen voorkomen als kindelementen van het **JAVA** element dat het type **JavaType** heeft:

Voorbeeld 4.44. JavaType

```
<xs:complexType name="JavaType">
   <xs:sequence>
      <xs:element name="NAME" minOccurs="0" maxOccurs="1"
      type="xs:string"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="PACKAGE" type="composedIdentifierType"/>
         <xs:element name="COMMENT" type="CommentType"/>
         <xs:element name="IMPORT" type="composedIdentifierType"/>
         <xs:element name="DOTS" type="EmptyType"/>
         <xs:element name="CLASS" type="ClassType"/>
         <xs:element name="INTERFACE" type="InterfaceType"/>
         <xs:element name="VARIABLE" type="VariableType"/>
         <xs:element name="METHOD" type="MethodType"/>
         <xs:element name="ASSIGNMENT" type="AssignmentType"/>
         <xs:element name="TRYBLOCK" type="TryBlockType"/>
         <xs:element name="IFBLOCK" type="IfBlockType"/>
         <xs:element name="FORLOOP" type="ForLoopType"/>
         <xs:element name="CONSTRUCTION" type="ConstructionType"/>
         <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
      </xs:choice>
   </xs:sequence>
   <xs:attribute ref="id" use="optional"/>
</xs:complexType>
```

We leggen hier enkel op dat de omschrijving (**NAME**) die we optioneel aan het **JAVA** element meegeven als eerste kindelement moet voorkomen. Voor de rest maakt de volgorde niet veel uit. We hebben het hier aan het gezond verstand van de maker van het XML document overgelaten om de elementen in een logische volgorde te plaatsen.

4.2.7. Ontwikkeld XML formaat voor litteratuurstudie

Ook wij zijn aan onze thesis begonnen met een litteratuurstudie. Voor deze litteratuurstudie was ook een formaat nodig en ook hiervoor hebben wij zelf een formaat ontwikkeld. Het element **READINGS** heeft als type **ReadingsType**. Dit type is gedefinieerd als:

Voorbeeld 4.45. ReadingsType, ReaderType, ArticleType, BookType, ReviewerType

```
<xs:complexType name="ReadingsType">
   <xs:sequence>
      <xs:element name="READER" maxOccurs="unbounded"
      type="ReaderType"/>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
         <xs:element name="ARTICLE" type="ArticleType"/>
         <xs:element name="BOOK" type="BookType"/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="ReaderType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:element name="EMAIL" type="xs:string"/>
      <xs:element name="ID" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="ArticleType">
   <xs:sequence>
      <xs:element name="TITLE" type="xs:string"/>
      <xs:element name="URI" minOccurs="0" type="xs:anyURI"/>
      <xs:element name="CONTENTS" type="xs:string"/>
      <xs:element name="AUTHOR" maxOccurs="unbounded" type="xs:string"/>
      <xs:element name="REVIEWER" minOccurs="0" maxOccurs="unbounded"
      type="ReviewerType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="BookType">
   <xs:sequence>
      <xs:element name="TITLE" type="xs:string"/>
      <xs:element name="AUTHOR" maxOccurs="unbounded" type="xs:string"/>
      <xs:element name="ISBN" minOccurs="0" type="xs:string"/>
      <xs:element name="CONTENTS" type="xs:string"/>
      <xs:element name="REVIEWER" minOccurs="0" maxOccurs="unbounded"
      type="ReviewerType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="ReviewerType">
   <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="REVIEW" type="xs:string"/>
      <xs:element name="RATING">
         <xs:restriction base="xs:string">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="10"/>
         </xs:restriction>
      </xs:element>
   </xs:sequence>
</xs:complexType>
```

Voor een litteratuurstudie zijn er één of meer lezers (**READER**), die een aantal artikels (**ARTICLE**) of boeken (**BOOK**) lezen. Een lezer wordt geïdentificeerd door een naam (**NAME**), e-mailadres (**EMAIL**) een een identifier (**ID**). Deze identifier moet uniek zijn onder de lezers onderling want deze identifier wordt gebruikt om te refereren aan een bepaalde lezer.

Als artikels hebben wij alleen artikels op het Internet beschouwd. Vandaar dat een artikel geïdentificeerd wordt met behulp van een URI (**URI**). Een artikel heeft ook nog titel (**TITLE**), eem omschrijving van de inhoud (**CONTENTS**), één of meer auteurs (**AUTHOR**). Het artikel kan, maar moet niet, nagelezen zijn en beoordeeld (**REVIEWER**). Dezelfde beschrijving geldt voor een boek (**BOOK**), met een ISBN nummer (**ISBN**) in plaats van een URI en een, om historische redenen, iets andere volgorde van de elementen.

Een reviewer heeft een identifier (**ID**) waarmee we naar een identifier van de lezers verwijzen. Hierdoor moeten we niet elke keer de volledige naam herhalen. Een reviewer heeft een review geschreven voor het desbetreffende boek of artikel en geeft een beoordeling, tussen 0 en 10, aan dat boek of artikel.

4.2.8. Ontwikkeld XML formaat voor bibliografie

We hebben ook een eigen formaat ontwikkeld om onze bibliografie te kunnen beheren. Alle ingangen in onze biblografie zijn kindelementen van het element **BIBLIOGRAPHY**. Dit element heeft het type **BibliographyType** dat gedefinieerd is als:

Voorbeeld 4.46. BibliographyType, BibItemType

```
<xs:complexType name="BibliographyType">
   <xs:sequence>
      <xs:element name="BIBITEM" maxOccurs="unbounded"
      type="BibItemType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="BibItemType">
   <xs:sequence>
      <xs:element name="REFERENCE" type="xs:string"/>
      <xs:element name="DESCRIPTION" type="xs:string"/>
      <xs:element name="URL" minOccurs="0" type="xs:anyURI"/>
   </xs:sequence>
</xs:complexType>
```

Het type **BibliographyType** op dat er minstens één ingang (**BIBITEM**) is in de bibliografie. Een ingang wordt beschreven door een unieke naam **REFERENCE**, waarbij we zelf moeten zorgen dat deze uniek is. Het is naar deze referentie dat we met een **BIBREFERENCE** verwijzen. Daarnaast is er ook nog een beschrijving (**DESCRIPTION**) en een locatie (**URL**) voor deze ingang voorzien. Merk op dat we geen mogelijkheid voorzien hebben om bijvoorbeeld een **ISBN** nummer op te geven. Dit komt omdat alle bronnen die we geraadpleegd hebben on-line terug te vinden zijn. Het probleem met de klassieke boeken is dat deze reeds verouderd zijn op het moment dat ze kunnen uitgegeven worden. Dit wordt onder andere veroorzaakt door de snelle evolutie in de hele XML wereld. Deze boeken hebben zeker hun nut als naslagwerk, maar waren voor ons toch niet recent genoeg (of te duur).

4.2.9. Bedenking

We zouden bij ons schema twee bedenkingen willen formuleren. Eerst en vooral hebben we voor alle elementnamen gekozen om met hoofdletters te werken. De enige reden hiervoor is om een beter zichtbaar onderscheid te hebben met gewone tekst die de inhoud van de elementen vormt. Dit komt omdat we gebruik hebben gemaakt van pure tekst editors en niet van een speciale XML editor. De volwassen XML editors waren te duur (in de buurt van $400) en de freeware XML editors stonden nog in hun kinderschoenen (lees: teveel foutmeldingen om er mee te kunnen werken).

Een tweede bedenking is dat we bij het specificeren van dit schema geen gebruik hebben gemaakt van de volledige kracht van de W3C XML Schema Language. Zo zou de aandachtige lezer kunnen opmerken dat we opvallend weinig gebruik hebben gemaakt van **xs:group** terwijl deze mogelijkheid er wel degelijk is. De reden waarom we dit niet gedaan hebben is dat pas heel op het einde gebleken gebleken is dat dit mogelijk is. We hebben ons XML formaat (en dit schema) incrementeel opgebouwd. Bij de eerste versie van de meeste elementen was niet duidelijk dat ze zoveel gemeenschappelijk hadden. Dit is pas gebleken toen we deze elementen nog eens bekeken en nog wat gingen verfijnen. Beslist is om dit zo te laten om duidelijk aan te tonen dat je niet alles op voorhand kan voorzien en dat het maken van een XML formaat een incrementeel proces is.

4.3. Problemen: speciale tekens

Zoals in het hoofdstuk over XML Schema aangehaald is het niet mogelijk om in XML Schema entiteiten te definiëren [Sectie 3.7.] . We willen deze uitspraak enigszins nuanceren. Het is wel mogelijk om een soortgelijk, maar niet identiek, effect te bekomen. We zullen dit uitleggen aan de hand van de entiteit **eacute**. Wanneer we entiteiten gebruiken dan gebruiken we é om é in onze tekst te krijgen. Een soortgelijk effect kunnen we bekomen door de

volgende elementdefinitie te gebruiken:

Voorbeeld 4.47.

```
<xs:element name="eacute" type="xs:token" fixed="é"/>
```

Om é in onze tekst te krijgen gebruiken we dan het element **eacute**. **Let wel**: het vervangen van het element door é zou door een schemaprocessor moeten gebeuren [TRXS0-C]. Het matchen van strings wordt hierdoor extra gecompliceerd.

Vermits Cocoon 2 niet over een schemaprocessor beschikt, hadden we ook nog de mogelijkheid om het vervangen van deze element door entiteiten in onze XSLT transformaties op te lossen. We hebben dit bewust niet gedaan om onze transformaties niet te overladen en het XML Schema voor onze tekst niet nodeloos ingewikkeld te maken.

Als oplossing hebben we gekozen voor een DTD bestand waarin een groot aantal entiteiten gedefinieerd zijn [Sectie 2.3.3.] . In elk XML bestand dat we aanmaken verwijzen we naar dit bestand. Het voordeel is dat we gewoon de entiteiten kunnen gebruiken. Wel heeft dit als nadeel dat, indien we een XML bestand verplaatsen, we de verwijzing naar het DTD bestand moeten aanpassen of (een kopie van) het externe DTD bestand mee moeten verhuizen. We vonden dit slechts een klein nadeel tegenover de andere mogelijkheden die we hier zojuist beschreven hebben.

5. XSL(T)

In het hoofstuk over XML [Hoofdstuk 2.] hebben wij beschreven wat XML is en hoe het gebruikt kan worden. Nu is een XML document op zichzelf eigenlijk niet erg bruikbaar, het wordt pas interessant als we dit XML document gaan omzetten naar andere soorten documenten of de voor ons relevante informatie eruit halen. Dat is precies waaruit de taak van XSL bestaat. In dit hoofdstuk beschrijven wij eerst kort wat XSL is en we belichten kort de ontstaansgeschiedenis. Vervolgens gaan we dieper in op de syntax van de transformatietaal XSLT, die een onderdeel is van XSL. Omdat XSLT niet zonder XPath kan, de taal die gebruikt wordt voor adressering binnen een XML document, eindigen we het hoofdstuk met een uitgebreide beschrijving over hoe je XPath kan gebruiken. In het hoofdstuk over XSL-FO [Hoofdstuk 7.] zullen we dan het tweede onderdeel van XSL, de vormgevingstaal XSL-FO, bespreken.

Voor dit hoofdstuk baseerden wij ons voornamelijk op [TWVB], [TRXSLT], [RDXSL], , en [TRXPA].

5.1. Wat is XSL?

De eXtensible Stylesheet Language (XSL) bestaat eigenlijk uit twee aparte talen. Je hebt de transformatietaal XSL Transformations (XSLT) en de formatteringstaal XSL Formatting Objects (er wordt hiernaar verwezen met FO, XSL-FO, XSL:FO of gewoon XSL). De syntax van XSLT en XSL-FO is een XML syntax die gedefinieerd is in twee XML Document Type Definitions. Aangezien XSL-FO en XSLT beiden gebruik maken van een XML syntax, is bijgevolg elk XSL-FO of XSLT document een well-formed XML document. XSLT en XSL-FO zijn volkomen onafhankelijk van elkaar.

De transformatietaal levert elementen die de regels (templates) vastleggen waarin gedefinieerd wordt hoe een XML document naar een ander soort document (meestal XML of HTML) omgezet moet worden. Dit maakt van XSLT een zeer belangrijke component van XML-gebaseerde elektronische handel, uitwisseling van elektronische data en metadata en van elke applicatie die omzettingen moet maken tussen verschillende XML representaties van dezelfde gegevens. XSL-FO aan de andere kant is een taal voor het beschrijven van de manier waarop een XML document vormgegeven moet worden.

Wie XSLT gebruikt, kan niet buiten XPath. XPath is een taal om bepaalde informatie in de structuur van het XML document terug te vinden. Ze is ontworpen om zowel door XSLT als XPointer [TRXPOINTER] (taal voor adressering binnen de interne structuren van XML documenten en extern geparste entiteiten) gebruikt te worden. Soms wordt XPath als het derde onderdeel van XSL beschouwd. In de literatuur bestaat hierover echter geen consensus.

Voorts is het belangrijk op te merken dat XSL nog steeds in de ontwikkelingsfase zit.

5.2. Korte geschiedenis van XSL

Voor deze sectie gebruikten wij [TWVB] als bron.

5.2.1. Cascading Style Sheets (CSS)

De oorsprong van XSL ligt in Cascading Style Sheets [TRCSS], waarbij een "stylesheet" wordt gebruikt om formattering toe te voegen aan een HTML file. XSLT heeft echter een heel

andere functie dan CSS stylesheets. CSS maakt het enkel mogelijk om de kleuren, achtergronden en font-types voor een HTML webpagina te definiëren. XSLT daarentegen maakt het mogelijk een XML file in een HTML file of een ander tekstgebaseerd formaat om te zetten. Maar het oorspronkelijke idee achter beide, de wens om de informatie-inhoud van de presentatievorm te scheiden, is hetzelfde. Voor verdere informatie over de analogiëen tussen CSS en XSLT verwijzen wij naar [HSTED].

5.2.2. XML Query Language

Aangezien XML toelaat je eigen markup tags te definiëren voor je documenten, werd het omzetten van XML documenten een algemene vereiste. Omdat de meeste browsers niet direct XML documenten kunnen tonen, was het nodig om XML in HTML om te zetten opdat de browsers het document zouden kunnen tonen als een webpagina. Om aan deze noden tegemoet te komen, dienden Microsoft, Texcel en webMethods in september 1998 een voorstel in bij het W3C, de XML Query Language of XQL [MMW3]. Een deel van dat voorstel was het gebruik van de XSL taal. Het voornaamste kenmerk van deze taal was het gebruik van matchpatronen als de basis voor een algemeen query (vraag) mechanisme voor XML documenten.

5.2.3. eXtensible Stylesheet for Transformation

In mei 1999 besliste het W3C om al het onderzoek dat gaande was op zowel het vlak van CSS als van XML Query Language in verband met "a common core semantic model for querying" samen te vatten en het resultaat was de introductie van de eXtensible Stylesheet Language for Transformation of XSLT.

5.2.4. XPath

Gedurende de ontwikkeling van XSLT, werd ook een ander lid van de XML familie gedefinieerd, met name XPointer [TRXPOINTER]. XPointer specificeert een taal die adressering in de interne structuren van XML documenten mogelijk maakt. Zowel XPointer als XSLT hadden behoefte aan een manier om naar verschillende delen van een document te wijzen. XSLT moest het deel van het document dat getransformeerd zou worden, kunnen selecteren en XPointer moest twee documenten kunnen linken. Het voorstel was om een gemeenschappelijke syntax en semantiek te leveren die zowel door XSLT als door XPointer zou kunnen gebruikt worden. Deze nieuwe subset werd XPath [TRXPA], [Sectie 5.4.] genoemd en alhoewel XPath een subset is van XSLT, kan het ook op zichzelf gebruikt worden. XPath wordt verderop in dit hoofdstuk uitgebreid besproken.

5.3. XSL Transformaties

Bij een XSL tranformatie leest de XSLT processor een XML document en een bijhorende XSLT stylesheet. De processor genereert een nieuw document, zich baserend op de instructies die hij vindt in de XSLT stylesheet. Een XSLT stylesheet is zelf ook een XML document dat volgens een welbepaalde syntax is opgebouwd.

XSLT is in eerste instantie ontwikkeld voor XML-naar-XML en XML-naar-HTML transformaties, maar het is ook mogelijk andere omzettingen te doen (via een XSLT transformatie naar een tussenliggend formaat kan een omzetting naar bijvoorbeeld PostScript of PDF gebeuren [Hoofdstuk 7.]). XSLT kan in bepaalde gevallen ook gebruikt worden om SGML en HTML documenten te transformeren, maar dan wel op voorwaarde dat deze

documenten goed gevormd (well-formed) zijn.

Er bestaan drie verschillende werkwijzen om XML documenten naar andere formaten om te zetten met behulp van een XSLT stylesheet:

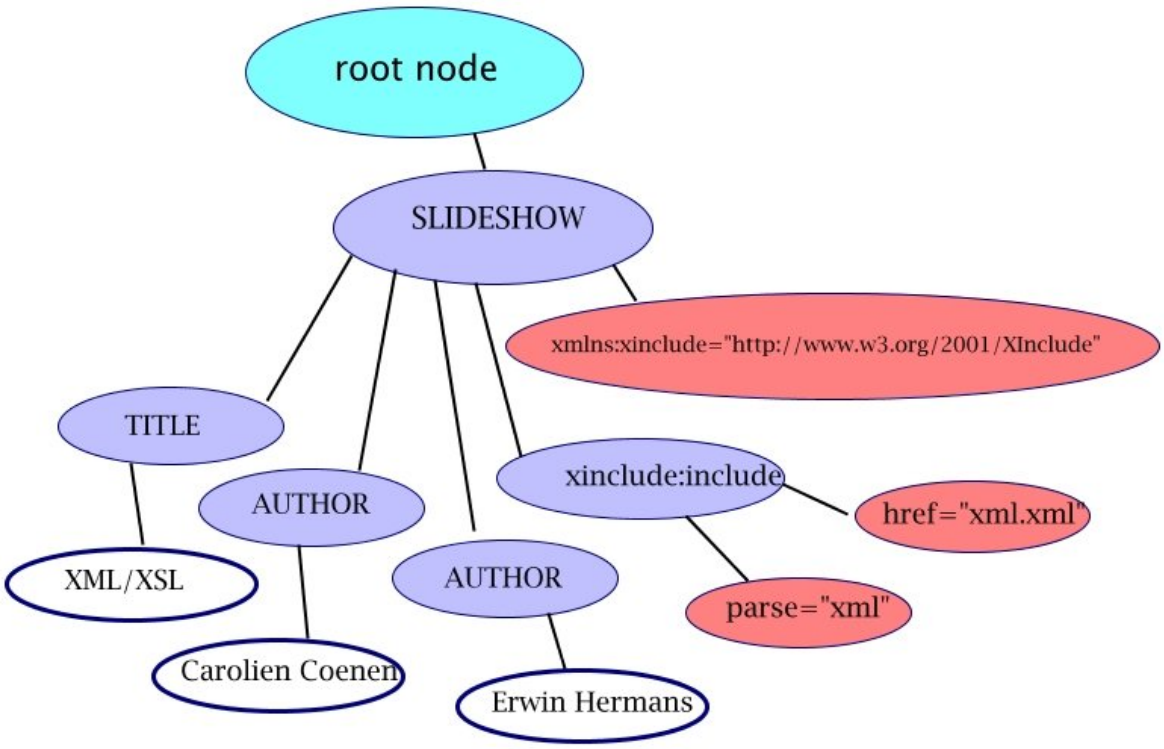
- Het XML document en de daarmee geassocieerde stylesheet worden beiden doorgegeven aan de client (web browser). De transformatie van het document gebeurt bij de client zoals gespecificeerd wordt door de stylesheet. Om een bepaalde stylesheet te verbinden aan een XML document moet de processing instruction `<?xml-stylesheet type="text/xml" href="URI van de stylesheet"?>` vlak na de XML declaratie toegevoegd worden.
- De server transformeert een XML document aan de hand van een XSLT stylesheet en zendt dit getransformeerde document naar de client. Voor deze manier van transformeren heb je speciale web server software nodig, zoals het web publishing framework, Cocoon, waarop wij later zullen terugkomen [Hoofdstuk 9.] .
- Een derde programma (bijvoorbeeld XALAN [XALAN], SAXON [SAXON] of een andere XSLT processor) transformeert het originele XML document in een ander formaat voordat het document op de server geplaatst wordt. Zowel de server als de client hebben alleen met het getransformeerde document te maken.

XSLT is verschillend van conventionele programmeertalen in die zin dat XSLT gebaseerd is op template regels die specificeren op welke wijze XML documenten verwerkt moeten worden. Een template regel dient om aan te geven welke output geproduceerd moet worden wanneer een bepaald patroon in het XML document wordt gematcht. In tegenstelling tot conventionele programmeertalen die vaak sequentieel zijn, kunnen deze template regels in eender welke volgorde geplaatst worden, omdat XSLT een declaratieve taal is. In een stylesheet worden al de templates verzameld die dienen om een bepaald document te transformeren.

Een XSLT processor modelleert een XML document als een boom (de bronboom van het document) die de volgende zeven soorten knooppunten kan bevatten:

- Het rootelement
- Elementen
- Tekst
- Attributen
- Namespaces
- Processing instructions
- Commentaar

Document Type Definities (DTD's), document type declaraties (dienen om aan te duiden welke DTD met een document geassocieerd wordt) en de XML declaratie worden niet als knooppunten aanzien en bijgevolg niet opgenomen in de boomstructuur waarmee de XSLT processor werkt. Op deze boom worden door de XSLT processor vervolgens allerlei bewerkingen (selecteren van knooppunten, ordenen van knooppunten, enzovoort) uitgevoerd. XSLT kan ook totaal nieuwe knooppunten aan het uitvoerdocument toevoegen of juist elementen weglaten uit de oorspronkelijke boomstructuur. Hieronder tonen wij een voorbeeldje van zulk een boom. Dit is hetzelfde voorbeeld als in het hoofdstuk over XML gebruikt werd [Sectie 2.3., Figuur 2.1.] . Bemerk dat in onderstaande versie inderdaad de XML declaratie is weggevallen.



Figuur 5.1. De boomstructuur waarmee de XSLT processor werkt

Het transformatieproces wordt vaak beschreven als volgt: XSLT wordt gebruikt om een XML bronboom te transformeren naar een resulterende boom of om een XML brondocument te transformeren naar een resulterend document (het uitvoerdocument). In het algemeen kan je zeggen dat de transformatie uit drie stappen bestaat: de invoer, enkele verwerkingsstappen en daarna de uitvoer. De te verwerken invoer wordt geselecteerd aan de hand van matchpatronen die een subset van de XPath expressies vormen. De elementen **xsl:apply-templates**, **xsl:for-each**, **xsl:if**, **xsl:choose**, **xsl:when** en **xsl:otherwise** kunnen gebruikt worden om de invoer verder te verwerken. De uitvoer die gegeneerd wordt, kan door een groot aantal elementen bepaald worden. We sommen er enkele op: **xsl:element**, **xsl:attribute**, **xsl:sort**, **xsl:number**, **xsl:copy-of**, enzovoort. Wij behandelen deze elementen in de volgende secties.

5.3.1. XSLT syntax

In deze paragraaf geven we een overzicht van de syntax van XSL transformaties. Dit overzicht is niet volledig, maar beperkt zich tot de, volgens ons, belangrijkste onderdelen. We beginnen met uit te leggen hoe een XSLT stylesheet eruit ziet [Sectie 5.3.1.1.] . De belangrijkste bouwstenen waaruit een stylesheet bestaat, zijn templates [Sectie 5.3.1.2.] . Templates zijn regels die op zoek gaan naar patronen om te matchen en vervolgens een bijhorende uitvoer genereren. Bij de uiteenzetting over templates bespreken wij ook uitvoerig de verschillende matchpatronen die kunnen voorkomen [Sectie 5.3.1.2.1.] . Dit is belangrijk omdat deze matchpatronen fundamenteel zijn om te kunnen werken met templates.

Na de uiteenzetting over templates, matchpatronen en het zeer belangrijke **xsl:apply-templates** element dat gebruikt wordt om templates recursief op te roepen [Sectie 5.3.1.2.2.] , bespreken wij hoe de waardes van knooppunten berekend worden [Sectie 5.3.1.3.] . We bespreken vervolgens de default template regels die gebruikt worden als er geen andere template regels voor handen zijn [Sectie 5.3.1.4.] .

Het is vaak belangrijk om nieuwe knooppunten aan je uitvoerdocument toe te voegen, hoe dit gebeurt bespreken we in de sectie over het toevoegen van markup aan de uitvoer [Sectie 5.3.1.5.] . Als je dezelfde inhoud meerdere keren op een verschillende manier wilt verwerken in je uitvoerdocument, moet je gebruik maken van modes [Sectie 5.3.1.6.] . Het is ook mogelijk om keuzes te maken gebaseerd op de inhoud van het invoerdocument, wij bespreken dit in [Sectie 5.3.1.7.] . Als je grote delen van je invoerdocument wilt behouden, kan het kopiëren van knooppunten zeer handig zijn [Sectie 5.3.1.8.] .

Wij wijden ook een sectie aan het nummeren van knooppunten [Sectie 5.3.1.9.] . In deze sectie wordt uiteengezet hoe je knooppunten automatisch kan laten tellen en ze een nummer geven. Het is ook mogelijk de knooppunten van je invoerdocument te sorteren volgens een bepaald criterium (bijvoorbeeld alfabetisch) in je uitvoerdocument. Dit bespreken wij in [Sectie 5.3.1.10.] . Door gebruik te maken van variabelen in je stylesheets is het mogelijk veel voorkomende codefragmenten maar één keer te moeten definiëren [Sectie 5.3.1.11.1.] . In de rest van de stylesheet kan dan altijd naar deze variabelen verwezen worden. Aansluitend bij het gebruik van variabelen behandelen wij templates met een naam [Sectie 5.3.1.11.2.] . Samen met variabelen en templates met een naam, behandelen wij ook het gebruik van parameters [Sectie 5.3.1.11.3.] . Variabelen, parameters en templates met een naam hebben veel raakpunten, vandaar dat wij ze samen behandelen.

Vervolgens bespreken wij hoe witruimte [Sectie 5.3.1.13.] verwerkt wordt. Een zeer belangrijk onderdeel van deze tekst met betrekking tot uitbreidbaarheid, is de behandeling van de verschillende manieren om meerdere stylesheets samen te voegen [Sectie 5.3.1.14.] . Er zijn twee verschillende elementen voorzien om dit te bewerkstelligen: **xsl:import** en **xsl:include**. We eindigen de sectie over XSLT syntax met het beschrijven en een aantal verschillende uitvoermethoden [Sectie 5.3.1.15.] en een voorbeeld van een uitgewerkte stylesheet [Sectie 5.3.2.] .

5.3.1.1. XSLT stylesheet

Een XSLT processor transformeert een XML document aan de hand van een XSLT stylesheet. Een XSLT stylesheet bestaat uit een aantal template regels. Deze regels bevatten een bepaald patroon waarnaar in het XML document gezocht moet worden. De XSLT processor doorloopt de opgebouwde boom van het XML document en vergelijkt elk knooppunt van de boom met het patroon van de template. Als dit patroon gevonden (gematcht) wordt, genereert de XSLT processor de output die in deze template regel gespecificeerd wordt.

Het rootelement van elk XSLT document ziet er als volgt uit:

Voorbeeld 5.1. Het rootelement van een XSLT document

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

In plaats van het **xsl:stylesheet** element kan je ook het **xsl:transform** element gebruiken, dit is een exact synoniem met dezelfde syntax, semantiek en attributen. De attributen **version** en **xmlns:xsl** zijn verplicht en moeten altijd de hierboven gegeven waardes hebben. Als je bijvoorbeeld een fout typt in de namespace URI, krijg je als uitvoer je stylesheet terug in plaats van het verwerkte XML document. Het is echter niet strikt noodzakelijk om het **xsl** voorvoegsel aan de *http://www.w3.org/1999/XSL/Transform* namespace te koppelen. De keuze van het voorvoegsel is volledig vrij. In onze tekst zullen wij verder altijd gebruik

maken van het **xsl** voorvoegsel.

In de eerste template regel van elke XSLT stylesheet moet het rootelement van het te transformeren XML document gematcht worden. Die eerste template moet er volgens de W3C standaarden dus als volgt uitzien:

Voorbeeld 5.2. Template volgens de W3C standaard

```
<xsl:template match="root_element">
  <!--Hier wordt beschreven hoe de gematchte knoop verwerkt zal worden
  in het uitvoerdocument-->
</xsl:template>
```

Opgelet echter, als je stylesheets wilt schrijven die bruikbaar zijn onder Internet Explorer [IE], dan moet deze stylesheet aan andere regels voldoen. Voor meer uitleg hierover verwijzen wij naar [Appendix B.] .

5.3.1.2. Templates

Template regels zijn het belangrijkste deel van een XSLT stylesheet. Ze associëren een bepaalde uitvoer met een bepaalde invoer. Elk **xsl:template** element heeft een **match** attribuut dat specificeert voor welke knoop van het invoerdocument de template geïnstantieerd wordt. De inhoud van het **xsl:template** element geeft aan welke uitvoer er gegenereerd moet worden als de template gematcht wordt.

Een template kan zowel gewoon tekst als XSLT instructies bevatten. De tekst in de template wordt letterlijk overgenomen in het uitvoerdocument. De XSLT instructies voeren bewerkingen uit op het invoerdocument en kopiëren het resultaat vervolgens naar het uitvoerdocument. Alle XSLT instructies zitten in de *http://www.w3.org/1999/XSL/Transform* namespace, zodat het gemakkelijk is het onderscheid te maken tussen elementen die letterlijk gekopieerd moeten worden naar de uitvoer en instructies. Deze laatste beginnen in ons geval dus altijd met **xsl:**.

Hieronder geven we een eenvoudig voorbeeld van een template regel. Als de XSLT processor bij het doorlopen van de boom van het XML document een knoop **HOMEPAGE** tegenkomt (de knoop wordt gematcht), dan wordt de tekst <b>Homepage:<b> naar het uitvoerdocument geschreven. Vervolgens wordt de inhoud van het **HOMEPAGE** element, geselecteerd door de **xsl:value-of** instructie naar de uitvoer geschreven. Het **xsl:value-of** wordt besproken in [Sectie 5.3.1.3.] .

Voorbeeld 5.3. Voorbeeld van een template regel

```
<xsl:template match="HOMEPAGE">
  <b> Homepage: </b>
  <xsl:value-of select="."/>
</xsl:template>
```

5.3.1.2.1. Matchpatronen

In deze sectie gaan we dieper in op het match attribuut dat bij elk **xsl:template** element hoort. Het match attribuut maakt gebruikt van een vrij complexe syntax om precies aan te duiden welke knooppunten gematcht moeten worden. Deze syntax is een subset van een vrij

krachtige syntax XPath, waar we later in deze tekst nog op zullen terugkomen. Wij overlopen nu een aantal veelgebruikte patronen die als matchpatronen kunnen voorkomen.

5.3.1.2.1.1. Matchen van het rootknooppunt

Het rootknooppunt is het element op het hoogste niveau waarvan de XML declaratie en het rootelement afhangen. Om het rootknooppunt te matchen, wordt aan het **match** attribuut de waarde "/" gegeven:

Voorbeeld 5.4. Matchen van het rootknooppunt

```
<xsl:template match="/">
  <!--Lichaam van de template-->
</xsl:template>
```

Zoals reeds eerder vermeld, is deze templateregel vooral belangrijk voor XSLT stylesheets bestemd om te gebruiken onder Internet Explorer. Deze regel overschrijft de default templateregel voor het rootknooppunt [Sectie 5.3.1.4.1.] .

5.3.1.2.1.2. Matchen van elementnamen

Het meest elementaire patroon bevat een enkele elementnaam en matcht alle elementen met die naam.

Voorbeeld 5.5. Matchen van elementnamen

```
<xsl:template match="SLIDE">
  <!--Lichaam van de template-->
</xsl:template>
```

5.3.1.2.1.3. Matchen met behulp van wild cards

Soms wil je dat één enkele template van toepassing is op meer dan één element. Je kan aangeven dat een template eender welk element matcht door gebruik te maken van een asterisk (*) wild card in plaats van een elementnaam in het **match** attribuut. De formulering zoals ze hier gehanteerd wordt, kan aanleiding geven tot verwarring. * matcht wel alle elementen, maar de processing moet dan ook tot bij die elementen geraken. Dus als er in het lichaam van de template geen **xsl:apply-templates** element staat, dan stopt de verwerking. Wij hebben dit zelf getest, omdat de formulering van de technische aanbeveling ons hierover gaan uitsluitsel gaf. Voor meer uitleg over het **xsl:apply-templates** element, verwijzen wij naar [Sectie 5.3.1.2.2.] .

In onderstaand voorbeeldje worden alle elementen gematcht. Als dit de enige template is in de stylesheet en er wordt niet door middel van **xsl:apply-templates** gezegd dat ook de kindelementen van het contextknooppunt (zie ook [Sectie 5.4.2.]) verwerkt moeten worden, dan wordt alleen het rootelement gematcht.

Voorbeeld 5.6. Matchen met behulp van de wild card

```
<xsl:template match="*">
  <!--Lichaam van de template-->
</xsl:template>
```

Stel nu dat je een aantal meer specifieke templates gedefinieerd hebt en dan al de overblijvende elementen (de elementen waarvoor geen expliciete templates voorzien zijn) wilt matchen met deze template. Dit levert geen enkele probleem op omdat in het geval dat twee templateregels allebei hetzelfde knooppunt matchen, bij default de meer specifieke regel de voorkeur krijgt.

Je kan ook alle elementen in een bepaalde namespace matchen door een namespace voorvoegsel voor de asterisk te plaatsen. Bijvoorbeeld:

Voorbeeld 5.7. Matchen met een bepaalde namespace voorvoegsel

```
<xsl:template match="svg:*">
   <!--Lichaam van de template-->
</xsl:template>
```

5.3.1.2.1.4. Matchen van kindknooppunten

De voorwaartse slash / kan gebruikt worden om specifieke hiërarchiëen van elementen te matchen. Wanneer het symbool / alleen voorkomt, refereert dit naar het rootknooppunt, zoals eerder vermeld. Als je het echter gebruikt tussen opeenvolgende elementnamen dan betekent dit dat het tweede vermelde element een kind van het eerste is. Als je bijvoorbeeld matcht op **SLIDE/LINE**, dan wordt er gezocht naar alle **LINE** elementen die kinderen zijn van een **SLIDE** element. Het is mogelijk om meerdere elementnamen achter elkaar te zetten, zodat je echt een deel van de boomstructuur kunt selecteren. Je kan ook gebruik maken van de wildcard om bepaalde elementen in de hiërarchie te vervangen. Zo refereert /* bijvoorbeeld naar het rootelement van het document, ongeacht wat dit is. Merk op dat dit niet hetzelfde is als het / symbool, dit verwijst immers naar het root**knooppunt**.

Voorbeeld 5.8. Matchen van kindknooppunten

```
<xsl:template match="SLIDE/LINE">
   <!--Lichaam van de template-->
</xsl:template>
```

5.3.1.2.1.5. Matchen van nakomelingen

Soms is het handig niet de volledige hiërarchie te moeten opsommen, maar gewoon aan te kunnen geven dat een element een nakomeling is van een ander element, gesitueerd op een willekeurig niveau in de boomstructuur. Om dit te bewerkstelligen wordt de dubbele slash // gebruikt. In onderstaand voorbeeld worden alle **LINE** elementen gematcht die een kind of een kleinkind of een achterkleinkind enzovoort zijn van een **SLIDE** element. Als de dubbele slash aan het begin van een patroon voorkomt, wordt eender welke nakomeling van de root node met die naam geselecteerd.

Voorbeeld 5.9. Matchen van nakomelingen

```
<xsl:template match="SLIDE//LINE">
   <!--Lichaam van de template-->
</xsl:template>
```

5.3.1.2.1.6. Matchen op ID

Soms wil je een speciale vormgeving opleggen aan één enkel element zonder al de andere elementen van hetzelfde type te veranderen. De eenvoudigste manier om dat te doen in XSLT is om de vormgeving te verbinden aan het **ID**-type attribuut van dat element. Dit kan gedaan worden met behulp van de **id()** functie die het **ID**-type attribuut van het element selecteert. De **id()** functie heeft als argument de waarde van het **ID**-type attribuut tussen afkappingstekens geplaatst. De onderstaande template zorgt er bijvoorbeeld voor dat het element met **ID** a85 cursief wordt gezet.

Voorbeeld 5.10. Matchen op ID

```
<xsl:template match="id('a85')">
   <I>
      <xsl:value-of select="."/>
   </I>
</xsl:template>
```

Bij het gebruik van deze functie, gaan we er natuurlijk vanuit dat het element dat je op deze manier wilt selecteren een uniek gedefinieerd **ID**-type attribuut heeft in de DTD van het brondocument. Het is heel waarschijnlijk dat dit echter niet het geval is. Veel documenten hebben bijvoorbeeld geen DTD's. Ze zijn alleen goedgevormd, niet geldig. En zelfs als het document een DTD heeft, is er geen garantie dat een element een **ID**-type attribuut heeft.

5.3.1.2.1.7. Matchen van attributen

Om een attribuut te matchen laat je gewoon de naam van het attribuut dat je wilt selecteren voorafgaan door het @ teken.

Voorbeeld 5.11. Matchen van attributen

```
<xsl:template match="@UNITS">
   <I>
      <xsl:value-of select="."/>
   </I>
</xsl:template>
```

Opgelet echter, door gewoon de bovenstaande template aan de stylesheet toe te voegen, zullen de **UNITS** in het invoerdocument niet automatisch tussen italic tags in het uitvoerdocument verschijnen. Dit komt omdat attributen geen kinderen zijn van de elementen waartoe zij behoren. Als een XSLT processor de boom overloopt ziet hij namelijk geen attribuutknooppunten. Om er voor te zorgen dat bovenstaande template toegepast wordt en alle **UNITS** attributen tussen italic tags in het uitvoerdocument verschijnen, moet je expliciet aangeven dat het **UNITS** attribuut verwerkt moet worden. Je kan deze template oproepen door aan het het **select** attribuut van het **xsl:apply-templates** element de waarde **@UNITS** te geven. Voor meer uitleg over het **xsl:apply-templates** element verwijzen wij naar [Sectie 5.3.1.2.2.] .

Onderstaand voorbeeld illustreert hoe expliciet wordt aangegeven dat de attributen verwerkt moeten worden.

Voorbeeld 5.12. Expliciet selecteren van attributen om ze te kunnen verwerken

```
<xsl:template match="MELTINGPOINT">
   <xsl:value-of select="."/>
   <xsl:apply-templates select="@UNITS"/>
</xsl:template>
<xsl:template match="@UNITS">
   <I>
      <xsl:value-of select="."/>
   </I>
</xsl:template>
```

Het is altijd mogelijk attributen met elementen te combineren door gebruik te maken van de verschillende hiërarchie-operatoren en de wild card. In het volgende voorbeeld worden alle **UNITS** attributen van een kind van een **ATOM** element gematcht.

Voorbeeld 5.13. Gebruik van de wildcard om attributen te matchen

```
<xsl:template match="ATOM/*/@UNITS">
   <!--Lichaam van de Template-->
</xsl:template>
```

5.3.1.2.1.8. Matchen van processing instructions

De **processing-instruction** functie matcht processing instructions. Deze functie is één van de vier mogelijke knooppunttesten die wij opsommen in de sectie over XPath [Sectie 5.4.3.1.2.] . Functies zijn een onderdeel van XPath. Het argument dat aan **processing-instruction** wordt meegegeven is een string tussen aanhalingstekens die het doel aangeeft van de te selecteren processing instruction. Het is ook mogelijk de **processing-instruction** functie te gebruiken zonder argumenten. Twee voorbeelden:

Voorbeeld 5.14. Matchen van processing instructions

```
<xsl:template match="/processing-instruction()">
   <xsl:processing-instruction name="xml-stylesheet"> type="text/xml"
   href="stylesheet.xsl" </xsl:processing-instruction>
</xsl:template>
```

Deze template matcht alle processing instruction kinderen van het rootknooppunt. Deze template vertrekt van de veronderstelling dat de enige processing instruction in het brondocument hoogstwaarschijnlijk een **xml-stylesheet** processing instruction is. Als er een andere processing instruction gematcht wordt, is de gegenereerde uitvoer niet meer in overeenstemming met de uitvoer. Zoals verder zal blijken in het hoofdstuk over Cocoon2 [Sectie 9.2.] , is het mogelijk de **xml-stylesheet** instructie te vermijden, wat leidt tot makkelijker te onderhouden en beter te begrijpen XML documenten. Het **xsl:processing-instruction** element in bovenstaand voorbeeld voegt een processing instruction met de gespecificeerde naam en waarde toe aan het uitvoerdocument. We komen hier op terug in [Sectie 5.3.1.5.4.] . In het volgende voorbeeld wordt weer de **xml-stylesheet** processing instruction gematcht, maar ditmaal aan de hand van zijn naam.

Voorbeeld 5.15. Matchen op de naam van een processing instruction

```
<xsl:template match="processing-instruction('xml-stylesheet')">
   <xsl:processing-instruction name="xml-stylesheet">
      <xsl:value-of select="."/>
```

```
    </xsl:processing-instruction>
  </xsl:template>
```

5.3.1.2.1.9. De of-operator

Het vertikale streepje | laat toe te matchen op meerdere patronen. Als één of meer van de patronen gematcht worden, zal de template geactiveerd worden.

Voorbeeld 5.16. De of-operator

```
  <xsl:template match="TEXT|XMLEXAMPLE">
    <xsl:apply-templates/>
  </xsl:template>
```

Bij gecombineerd gebruik van de hiërarchie-operator en de of-operator, heeft de hiërarchie-operator "/" voorrang op de of-operator "|".

Voorbeeld 5.17. gecombineerd gebruik van / en |

```
  <xsl:template match="LINE/TEXT | XMLEXAMPLE">
    <xsl:apply-templates/>
  </xsl:template>
```

Dit wil zeggen dat in bovenstaand voorbeeld gematcht wordt op een **TEXT** kind van **LINE** of op een **XMLEXAMPLE** kind van een ongespecificeerde knooppunt, dat niet noodzakelijk een **LINE** knooppunt moet zijn.

5.3.1.2.1.10. Predikaten

Tot nu toe hebben we alleen voorbeelden behandeld waarin gematcht werd op bepaalde knooppunten. Het is ook mogelijk om de te matchen patronen te verfijnen met behulp van predikaten. Predikaten onderwerpen de geselecteerde knooppunten aan een test. De test wordt tussen vierkante haken [] achter het knooppunt dat je wenst te testen geplaatst. Er zijn veel verschillende tests mogelijk, onder andere:

- Of een element een bepaald kind, attribuut of ander knooppunt bevat
- Of de waarde van een attribuut een bepaalde string is
- Of de waarde van een element overeenkomt met een bepaalde string
- Welke positie een gegeven knoop bezet in de hiërarchie

In dit voorbeeld worden alleen de **LINE** elementen gematcht die een **TEXT** knooppunt als kind hebben:

Voorbeeld 5.18. Gebruik van knooppunttesten

```
  <xsl:template match="LINE[TEXT]">
    <!--Lichaam van de template-->
  </xsl:template>
```

Merk op dat hier wel degelijk het **LINE** element gematcht wordt en niet het **TEXT** knooppunt zoals dat het geval is bij de uitdrukking: **LINE/TEXT**.

De test haakjes kunnen veel meer bevatten dan gewoon de naam van een kindelement. In feite kunnen ze eender welke XPath expressie bevatten. (XPath expressies zijn een superset van de matchpatronen, die we in een aparte sectie van deze tekst bespreken.) Als het gespecificeerde element een kind heeft dat overeenkomt met de test-expressie, dan matcht dit element het hele patroon. In het volgende voorbeeld worden alle **ATOM** elementen die een **DENSITY** kind element hebben met een **UNITS** attribuut gematcht:

Voorbeeld 5.19. Testpatroon met een XPath expressie

```
<xsl:template match="ATOM[DENSITY/@UNITS]">
   <!--Lichaam van de template-->
</xsl:template>
```

Een testpatroon dat zeer nuttig kan zijn, is de test op stringgelijkheid. Een gelijkheidsteken kan testen of de inhoud van een knoop overeenkomt met een gegeven string. Onderstaande template matcht het **SLIDE** element dat een **TITLE** kindelement bevat waarvan de inhoud de string "XSLT: What's it good for?" is.

Voorbeeld 5.20. Testen op stringgelijkheid

```
<xsl:template match="SLIDE[TITLE='XSLT: What's it good for?']">
   <!--Lichaam van de template-->
</xsl:template>
```

Het is vrij moeilijk om de stringgelijkheid van de inhoud van een element te testen, omdat je de waarde waarmee je vergelijkt exact juist moet hebben, dat wil zeggen dat je ook de witruimte in dat element juist moet hebben. Het is eenvoudiger om attribuutwaardes te testen, omdat deze minder kans hebben op het bevatten van onbetekenende witruimte.

Voor meer uitleg over het gebruik van predikaten bij XPath expressies verwijzen wij naar [Sectie 5.4.3.1.3.] .

5.3.1.2.1.11. Matchen van commentaar

Meestal wordt commentaar in XML documenten gewoon genegeerd, omdat over het algemeen wordt aangenomen dat het een slecht idee is om van commentaar een essentieel onderdeel van je document te maken. Toch is er in de standaard een manier voorzien om te matchen op commentaar, met name de **comment()** knooppuntstest [Sectie 5.4.3.1.2.] . Ondanks het gebruik van haakjes bij deze functie, worden er nooit argumenten aan **comment()** meegegeven. Ook dit patroon kan in combinatie met de hiërarchie-operatoren gebruikt worden.

Voorbeeld 5.21. Matchen van commentaar

```
<xsl:template match="SLIDE/comment()">
   <I>
      <xsl:value-of select="."/>
   </I>
</xsl:template>
```

Wij willen nogmaals benadrukken dat je beter nooit belangrijke informatie in commentaar zet. De enige reden dat de XSLT standaard toestaat commentaar te selecteren is om bij de transformatie van één XML document naar een ander de commentaar intact te houden:

Voorbeeld 5.22. Het behouden van de oorspronkelijke commentaar in het uitvoerdocument

```
<xsl:template match="comment()">
   <xsl:comment>
      <xsl:value-of select="."/>
   </xsl:comment>
</xsl:template>
```

5.3.1.2.1.12. Matchen van tekstknooppunten

Tekstknooppunten worden meestal genegeerd als knooppunten, alhoewel hun waardes wel opgenomen worden als onderdeel van de waarde van een geselecteerd element. Een tekstknooppunt komt overeen met de gewone tekstuele inhoud van een element. Met de **text()** operator is het mogelijk om het tekstkind van een element te selecteren. Deze operator heeft, net zoals **comment()** geen argumenten. De belangrijkste bestaansreden voor deze operator is dat deze gebruikt wordt voor de defaultregels. Verderop in deze tekst zullen we beschrijven wat die defaultregels juist inhouden [Sectie 5.3.1.4.] . De template in onderstaand voorbeeld zorgt er voor dat alle tekst in een XML document tussen italic tags komt te staan.

Voorbeeld 5.23. Matchen van tekstknooppunten

```
<xsl:template match="text()">
   <I>
      <xsl:value-of select="."/>
   </I>
</xsl:template>
```

5.3.1.2.2. Het xsl:apply-templates element

In de voorgaande paragrafen hebben wij behandeld hoe de invoer gematcht moet worden. In deze sectie gaan we dieper in op hoe deze gematchte invoer vervolgens verwerkt moet worden. Het **xsl:apply-templates** element zorgt hiervoor.

Meestal wil je dat als een bepaald knooppunt gematcht wordt ook de kinderen van dat knooppunt verwerkt worden om een bepaalde inhoud in het uitvoerdocument te genereren. Het element dat dit realiseert is **xsl:apply-templates**. Dit element zorgt ervoor dat je de knopen van je invoerdocument recursief kan doorlopen. Als je het **xsl:apply-templates** element opneemt in de uitvoertemplate, wordt elk kindelement van het gematchte bronelement op zijn beurt vergeleken met de templates in je stylesheet tot er een match wordt gevonden. De template voor het knooppunt dat gematcht wordt, kan op zijn beurt weer **xsl:apply-templates** elementen bevatten om te zoeken naar matches voor zijn kinderen enzovoort. Bij het verwerken van de knooppunten wordt elke knooppunt behandeld als een volledige boom. Dit is het voordeel van de boomstructuur van het document: elk deel kan op dezelfde manier behandeld worden als het geheel. Een voorbeeldje van het gebruik van **xsl:apply-templates**:

Voorbeeld 5.24. Illustratie van het gebruik van xsl:apply-templates

```
<xsl:template match="SLIDESHOW">
   <html>
      <body>
         <xsl:apply-templates/>
```

```
      </body>
   </html>
</xsl:template>
```

5.3.1.2.2.1. Het select attribuut

Door aan het **xsl:apply-templates** element een **select** attribuut mee te geven, kan je aanduiden dat je in plaats van alle kinderen van een gematcht knooppunt slechts een deel van zijn kinderen wilt selecteren. Het select attribuut gebruikt dezelfde soort patronen als het **match** attribuut van het **xsl:template** element (zie hoger). Als er geen **select** attribuut aanwezig is, worden alle kind elementen, tekst, commentaar en processing instructions geselecteerd. Attributen en namespace knooppunten (de attributen die gebruikt worden voor namespace declaratie, zie ook [Sectie 2.4.]) worden niet geselecteerd. Een voorbeeld:

Voorbeeld 5.25. Het select attribuut van xsl:apply-templates

```
<xsl:template match="SLIDESHOW">
   <html>
      <body>
         <xsl:apply-templates select="SLIDE"/>
      </body>
   </html>
</xsl:template>
```

In bovenstaand voorbeeld is **SLIDE** een relatief pad ten opzicht van het contextknooppunt. Voor meer uitleg over de context verwijzen wij naar [Sectie 5.4.2.] . Een absoluut pad is een pad dat vertrekt vanaf het rootknooppunt van het document. Een absoluut pad begint altijd met /.

Wanneer in je XSLT stylesheet een template voorkomt waarvan het lichaam een aantal keren het **xsl:apply-templates** element bevat met telkens een ander **select** attribuut, dan bepaalt alleen de volgorde waarin de elementen door het **xsl:apply-templates** element geselecteerd worden, de uiteindelijke volgorde van deze elementen in het uitvoerdocument. De volgorde waarin deze elementen in het invoerdocument staan is dus niet van belang. Als éénzelfde template meer dan één element matcht, dan verschijnen die elementen in dezelfde volgorde in het uitvoerdocument als die waarin ze in het invoerdocument staan.

Stel dat je het volgende XML document hebt:

Voorbeeld 5.26. Eenvoudig document BOOK

```
<BOOK>
   <AUTHOR>
      <FIRSTNAME> Jan </FIRSTNAME>
      <SURNAME> Dockx </SURNAME>
   </AUTHOR>
   <AUTHOR>
      <FIRSTNAME> Kristof </FIRSTNAME>
      <SURNAME> Mertens </SURNAME>
   </AUTHOR>
   <AUTHOR>
      <FIRSTNAME> Nele </FIRSTNAME>
      <SURNAME> Smeets </SURNAME>
   </AUTHOR>
```

```
</BOOK>
```

En we passen daarop de volgende stylesheet toe:

Voorbeeld 5.27. Stylesheet toe te passen op BOOK

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:template match="BOOK">
        <html>
           <body>
              <xsl:apply-templates select="AUTHOR"/>
           </body>
        </html>
     </xsl:template>
     <xsl:template match="AUTHOR">
        <xsl:apply-templates select="SURNAME"/>
        <xsl:apply-templates select="FIRSTNAME"/>
        <BR/>
     </xsl:template>
     <xsl:template match="FIRSTNAME">
        <xsl:value-of select="."/>
     </xsl:template>
     <xsl:template match="SURNAME">
        <xsl:value-of select="."/>
     </xsl:template>
  </xsl:stylesheet>
```

Het **SURNAME** element wordt eerst geselecteerd door **xsl:apply-templates**. De uitvoer van deze template komt dus eerst in het uitvoerdocument. Het **FIRSTNAME** element wordt als tweede geselecteerd door **xsl:apply-templates**. De uitvoer van deze template komt dus na de uitvoer van de template die **SURNAME** matcht in het uitvoerdocument. De **AUTHOR** elementen komen in de uitvoer terecht in dezelfde volgorde als die waarin ze in het invoerdocument staan. De uitvoer van deze template ziet er als volgt uit:

```
Dockx Jan
Mertens Kristof
Smeets Nele
```

Het **xsl:apply-templates** element wordt gewoonlijk gebruikt om knooppunten te verwerken die kinderen zijn van het context knooppunt. Als het **xsl:apply-templates** element op deze manier gebruikt wordt, kunnen er geen oneindige lussen optreden. Het is echter mogelijk om **xsl:apply-templates** elementen te laten verwerken die geen nakomelingen zijn van het context knooppunt. In dit geval is het wel mogelijk dat er oneindige lussen ontstaan. In het onderstaande voorbeeld is dit het geval.

Voorbeeld 5.28. Voorbeeld van een oneindige lus

```
  <xsl:template match="THESIS">
     <xsl:apply-templates select="."/>
  </xsl:template>
```

De XSLT processor kan zo geïmplenteerd zijn dat hij in staat is zulke lussen te ontdekken, maar de mogelijkheid bestaat dat een stylesheet in een oneindige lus terecht komt die niet

ontdenkt kan worden. We merken op dat dit feit een niet onbelangrijk risico inhoudt.

5.3.1.2.3. Imperatief of declaratief?

Er bestaan twee manieren om meerdere elementen na mekaar te verwerken. De eerste methode maakt gebruik van **xsl:apply-templates** en hebben we al eerder behandeld. De tweede methode maakt gebruikt van het **xsl:for-each** element. Het **xsl:for-each** element verwerkt achtereenvolgens ieder element dat overeenkomt met zijn **select** attribuut. Een template die **for-each** gebruikt kan meestal omgezet worden naar twee equivalente templates waarbij één van beiden een **xsl:apply-templates** element bevat. Onderstaande voorbeelden illustreren dit:

Voorbeeld 5.29. Gebruik van xsl:for-each

```
<xsl:template match="SLIDESHOW">
   <html>
      <body>
         <xsl:for-each select="AUTHOR">
            <B> Author: </B>
            <xsl:value-of select="."/>
         </xsl:for-each>
      </body>
   </html>
</xsl:template>
```

Voorbeeld 5.30. Gebruik van xsl:apply-templates

```
<xsl:template match="SLIDESHOW">
   <html>
      <body>
         <xsl:apply-templates select="AUTHOR"/>
      </body>
   </html>
</xsl:template>
<xsl:template match="AUTHOR">
   <B> Author: </B>
   <xsl:value-of select="."/>
</xsl:template>
```

We kunnen hier parallellen trekken met declaratieve en imperatieve programmeertalen. Daar waar **xsl:for-each** staat voor de imperatieve manier van programmeren, staat **xsl:apply-templates** voor de declaratieve manier van werken. De voorkeur van de meeste auteurs die wij geconsulteerd hebben, gaat uit naar de declaratieve manier van werken en wij hebben dan ook geprobeerd het gebruik van **xsl:for-each** in onze templates tot een minimum te beperken.

Een andere manier om naar het verschil tussen **xsl:apply-templates** en **xsl:for-each** te kijken, is de volgende. Bij **xsl:for-each** kan je zeggen dat het document op een iteratieve manier doorlopen wordt. Alle elementen die aan het **select** attribuut voldoen worden achtereenvolgens verwerkt, tot er zo geen element meer te vinden is. Bij het **xsl:apply-templates** element wordt er duidelijk gebruik gemaakt van recursiviteit. Binnen een template wordt recursief eenandere template opgeroepen.

In de literatuur wordt het onderscheid dat wij hier belicht hebben, ook uitgedrukt aan de hand van de termen Push en Pull.

5.3.1.3. Het berekenen van waardes

Het **xsl:value-of** element berekent de waarde van iets (meestal is dit iets in het invoerdocument, maar dit hoeft niet altijd zo te zijn) en kopieert dit vervolgens naar het uitvoerdocument. Het **select** attribuut van het **xsl:value-of** element specificeert exact welke waarde berekend moet worden. Bijvoorbeeld:

Voorbeeld 5.31. Illustratie van het gebruik van xsl:value-of

```
<xsl:template match="AUTHOR">
   <B> First Name Author: </B>
   <xsl:value-of select="FIRSTNAME"/>
</xsl:template>
```

In het uitvoerdocument komt na `<B> Author: </B>` de waarde van de **FIRSTNAME** knoop te staan. Het knooppunt waarvan de waarde wordt geselecteerd, hier de **FIRSTNAME** knoop, is relatief ten opzichte van de huidige knoop. De huidige knoop is deze die gematcht wordt door de template, hier het **AUTHOR** element.

De waarde van een knoop is altijd een string, eventueel een lege string. De exacte inhoud van deze string hangt af van het knooppunttype. Het knooppunttype dat het meest voorkomt, is een element en de waarde van een elementknooppunt is simpelweg de aaneenschakeling van al de tekens (uitgezonderd de markup) tussen de start- en eindtag van een element. Een kort voorbeeldje om dit te verduidelijken:

Voorbeeld 5.32.

```
<SLIDE>
   <!--Dit is een willekeurige slide uit een presentatie-->
   <TITLE> XML: What's it good for? </TITLE>
   <LINE>
      <TEXT> Markup: Structure your documents </TEXT>
   </LINE>
   <LINE>
      <TEXT> Presentation versus content </TEXT>
   </LINE>
   <LINE>
      <TEXT> Designed for the web </TEXT>
   </LINE>
   <LINE>
      <TEXT> Easier exchange of information </TEXT>
   </LINE>
</SLIDE>
```

De waarde van het **SLIDE** element ziet er als volgt uit: `

```
XML: What's it good for?
   Markup: Structure your documents
   Presentation versus content
   Designed for the web
   Easier exchange of information
```

De tags en de commentaren zijn verdwenen. Al de rest (de inhoud van de subelementen van **SLIDE** én de oorspronkelijke witruimte, waaronder de end-of-lines) werd behouden.

De waardes van de zes andere knooppunttypes worden op een gelijkaardige manier berekend. Onderstaande tabel geeft een overzicht:

Tabel 5.1. Waardes van knooppunten

knooppunttype	Waarde
Root	De waarde van het rootelement
Element	De aaneenschakeling van alle verwerkte (geparste) tekengegevens (geen markup!) die het element bevat, inclusief tekengegevens in de nakomelingen van het element
Tekst	De tekst van het knooppunt; eigenlijk de knoop zelf
Attribuut	De genormaliseerde attribuutwaarde, zoals gespecificeerd in de XML 1.0 technische aanbeveling van het W3C (dit komt neer op de attribuut waarde, nadat entiteiten vervangen zijn en witruimte aan het begin en einde verwijderd is), zonder de naam van het attribuut, het gelijkheidsteken en de aanhalingstekens
Namespace	De URI van de namespace
Processing instruction	De data in de processing instruction, zonder de processing instruction zelf en zonder <? en ?>
Commentaar	De tekst van de commentaar, zonder <-- en -->

5.3.1.4. De default template regels

Het vraagt vaak vrij veel werk om heel de hiërarchie van een XML document te vertalen naar templates in de XSLT stylesheet. Dit is vooral zo als het document in kwestie geen vaste, voorspelbare orde volgt, maar, zoals vele webpagina's, elementen bijna willekeurig door mekaar gebruikt. In zulke gevallen is het handig algemene regels te hebben die een element kunnen vinden en daarop templates toepassen, ongeacht de plaats waarop dit element zich bevindt in het invoerdocument.

XSLT voorziet verscheidene default template regels die impliciet in alle stylesheets opgenomen worden. De **eerste default regel** matcht het rootknooppunt en de element knooppunten en past templates toe op alle kindknooppunten. De **tweede default regel** matcht tekstknooppunten en attributen door hun waardes te kopiëren naar de uitvoer. De eerste en de tweede default regel zorgen ervoor dat zelfs een stylesheet met daarin één leeg **xsl:stylesheet** element altijd de onbewerkte data van het invoerdocument als uitvoer zal geven. Hieronder bespreken we de voorgedefinieerde templateregels meer in detail.

5.3.1.4.1. Regels voor elementen en het rootknooppunt

De **eerste default regel** ziet er als volgt uit:

Voorbeeld 5.33. Default template regel voor elementen en root

```
<xsl:template match="*|/">
   <xsl:apply-templates/>
</xsl:template>
```

Deze default regel zorgt ervoor dat alle elementen recursief verwerkt worden zelfs als ze niet gematcht worden door de expliciete template regels. Dat wil zeggen dat, tenzij er een template regel bestaat die deze regel overschrijft, alle elementknooppunten verwerkt zullen worden. Deze regel matcht immers al de elementen in een document en gaat vervolgens op zoek naar template regels om toe te passen op de kinderen van deze elementen. Deze kinderen worden door de default template gematcht en vervolgens wordt weer **xsl:apply-templates** toegepast

op hun kinderen, enzovoort. Tenzij deze regel overschreven wordt, worden dus alle elementknooppunten verwerkt.

Opgelet, als er een expliciete regel voor een ouder van een element aanwezig is, wordt deze regel niet meer geactiveerd voor de kinderen van dat element, tenzij de template regel voor de ouder een **xsl:apply-templates** kind bevat. Als je bijvoorbeeld alleen het rootknooppunt matcht en de kinderen verder niet meer verwerkt met behulp van **xsl:apply-templates** of **xsl:for-each**, stopt de verwerking van het document en wordt er geen uitvoer meer gegenereerd.

Voorbeeld 5.34.

```
<xsl:template match="/"/>
```

5.3.1.4.2. Regels voor tekstknooppunten en attributen

Voorbeeld 5.35. Default template regel voor tekstknooppunten en attributen

```
<xsl:template match="text()|@*">
   <xsl:apply-templates/>
</xsl:template>
```

De **tweede default template regel** voor tekst- en attribuutknooppunten matcht alle tekst- en attribuutknooppunten en kopieert de waarde van deze knooppunten naar het uitvoerdocument. Deze regel zorgt ervoor dat tenminste de tekst van een element verschijnt in het uitvoerdocument als er geen regel bestaat die deze specifiek matcht. Een andere regel kan de default regel overschrijven voor elementen waarvoor je meer of minder uitvoer wilt, dan alleen de tekstuele inhoud van een element.

Deze regel kopieert ook de attribuutwaardes (alleen de waardes, niet hun bijhorende namen) naar de uitvoer, maar maakt er wel gewone tekst van in het uitvoerdocument. Omdat er geen default template regel is die templates toepast op attributen, zal deze regel voor attributen nooit geactiveerd worden, tenzij je specifiek een niet-default regel toevoegt ergens in je stylesheet die uitdrukkelijk templates toepast op de attributen van één of meerdere elementen. Zie ook [Sectie 5.3.1.2.1.7.] .

5.3.1.4.3. Regels voor processing instructions en commentaren

De default regel voor processing instructions en commentaar is heel eenvoudig: de processing instructions en commentaren worden gewoon uit de uitvoer weggelaten alsof ze niet bestaan.

Voorbeeld 5.36. Default template regel voor processing instructions en commentaren

```
<xsl:template match="processing-instruction()|comment()"/>
```

Samen zorgen deze regels ervoor dat een lege stylesheet gewoon de inhoud van de elementen kopieert van de invoer naar de uitvoer, waarbij alle markup wordt weggelaten. Het is belangrijk te onthouden dat het hier over regels van lage prioriteit gaat en dat dus elke andere match voorrang heeft op de defaultregels.

5.3.1.5. Toevoegen van markup aan de uitvoer

Beslissingen over welke markup op te nemen in het uitvoerdocument wil je vaak laten afhangen van de elementen en attributen in het invoerdocument. Je wilt bijvoorbeeld de inhoud van een **FILENAME** element (de naam van de file) uit de invoer veranderen in een link naar deze file in je uitvoerdocument. Dit doe je door de waarde van je **FILENAME** element toe te kennen aan het **HREF** attribuut van een **A** element in de uitvoer. Of je wilt één elementtype in de invoer vervangen door verscheidene elementtypes in de uitvoer afhankelijk van de attribuutwaarde van dat element in het invoerdocument. Om dit te doen, zijn er de elementen **xsl:element**, **xsl:attribute**, **xsl:processing-instruction**, **xsl:comment** en **xsl:text** voorzien. In de inhoud van de opgesomde elementen worden XSLT instructies gebruikt om de uitvoer van de elementen te bepalen. Om de attribuutwaardes voor de attributen bij de opgesomde elementen te specificeren worden attribuutwaarde templates gebruikt. [Sectie 5.3.1.5.5.]

5.3.1.5.1. Elementen toevoegen

Elementen worden meestal opgenomen in het uitvoerdocument door gewoon de letterlijke start- en eindtag in de inhoud van de template te zetten. Als je bijvoorbeeld een **b** element in de uitvoer wil krijgen, voeg je gewoon en toe op de juiste plaats in je stylesheet. Soms wil je echter details van het invoerdocument gebruiken om te bepalen welk element in het uitvoerdocument geplaatst moet worden. Dit kan je doen met behulp van het **xsl:element** element dat een element toevoegt aan het uitvoerdocument. De naam van het element wordt gegeven door een attribuutwaarde template in het **name** attribuut van **xsl:element**. De inhoud van het element wordt afgeleid uit de inhoud van het **xsl:element** element, dat op zijn beurt **xsl:attribute**, **xsl:processing-instruction** en **xsl:comment** elementen kan bevatten.

In onderstaand voorbeeld wordt geïllustreerd hoe het **xsl:element** element gebruikt wordt om een element toe te voegen aan het uitvoerdocument. Wij maken hierbij ook gebruik van het **xsl:attribute** element dat we in de volgende sectie behandelen.

Voorbeeld 5.37. Toevoegen van elementen

```
<xsl:template match="URL">
   <xsl:element name="a">
      <xsl:attribute name="href">
         <xsl:value-of select="."/>
      </xsl:attribute>
   </xsl:element>
</xsl:template>
```

5.3.1.5.2. Attributen toevoegen

Je kan attributen toevoegen aan je uitvoerdocument door eenvoudigweg de letterlijke attributen zelf toe te voegen aan je template. Je kan bijvoorbeeld een **DIV** element dat een **ALIGN** attribuut heeft met de waarde **CENTER** toevoegen door gewoon `<DIV ALIGN="CENTER"/>` op de juiste plaats in je template te zetten. Toch zal je, net zoals bij het toevoegen van elementen, vaak rekening moeten houden met gegevens die je leest uit het invoerdocument om te bepalen welke waarde je attribuut zal krijgen en soms zelfs om te bepalen welke de naam van je attribuut is. Een voorbeeldje:

Voorbeeld 5.38. Toevoegen van attributen

```
<xsl:template match="URL">
   <a>
      <xsl:attribute name="href">
         <xsl:value-of select="."/>
      </xsl:attribute>
   </a>
</xsl:template>
```

Merk op dat dit hetzelfde voorbeeld is als we gebruikt hebben om het toevoegen van elementen te illustreren. Alleen is in dit voorbeeld de uitgebreide schrijfwijze voor het toevoegen van elementen vervangen door de kortere versie waarin het element <a> gewoon letterlijk wordt opgenomen in de template.

Alle **xsl:attribute** elementen moeten staan vóór elke andere inhoud van hun ouderelement. Het is dus verboden om een attribuut toe te voegen aan een element nadat je al begonnen bent met het uitschrijven van de inhoud van dat element.

5.3.1.5.3. Definiëren van attribuutsets

Het komt vaak voor dat je dezelfde groep van attributen wilt toepassen op veel verschillende elementen van hetzelfde of een verschillend elementtype. Om dit gemakkelijker te maken, kan je met behulp van het **xsl:attribute-set** element één of meer attributen als leden van een attribuutset definiëren op het hoogste niveau van de stylesheet. Deze attribuutset kan je dan gebruiken in een element met een **xsl:use-attribute-sets** attribuut.

We illustreren dit aan de hand van een voorbeeldje. Het **xsl:attribute-set** element definieert een set van attributen met de naam **cellstyle** dat de volgende attributen bevat: **font-family** attribuut met als waarde **New York, Times New Roman, Times, serif** en een **font-size** attribuut met als waarde **12pt.**.

Voorbeeld 5.39. Definiëren van een attribuutset

```
<xsl:attribute-set name="cellstyle">
   <xsl:attribute name="font-family"> New York, Times New Roman, Times,
   serif </xsl:attribute>
   <xsl:attribute name="font-size"> 12pt. </xsl:attribute>
</xsl:attribute-set>
```

Onderstaande template regel past nu deze attributen toe op de **td** elementen van het uitvoerdocument.

Voorbeeld 5.40. Gebruik van attribuutsets in een element

```
<xsl:template match="DAY">
   <tr>
      <td xsl:use-attribute-sets="cellstyle">
         <xsl:value-of select="DATE"/>
      </td>
      <td xsl:use-attribute-sets="cellstyle">
         <xsl:value-of select="TODO"/>
      </td>
   </tr>
</xsl:template>
```

Een element kan meer dan één attribuutset gebruiken door in de waarde van het

xsl:use-attributes-sets attribuut de namen van de te gebruiken attribuutsets op te nemen in een lijst, van elkaar gescheiden door witruimten. Dat ziet er dan als volgt uit:

Voorbeeld 5.41. Meerdere attribuutsets in een element

```
<xsl:template match="DAY">
   <tr>
      <td xsl:use-attribute-sets="cellstyle numberstyle">
         <xsl:value-of select="DATE"/>
      </td>
   </tr>
</xsl:template>
```

Als meer dan één attribuutset hetzelfde attribuut definieert, dan wordt voor de definitie van dat attribuut de laatst vermelde attribuutset gebruikt. Als er meer dan één attribuutset met dezelfde naam bestaat (een situatie die zich zou kunnen voordoen wanneer een stylesheet een andere importeert), dan worden de attributen die in de sets gedefinieerd worden, samengevoegd. Als beide attribuutsets met dezelfde naam eenzelfde attribuut definiëren, dan wordt de attribuutwaarde uit de set met hoogste graad van belangrijkheid gekozen. Een stylesheet waarin meerdere attribuutsets met dezelfde naam en dezelfde graad van belangrijkheid, hetzelfde attribuut definiëren is verkeerd. De graad van belangrijkheid van een attribuutset hangt af van de graad van belangrijkheid van het **xsl:stylesheet** element dat deze attribuutset bevat. De graad van belangrijkheid van een stylesheet wordt op zijn beurt bepaald door de voorrangsregels die gelden voor de **xsl:include** en **xsl:import** elementen. Voor meer informatie hierover verwijzen wij naar de sectie over voorrangsregels bij het samenvoegen van meerdere stylesheets. [Sectie 5.3.1.14.3.]

Het is ook mogelijk attribuutsets aan bepaalde elementen toe te voegen door een **use-attribute-sets** attribuut toe te voegen aan een **xsl:element**, een **xsl:copy** of een **xsl:attribute-set** element, zoals in het onderstaande voorbeeld.

Voorbeeld 5.42. Het use-attribute-sets attribuut gebruikt samen met xsl:element

```
<xsl:template match="DAY">
   <tr>
      <xsl:element name="td" use-attribute-sets="cellstyle">
         <xsl:value-of select="DATE"/>
      </xsl:element>
   </tr>
</xsl:template>
```

Merk op dat hier het voorvoegsel **xsl:** niet gebruikt wordt voor **use-attribute-sets**. In feite is het gebruik van dit voorvoegsel op deze plaats niet toegestaan, omdat **use-attribute-sets** een attribuut is van een XSLT element in plaats van een attribuut van een gewoon element uit de resultaatset.

5.3.1.5.4. Processing instructions toevoegen

Het **xsl:processing-instruction** element plaatst, zoals de naam het al aangeeft, een processing instruction in het uitvoerdocument. Het doel (target) van de processing instruction wordt aangeduid door een verplicht **name** attribuut. De inhoud van het **xsl:processing-instruction** element wordt de inhoud van de processing instruction. Het is mogelijk om **xsl:value-of** elementen of **xsl:apply-templates** elementen op de nemen in het **xsl:processing-instruction** element, op voorwaarde dat het resultaat van deze instructies gewone tekst is. Onderstaande regel vervangt bijvoorbeeld **PROGRAM** elementen door een **gcc** processing instruction en

maakt gebruik van **xsl:value-of**.

Voorbeeld 5.43. Toevoegen van een processing instruction aan de uitvoer

```
<xsl:template match="PROGRAM">
   <xsl:processing-instruction name="gcc"> -04
      <xsl:value-of select="NAME"/>
   </xsl:processing-instruction>
</xsl:template>
```

Het is belangrijk er op te letten dat het **xsl:processing-instruction** element geen **xsl:element** instructie of een andere instructie mag bevatten waardoor elementen of attributen in het resultaat kunnen verschijnen. Er mag in het **xsl:processing-instruction** element geen instructie of letterlijke tekst staan waardoor er ?> in de uitvoer kan verschijnen, anders zou de XSLT processor denken dat de processing instructie op die plaats al afgesloten wordt.

5.3.1.5.5. Attribuutwaarde templates

Attribuutwaarde templates kopiëren gegevens van het invoerdocument naar attribuutwaardes in het uitvoerdocument. Stel nu dat je een elementgebaseerd invoerdocument om wilt zetten naar een uitvoerdocument met een attribuutgebaseerde vorm. (Dit voorbeeld is louter hypothetisch, want zoals wij al eerder uitgelegd hebben, is het gebruik van attributen best zoveel mogelijk te vermijden. [Sectie 2.3.2.]) Dit wil je waarschijnlijk bereiken met een template van de volgende vorm:

Voorbeeld 5.44. Onjuist voorbeeld ter illustratie van het probleem

```
<xsl:template match="ELEMENT_NAME">
   <ELEMENT_NAME ATTRIBUTENAME1="<xsl:value-of select="CHILD_NAME1">"
   ATTRIBUTENAME2="<xsl:value-of select="CHILD_NAME2">"/>
</xsl:template>
```

Het doel van deze template is dus de waarde van de **CHILD_NAME1** en **CHILD_NAME2** elementen toekennen aan de attribuutwaardes van de attributen **ATTRIBUTENAME1** en **ATTRIBUTENAME2** van het element **ELEMENT_NAME**.

Dit is echter geen goedgevormde XML. Het is niet toegelaten het < teken binnen de waarde van een attribuut te gebruiken. In plaats daarvan moeten binnen attribuutwaardes de **xsl:value-of** elementen vervangen worden door gegevens omvat door accolades {}, dit noemen we attribuutwaarde templates. De juiste schrijfwijze van bovenstaande template wordt dus:

Voorbeeld 5.45. Gecorrigeerd voorbeeld met attribuutwaarde templates

```
<xsl:template match="ELEMENT_NAME">
   <ELEMENT_NAME ATTRIBUTENAME1="{CHILD_NAME1}"
   ATTRIBUTENAME2="{CHILD_NAME2}"/>
</xsl:template>
```

Deze template heeft als gevolg dat in de uitvoer respectievelijk **{CHILD_NAME1}** en **{CHILD_NAME2}** vervangen worden door de waarde van de **CHILD_NAME1** en **CHILD_NAME2** kinderen van het gematchte **ELEMENT_NAME** element.

Attribuutwaarde templates kunnen ook meer gecompliceerde patronen gebruiken dan enkel

elementnamen. In feite kan eender welke XPath expressie gebruikt worden in een attribuutwaarde template. Een voorbeeld:

Voorbeeld 5.46. Gebruik van XPath expressies in attribuutwaarde templates

```
<xsl:template match="ELEMENT_NAME">
   <ELEMENT_NAME ATTRIBUTENAME1="{../NAME1}"
   ATTRIBUTENAME2="{position()}"/>
</xsl:template>
```

De waardes van attributen kunnen meer dan één attribuutwaarde template bevatten. Het is perfect mogelijk in de waarde van een bepaald attribuut een attribuutwaarde template te combineren met gewone tekst of met andere attribuutwaarde templates. In onderstaande template wordt een **ELEMENT_NAME** element gematcht en vervangen door zijn **NAME** kind. In het uitvoerdocument vormt dit **NAME** kind een link naar een file met volgend formaat: "waarde van het **NAME** kind" "waarde van het **NUMBER** attribuut".html.

Voorbeeld 5.47. Combinatie van tekst en verschillende attribuutwaarde templates in een attribuutwaarde

```
<xsl:template match="ELEMENT_NAME">
   <A HREF="{NAME}{@NUMBER}.html">
      <xsl:value-of select="NAME"/>
   </A>
</xsl:template>
```

Attribuutwaarde templates maken het mogelijk de beslissing welk element, attribuut of processing instruction in de uitvoer komt, af te laten hangen van de waardes van elementen en attributen in het invoerdocument. Attribuutwaarde templates kunnen in veel attributen van een XSLT stylesheet gebruikt worden. Deze attribuutwaarde templates zijn vooral belangrijk voor **xsl:element**, **xsl:attribute** en **xsl:processing-instruction** elementen, die wij hierboven behandeld hebben. Het is belangrijk te onthouden, dat als je meer ingewikkelde patronen wilt meegeven aan de waardes van de **name** attributen van **xsl:element**, **xsl:attribute** of **xsl:processing-instruction**, je steeds gebruik moet maken van attribuutwaarde templates. Een voorbeeld:

Voorbeeld 5.48. Gebruik van attribuutwaarde templates door xsl:element en xsl:attribuut

```
<xsl:template match="ELEMENT_NAME">
   <xsl:element name="{ELEMENT_NAME}">
      <xsl:attribute name="{@ATTRIBUTE_NAME}">
         <xsl:value-of select="CHILD_NAME"/>
      </xsl:attribute>
   </xsl:element>
</xsl:template>
```

Attribuutwaarde templates kunnen **niet** gebruikt worden als de waarde van een **select** attribuut, een **match** attribuut of een **xmlns** attribuut. Ze kunnen ook niet de waarde zijn van een attribuut dat de naam van een ander XSLT instructie-element geeft of van een element op het hoogste niveau (een element dat een direct kind is van **xsl:stylesheet**).

5.3.1.5.6. Commentaar toevoegen

Het **xsl:comment** element voegt commentaar toe aan het uitvoerdocument. Dit element heeft geen attributen. De inhoud van het element is de tekst die in de commentaar zal verschijnen. De inhoud van het **xsl:comment** element kan, analoog aan het **xsl:processing-instruction** element, **xsl:value-of** elementen en **xsl:apply-templates** elementen bevatten, weer op voorwaarde dat de resultaten van deze instructies gewoon tekst zijn. Het **xsl:comment** element mag ook geen **xsl:element** elementen bevatten, net zo min als andere instructies die elementen of attributen opleveren in het resultaat. Ook letterlijke tekst of instructies die als gevolg hebben dat er het symbool "--" in de commentaar verschijnt, zijn niet toegelaten. Een voorbeeld:

Voorbeeld 5.49. Toevoegen van commentaar aan de uitvoer

```
<xsl:template match="SLIDE">
   <xsl:comment> Er staat nu deze commentaar in de uitvoer in plaats van
   het SLIDE element. </xsl:comment>
</xsl:template>
```

5.3.1.5.7. Tekst toevoegen

De inhoud van een **xsl:text** element wordt als letterlijke tekst toegevoegd aan het uitvoerdocument. Het **xsl:text** element wordt niet vaak gebruikt omdat het meestal gemakkelijker is om de tekst gewoon in de template op te nemen. Toch is er een voordeel aan het **xsl:text** element: het bewaart namelijk de witruimte, zelfs als het knooppunt alleen maar uit witruimte bestaat. Standaard worden door de XSLT processor alle tekstknooppunten uit de stylesheet verwijderd die alleen maar witruimte bevatten, maar in sommige gevallen (denk aan broncode of poëzie) kan witruimte wel betekenisvol zijn. Een voorbeeldje:

Voorbeeld 5.50. Toevoegen van tekst aan de uitvoer

```
<xsl:template match="SLIDE">
   <xsl:text> Het oorspronkelijke SLIDE element wordt in de uitvoer
   vervangen door deze tekst. </xsl:text>
</xsl:template>
```

5.3.1.6. Modes

Het komt vaak voor dat je één en dezelfde inhoud van een brondocument meerdere keren, maar op een andere manier verwerkt, wilt opnemen in je uitvoerdocument. Je hebt bijvoorbeeld een tekst in XML formaat geschreven en je wilt deze omzetten naar HTML. Aan het begin van je uitvoerdocument wil je een inhoudstafel genereren die links bevat naar de verschillende hoofdstukken van je tekst. Dit wil dus zeggen dat je twee verschillende templates moet toepassen op dezelfde bronelementen op verschillende plaatsen in je document. Hoe los je dit op?

De oplossing van dit probleem is vrij simpel: je definieert twee verschillende templates die matchen op hetzelfde element maar die een verschillend **mode** attribuut hebben. Vervolgens kan je kiezen welke template je wilt gebruiken door het **mode** attribuut van het **xsl:apply-templates** element op de juiste waarde te zetten. Een deel van een stylesheet die wij gebruiken om onze tekst naar html om te zetten, illustreert het gebruik van het mode attribuut:

Voorbeeld 5.51. Het gebruik van het mode attribuut

```
<xsl:stylesheet version="1.0"
xmlns="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="THESIS">
      <html>
         <head>
            <title> Thesis Erwin Hermans en Carolien Coenen </title>
         </head>
         <body bgcolor="#FFFFFF" text="#000000">
            <xsl:apply-templates select="CHAPTER" mode="TOC"/>
            <xsl:apply-templates select="CHAPTER"/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="CHAPTER" mode="TOC">
      <H4>
         <a href="#generate-id(.))">
            <xsl:value-of select="TITLE"/>
         </a>
      </H4>
   </xsl:template>
   <xsl:template match="CHAPTER">
      <H1>
         <a name="{generate-id(.))}">
            <xsl:value-of select="TITLE"/>
         </a>
      </H1>
      <xsl:apply-templates/>
   </xsl:template>
</xsl:stylesheet>
```

In dit voorbeeld wordt gebruik gemaakt van de **generate-id()** functie. Deze functie genereert een unieke indentifier voor het knooppunt dat als argument wordt meegegeven. In dit voorbeeld wordt met behulp van die unieke indentifier een link gelegd van het hoofdstuk in de inhoudstafel (in de TOC mode) naar een anker rond de titel van dat hoofdstuk in de eigenlijke tekst. Doordat het twee keer om hetzelfde **CHAPTER** element gaat, is de gegenereerde identifier in beide gevallen hetzelfde .

5.3.1.7. Keuzes maken

XSLT voorziet twee elementen die het mogelijk maken, gebaseerd op de invoer, de uitvoer te beïnvloeden. Afhankelijk van welke patronen er in de invoer aanwezig zijn, geeft het **xsl:if** element een gegeven XML fragment wel of niet door aan de uitvoer. Het **xsl:choose** element kiest één fragment uit een aantal mogelijke XML fragmenten, afhankelijk van welke patronen aanwezig zijn in het invoerdocument. De meeste dingen die je met **xsl:if** en **xsl:choose** kunt doen, zijn ook te bereiken door een gepaste toepassing van templates. Alhoewel het gebruik van **xsl:if** en **xsl:choose** vaak gemakkelijker is, geven wij de voorkeur aan het gebruik van templates.

5.3.1.7.1. Het xsl:if element

Het **xsl:if** element biedt de mogelijkheid om de uitvoer te veranderen gebaseerd op een patroon in de invoer. Het **test** attribuut van **xsl:if** bevat een uitdrukking die een boolean als resultaat geeft. Als de uitdrukking waar is, wordt de inhoud van het **xsl:if** element doorgegeven aan de uitvoer. Als de uitdrukking niet waar is, gebeurt er niets. Onderstaande template geeft een opsomming van URL's. De **position()** functie geeft de positie van het gematchte element in de context knooppuntenlijst weer (we komen later nog terug op deze functie in de sectie over XPath [Sectie 5.4.] . Deze template zet achter elke URL een komma, behalve achter de laatste.

Voorbeeld 5.52. Gebruik van het xsl:if element

```
<xsl:template match="URL">
   <xsl:value-of select="."/>
   <xsl:if test="position()!=last()"> , </xsl:if>
</xsl:template>
```

Wij hebben dit voorbeeld gekozen omdat dit één van de templates is waar een alternatieve manier van uitwerken aan de hand van templates niet zo voor de hand ligt. Wij gebruiken deze template dan ook vrij frequent telkens wanneer er een opsomming van elementen voorkomt waartussen een komma moet staan.

Er zijn geen **xsl:else** of **xsl:else-if** elementen voorzien. De functionaliteit van deze elementen wordt vervuld door **xsl:choose**.

5.3.1.7.2. Het xsl:choose, het xsl:when en het xsl:otherwise element

Het **xsl:choose** element kiest één mogelijkheid uit een aantal uitvoermogelijkheden. Welke mogelijkheid gekozen wordt, is afhankelijk van een aantal voorwaarden. Elke voorwaarde en zijn bijhorende uitvoertemplate, wordt uitgedrukt met behulp van een **xsl:when** kindelement van **xsl:choose**. Het **test** attribuut van het **xsl:when** element is een XPath expressie die een boolean teruggeeft als waarde. Als er meerdere voorwaarden waar zijn, wordt alleen de eerste template waarvan de voorwaarde waar is, uitgevoerd. Het is dus niet nodig dat de voorwaarden disjunct zijn. Als geen enkele van de **xsl:when** elementen een voorwaarde heeft die waar is, dan wordt het **xsl:otherwise** kindelement van **xsl:choose** geïnstantieerd. Als er geen **xsl:otherwise** element aanwezig is, wordt er in dit geval niks gedaan.

Voorbeeld 5.53. Gebruik van het xsl:choose element

```
<xsl:template match="AUTHOR">
   <xsl:choose>
      <xsl:when test="NAME='Erwin'">
         <font color="#0000CC">
            <xsl:value-of select="."/>
         </font>
      </xsl:when>
      <xsl:when test="NAME='Carolien'">
         <font color="#FF0000">
            <xsl:value-of select="."/>
         </font>
      </xsl:when>
      <xsl:otherwise>
         <font color="#000000">
            <xsl:value-of select="."/>
         </font>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

Deze template zet de tekst "Erwin" in het blauw in het uitvoerdocument als het **NAME** kindelement van **AUTHOR** gelijk is aan "Erwin". Als dit kindelement gelijk is aan "Carolien", wordt de uitvoer "Carolien" in het rood weergegeven. Als het kindelement verschillend is van "Erwin" of "Carolien", wordt de inhoud van dat element in het zwart gezet.

In het voorbeeld hieronder wordt een alternatieve manier gegeven om hetzelfde resultaat te bekomen als in bovenstaand voorbeeld:

Voorbeeld 5.54. Alternatief voor het gebruik van xsl:choose

```
<xsl:template match="AUTHOR">
   <font color="#000000">
      <xsl:value-of select="NAME"/>
   </font>
</xsl:template>
<xsl:template match="AUTHOR[NAME="Erwin"]">
   <font color="#0000CC">
      <xsl:value-of select="NAME"/>
   </font>
</xsl:template>
<xsl:template match="AUTHOR[NAME="Carolien"]">
   <font color="#FF0000">
      <xsl:value-of select="NAME"/>
   </font>
</xsl:template>
```

5.3.1.8. Kopiëren van knooppunten

Het **xsl:copy** element kopieert het contextknooppunt naar de uitvoerboom. Kindelementen, attributen en andere inhoud worden niet automatisch meegekopieerd. Als je deze toch wilt meekopiëren, dan kan je in de inhoud van het **xsl:copy** element het **xsl:apply-templates** element gebruiken om te selecteren wat je wilt kopiëren. Dit element is vaak nuttig om een document van één markup woordenschat naar dezelfde of een verwante markup woordenschat om te zetten. Bijvoorbeeld om een XML document om te zetten naar een vrijwel identiek XML document waarbij alleen de oorspronkelijke commentaar wordt weggelaten. Enkele voorbeelden:

Een stylesheet die een document transformeert naar zichzelf:

Voorbeeld 5.55. De identieke transformatie

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="*|@*|comment()|processing-instruction()|text()">
      <xsl:copy>
         <xsl:apply-templates
         select="*|@*|comment()|processing-instruction()|text()"/>
      </xsl:copy>
   </xsl:template>
 </xsl:stylesheet>
```

Als we de identieke transformatie licht aanpassen door het **comment()** knooppunt uit de **match** en de **select** attributen weg te laten, bekomen we een stylesheet die het hele document behoudt, behalve de commentaarknooppunten.

Voorbeeld 5.56. Stylesheet die heel het document kopieert behalve de commentaarknooppunten

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="*|@*|processing-instruction()|text()">
      <xsl:copy>
```

```
        <xsl:apply-templates
        select="*|@*|processing-instruction()|text()"/>
      </xsl:copy>
    </xsl:template>
  </xsl:stylesheet>
```

Het **xsl:copy** element kopieert alleen het contextknooppunt. Het is ook mogelijk andere knooppunten te kopiëren, gebruik makend van **xsl:copy-of**. Het **select** attribuut van **xsl:copy-of** bepaalt welke knooppunten gekopieerd worden. In het onderstaande voorbeeld wordt een nieuw XML document opgebouwd dat alleen de titels van de hoofdstukken en de secties bevat:

Voorbeeld 5.57. Gebruik van xsl:copy-of

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="THESIS">
      <THESIS>
        <xsl:apply-templates select="CHAPTER"/>
      </THESIS>
    </xsl:template>
    <xsl:template match="CHAPTER">
      <xsl:copy-of select="TITLE"/>
      <xsl:apply-templates select="SECTION"/>
    </xsl:template>
    <xsl:template match="SECTION">
      <xsl:copy-of select="TITLE"/>
    </xsl:template>
  </xsl:stylesheet>
```

5.3.1.9. Tellen van knooppunten

Voor deze sectie hebben wij ons gebaseerd op [TRXSLTNU].

Het **xsl:number** element voegt een geheel getal met een bepaalde opmaak toe aan het uitvoerdocument. De waarde van het geheel getal wordt gegeven door het **value** attribuut van **xsl:number**. Het **value** attribuut bevat een expressie die geëvalueerd wordt en omgezet naar een getal. Het bekomen getal wordt afgerond tot het dichtsbijzijnde geheel getal en omgezet naar de vorm gegeven door de waarde van het **format** attribuut. [Sectie 5.3.1.9.2.1.] Het **value** attribuut kan eender welke gegevens bevatten die door XPath omgezet kunnen worden naar een getal. Er zijn defaultwaardes voorzien voor het **value** attribuut en het **format** attribuut.

We illustreren dit met een voorbeeld.

Voorbeeld 5.58. Gebruik van xsl:number met value attribuut

```
  <xsl:template match="THESIS">
    <html>
      <body>
        <xsl:apply-templates select="CHAPTER"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="CHAPTER">
```

```
      <H1>
         <xsl:number value="NUMBER"/>
         <xsl:text> : </xsl:text>
         <xsl:value-of select="TITLE"/>
      </H1>
   </xsl:template>
```

We passen deze template toe op onderstaand XML document:

Voorbeeld 5.59. Eenvoudig XML document voor een tekstraamwerk

```
   <THESIS>
      <?xml version="1.0"?>
      <CHAPTER>
         <TITLE> Hoofdstuk 1 </TITLE>
         <NUMBER> 1 </NUMBER>
      </CHAPTER>
      <CHAPTER>
         <TITLE> Hoofdstuk 2 </TITLE>
         <NUMBER> 2 </NUMBER>
      </CHAPTER>
   </THESIS>
```

We bekomen dan als resultaat:

Voorbeeld 5.60. Uitvoer van de stylesheet met het value attribuut

```
   <html>
      <body>
         <H1> 1:Hoofdstuk 1 </H1>
         <H2> 2:Hoofdstuk 2 </H2>
      </body>
   </html>
```

5.3.1.9.1. Defaultwaardes en hoe ze aan te passen

Als het **value** attribuut weggelaten wordt bij het **xsl:number** element, dan wordt de positie van het huidige knooppunt in de boom van het brondocument gebruikt als het nummer. Met andere woorden, de defaultwaarde die berekend wordt door **xsl:number** als er geen **value** attribuut aanwezig is, is de positie van het knooppunt ten opzichte van de broer- of zusterknooppunten met dezelfde naam als het huidige element. Voorbeelden hiervan volgen later. We willen hieraan toevoegen dat dit niet hetzelfde is als de waarde die teruggegeven wordt door de **position()** functie [Sectie 5.4.4.1., Tabel 5.4.] . De **position()** functie berekent alleen de positie van het huidige knooppunt, relatief ten opzichte van de andere knooppunten in de context knooppuntenlijst. De context knooppuntenlijst bestaat uit de knooppunten die gematcht worden door dezelfde template als degene die het huidige knooppunt matcht. De knooppunten in de context knooppuntenlijst delen bovendien dezelfde ouder. Het is mogelijk aan te passen wat er door **xsl:number** geteld wordt met behulp van de attributen **level**, **count** en **from**.

5.3.1.9.1.1. Het level attribuut

Als er geen **value** attribuut aanwezig is, telt het **xsl:number** element default de broer- of zusterknooppunten van het huidige knooppunt van hetzelfde type. Dus als je de **TITLE** elementen zou tellen in het bovenstaande voorbeeld [Sectie 5.3.1.9., Voorbeeld 5.59.] van

een eenvoudig XML document, dan zouden de twee **TITLE** elementen allebei het nummer 1 hebben, omdat ze geen broer of zus van mekaar zijn.

Als je het **level** attribuut de waarde **any** geeft, dan worden alle elementen van hetzelfde type als het huidige knooppunt in het document geteld, eender waar zij zich in de boomstructuur bevinden. Een voorbeeld:

Voorbeeld 5.61. Gebruik van xsl:number met level attribuut

```
<xsl:template match="THESIS">
   <html>
      <body>
         <xsl:apply-templates select="CHAPTER"/>
      </body>
   </html>
</xsl:template>
<xsl:template match="CHAPTER">
   <xsl:apply-templates select="TITLE"/>
</xsl:template>
<xsl:template match="TITLE">
   <H1>
      <xsl:number level="any"/>
      <xsl:text> : </xsl:text>
      <xsl:value-of select="TITLE"/>
   </H1>
</xsl:template>
```

Als we deze stylesheet toepassen op ons eenvoudig XML document bekomen we als resultaat:

Voorbeeld 5.62. Uitvoer van de stylesheet met de waarde any voor het level attribuut

```
<html>
   <body>
      <H1> 1:Hoofdstuk 1 </H1>
      <H2> 2:Hoofdstuk 2 </H2>
   </body>
</html>
```

De defaultwaarde van het **level** attribuut is **single**. Als we de defaultwaarde zouden meegeven aan het **level** attribuut in bovenstaande stylesheet, dan zou er voor de dubbele punt in het uitvoerdocument twee keer 1 komen te staan. Je kan het **level** attribuut ook de waarde **multiple** meegeven. Als de waarde van het **level** attribuut op **multiple** staat, wordt een lijst opgesteld met daarin alle voorouders van het huidige knooppunt die dezelfde naam hebben, het huidige knooppunt zelf inbegrepen. Elk knooppunt in deze lijst krijgt dan een nummer dat overeenstemt met het aantal voorafgaande broer- of zusterknooppunten van hetzelfde type, verhoogd met één (het knooppunt zelf wordt ook meegeteld). In het voorbeeld [Sectie 5.3.1.9.1.2., Voorbeeld 5.63.] in de volgende sectie wordt dit geïllustreerd. We willen in dat voorbeeld alle **CHAPTER** elementen tellen die voorafgaan aan het **CHAPTER** element waarvan het context knooppunt een nakomeling is en dit met inbegrip van het huidige **CHAPTER** element zelf. Om dit te doen moeten we het **level** attribuut op **multiple** zetten.

5.3.1.9.1.2. Het count attribuut

Als er geen **value** attribuut aanwezig is, worden default elementen van hetzelfde type als het huidige element geteld. Door aan het **count** attribuut van **xsl:number** een gepaste uitdrukking mee te geven, kan je aangeven wat er geteld moet worden. Een licht gewijzigd uittreksel uit

de templates die wij gebruiken om onze thesistekst naar HTML om te zetten:

Voorbeeld 5.63. Gebruik van xsl:number met count attribuut en from attribuut

```
<xsl:template match="CHAPTER//EXPLANATIONS">
  <H4> Tabel
    <xsl:number level="multiple" count="CHAPTER"/>
    <xsl:text> . </xsl:text>
    <xsl:number level="any" count="EXPLANATIONS" from="CHAPTER"/>
    <xsl:element name="A">
      <xsl:attribute name="NAME">
        <xsl:value-of select="TITLE"/>
      </xsl:attribute>
      <xsl:value-of select="TITLE"/>
    </xsl:element>
  </H4>
  <Table width="80%" border="1">
    <xsl:apply-templates select="HEADING | EXPITEM"/>
  </Table>
</xsl:template>
```

In deze template worden eerst de **CHAPTER** elementen geteld, dit gebeurt door het **level** attribuut de waarde **multiple** mee te geven. Hierdoor worden alle **CHAPTER** voorouders van het **EXPLANATIONS** element geteld. Dus als het **EXPLANATIONS** element zich in het derde hoofdstuk bevindt, is het eerste getal dat na de tekst "Tabel" komt een 3. Na het eerste getal volgt een punt. Vervolgens wordt het aantal **EXPLANATIONS** elementen van het **CHAPTER** element geteld door het **level** attribuut de waarde **any** en het **from** attribuut de waarde **CHAPTER** te geven . Dus als het **EXPLANATIONS** element het derde element van dat type in het hoofdstuk is, dan krijgen we als uiteindelijk waarde van het getal: 3.3.

In het voorgaande voorbeeld is **count="EXPLANATIONS"** in feite overbodig. Wij geven er echter de voorkeur aan dit attribuut steeds neer te schrijven, omdat dit het makkelijker maakt in de gaten te houden wat je nu juist aan het tellen bent.

5.3.1.9.1.3. Het from attribuut

In bovenstaand voorbeeld [Sectie 5.3.1.9.1.2., Voorbeeld 5.63.] wordt ook gebruik gemaakt van het **from** attribuut bij het **xsl:number** element. Het **from** attribuut bevat een XPath expressie die aangeeft bij welk knooppunt het tellen begint in de invoerboom. Opgelet, het tellen begint altijd vanaf 1. Het **from** attribuut verandert alleen welk knooppunt als eerste knooppunt wordt beschouwd. Dit attribuut heeft alleen effect als het samen met het **level** attribuut wordt gebruikt. In elk ander geval heeft het geen effect op de uitvoer.

5.3.1.9.2. Omzetting van getal naar string

Tot nu toe hebben we alleen geteld met getallen van de vorm 1, 2, 3, enzovoort. Er bestaan echter nog andere mogelijkheden. Neem bijvoorbeeld de inhoudstafel van een boek, vaak worden deze bladzijden genummerd met Romeinse cijfers: i, ii, iii, enzovoort. Verschillende landen gebruiken vaak ook verschillende conventies voor het voorstellen van reële getallen en het groeperen van de cijfers van een groot getal. Deze eigenschappen kunnen allemaal aangepast worden met behulp van de volgende vier attributen van **xsl:number**: **format**, **letter-value**, **grouping-separator** en **grouping-size**.

5.3.1.9.2.1. Het format attribuut

De stijl van nummering die gebruikt wordt door **xsl:number** kan aangepast worden met behulp van het **format** attribuut. Dit attribuut heeft in het algemeen één van de volgende waardes:

- i: Romeinse cijfers in kleine letters i, ii, iii, iv, v, vi,...
- I: Romeinse cijfers in hoofdletters I, II, III, IV, V, V, VI,...
- a: kleine letters a, b, c, d, e,...
- A: grote letters A, B, C, D, E,...

Het is ook mogelijk bij je nummering gebruik te maken van leidende nullen door het aantal leidende nullen aan te geven in het **format** attribuut: **format="001"**. Je kan ook punten, komma's of witruimte opnemen in de waarde van het **format** attribuut. Dit laat toe om bijvoorbeeld getallen van de vorm: "1.2.3 " op te bouwen.

Een voorbeeld van het gebruik van het **format** attribuut letterlijk uit de stylesheets voor onze thesistekst geplukt:

Voorbeeld 5.64. Gebruik van xsl:number met format attribuut

```
<xsl:template match="CHAPTER//EXPLANATIONS">
   <H4> Tabel
      <xsl:number level="multiple" count="CHAPTER" format="1."/>
      <xsl:number level="any" count="EXPLANATIONS" from="CHAPTER"
      format="1 "/>
      <xsl:element name="A">
         <xsl:attribute name="NAME">
            <xsl:value-of select="TITLE"/>
         </xsl:attribute>
         <xsl:value-of select="TITLE"/>
      </xsl:element>
   </H4>
   <Table width="80%" border="1">
      <xsl:apply-templates select="HEADING | EXPITEM"/>
   </Table>
</xsl:template>
```

Je kan het **format** attribuut ook gebruiken om de nummering van verschillende knooppunten te combineren.

Voorbeeld 5.65. Gebruik van format attribuut bij het nummeren van verschillende knooppunten

```
<xsl:template match="CHAPTER//SECTION">
   <H2>
      <xsl:number level="multiple" count="CHAPTER|SECTION"
      format="1.1.1"/>
      <xsl:value-of select="TITLE"/>
   </H2>
   <xsl:apply-templates/>
</xsl:template>
<xsl:template match="APPENDIX//SECTION">
   <H2>
      <xsl:number level="multiple" count="APPENDIX|SECTION"
      format="1.1.1"/>
      <xsl:value-of select="TITLE"/>
   </H2>
   <xsl:apply-templates/>
</xsl:template>
```

Het resultaat van het toepassen van bovenstaande template is dat de nummering van de secties

in de hoofstukken begint met het nummer van het hoofdstuk of de appendix waarin de sectie zich bevindt. Sectie 2 in hoofdstuk 2 heeft als nummer 2.2. Een subsectie 3 van deze sectie heeft als nummer 2.2.3. Een subsectie 4 van deze subsectie heeft als nummer 2.2.3.4, enzovoort. De diepte van de subsectie bepaalt de diepte van de nummering. De nummering bij de appendices verloopt analoog, met dit verschil dat de appendices met letters genummerd worden.

5.3.1.9.2.2. Het letter-value attribuut

Het **letter-value** attribuut maakt het onderscheid tussen letters die geïnterpreteerd worden als nummers (I, II, III, IV,..) en letters die geïnterpreteerd worden als letters (I, J, K, L, M,...). Als je in plaats van de Romeinse nummering de lettersequentie startend vanaf I (I, J, K, L, M,...) wilt gebruiken, dan moet je het **letter-value** attribuut op het sleutelwoord **alphabetic** zetten. Het sleutelwoord **traditional** (dit is de default) specificeert dat bij het tellen een numerieke opeenvolging van cijfers wordt gebruikt. Een voorbeeld:

Voorbeeld 5.66. Gebruik van xsl:number met format attribuut

```
<xsl:template match="CHAPTER">
   <H4>
      <xsl:number level="multiple" count="CHAPTER" format="I"
      letter-value="alphabetic"/>
      <xsl:element name="A">
         <xsl:attribute name="NAME">
            <xsl:value-of select="TITLE"/>
         </xsl:attribute>
         <xsl:value-of select="TITLE"/>
      </xsl:element>
   </H4>
</xsl:template>
```

5.3.1.9.2.3. Het grouping-separator en het grouping-size attribuut

Het **grouping-separator** attribuut duidt aan welk scheidingsteken gebruikt wordt tussen groepen van cijfers. In Amerika gebruikt men in grote getallen bijvoorbeeld een komma om de drie cijfers: 4,598,256,254,235. In andere landen worden dan weer de cijfers in groepen van vier samengenomen of wordt een punt als scheidingsteken gebruikt. Het **grouping-size** attribuut duidt aan hoeveel cijfers er samengenomen moeten worden per groep. In het algemeen hangt de waarde van de attributen samen met de taal die je gebruikt. Een voorbeeld:

Voorbeeld 5.67. Gebruik van het grouping separator en het grouping-size attribuut

```
<xsl:number grouping-separator="," grouping-size="3"/>
```

5.3.1.10. Sorteren van uitvoerelementen

Het **xsl:sort** element sorteert de uitvoerelementen in een bepaalde volgorde, die niet noodzakelijk gelijk is aan de volgorde in het invoerdocument. Een **xsl:sort** element is een kind van een **xsl:apply-templates** element of een **xsl:for-each** element. Het **select** attribuut van het **xsl:sort** element definieert de sleutel die gebruikt wordt om de uitvoer van het **xsl:apply-templates** of het **xsl:for-each** element te sorteren.

Default gebeurt het sorteren in alfabetische volgorde van de sleutels. Als er meer dan één **xsl:sort** element aanwezig is binnen een gegeven **xsl:apply-templates** of **xsl:for-each** element, dan worden de elementen eerst gesorteerd volgens de eerste sleutel, dan volgens de tweede sleutel, enzovoort. Als er dan nog elementen zijn die hetzelfde zijn, dan verschijnen ze in de uitvoer in dezelfde volgorde als in het invoerdocument. Als voorbeeld geven wij de template die wij gebruiken om de verschillende items in de bibliografie van onze thesistekst te sorteren met als sleutel het **REFERENCE** element:

Voorbeeld 5.68. Gebruik van het xsl:sort element

```
<xsl:apply-templates select="BIBITEM">
   <H1> Bibliografie
      <xsl:apply-templates select="BIBITEM">
         <xsl:sort select="REFERENCE"/>
      </xsl:apply-templates>
   </H1>
</xsl:apply-templates>
```

Het is mogelijk de default alfabetische volgorde te veranderen door het optionele **data-type** attribuut een andere waarde mee te geven. Als je bijvoorbeeld wilt sorteren op getalwaarde in plaats van op alfabetische volgorde, dan moet je de waarde **number** aan het **data-type** attribuut meegeven. Deze manier van sorteren kan je natuurlijk alleen maar toepassen op elementen die een getal als inhoud hebben. Als je de default stijgende volgorde van sorteren wilt veranderen naar een dalende volgorde van moet je het **order** attribuut de waarde **descending** meegeven. We combineren beide attributen in onderstaand voorbeeldje:

Voorbeeld 5.69. Gebruik van het data-type en het order attribuut

```
<xsl:sort order="descending" data-type="number" select="NUMBER"/>
```

Alfabetisch sorteren is natuurlijk afhankelijk van het alfabet dat gebruikt wordt. Het **lang** attribuut geeft aan welke de taal van de sleutels is. De waarde van dit attribuut moet een ISO 693 taalcode zijn, zoals bijvoorbeeld **en** voor Engels [ISO693]. Processoren worden wel niet verondersteld te weten hoe ze moeten sorteren in al de verschillende talen die ze eventueel zouden kunnen tegenkomen. Dit zou de sorteeralgoritmes te gecompliceerd maken. Er bestaan namelijk natuurlijke talen waarin verschillende standaardmethoden bestaan om te sorteren gebaseerd op verschillende criteria. Het **lang** attribuut wordt genegeerd als het **data-type** de waarde **number** heeft.

Ten slotte bestaat er ook nog een **case-order** attribuut dat twee waardes kan aannemen: **upper-first** of **lower-first** om aan te duiden of hoofdletters gesorteerd moeten worden vóór kleine letters of omgekeerd. De defaultwaarde is afhankelijk van de gebruikte taal.

5.3.1.11. Variabelen, templates met een naam en parameters

In deze sectie behandelen we variabelen, templates met een naam en parameters. Wij bespreken deze begrippen samen omdat er heel wat onderlinge overeenkomsten zijn. Variabelen, templates met een naam en parameters zijn in feite niet meer dan een wijze om een naam aan een bepaalde waarde te verbinden. Deze waarde kan constant zijn of in sommige gevallen variabel.

Variabelen en parameters introduceren een bijkomend gegevenstype in de XPath taal, met

name het resulterend boomfragment. Een variabele of parameter kan verbonden worden aan een resulterend boomfragment in plaats van aan één van de vier basis gegevenstypes: string, getal, boolean en sets van knooppunten. Resulterende boomfragmenten komen nog kort ter spraken in de sectie over XPath [Sectie 5.4.4.5.] .

5.3.1.11.1. Variabelen

Soms kan het handig zijn om variabelen te gebruiken om een nettere code te verkrijgen. Variabelen maken het mogelijk om vaak gebruikte tekst te vervangen door een simpele naam en een verwijzing naar de definitie van de variabele. Variabelen maken het ook gemakkelijk om tekst aan te passen die op verschillende plaatsen in het document aanwezig is, door gewoon de definitie van de variabele aan te passen. Het **xsl:variable** element definieert een string met een naam, die aan de hand van een attribuutwaarde template [Sectie 5.3.1.5.5.] gebruikt kan worden op een andere plaats in de stylesheet.

Het **xsl:variable** element heeft één verplicht attribuut, het **name** attribuut en één optioneel attribuut, het **select** attribuut. Het **name** attribuut levert de naam waarmee naar de variabele verwezen kan worden. De waarde van het **select** attribuut kan eender welke XPath expressie zijn [Sectie 5.4.] . De waarde van de variabele kan op verschillende manier gespecificeerd worden:

- Als er een **select** attribuut aanwezig is, dan is de waarde van dit attribuut een expressie. De waarde van de variabele is het resultaat van de evaluatie van deze expressie. In dit geval moet de inhoud van het element leeg zijn.
- Als er geen **select** attribuut aanwezig is, dan levert de inhoud van het **xsl:variable** element de waarde waardoor de verwijzing naar de variabele vervangen moet worden.
- Als het **xsl:variable** element leeg is en geen **select** attribuut heeft, dan is de waarde van de variabele gelijk aan de lege string.

Als aan een variable meerdere keren een verschillende waarde toegekend wordt, dan wordt de laatst toegekende waarde als de effectieve waarde van de variabele aanzien.

In het onderstaande voorbeeld definieert het **xsl:variable** element een variabele met de naam **Copy** en de waarde **Copyright 2002 Erwin Hermans en Carolien Coenen**. Merk op dat we geen gebruik maken van het **select** attribuut.

Voorbeeld 5.70. Definiëren van variabelen aan de hand van de inhoud van xsl:variable

```
<xsl:variable name="copy"> Copyright 2002 Erwin Hermans en Carolien
Coenen </xsl:variable>
```

Hieronder definiëren we dezelfde variabele, maar nu maken we gebruik van het **select** attribuut om de variabele een waarde te geven. Bemerk dat de inhoud van het **xsl:variable** element leeg is.

Voorbeeld 5.71. Definiëren van variabelen met het select attribuut

```
<xsl:variable name="copy" select="'Copyright 2002 Erwin Hermans en
Carolien Coenen'"/>
```

Op het eerste gezicht lijken de bijkomende enkelvoudige aanhalingstekens rond de tekst vreemd. Deze zijn echter nodig om aan te geven dat de waarde van dit **select** attribuut een

string is. Wij hebben getest wat er gebeurt als deze aanhalingstekens zouden weggelaten worden. Het resultaat hiervan is dat bij het oproepen van de variabele, deze niet meer in het uitvoerdocument verschijnt. Het is dus belangrijk deze aanhalingstekens niet te vergeten wanneer je een stringinhoud aan je variabele wilt toekennen.

Onze persoonlijk voorkeur gaat uit naar de eerste methode waar de waarde van de variabele bepaald wordt door de inhoud van het **xsl:variabele** element. Deze werkwijze lijkt ons minder verwarrend.

Om de waarde van deze variabele te gebruiken, zet je een dollarteken voor de naam van de variabele. Om een variabele op te nemen in de waarde van een attribuut, moet je gebruik maken van een attribuutwaarde template. Bijvoorbeeld:

Voorbeeld 5.72. Verwijzen naar een variabele in een attribuutwaarde

```
<THESIS COPYRIGHT="{$copy}"/>
```

Je kan **xsl:value-of** gebruiken om de tekst waardoor de variabele vervangen moet worden als tekst in het uitvoerdocument te krijgen. Merk op dat we in dit voorbeeld geen attribuutwaarde template gebruiken, omdat zoals reeds eerder vermeld [Sectie 5.3.1.5.5.] dit niet toegestaan is in combinatie met een **select** attribuut.

Voorbeeld 5.73. De variabele vervangen door zijn tekst

```
<xsl:value-of select="$copy"/>
```

De inhoud van het **xsl:variable** element mag markup, inclusief andere XSLT instructies, bevatten. Dit betekent dat je de waarde van een variabele kan berekenen gebaseerd op andere informatie en zelfs op waardes van andere variabelen. Het is echter verboden om een variabele recursief naar zichzelf te laten verwijzen. Een gelijkaardige situatie doet zich voor als je twee variabelen naar elkaar laat verwijzen. Ook dit is fout.

xsl:variable elementen kunnen ofwel kinderen op het hoogste niveau van het **xsl:stylesheet** element zijn, ofwel kunnen ze opgenomen worden binnen template regels. Een variabele die gedefinieerd wordt op het hoogste niveau van de stylesheet kan van eender waar in de stylesheet opgeroepen worden. Het is een globale variabele. Een variabele die gedeclareerd wordt binnen een template regel is daarentegen alleen toegankelijk binnen die template regel. Dit wil zeggen dat de broer- of zusterelementenvan het **xsl:variable** element en hun respectievelijke kinderen deze variabele kunnen gebruiken. Dit noemen we de scope van de variabele. Het **xsl:variable** element kan eender waar binnen een template gebruikt worden. Lokale variabelen hebben voorrang op globale variabelen met dezelfde naam. Lokale variabelen kunnen ook andere lokale variabelen overschrijven. Als er een conflict is tussen twee variabelen met dezelfde naam, dan wordt de dichtsbijzijnde lokale variabele met dezelfde naam gebruikt.

Er bestaan parallellen tussen XSLT variabelen en Java. Wij bespreken deze in een apart hoofdstuk [Sectie 14.1.5.] .

5.3.1.11.2. Templates met een naam

Variabelen zijn beperkt tot eenvoudige tekst en markup. XSLT levert een krachtiger macromechanisme dat het mogelijk maakt standaard markup en tekst rond veranderende data te zetten. Stel dat je een bepaalde vrij complexe markup hebt die op verscheidene plaatsen in je stylesheet binnen andere templates terugkomt, dan kan je ervoor kiezen om deze in een template met een naam om te vormen. Alhoewel je in principe hetzelfde effect kan bereiken met het **xsl:variable** element, wordt voor grotere stukken code de voorkeur gegeven aan een template met een naam. Templates met een naam lijken op variabelen, maar ze doen nog iets meer: ze laten je toe om gegevens van de plaats waar de template met een naam wordt toegepast te gebruiken, dit in tegenstelling tot het gewoon invoegen van vaste tekst.

Het **xsl:template** element kan een **name** attribuut hebben waardoor deze template expliciet kan opgeroepen worden. Een voorbeeld van een template met een naam:

Voorbeeld 5.74. Template met een naam

```
<xsl:template name="CELL">
   <td>
      <font face="Times, serif" color="blue" size="2">
         <xsl:value-of select="."/>
      </font>
   </td>
</xsl:template>
```

Het `<xsl:value-of select="."/>` element in het midden van de template met de naam "CELL" wordt telkens vervangen door de inhoud van het huidige knooppunt van waaruit deze template wordt opgeroepen. Het is mogelijk de beide attributen van het **xsl:template** element, met name **name** en **match** samen te gebruiken.

Het **xsl:call-template** element kan enkel voorkomen in de inhoud van een template regel. Het heeft een noodzakelijk **name** attribuut dat als waarde de naam van de op te roepen template krijgt. Als het **xsl:call-template** element verwerkt wordt door de processor, wordt het vervangen door de inhoud van het **xsl:template** element met dezelfde naam. Een illustratie van het gebruik van het **xsl:call-template** element:

Voorbeeld 5.75. Het xsl:call-template element

```
<xsl:template match="TABLE_ELEMENT">
   <xsl:call-template name="CELL"/>
</xsl:template>
```

In dit voorbeeld worden er door het gebruik van de template met een naam slechts een paar regels code uitgespaard. Als de complexiteit van deze template echter groter wordt en vaker herbruikt wordt, heeft dit het voordeel dat de complexiteit van je eigenlijke stylesheet sterk gereduceerd wordt. Templates met een naam hebben ook het voordeel dat ze, net zoals variabelen, gemeenschappelijke patronen uit de stylesheet halen zodat je deze maar op één plaats meer moet aanpassen.

Een stylesheet die meer dan één template met dezelfde naam en dezelfde graad van belangrijkheid bevat is verkeerd. Voor meer informatie over het bepalen van de graad van belangrijkheid verwijzen wij naar de sectie over voorrangsregels bij het samenvoegen van meerdere stylesheets. [Sectie 5.3.1.14.3.]

We eindigen deze sectie met nog een licht gewijzigd voorbeeld uit een stylesheet die wij geschreven hebben als inleidende studie op XML en XSL. De template met als naam

HEADHTML wordt in een aparte stylesheet **general.xsl** gedefinieerd.

Voorbeeld 5.76. general.xsl

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:template name="HEADHTML">
        <head>
           <TITLE>
              <xsl:value-of select="TITLE"/>
           </TITLE>
           <LINK rel="stylesheet" href="{URL}"/>
        </head>
     </xsl:template>
  </xsl:stylesheet>
```

In de volgende stylesheet wordt de template opgeroepen. Merk op dat eerst de stylesheet **general.xsl** met behulp van het **xsl:include** element in de stylesheet **slides.xsl** opgenomen moet worden.

Voorbeeld 5.77. slides.xsl

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:include href="general.xsl"/>
     <xsl:template match="SLIDESHOW">
        <html>
           <xsl:call-template name="HEADHTML"/>
           <!--Rest van de template-->
        </html>
     </xsl:template>
  </xsl:stylesheet>
```

5.3.1.11.3. Doorgeven van parameters

Elke oproep van een template kan parameters doorgeven aan de template om zijn uitvoer aan te passen. Dit gebeurt op dezelfde manier voor templates mét en templates zonder naam. In het **xsl:template** element, worden de parameters voorgesteld als **xsl:param** kindelementen. In het **xsl:call-template** of het **xsl:apply-templates** element, worden parameters voorgesteld als **xsl:with-param** elementen.

Parameters worden gebruikt als er aan een template een bij elke oproep van de template veranderende waarde moet worden meegegeven. Het verplichte **name** attribuut van het **xsl:param** element geeft de parameter een naam. Dit is vooral belangrijk als er meerdere argumenten zijn. Het **xsl:param** element heeft een optioneel **select** attibuut. Het definiëren van een parameter gebeurt op dezelfde manier als het definiëren van een variabele. [Sectie 5.3.1.11.1.] De inhoud van het **xsl:param** element of de waarde van het **select** attribuut levert een default waarde voor de parameter, die gebruikt wordt als de oproep van de template geen waarde meegeeft.

Wat betreft de scope van de parameter, ook hier zijn dezelfde regels van toepassing die we al eerder beschreven hebben voor variabelen [Sectie 5.3.1.11.1.] . Er wordt een onderscheid gemaakt tussen globale en lokale parameters. Het enige verschilpunt is dat in tegenstelling tot variabelen een **xsl:param** element alleen maar in het begin van een **xsl:template** element mag voorkomen.

Hernemen we het voorbeeld uit de vorige sectie waarin de template met als naam **CELL** wordt gedefinieerd. Nu willen we dat in iedere cel van de tabel een link wordt opgenomen die afhangt van het gematchte element waarvoor de **CELL** template wordt opgeroepen. De defaultwaarde voor de **file** parameter is **index.html**.

Voorbeeld 5.78. Gebruik van xsl:param

```
<xsl:template name="CELL">
   <xsl:param name="file"> index.html </xsl:param>
   <td>
      <font face="Times, serif" color="blue" size="2">
         <a href="{$file}">
            <xsl:value-of select="."/>
         </a>
      </font>
   </td>
</xsl:template>
```

Als deze template wordt opgeroepen, levert een **xsl:with-param** kind van het **xsl:apply-templates** element de waarde van de parameter. Het **name** attribuut dient om de gebruikte parameter te identificeren en de inhoud van het **xsl:with-param** element geeft de waarde van de parameter.

Voorbeeld 5.79. Het xsl:with-param element

```
<xsl:template match="TABLE_ELEMENT">
   <xsl:call-template name="CELL">
      <xsl:with-param name="file"> table_element.html </xsl:with-param>
   </xsl:call-template>
</xsl:template>
```

Er is wel degelijk een verschil tussen **xsl:variable** en **xsl:param**. Het verschil bestaat hierin dat de waarde die we voor een parameter definieren slechts een default waarde is. Geven we bij het oproepen van een bepaalde, geparametriseerde template een waarde mee voor een parameter, dan zal die waarde de default waarde overschrijven. Indien we globaal een variabele definieren en we definieren lokaal, als eerste kindelement van een template, ook een variabele met dezelfde naam, dan zal, binnen die template, alleen met de lokale definitie rekening houden worden. Kort samengevat: een variabeledefinitie overschrijft en een parameterdefinitie kan overschreven worden.

5.3.1.12. Sleutels

Sleutels maken het mogelijk om een structuur van impliciete kruisverwijzingen in je documenten op te nemen, dit in tegenstelling tot de stuctuur van expliciete kruisverwijzingen die ID-type attributen leveren. Sleutels hebben een aantal voordelen ten opzicht van ID-type attributen. We sommen deze voordelen hieronder kort op:

- Sleutels worden gedeclareerd in de stylesheet en niet in een DTD die vaak zelfs niet eens ingelezen wordt.
- De sleutelwaardes kunnen eender waar in een element geplaatst worden. De ID van een element kan echter alleen maar gespecificeerd worden in een attribuut.
- Het is mogelijk meer dan één sleutel aan een element te verbinden
- Sleutels kunnen eender welke waarde bevatten, dus ook witruimte. ID's moeten aan de voorwaarde van XML namen voldoen. [Sectie 2.3.1.] , [TRXMLNAMES]
- Een sleutel moet niet uniek zijn, een ID wel.

Een sleutel is een soort van veralgemeende identifier die hoort bij een knooppunt. Een sleutel bestaat uit drie delen:

- de naam van de sleutel (een gekwalificeerde naam)
- het knooppunt waarop de sleutel van toepassing is
- de waarde voor de sleutel (een string)

De sleutel kan beschouwd worden als de afbakening van een gescheiden, onafhankelijke ruimte van identifiers. De waarde die een sleutel in een bepaald knooppunt heeft, kan op eender welke plaats in het knooppunt gespecificeerd worden. Bijvoorbeeld in een attribuut, in een kindelement van het knooppunt of in de inhoud. Om de waarde van een sleutel te vinden, wordt gebruik gemaakt van XPath expressies. [Sectie 5.4.]

In onderstaand voorbeeld wordt een sleutel voor de **INDEX** knooppunten gedeclareerd met als naam **INDICES** en als waarde de string-waarde van het knooppunt zelf.

Voorbeeld 5.80. Sleuteldeclaratie met behulp van het xsl:key element

```
<xsl:key name="INDICES" match="INDEX" use="."/>
```

Het is mogelijk dat er in hetzelfde document meerdere sleutels bestaan die gelden voor hetzelfde knooppunt, die dezelfde naam hebben, maar waarvan de sleutelwaardes verschillend zijn. Het is ook mogelijk dat er verschillende sleutels in het document zijn die dezelfde naam en dezelfde waarde hebben, maar die gelden voor verschillende knooppunten.

Een stylesheet declareert een aantal sleutels voor het document door gebruik te maken van het **xsl:key** element. Het **xsl:key** element is een element van topniveau, dit wil zeggen dat het alleen maar rechtstreeks onder het **xsl:stylesheet** element kan voorkomen. Het **xsl:key** element heeft drie attributen: **name**, **match** en **use**.

De waarde van het **name** attribuut is een gekwalificeerde naam (bestaande uit een optioneel namespace voorvoegsel en het lokale deel van de naam). De waarde van het **match** attribuut is een Xpath expressie. [Sectie 5.4.]

Het **xsl:key** element geeft informatie over de sleutels van elk knooppunt dat het patroon matcht dat gespecificeerd wordt in het **match** attribuut op eender welke diepte in de boom.

Het **use** attribuut bevat een uitdrukking die de waardes van de sleutel specificeert. De uitdrukking wordt één keer geëvalueerd voor elk knooppunt dat het patroon matcht. Elk knooppunt dat het patroon matcht heeft dus een sleutel met een bepaalde naam, waarbij de waarde van die sleutel bekomen wordt door de uitdrukking in het **use** attribuut te evalueren voor dat knooppunt en om te zetten naar een string-waarde. De omzetting naar een string-waarde, wordt behandeld in de sectie over XPath. [Sectie 5.4.4.4.] Als het resultaat van de evaluatie van deze uitdrukking een set van knooppunten is, dan is de waarde van de sleutel gelijk aan de string-waarde van één of meer van de knooppunten in deze set.

Dus een knooppunt **X** behoort tot een sleutel met naam **Y** en waarde **Z** als en slechts als er een **xsl:key** element bestaat zodat:

- **X** het patroon matcht dat gespecificeerd wordt in het **match** attribuut van het **xsl:key** element
- de waarde van het **name** attribuut van het **xsl:key** element gelijk is aan **Y**
- wanneer de uitdrukking die gespecificeerd wordt in het **use** attribuut van het **xsl:key**

element, geëvalueerd wordt voor **X**, het resultaat van deze evaluatie na string-omzetting gelijk aan **Z**.

Het **xsl:key** element wordt altijd gebruikt in combinatie met de functie **key(String sleutelnaam, Object waarde)**. De **key(String sleutelnaam, Object waarde)** functie selecteert alle knooppunten waarvan de waarde voor de opgegeven sleutel (met **sleutelnaam**) overeenkomt met de opgegeven **waarde**. We komen hier in de sectie over XPath meer uitgebreid op terug. [Sectie 5.4.4.1., Voorbeeld 5.101.]

5.3.1.13. Witruimte

Het default gedrag voor tekstknooppunten die vanuit een invoerdocument gelezen worden, bestaat uit het behouden van alle witruimte. Een tekstknooppunt is de gewone tekstuele inhoud van een element. Als je zo'n tekstknooppunt naar de uitvoer schrijft, ontstaat er dus vaak witruimte die geen specifieke betekenis heeft. Neem het volgende element:

Voorbeeld 5.81.

```
<LINE>
   <!--Dit is een voorbeeld element-->
Dit is het tekstknooppunt dat bij het element LINE hoort. </LINE>
```

Als we van dit element de waarde nemen, krijgen we:

```
Dit is het tekstknooppunt dat bij het element LINE hoort.
```

Nu is voor een computer onbetekenende witruimte niet belangrijk, maar voor een mens kan dit vaak storend werken. Om de witruimte aan het begin en op het einde van een string weg te halen, bestaat er de functie **normalize-space()**. Je kan deze functie als volgt gebruiken: `<xsl:value-of select="normalize-space(LINE)"/>` . De functie **normalize-space()** wordt ook behandeld in de sectie over XPath. [Sectie 5.4.4.4., Tabel 5.5.]

Het is ook mogelijk knooppunten die alleen uit witruimte bestaan automatisch te deleten door gebruik te maken van het **xsl:strip-space** element. Het **elements** attribuut van dit hoogste niveau element bevat een lijst van elementen waarvan de tekstknooppunten die alleen maar witruimte bevatten verwijderd moeten worden. Een voorbeeld:

Voorbeeld 5.82. Het xsl:strip-space element

```
<xsl:strip-space elements="SLIDE LINE"/>
```

Je kan al de knooppunten die alleen maar witruimte bevatten uit het document verwijderen als volgt:

Voorbeeld 5.83. Verwijderen van knooppunten die alleen witruimte bevatten

```
<xsl:strip-space elements="*"/>
```

Er bestaat ook een **xsl:preserve-space** element met een gelijkaardige syntax maar een tegengestelde betekenis. Aangezien de default het behouden van witruimte is, wordt dit

element zelden gebruikt. De belangrijkste bestaansredenen van dit element zijn het overschrijven van **xsl:strip-space** elementen die geïmporteerd worden uit andere stylesheets of het specificeren van enkele elementen waarvan je de witruimte wel wilt behouden als de defaultbehandeling het weghalen van witruimte is.

Tekstknooppunten in de stylesheet die alleen maar witruimte bevatten, worden, in tegenstelling tot de behandeling van zulke knooppunten in het invoerdocument, default verwijderd. Als je zo een knooppunt toch wilt bewaren, dan moet je een **xml:space** attribuut met als waarde **preserve** toevoegen aan het ouderelement of één van de voorouders van dat tekstknooppunt. Het **xml:space** attribuut werd ook al behandeld in de sectie over witruimte in het hoofdstuk over XML [Sectie 2.3.8.1.] . Het is vaak nog het gemakkelijkst om gewoon het **xsl:text** element te gebruiken om betekenisvolle witruimte in een stylesheet te zetten. witruimte binnen een **xsl:text** element wordt, zoals reeds eerder vermeld, letterlijk behandeld en nooit verwijderd.

5.3.1.14. Meerdere stylesheets samenvoegen

Een XML document kan gebruik maken van verschillende voorgedefinieerde markup woordenschatten die aan de hand van XML Schema [Hoofdstuk 3.] beschreven kunnen worden. Voor deze verschillende woordenschatten zullen een aantal standaard stylesheets geïmplementeerd zijn. Natuurlijk wil je om de hoeveelheid werk aan je eigen stylesheets te verkleinen zoveel mogelijk gebruik maken van deze standaard stylesheets die bij een woordenschat horen. Aan de andere kant wil je ook regels toevoegen die van toepassing zijn op je eigen specifieke XML document. Het **xsl:import** en het **xsl:include** element maken het mogelijk om meerdere stylesheets samen te voegen, zodat stylesheets kunnen hergebruikt worden voor verschillende woordenschatten en verschillende doeleinden.

5.3.1.14.1. Het xsl:import element

Het **xsl:import** element is een hoogste niveau element met een **href** attribuut dat de URI bevat van de stylesheet die geïmporteerd moet worden. Alle **xsl:import** elementen moeten vóór ieder ander kindelement van het **xsl:stylesheet** rootelement staan. Een voorbeeld:

Voorbeeld 5.84. Gebruik van het xsl:import element

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:import href="html/xml.xsl"/>
     <xsl:import href="html/bib.xsl"/>
     <xsl:import href="html/dtd.xsl"/>
     <!--De andere kindelementen van xsl:stylesheet-->
  </xsl:stylesheet>
```

Het is mogelijk dat regels in de geïmporteerde stylesheet conflicteren met regels in de importerende stylesheet. Er kan bijvoorbeeld twee keer hetzelfde element gematcht worden. In dit geval hebben de regels in de importerende stylesheet voorrang. Als twee regels in verschillende geïmporteerde stylesheets conflicteren, dan heeft de regel in de laatst geïmporteerde stylesheet voorrang.

Het **xsl:apply-imports** element is een variant van het **xsl:apply-templates** element dat alleen gebruikt wordt voor geïmporteerde regels. Het gebruikt geen regels van de importerende stylesheet. Dit element laat toe geïmporteerde regels te gebruiken die anders overschreven

zouden worden door regels in de importerende stylesheet. Indien een template met hetzelfde **match** attribuut in verschillende geïmporteerde stylesheets voorkomt, wordt door het **xsl:apply-imports** element de template in laatst geïmporteerde stylesheet gebruikt. Er is geen manier om aan te geven dat in een specifieke geïmporteerde stylesheet naar een template gezocht moet worden. De syntax van het **xsl:apply-imports** element is identiek aan die van het **xsl:apply-templates** element.

5.3.1.14.2. Het xsl:include element

Het **xsl:include** element is een hoogste niveau element dat een andere stylesheet in de huidige stylesheet kopieert op de plaats waar het voorkomt. Dit wil zeggen dat het **xsl:include** element de inhoud van het **xsl:style-sheet** of het **xsl:transform** element van het opgegeven document kopieert in het huidige document. Het **href** attribuut levert de URI van de te kopiëren stylesheet. Een **xsl:include** element is een hoogste niveau kindelement van **xsl:style-sheet** dat eender waar kan voorkomen, maar altijd achter het laatste **xsl:import** element moet komen.

In tegenstelling tot **xsl:import** elementen, hebben de template regels die in het document opgenomen worden door **xsl:include** dezelfde voorrangsregels als zouden ze gewoon gekopieerd en geplakt worden van één stylesheet naar een andere. Voor de XSLT processor is er geen verschil tussen een regel die met **xsl:include** opgenomen is in de stylesheet en een regel die effectief in de stylesheet staat.

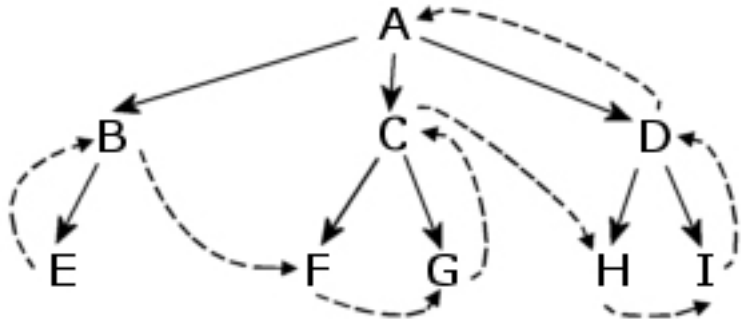
5.3.1.14.3. Voorrangsregels

De **xsl:stylesheet** elementen die de XSLT processor tegenkomt tijdens het verwerken van een stylesheet met **xsl:import** elementen, worden gebruikt voor het opbouwen van een "import-boom". In de import-boom heeft elk **xsl:stylesheet** element één import-kind voor elk **xsl:import** element dat het bevat. Voordat de import-boom kan opgebouwd worden, moeten eerst alle **xsl:include** elementen verwerkt worden. Een **xsl:stylesheet** element in de import-boom heeft een lagere graad van belangrijkheid dan een ander **xsl:stylesheet** element in de import-boom, indien dit element eerder dan het andere **xsl:stylesheet** element bezocht wordt bij het van links naar rechts en van onder naar boven (post-order) doorlopen van de import-boom. Bij deze manier van doorlopen van de import-boom wordt elk **xsl:stylesheet** element bezocht **na** zijn import-kinderen. Elke definitie en elke template regel heeft dezelfde graad van belangrijkheid als de **xsl:stylesheet element** die het bevat.

We illustreren dit met een voorbeeld.

- stylesheet A importeert de stylesheets B, C en D in die volgorde
- stylesheet B importeert stylesheet E
- stylesheet C importeert stylesheet F en G
- stylesheet D importeert stylesheet H en I

De opgebouwde overervingsboom ziet er als volgt uit:



Figuur 5.2. De opgebouwde importboom

Merk op dat wij hier de meest specifieke stylesheet vanboven geplaatst hebben, en niet vanonder zoals je dat misschien gewoon bent in andere overervingsstructuren.

Er ontstaat de volgende ordening naar stijgende graad van belangrijkheid: E, B, F, G, C, H, I, D, A.

In het algemeen heeft een template regel met een hogere graad van belangrijkheid voorrang op een definitie of template regel met een lagere graad van belangrijkheid. Een stylesheet die zichzelf direct of indirect importeert wordt niet toegestaan. Het geval waarin een bepaalde stylesheet op meerdere plaatsen wordt geïmporteerd wordt niet op een speciale manier behandeld. De import-boom zal gewoon een afzonderlijk **xsl:stylesheet** element hebben voor elke plaats waar de stylesheet wordt geïmporteerd.

Voor een vergelijking tussen het overervingsmechanisme van Java en het in deze sectie besproken gebruik van **xsl:import** en **xsl:include**, verwijzen wij naar .

5.3.1.15. Uitvoermethoden

De voorbeelden die we tot nu toe in onze tekst aangehaald hebben, zijn vooral bedoeld voor het omzetten van XML naar goedgevormde HTML. De meeste XSLT processoren ondersteunen echter drie verschillende uitvoermethoden: XML, HTML en tekst. De XSLT processor reageert anders afhankelijk van welke van deze uitvoermethoden er gebruikt wordt.

Het **XML** formaat is de default uitvoermethode en eigenlijk het eenvoudigst, omdat de uitvoer meestal exact is wat je opgelegd hebt in je stylesheet. Aangezien het in goedgevormde XML niet toegestaan is kleiner-dan tekens en ampersands op te nemen, worden bij de omzetting van één XML document naar een ander de entiteitsreferenties < en & behouden, ze worden dus niet omgezet naar < en &. Er bestaan echter manieren om dit te omzeilen [Sectie 2.3.6.] .

De **HTML** uitvoermethode is ontworpen om standaard HTML 4.0 [TRHTML] te produceren. Deze standaard is verschillend van de goedgevormde HTML die wij altijd gebruiken in onze tekst. Lege tags zien er uit als
 , processing instructions worden afgesloten met een > in plaats van een ?> en < tekens die gebruikt worden in JavaScript worden niet omgezet naar <. Dit maakt het gemakkelijker om HTML te produceren die werkt onder verschillende browsers en platformen, zonder de rare effecten die ontstaan wanneer HTML in een XML harnas gedwongen wordt. De HTML uitvoermethode wordt automatisch geselecteerd zodra de formatter bemerkt dat het rootelement van de uitvoer één of andere afkorting (niet

hoofdlettergevoelig) van Hypertext Markup Language is: html, HTML, HtMl, enzovoort.

De laatste uitvoermethode is gewoon **tekst**. De tekst uitvoermethode bouwt eerst een volledige resulterende boom op net zoals de XML uitvoermethode, maar geeft dan alleen de stringwaarde van die boom terug. Dit is nuttig om niet-XML formaten zoals RTF [RTF] of TeX [TUG] te transformeren. Het grootste voordeel van het tekst uitvoerformaat is dat kleiner-dan tekens en ampersands niet omgezet worden naar < en &. Dit maakt het mogelijk om effectief willekeurige tekst te produceren.

5.3.1.15.1. Het xsl:output element

Default zal een XSLT processor de XML uitvoermethode gebruiken, tenzij hij het rootelement van de uitvoer herkent als HTML. Je kan dit aanpassen door het hoogste niveau element **xsl:output** te gebruiken. Het **method** attribuut van het **xsl:output** element duidt aan welke uitvoermethode gebruikt moet worden en kan normaal de volgende drie waardes aannemen: **xml**, **html** of **text**. Sommige XSLT processoren ondersteunen ook andere waardes (alhoewel er maar drie door de W3C standaard zijn toegestaan), maar de meeste doen dit niet.

Het **xsl:output** element beschikt ook over een aantal andere toegestane attributen die de uitvoer bepalen. Deze maken het mogelijk de proloog van het document te veranderen, de uitvoer te indenteren met betekenisvolle witruimte en te bepalen welke elementen CDATA secties gebruiken in plaats van < en & tekens te escapen. Deze attributen worden in volgende secties behandeld.

5.3.1.15.1.1. Aanpassen van de XML declaratie

De attributen **omit-xml-declaration**, **version**, **standalone** en **encoding** beïnvloeden de XML declaratie, dit op voorwaarde dat de uitvoermethode XML is natuurlijk.

Het **omit-xml-declaration** attribuut heeft waarde **yes** of **no**. De defaultwaarde is **no**. Als de waarde **yes** is, wordt er geen XML declaratie opgenomen in het document, in het andere geval wel. Dus om expliciet een XML declaratie op te nemen in je uitvoerdocument, voeg je het volgende element toe op het hoogste niveau van je stylesheet:

Voorbeeld 5.85. Gebruik van het omit-xml-declarationattribuut van xsl:output

```
<xsl:output method="xml" omit-xml-declaration="no"/>
```

Het is ook mogelijk om **method** en **omit-xml-declaration** in twee aparte **xsl:output** elementen op te nemen.

De default waarde van het **version** attribuut van de XML declaratie is 1.0. Dit is op dit moment ook de enige toegestane waarde. Als dit in de toekomst zal veranderen, laat het **version** attribuut van het **xsl:output** element toe om de versie gebruikt in de XML declaratie te veranderen op volgende wijze:

Voorbeeld 5.86. Gebruik van het version attribuut van xsl:output

```
<xsl:output version="1.1"/>
```

Het **standalone** attribuut van de XML declaratie kan op de waarde **yes** of **no** gezet worden

aan de hand van het **standalone** attribuut van het **xsl:output** element. Als het **standalone** attribuut van de XML declaratie de waarde **yes** heeft, wil dit zeggen dat indien er bij het begin van het XML document naar een uitwendige DTD verwezen wordt, de XSLT processor deze niet moet verwerken.

Voorbeeld 5.87. Gebruik van het standalone attribuut van xsl:output

```
<xsl:output method="xml" omit-xml-declaration="no" standalone="yes"/>
```

Het **encoding** attribuut van het **xsl:output** element dient om het **encoding** attribuut van de XML declaratie aan te passen. De waarde kan elke legale coderingsnaam zijn die geregistreerd is bij de Internet Assigned Numbers Authorithy [IANA]. Een voorbeeld waarbij het XML uitvoerdocument een ISO-8859-15 codering heeft.

Voorbeeld 5.88. Gebruik van het encoding attribuut van xsl:output

```
<xsl:output method="xml" omit-xml-declaration="no"
encoding="ISO-8859-15"/>
```

Dit verandert ook de default UTF-8 codering die de XSLT processor normaal gebruikt voor het uitvoerdocument. Opgelet echter, niet alle processoren ondersteunen alle mogelijke coderingen!

5.3.1.15.1.2. Indentatie

Als de meeste witruimte in je uitvoerdocument geen specifieke betekenis heeft, is het mogelijk ervoor te zorgen dat de gegenereerde XML even mooi geordend is als het handgecodeerde invoerdocument. In een mooi geordend document wordt de nesting van de verschillende elementen aangegeven door de indentatie. We moeten dus extra witruimte aan het uitvoerdocument gaan toevoegen om dit te bereiken. Als de waarde van het attribuut **indent** van **xsl:output** op **yes** gezet wordt, mag (dit is wel geen vereiste) de processor extra witruimte toevoegen aan de uitvoer. De processor mag echter in geen geval witruimte uit het document verwijderen. De default waarde van het **indent** attribuut is **no**.

Het is echter niet mogelijk om aan te geven hoe diep je wilt dat ieder niveau inspringt. Dit wordt volledig overgelaten aan de XSLT processor. Het **indent** attribuut van **xsl:output** zorgt ervoor dat de gegenereerde uitvoercode bijna net zo aantrekkelijk is als handgecodeerde code.

5.3.1.15.1.3. CDATA secties

In standaard XSLT is het niet toegestaan om CDATA secties toe te voegen op willekeurige plaatsen in een XML document dat gegenereerd is door een XSL transformatie. Je kan echter wel aangeven dat de tekstinhoud van een bepaald element in een CDATA sectie geplaatst moet worden. In dit geval moeten de < en & tekens binnen de CDATA sectie niet gecodeerd worden als < en & zoals dat gewoonlijk het geval is.

In onderstaand voorbeeld wordt de tekstinhoud van het **SCRIPT** en het **CODE** element omgeven door een CDATA sectie, door aan de waarde van het **cdata-section-elements** attribuut van **xsl:output** de naam van deze twee elementen mee te geven.

Voorbeeld 5.89. Gebruik van het cdata-section-elements attribuut van xsl:output

```
<xsl:output cdata-section-elements="SCRIPT CODE"/>
```

Er zijn nog andere attributen bij het **xsl:output** element. Wij vermelden ze hier voor de volledigheid maar zullen ze niet in detail bespreken. Deze attributen zijn: **doctype-system**, **doctype-public** en **media-type**. Voor hun exacte betekenis verwijzen wij naar de W3C standaard [TRXSL].

5.3.2. Een uitgewerkt voorbeeld

De onderstaande XSL transformatie illustreert hoe een slideshow in XML formaat omgezet kan worden naar een slideshow in HTML formaat. Deze stylesheets zijn een onderdeel van de stylesheets die we gebruikt hebben om een website die oorspronkelijk in HTML was geschreven om te zetten naar een website die met XML en XSL werkte.

Om duidelijk te maken voor welk invoerdocument we deze stylesheets geschreven hebben, geven we eerst een verkorte versie van de slideshow in XML formaat.

Voorbeeld 5.90. slides.xml

```
<?xml version="1.0"?>
 <SLIDESHOW>
    <SLIDE>
       <TITLE> Introduction </TITLE>
    </SLIDE>
    <SLIDE>
       <TITLE> It is Not the Language </TITLE>
       <LINE>
          <TEXT> This course is not about an object oriented programming
          language </TEXT>
          <LINE>
             <TEXT> Different OOP languages all offer more or less the
             same technical concepts </TEXT>
          </LINE>
       </LINE>
       <LINE>
          <TEXT> Knowing the syntax of an OOP language doesn't make you an
          OO programmer </TEXT>
          <LINE>
             <TEXT> Languages are the playground of low-level techies
             </TEXT>
             <LINE>
                <TEXT> interested very much in performance </TEXT>
             </LINE>
             <LINE>
                <TEXT> if it is possible, we need it </TEXT>
             </LINE>
          </LINE>
          <LINE>
             <TEXT> There are criteria for the quality of software much
             more important than performance </TEXT>
             <LINE>
                <TEXT> correctness </TEXT>
             </LINE>
             <LINE>
                <TEXT> adaptability, extensibility, stability under changes
                </TEXT>
             </LINE>
             <LINE>
                <TEXT> reusability </TEXT>
             </LINE>
```

```
            <LINE>
               <TEXT> robustness </TEXT>
            </LINE>
            <LINE>
               <TEXT> ... </TEXT>
            </LINE>
         </LINE>
         <LINE>
            <TEXT> Most important criteria are born out of the highest
            cost for software: the human developers </TEXT>
            <LINE>
               <TEXT> far more costly than faster hardware </TEXT>
            </LINE>
         </LINE>
      </LINE>
   </SLIDE>
   <SLIDE>
      <TITLE> OOP Craftsmanship </TITLE>
      <LINE>
         <TEXT> Over the years, we have learned ... </TEXT>
         <LINE>
            <TEXT> Which techniques offered by OOP languages can be used
            safely </TEXT>
         </LINE>
         <LINE>
            <TEXT> Which techniques offered by OOP languages we should
            avoid using </TEXT>
            <LINE>
               <TEXT> although they are technically feasible </TEXT>
            </LINE>
         </LINE>
         <LINE>
            <TEXT> How to use the techniques offered by OOP languages in
            a good way </TEXT>
            <LINE>
               <TEXT> most of the offered techniques can be used in a bad
               way </TEXT>
            </LINE>
         </LINE>
      </LINE>
      <LINE>
         <TEXT> A programming language is a sharp tool </TEXT>
         <LINE>
            <TEXT> You can do beautiful things with it, if you know how
            to use the tool correctly </TEXT>
         </LINE>
         <LINE>
            <TEXT> You will hurt yourself if you use the tool without
            proper training </TEXT>
         </LINE>
         <LINE>
            <TEXT> Craftsmanship </TEXT>
         </LINE>
      </LINE>
   </SLIDE>
</SLIDESHOW>
```

We hebben eerst in een aparte stylesheet **general.xsl** de template **HEADHTML** gedefinieerd, omdat dezelfde heading nog op enkele andere plaatsen in de stylesheets van de site gebruikt werd.

Voorbeeld 5.91. general.xsl

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
```

```
  ="http://www.w3.org/1999/XSL/Transform">
     <xsl:template name="HEADHTML">
        <head>
           <TITLE> OOP with Java - Parts </TITLE>
           <LINK rel="stylesheet" href="support/javacoursesite.css"/>
        </head>
     </xsl:template>
  </xsl:stylesheet>
```

In onderstaande stylesheet wordt dan de HTML slideshow opgebouwd met behulp van de parameter **slide**. De waarde van de **slide** parameter komt overeen met de positie van het **SLIDE** element binnen het **SLIDESHOW** element. Op elke slide die opgebouwd wordt, verschijnt een link naar de vorige slide, de index (overzicht van alle slides opgebouwd aan de hand van de mode **toc**) en de volgende slide.

Je zal bemerken dat we geen consistente werkwijze hebben gevolgd bij het gebruik van elementen en attributen, dit hebben we gedaan om te illusteren dat zowel de korte als de uitgebreide versie met **xsl:element** en **xsl:attribute** mogelijk zijn in vele gevallen. We willen er wel op wijzen dat wanneer er binnen een element of een attribuut gebruikt gemaakt wordt van XSLT instructies de uitgebreide vorm noodzakelijk is.

Voorbeeld 5.92. slides.xsl

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:include href="general.xsl"/>
     <xsl:param name="slide"/>
     <xsl:template match="SLIDESHOW">
        <html>
           <xsl:call-template name="HEADHTML"/>
           <body>
              <xsl:if test="not($slide)">
                 <TABLE width="100%">
                    <TH>
                       <xsl:value-of select="TITLE"/>
                       <BR/>
                       <xsl:value-of select="DATE"/>
                       <BR/>
                    </TH>
                    <TR>
                       <xsl:element name="TD">
                          <xsl:attribute name="valign"> top </xsl:attribute>
                          <apply-templates select="SLIDE" mode="toc"/>
                       </xsl:element>
                       <xsl:element name="TD">
                          <xsl:attribute name="valign"> top </xsl:attribute>
                          <apply-templates
                          select="child::*[local-name()='AUTHOR' or
                          local-name()='EMAIL' or local-name()='HOMEPAGE']"/>
                       </xsl:element>
                    </TR>
                 </TABLE>
              </xsl:if>
              <xsl:if test="$slide">
                 <TABLE width="100%">
                    <TR>
                       <TD>
                          <xsl:if test="$slide > 1">
                             <xsl:element name="a">
                                <xsl:attribute name="href"> slides.xml?slide=
```

```
                        <xsl:value-of select="$slide - 1"/>
                      </xsl:attribute>
                    Vorige </xsl:element>
                  </xsl:if>
                </TD>
                <TD>
                  <a href="slides.xml"> Index </a>
                </TD>
                <TD>
                  <xsl:if test="$slide < count(SLIDE)">
                    <xsl:element name="a">
                      <xsl:attribute name="href"> slides.xml?slide=
                        <xsl:value-of select="$slide + 1"/>
                      </xsl:attribute>
                    Volgende </xsl:element>
                  </xsl:if>
                </TD>
              </TR>
            </TABLE>
            <xsl:apply-templates select="SLIDE[position()=$slide]"
            mode="full"/>
          </xsl:if>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="SLIDE" mode="toc">
      <xsl:element name="A">
        <xsl:attribute name="HREF"> slides.xml?slide=
          <xsl:value-of select="select="position()"/>
        </xsl:attribute>
        <xsl:value-of select="TITLE"/>
      </xsl:element>
      <BR/>
    </xsl:template>
    <xsl:template match="HOMEPAGE">
      <b> Homepage: </b>
      <xsl:element name="A">
        <xsl:attribute name="HREF">
          <xsl:value-of select="."/>
        </xsl:attribute>
        <xsl:value-of select="."/>
      </xsl:element>
      <BR/>
    </xsl:template>
    <xsl:template match="AUTHOR">
      <b> Author: </b>
      <xsl:value-of select="."/>
      <BR/>
    </xsl:template>
    <xsl:template match="EMAIL">
      <b> Email: </b>
      <xsl:element name="A">
        <xsl:attribute name="HREF"> mailto:
          <xsl:value-of select="."/>
        </xsl:attribute>
      </xsl:element>
      <BR/>
    </xsl:template>
    <xsl:template match="SLIDE" mode="full">
      <H1>
        <xsl:value-of select="TITLE"/>
      </H1>
      <ul>
        <xsl:apply-templates select="LINE"/>
      </ul>
    </xsl:template>
    <xsl:template match="LINE">
      <xsl:element name="LI">
        <xsl:apply-templates select="TEXT"/>
```

```
        </xsl:element>
        <ul>
          <xsl:apply-templates select="LINE"/>
        </ul>
      </xsl:template>
      <xsl:template match="TEXT">
        <xsl:apply-templates>
          <!--gebruik maken van de standaard templates-->
        </xsl:apply-templates>
      </xsl:template>
      <xsl:template match="EMPH">
        <b>
          <xsl:value-of select="."/>
        </b>
      </xsl:template>
      <xsl:template match="PARADIGM">
        <font color="green">
          <xsl:value-of select="."/>
        </font>
      </xsl:template>
    </xsl:stylesheet>
```

5.4. XPath

5.4.1. Wat is XPath?

XPath [TRXPA] is het resultaat van een poging om een gemeenschappelijke syntax en semantiek te ontwikkelen voor de functionaliteit die XSLT en XPointer delen. XPointer specificeert een taal die adressering in de interne structuren van XML documenten mogelijk maakt. XPath is ontstaan uit de gemeenschappelijke behoefte van XPointer [TRXPOINTER] en XSLT om verschillende delen van een document te kunnen adresseren. XSLT moest het deel van het document dat getransformeerd zou worden kunnen selecteren en XPointer moest twee documenten kunnen linken.

Het primaire doel van XPath is het adresseren van delen van een XML document. Door XPath te gebruiken kunnen we de plaats van bepaalde documentstructuren of data in een XML document aanduiden en dan deze informatie verwerken aan de hand van XSLT. XPath heeft ook enkele basisfaciliteiten om strings, getallen (numbers) en booleans te manipuleren. XPath maakt gebruik van een compacte, niet-XML syntax om het gebruik van XPath expressies als XSL attribuutwaardes (van bijvoorbeeld het attribuut **select**) of als URI's te vergemakkelijken. XPath maakt gebruik van de boomstructuur van een XML document. De naam XPath komt van het gebruik van een padnotatie zoals in URL's om door de hiërarchische structuur van een XML document te navigeren.

Naast het gebruik van XPath voor adressering, is XPath ook ontworpen zodat het beschikt over een subset die gebruikt kan worden om te matchen (testen of een knooppunt aan een bepaald patroon voldoet). Dit deel van XPath wordt beschreven in de XSLT aanbeveling [TRXSLT]. Dit verklaart waarom, alhoewel XPath en XSLT ontwikkeld zijn als twee onafhankelijke W3C standaarden, het vaak moeilijk is te onderscheiden waar XSLT stopt en waar XPath begint. De subset van XPath die gebruikt wordt om te matchen, hebben wij reeds eerder in deze tekst besproken. [Sectie 5.3.1.2.1.]

De primaire syntactische bouwsteen van XPath is de expressie. XPath expressies zijn zoals reeds eerder vermeld een superset van de patronen die gebruikt worden om te matchen. Dit wil zeggen dat alle matchpatronen expressies zijn, maar dat niet alle expressies

matchpatronen zijn. Matchpatronen maken het mogelijk knooppunten te matchen op hun naam, op de naam van hun kinderen, op hun attributen enzovoort. XPath expressies breiden deze mogelijkheden nog uit: je kan een meer ingewikkeld pad in de boomstructuur bewandelen om een knooppunt te selecteren door te verwijzen naar voorouderknooppunten, broer- of zusterknooppunten, enzovoort. XPath expressies kunnen ook booleans, getallen en strings teruggeven.

Waar wordt XPath nu concreet gebruikt? De waarde van het **select** attribuut dat gebruikt wordt in **xsl:apply-templates**, **xsl:value-of**, **xsl:for-each**, **xsl:copy-of**, **xsl:variable**, **xsl:param** en **xsl:sort** is een expressie geschreven in de XPath taal. De waarde van het **match** attribuut dat gebruikt wordt in **xsl:template** is op zijn beurt ook een XPath expressie.

In onderstaande secties bespreken we achtereenvolgens de context van een XPath expressie, locatiepaden en ten slotte XPath expressies zelf. Om te kunnen navigeren door de boom van het XML document is het belangrijk nooit uit het oog te verliezen wat de context is waarin je je op dat moment bevindt. Locatiepaden zijn een onderdeel van XPath expressies die alleen knooppunten als waarde kunnen teruggeven. Omdat dit zo'n belangrijk onderdeel vormt van XPath, bespreken we ze in een aparte sectie.

5.4.2. De context

Als we met XSLT werken, is de context van een query het knooppunt in het brondocument dat op dat moment verwerkt wordt. Anders gezegd: de context wordt geïnitialiseerd op de knoop die op dat moment door de template gematcht wordt.

Als we kijken naar de volgende matchuitdrukking:

Voorbeeld 5.93.

```
<xsl:template match="/">
   <!-- Inhoud van de template. -->
</xsl:template>
```

Op dit moment bevinden we ons in de context van de root van het XML document. Opgelet, dit is niet het rootelement, maar het **hele** document.

Nog een voorbeeldje:

Voorbeeld 5.94.

```
<xsl:for-each select="MENUITEM">
   <!-- Inhoud van het xsl:for-each element. -->
</xsl:for-each>
```

In een **xsl:for-each** lus is de context het knooppunt (node) waarover we op dit moment aan het itereren zijn.

Het is heel belangrijk dat je altijd goed weet in welke context je aan het werken bent. Als je je XSL stylesheet aan het opstellen of debuggen bent, is de eerste vraag die je jezelf moet stellen: *"Welke context wordt hier verwerkt?"*.

5.4.3. Locatiepaden

Locatiepaden zijn niet de meest algemene grammaticale bouwstenen in XPATH, ze vormen echter wel de meest belangrijke bouwstenen. Een locatiepad is een speciaal geval van een XPath expressie waarvan het resultaat altijd een knooppunt is. Elk locatiepad kan beschreven worden aan de hand van een recht toe recht aan syntax, die echter nogal langdradig is. Daarom bestaat er ook een afgekorte syntax. Je bent volledig vrij om te kiezen welke syntax jouw voorkeur wegdraagt.

Er zijn twee soorten locatiepaden: relatieve locatiepaden en absolute locatiepaden. Een relatief locatiepad bestaat uit een opeenvolging van één of meer locatiestappen, gescheiden door de hiërarchie-operator /. De stappen in een relatief locatiepad zijn samengesteld van links naar rechts. Elke stap selecteert op zijn beurt een set van knooppunten relatief ten opzichte van een context knooppunt. De opeenvolgende stappen worden op de volgende manier samengenomen:

- De initiële stap selecteert een set van knooppunten relatief ten opzichte van een context knooppunt.
- Elk knooppunt in die set wordt vervolgens gebruikt als een context knooppunt voor de volgende stap.
- De set van knooppunten die het resultaat zijn van het toepassen van de volgende stap worden samengevoegd en vormen het resultaat van de twee opeenvolgende locatiestappen

Deze procedure wordt herhaald tot er geen locatiestappen meer overblijven in het locatiepad.

Een absoluut locatiepad bestaat uit een /, eventueel gevolgd door een relatief locatiepad. Een / alleen selecteert, zoals we reeds eerder gezien hebben, het rootknooppunt van het document dat het context knooppunt bevat. Als de / gevolgd wordt door een relatief locatiepad, wordt de set van knooppunten teruggegeven die zouden geselecteerd worden door het relatieve locatiepad ten opzichte van het rootknooppunt.

Enkele voorbeelden van locatiepaden:

- **ancestor::CHAPTER** selecteert alle **CHAPTER** voorouderknooppunten van het context knooppunt
- **child::CHAPTER/descendant::PARA** selecteert de **PARA** elementen die nakomelingen zijn van de **CHAPTER** kinderen van het context knooppunt
- **child::*/child::PARA** selecteert alle **PARA** kleinkinderen van het context knooppunt
- **child::para[position()=1]** selecteert het eerste **PARA** kindknooppunt van het context knooppunt
- **child::*[self::CHAPTER or self::APPENDIX]** selecteert de **CHAPTER** en **APPENDIX** kinderen van het context knooppunt
- **following-sibling::chapter[position()=1]** selecteert het volgende **CHAPTER** broer- of zusterknooppunt van het context knooppunt

In de onderstaande secties wordt uitgelegd hoe bovenstaande voorbeelden opgebouwd worden.

5.4.3.1. Locatiestappen

Een locatiestap bestaat uit drie delen:

- Een axis: geeft de relatie aan tussen de knooppunten geselecteerd door de locatiestap en de context in de boom
- Een knooppunttest: geeft het knooppunttype en de uitgebreide naam (expanded-name) van de knopen geselecteerd door de locatiestap
- Nul of meer predikaten: gebruiken willekeurige uitdrukkingen om de set van knopen

geselecteerd door de locatiestap te verfijnen

De syntax voor een locatiestap ziet er uit als volgt:

naam_van_de_axis::knooppunttest[nul_of_meer_expressies]

Laten we dit illustreren met een voorbeeldje:

child::PARA[position()=1]

child is de naam van de axis, PARA is de knooppunttest en [position()=1] is een predikaat. Wat de exacte betekenis van elk van deze onderdelen is, wordt hieronder uitgelegd.

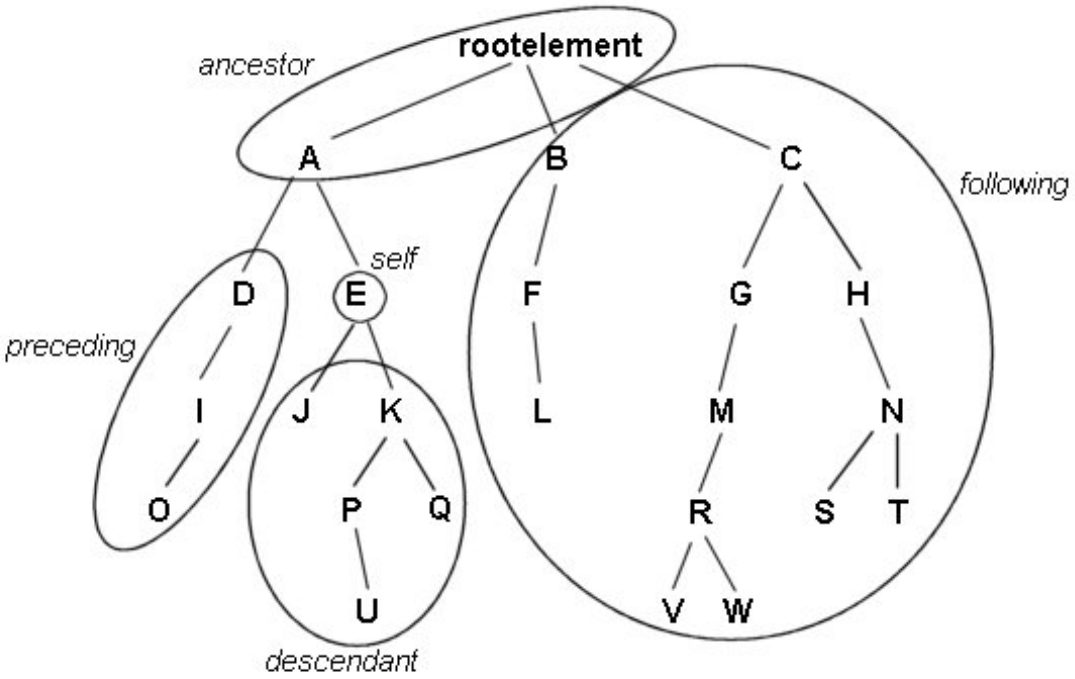
5.4.3.1.1. Axes

XPath heeft een aantal axes die je kan gebruiken om te selecteren uit verschillende delen van de boom relatief ten opzichte van de context. Hieronder geven wij een samenvatting van de verschillende mogelijkheden en hun betekenis:

Tabel 5.2. expressie Axes

Axis	Selecteert
`ancestor`	De ouder (parent) van het context knooppunt, de ouder van de ouder van de context, de ouder van de ouder van de ouder van de context, enzovoort
`ancestor-or-self`	De voorouders (ancestor) van het context knooppunt en het contextknooppunt zelf
`attribute`	Het attribuut van het context knooppunt, in het geval van meerdere attributen wordt alleen het eerste attribuut geselecteerd
`child`	De onmiddellijke kinderen van het context knooppunt
`descendant`	De kinderen van het context knooppunt, de kinderen van de kinderen van het context knooppunt, enzoverder
`descendant-or-self`	Het context knooppunt zelf en zijn afstammelingen (descendants)
`following`	Alle knooppunten die volgen na het context knooppunt (bekeken in de volgorde waarin de knooppunten voorkomen in het document), uitgezonderd nakomelingen van het context knooppunt, attribuut- en namespace knooppunten
`following-sibling`	Alle knooppunten die volgen na het context knooppunt en die dezelfde ouder (parent) hebben als het context knooppunt
`namespace`	De namespace van de context
`parent`	Het unieke ouderknooppunt van het context knooppunt
`preceding`	Alle knooppunten die vóór het context knooppunt staan (bekeken in de volgorde waarin de knooppunten voorkomen in het document), uitgezonderd voorouders van het context-knooppunt, attribuut- en namespace knooppunten
`preceding-sibling`	Alle knooppunten die vóór het begin van het context knooppunt staan en die dezelfde ouder hebben als het context knooppunt
`self`	De context zelf

De **ancestor**, **descendant**, **following**, **preceding** en **self** axes partitioneren een document (attribuut en namespace knooppunten worden genegeerd). Deze axes bevatten geen gemeenschappelijke knooppunten en samen bevatten ze alle knooppunten van het document. We illustreren dit in onderstaande figuur:



Figuur 5.3. Het partitioneren van de boom van het document

Het kiezen van een axis beperkt de verdere verwerking van de XPath expressie tot de set van knooppunten die beschreven zijn in bovenstaande tabel. Zoals reeds eerder vermeld wordt een axis meestal gevolgd door een "::" symbool en een knooppunttest die de knooppuntset nog verder beperkt. Het is echter ook perfect mogelijk gewoon de axes op zichzelf te gebruiken om knooppunten te selecteren. We illustreren dit met enkele voorbeelden.

Voorbeeld 5.95. Gebruik van axes

```
<xsl:template match="CHAPTER">
   <H1>
      <xsl:value-of select="child::TITLE"/>
   </H1>
   <xsl:apply-templates select="child::SECTION"/>
</xsl:template>
```

Deze template regel matcht **CHAPTER** elementen. Wanneer een **CHAPTER** element gematcht wordt, is dit element het context knooppunt. Alle **TITLE** en **SECTION** elementen worden vervolgens geselecteerd uit de verzameling van alle kinderen van het context knooppunt **CHAPTER**. Als er meer dan één **TITLE** element aanwezig is onder het **CHAPTER** element, dan worden al deze elementen geselecteerd, maar alleen de waarde van het eerste element wordt teruggegeven.

De **child** axis in bovenstaand voorbeeld is niet echt interessant, omdat deze axis in de verkorte vorm [Sectie 5.4.3.1.4.] gewoon weggelaten kan worden. Laten we daarom naar een ander voorbeeldje kijken waar de waarde van het **TITLE** kind van het voorouderknooppunt **CHAPTER** wordt geselecteerd:

Voorbeeld 5.96. Gebruik van de ancestor axis

```
<xsl:template match="SECTION">
   <H3>
      <xsl:value-of select="ancestor::CHAPTER/child::TITLE"/>
   </H3>
</xsl:template>
```

Merk op dat in dit voorbeeld gebruik gemaakt wordt van twee locatiestappen, met name **ancestor::CHAPTER** en **child::TITLE**, gescheiden door de hiërarchie-operator /. Deze beide locatiestappen samen vormen het locatiepad.

De andere axes kennen een gelijkaardig gebruik.

5.4.3.1.2. Knooppunttesten

De axis kan in plaats van door de naam van een knooppunt, ook gevolgd worden door één van de volgende knooppunttype functies:

- **comment()**
- **text()**
- **processing-instruction()**
- **node()**

Deze functies selecteren knooppunten op hun knooppunttype. De **comment()** functie selecteert een commentaarknooppunt. De **text()** functie selecteert een tekstknooppunt. De **processing-instruction()** functie selecteert een processing instruction knooppunt en de **node()** functie selecteert eender welk soort van knooppunt. De **node()** functie is verschillend van de * wild card, omdat deze wild card alleen elementknooppunten kan selecteren, terwijl de node() functie ook commentaar, processing-instruction, namespace, enzovoort knooppunten selecteert. De **processing-instruction()** functie kan een optioneel argument bevatten dat de naam van de te selecteren processing instruction aangeeft.

Een voorbeeld:

Voorbeeld 5.97. Gebruik van knooppunttesten

```
<xsl:template match="SECTION">
   <xsl:value-of select="descendant-or-self::comment()"/>
</xsl:template>
```

In dit voorbeeld wordt alle commentaar van zowel het **SECTION** element als al zijn nakomelingen geselecteerd.

5.4.3.1.3. Predikaten

Een predikaat filtert een set van knooppunten om een nieuwe knooppuntenset te bekomen. Voor elk knooppunt in de set van knooppunten wordt de predikaattest geëvalueerd met het te evalueren knooppunt als context knooppunt. Als de predikaattest waar is voor een knooppunt, wordt dat knooppunt opgenomen in de nieuwe set van knooppunten, als het onwaar is niet.

Een voorbeeldje:

Voorbeeld 5.98.

```
<xsl:template match="SLIDESHOW">
   <html>
      <body>
         <apply-templates select="child::*[local-name()='AUTHOR' or
         local-name()='EMAIL' or local-name()='HOMEPAGE'] "/>
      </body>
   </html>
</xsl:template>
```

In dit voorbeeld worden alle kindelementen van het **SLIDESHOW** element geselecteerd die als lokale naam **AUTHOR**, **EMAIL** of **HOMEPAGE** hebben. Op de functie **local-name()** komen we later nog terug. [Sectie 5.4.4.1., Tabel 5.4.]

5.4.3.1.4. Verkorte syntax

De verschillende axes die in de tabel [Sectie 5.4.3.1.1., Tabel 5.2.] opgesomd zijn, worden in de praktijk niet zo vaak gebruikt, omdat ze nogal veel typewerk vergen. XPath definieert ook een verkorte syntax die een alternatief is voor de meest gebruikte van deze axes. Onderstaande tabel geeft een overzicht van de meest gebruikte axes met hun bijhorende kortere vorm.

Tabel 5.3. Verkorte syntax voor XPath expressies

Volledige vorm	Verkorte vorm
`self::node()`	.
`parent::node()`	..
`child::name`	name
`attribute`	@
`attribute::name`	@name
`/descendant-or-self::node()/`	//

Opgelet! Match patronen [Sectie 5.3.1.2.1.] kunnen alleen de verkorte syntax en de **child** en **attribute** axes gebruiken. De volledige syntax die de axes uit tabel [Sectie 5.4.3.1.1., Tabel 5.2.] gebruiken, kan alleen gebruikt worden in expressies.

We hernemen de voorbeeldjes uit de sectie over axes [Sectie 5.4.3.1.1.] , maar gebruiken nu de verkorte schrijfwijze:

Voorbeeld 5.99. Gebruik van axes, verkorte schrijfwijze

```
<xsl:template match="CHAPTER">
   <H1>
      <xsl:value-of select="TITLE"/>
   </H1>
   <xsl:apply-templates select="SECTION"/>
</xsl:template>
```

Voorbeeld 5.100. Gebruik van de parent axis, verkorte schrijfwijze

```
<xsl:template match="CHAPTER">
   <H3>
```

```
      <xsl:value-of select="../TITLE"/>
    </H3>
  </xsl:template>
```

5.4.4. XPath Expressies

Elke XPath expressie evalueert naar één enkele waarde. De expressies die we tot nu toe behandeld hebben, hadden steeds een set van knooppunten als resultaat. Er zijn echter vijf verschillende mogelijkheden voor het resultaat van een XPath expressie:

- Sets van knooppunten
- Booleans
- Getallen
- Strings
- Resulterende boomfragmenten

5.4.4.1. Sets van knooppunten

Een set van knooppunten is een ongeordende groep van knooppunten uit het invoerdocument. Welke knooppunten in de knooppuntenset zitten, hangt af van het context knooppunt, de axis, de knooppunttest en het predikaat. De XPath expressie **child::CHAPTER/child::SECTION** geeft bijvoorbeeld alle **SECTION** elementen terug die een kind zijn van **CHAPTER** elementen, die op hun beurt weer een kind zijn van het context knooppunt (het gematchte knooppunt).

In onderstaande tabel staan de verschillende functies opgesomd die werken met sets van knooppunten of die sets van knooppunten teruggeven. Alhoewel sommige functies een getal of een string als resultaat teruggeven, worden deze volgens de W3C Recommendation [TRXPA] toch bij de functies voor sets van knooppunten gerekend.

Voor een goed begrip van deze tabel zullen wij eerst kort het begrip context knooppuntenlijst uitleggen. Het context knooppunt is een lid van de context knooppuntenlijst. De context knooppuntenlijst is die groep van elementen die op hetzelfde tijdstip allemaal dezelfde regel matchen. Meestal is zo een knooppuntenlijst het resultaat van een **xsl:apply-templates** of een **xsl:for-each** oproep. Als bijvoorbeeld het **select** attribuut van **xsl:apply-templates** de waarde **PARA** heeft, dan bestaat de context knooppuntenlijst uit al de gematchte **PARA** elementen.

Tabel 5.4. Functies voor sets van knooppunten

Functie	Geeft terug
`position()`	Geeft een getal terug dat de positie van het context knooppunt in de context knooppuntenlijst weergeeft; het eerste knooppunt in de lijst heeft positie 1
`last()`	Geeft een getal terug dat het aantal knooppunten in de context knooppuntenlijkst weergeeft; dit is hetzelfde als de positie van het laatste knooppunt in de lijst
`count(node-set)`	Geeft een getal terug dat het aantal knooppunten in node-set weergeeft
`id(string1 string2 string3)`	Geeft een set van knooppunten terug met al de elementen van het document die een ID hebben waarvan de naam in de argumentenlijst zit; de lege set wordt teruggegeven als er geen ele-

	ment is dat de gespecificeerde ID heeft
`key(String naam, Object waarde)`	Geeft een set van knooppunten terug die alle knooppunten van het document bevat die een sleutel hebben met de gespecificeerde waarde; sleutels worden gezet met het hoogste niveau element **xsl:key**
`local-name(node-set)`	Geeft een string terug die de lokale naam (alles achter het namespace voorvoegsel) van het eerste knooppunt in het node-set argument weergeeft; als de node-set die als argument wordt meegegeven leeg is, wordt een lege string teruggegeven; kan gebruikt worden zonder argument om de lokale naam van het context knooppunt te verkrijgen
`namespace-uri(node-set)`	Geeft een string terug die de URI van de namespace van het eerste knooppunt in de node-set weergeeft; als de node-set die als argument wordt meegegeven leeg is of als het eerste knooppunt in node-set geen gekwalificeerde naam heeft of de namespace URI leeg is, wordt een lege string teruggegeven; kan gebruikt worden zonder argument om de URI van de namespace van het context knooppunt te verkrijgen; geeft een lege string terug als het knooppunt niet in een namespace zit
`name(node-set)`	Geeft een string terug die de gekwalificeerde naam (zowel het voorvoegsel als het lokale deel) van het eerste knooppunt in het node-set argument weergeeft; als de node-set die als argument wordt meegegeven leeg is of als het eerste knooppunt in node-set geen gekwalificeerde naam heeft, wordt een lege string teruggegeven; kan gebruikt worden zonder argument om de gekwalificeerde naam van het context knooppunt te verkrijgen
`generate-id(node-set)`	Geeft een string terug die een unieke identifier is voor het eerste knooppunt in het node-set argument; kan gebruikt worden zonder argument om een ID te genereren voor het context knooppunt

Een voorbeeld van het gebruik van deze functies:

Voorbeeld 5.101. Gebruik van de key() en de generate-id() functie

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:key name="INDICES" match="INDEX" use="."/>
     <xsl:template match="THESIS">
        <html>
           <body>
              <H1> Index </H1>
              <xsl:apply-templates
              select="//INDEX[generate-id(.)=generate-id(key('INDICES',
              .))]" mode="INDEX">
                 <xsl:sort select="."/>
              </xsl:apply-templates>
           </body>
        </html>
     </xsl:template>
     <xsl:template match="INDEX">
        <A name="{generate-id(.)}">
           <xsl:value-of select="."/>
        </A>
     </xsl:template>
     <xsl:template match="INDEX" mode="INDEX">
```

```
      <A href="#{generate-id(.)}">
        <xsl:value-of select="."/>
      </A>
      <BR/>
    </xsl:template>
  </xsl:stylesheet>
```

In bovenstaande stylesheet wordt eerst met behulp van het **xsl:key** element [Sectie 5.3.1.12.] de sleutel met als naam **INDICES** verbonden aan elk **INDEX** element. De waarde van de sleutel is de inhoud van het **INDEX** element.

Als het **THESIS** element gematcht wordt, wordt er een index opgebouwd (denk aan de index achteraan in een boek). Alle **INDEX** nakomelingen van het **THESIS** element waarvan de gegenereerde identifier gelijk is aan de gegenereerde identifier van het eerste element uit de knooppuntenset **key('INDICES', .)**, worden geselecteerd met de mode **INDEX**. De elementen in de set **key('INDICES', .)** zijn alle knooppunten die dezelfde key **INDICES** hebben. Dit komt neer op alle **INDEX** elementen met dezelfde naam. De reden dat we deze vrij ingewikkelde constructie opzetten is om de dubbels uit de lijst van **INDEX** elementen in de index weg te halen. Dit kunnen we op deze manier doen, omdat voor elk **INDEX** element in de tekst, dus ook voor elementen waarvan de inhoud hetzelfde is, een andere identifier wordt gegenereerd.

Voorts wordt er ook nog gebruik gemaakt van het **generate-id()** element bij de twee templates die **INDEX** elementen matchen. Dit om een anker in de tekst te genereren en om een link naar dit anker in de indexlijst op te nemen.

5.4.4.2. Booleans

Een boolean heeft één van twee waardes: waar of onwaar. In XSLT is het mogelijk eender welke soort gegevens om te zetten naar een boolean. Vaak wordt dit impliciet gedaan wanneer een string, een getal of een set van knooppunten gebruikt wordt waar een boolean wordt verwacht. Deze impliciete omzetting gebeurt onder meer bij de waarde van het **test** attribuut van een **xsl:if** element. Deze omzettingen kunnen ook uigevoerd worden door de **boolean()** functie die een argument van eender welk type omzet naar een boolean volgens de volgende regels:

- Een lege set van knooppunten is onwaar. Alle andere sets zijn waar.
- Een getal is onwaar als het nul is of NaN (Not a Number, een symbool gebruikt om het resultaat van de deling door nul of een andere illegale operatie weer te geven). In alle andere gevallen is het getal waar.
- Een string waarvan de lengte nul is, is onwaar. Alle andere strings zijn waar.
- Een leeg fragment van de resulterende boom is onwaar. Alle andere fragmenten zijn waar.

Booleans kunnen ook het resultaat zijn van uitdrukkingen die gebruik maken van de volgende operatoren:

- = gelijk aan
- != niet gelijk aan
- < kleiner dan
- > groter dan
- <= kleiner dan of gelijk aan
- >= groter dan of gelijk aan

Opgepast echter, het < teken is niet toegestaan in attribuutwaardes. Dit moet bijgevolg vervangen worden door <, zelfs als het gebruikt wordt als de kleiner-dan operator.

Bovenvermelde operatoren worden het vaakst gebruikt in predikaattesten. Een XPath expressie kan niet alleen een patroon bevatten dat bepaalde knooppunten selecteert, maar ook een predikaat dat de set van geselecteerde knooppunten verder filtert. Elk knooppunt kan een onbeperkt aantal predikaten hebben. Maar in de praktijk wordt dit meestal beperkt tot één predikaat.

In dit voorbeeld wordt alleen het eerste **CHAPTER** kind van **THESIS** gematcht:

Voorbeeld 5.102. Gebruik van de booleaans operator = in predikaten

```
<xsl:template match="THESIS/CHAPTER[position()=1]">
   <xsl:value-of select="."/>
</xsl:template>
```

In dit voorbeeld worden alle **CHAPTER** kinderen van **THESIS** gematcht, behalve het eerste element (dat waarvan de positie 1 is).

Voorbeeld 5.103. Gebruik van de booleaanse operator > in predikaten

```
<xsl:template match="THESIS/CHAPTER[position()>1]">
   <xsl:value-of select="."/>
</xsl:template>
```

De sleutelwoorden **and** en **or** combineren twee booleaanse uitdrukkingen volgens de normale regels van de logica. Als de eerste voorwaarde van een **and** uitdrukking onwaar is, dan is de hele uitdrukking onwaar. Het tweede deel wordt bijgevolg niet meer nagekeken. De logische **or** kijkt na of beide uitdrukkingen waar zijn. Als de eerste uitdrukking waar is, dan is de volledige **or** uitdrukking waar. Bijgevolg wordt de tweede voorwaarde niet meer nagekeken. De **not()** functie keert het resultaat van een operatie om. XSLT heeft geen exclusieve **or** operator. De functionaliteit van deze operator kan geïmiteerd worden door een oordeelkundig gebruik van **and**, **or** en **not()**.

In volgend voorbeeldje wordt het gebruik van het sleutelwoord **and** geïllustreerd. Deze template matcht alleen een **CHAPTER** element als dit het eerste en het laatste (en dus het enige) kindelement van een **THESIS** element is, wat in ons geval geen gematchte elementen oplevert.

Voorbeeld 5.104. Gebruik van het sleutelwoord and

```
<xsl:template match="THESIS/CHAPTER[position()=1 and
position()=last()]">
   <xsl:value-of select="."/>
</xsl:template>
```

In het volgende voorbeeld worden alle **CHAPTER** elementen, behalve het laatste, gematcht:

Voorbeeld 5.105. Gebruik van het sleutelwoord not

```
<xsl:template match="CHAPTER[not(position()=last())]">
   <xsl:value-of select="."/>
</xsl:template>
```

We kunnen deze match expressie ook schrijven door gebruik te maken van het symbool "!=" in plaats van **not()**.

Voorbeeld 5.106. Gebruik van !=

```
<xsl:template match="CHAPTER[position()!=last()]">
   <xsl:value-of select="."/>
</xsl:template>
```

Tenslotte zijn er nog drie overblijvende functies die een boolean als resultaat hebben:

- true() geeft altijd waar terug
- false() geeft altijd onwaar terug
- lang(code) geeft waar als het huidige knooppunt dezelfde taal heeft (aangegeven door het **xml:lang** attribuut [Sectie 2.3.8.3.]) als het **code** argument

5.4.4.3. Getallen

XPath getallen zijn 64-bit IEEE 754 vlottende komma getallen [IEEE754], [MW754]. Waardes die geen getallen zijn, zoals booleans en strings worden automatisch omgezet naar getallen wanneer dit nodig is. De gebruiker kan ook expliciet een omzetting vragen door gebruik te maken van de **number()** functie die de volgende regels volgt:

- Booleaanse waardes zijn 1 als ze waar zijn en 0 als ze onwaar zijn.
- Een string wordt ontdaan van de whitespace voor en achter de string en dan omgezet naar een getal. De string "12" wordt bijvoorbeeld omgezet naar het getal 12. Als de string niet geïnterpreteerd kan worden als een getal, dan wordt de string omgezet naar het speciale symbool NaN.
- Sets van knooppunten en fragmenten van de resulterende boom worden omgezet naar strings [Sectie 5.4.4.4.] , waarop die string wordt omgezet naar een getal.

XPath maakt gebruik van de volgende rekenkundige operatoren:

- + voor de optelling
- - voor de aftrekking
- * voor de vermenigvuldiging
- **div** voor de deling
- **mod**, een binaire operator die de rest van de deling van twee getallen teruggeeft

Als je bijvoorbeeld `<xsl:value-of select="2+2"/>` in je template zet, wordt de string "4" in je uitvoerdocument opgenomen. Deze operatoren worden meer frequent gebruikt als onderdeel van een test. In het volgende voorbeeld worden alle **ATOM** elementen geselecteerd die een atoomgewicht hebben dat meer dan twee keer hun atoomgetal is:

Voorbeeld 5.107. Gebruik van rekenkundige operatoren

```
<xsl:template match="PERIODIC_TABLE">
   <HTML>
      <BODY>
```

```
        <xsl:apply-templates select="ATOM[ATOMIC_WEIGHT > 2 *
        ATOMIC_NUMBER]"/>
      </BODY>
    </HTML>
  </xsl:template>
```

In dit voorbeeld van het gebruik van de rekenkundige operator **mod**, zijn er twee afzonderlijke templates, één voor de even en één voor de oneven hoofdstukken.

Voorbeeld 5.108. Gebruik van de rekenkundige operator mod

```
  <xsl:template match="CHAPTER[position() mod 2 = 1]">
    <!--Doe iets met de oneven CHAPTER elementen-->
  </xsl:template>
  <xsl:template match="CHAPTER[position() mod 2 = 0]">
    <!--Doe iets met de even CHAPTER elementen-->
  </xsl:template>
```

Tenslotte zijn er nog vier functies die bewerkingen uitvoeren op getallen:

- floor() geeft het grootste geheel getal terug dat kleiner of gelijk aan het getal is
- ceiling() geeft het kleinste geheel getal terug dat groter dan of gelijk aan het getal is
- round() rondt het getal af tot het dichtsbijzijnde geheel getal
- sum() heeft als argument een set van knooppunten en geeft de som terug van de knooppunten die in deze set zitten

5.4.4.4. Strings

Een string is een opeenvolging van Unicode tekens [UNICODE]. Andere types van gegevens kunnen omgezet worden naar strings gebruik makend van de **string()** functie die de volgende regels volgt:

- Sets van knooppunten worden omgezet naar strings door gebruik te maken van de waarde van het eerste knooppunt in de set die berekend wordt door **xsl:value-of** volgens de eerder in dit hoofdstuk gegeven regels. [Sectie 5.3.1.3., Tabel 5.1.]
- De booleaanse waarde "onwaar" wordt omgezet naar het Engelse woord "false". De booleaanse waarde "waar" wordt omgezet naar het Engelse woord "true".
- Een getal wordt omgezet naar een getal in Europese stijl, zoals -89 en 16.44854. Dit wil zeggen dat als het getal een geheel getal is, dit getal voorgesteld wordt in decimale vorm zonder een decimale punt en zonder leidende nullen, eventueel voorafgegaan door een minteken. In het andere geval wordt het getal voorgesteld in de decimale vorm met een decimale punt en ten minste één cijfer na de decimale punt, eventueel voorafgegaan door een minteken. In deze vorm komen geen leidende nullen (leading zeros) en achterkomende nullen (trailing zeros) voor. [IEEE754]
- Fragmenten van de resulterende boom worden omgezet door ze op te nemen als de inhoud van één enkel denkbeeldig element en vervolgens de waarde van dat denkbeeldige element te nemen. Ook hier wordt de waarde van dit element berekend door **xsl:value-of**. [Sectie 5.3.1.3., Tabel 5.1.]

Naast **string()** bevat XSLT nog tien andere functies die strings kunnen manipuleren. We vatten deze samen in onderstaande tabel:

Tabel 5.5. XPath string functies

Functie	Geeft terug
`starts-with(main_string, prefix_string)`	Geeft de booleaanse waarde waar terug als **main_string** begint met **prefix_string**. Geeft onwaar terug in het andere geval
`contains(containing_string, contained_string)`	Geeft de booleaanse waarde waar terug als **contained_string** een deel is van **containing_string**. Onwaar in het andere geval
`substring(string, offset, length)`	Geeft een string van **length** tekens terug vanaf de gespecificeerde **offset** in de gegeven **string** óf alle tekens vanaf de **offset** tot het einde van de **string** als het **length** argument afwezig is. **length** en **offset** worden afgerond tot de dichtsbijzijnde integer als dit nodig is. Het eerste teken in de string heeft offset 1
`substring-before(string, marker-string)`	Geeft het deel van de **string** terug vanaf het eerste teken tot het eerste voorkomen van de **marker-string** (de marker-string zelf is niet inbegrepen in de resultaatstring)
`substring-after(string, marker-string)`	Geeft het deel van de **string** terug vanaf het eerste voorkomen van de **marker-string** tot het einde van de string (de **marker-string** zelf is niet inbegrepen in de resultaatstring)
`string-length(string)`	Geeft een getal terug dat het aantal tekens in de **string** weergeeft
`normalize-space(string)`	Geeft de **string** terug waarvan de whitespace in het begin en op het einde weggenomen werd en waarvan opeenvolgingen van witruimte vervangen werden door één enkele spatie. Als deze functie geen argument heeft, wordt de stringwaarde van het context knooppunt genormaliseerd
`translate(string, replaced_text, replacement_text)`	Geeft een string terug waarvan de tekens die voorkomen in **replaced_text** vervangen zijn door de overeenkomstige tekens uit **replacement_text**
`concat(string1, string2, . . . )`	Geeft een string terug die de aaneenschakeling is van alle strings die als argumenten aan de functie zijn meegegeven in de volgorde waarin ze voorkomen in de argumentenlijst

5.4.4.5. Resulterende boomfragmenten

Een resulterend boomfragment is een deel van een XML document dat geen compleet knooppunt of set van knooppunten vormt. Resulterende boomfragmenten kunnen teruggegeven worden door bepaalde uitbreidende functies (functies die horen bij een bepaalde XSLT implementatie of installatie) of door te verwijzen naar variabelen die resulterende boomfragmenten bevatten. Omdat de fragmenten geen goedgevormde XML zijn, kan je niet veel met ze doen. In feite zijn de enige operaties die toegestaan worden op fragmenten, het omzetten naar een string of een boolean door gebruik te maken van **string()** of **boolean()**. Zie ook [Sectie 5.3.1.11.] .

5.5. Conclusie

Zoals uit dit hoofdstuk blijkt, beschikt XSLT over een zeer uitgebreid arsenaal aan verschillende syntaxconstructies. Op het eerste zicht kan dit wel een beetje overweldigend lijken, maar uit onze eigen ervaring blijkt dat het niet zoveel tijd kost om een rudimentaire stylesheet te leren schrijven. Als je XML document goed gestructureerd is, kom je al heel ver

met een aantal elementaire templates en een aantal eenvoudige XPath expressies om wat eenvoudige uitvoer (meestal HTML) te genereren. Het verfijnen van de templates kost echter meer werk, vooral als je kruisverwijzingen in je documenten wilt verwerken. In dat geval moet je met vrij ingewikkelde XPath expressies, het **xsl:key** element en de **key()** functie werken en ook gebruik maken van de mogelijkheden om keuzes tussen verschillende opties te maken.

Wat de toekomst van XML en XSLT betreft, denken wij dat er voor standaardtoepassingen gedetailleerde stylesheets geschreven zullen worden. Als je deze standaardstylesheets wilt aanpassen, door bijvoorbeeld een paar templates te overschrijven, kan je dit doen door jouw specifieke stylesheet samen te voegen met de standaardstylesheet door gebruik te maken van **xsl:import** en **xsl:include**. Deze standaardstylesheets zullen natuurlijk gedocumenteerd moeten worden. Op dat vlak is nog niet zoveel onderzoek verricht, maar wij vermoeden dat dit in de toekomst niet kan uitblijven.

6. Gebruik van XSLT in onze tekst

Zoals we in [Hoofdstuk 4.] het gebruik van XML Schema in het kader van onze thesistekst hebben besproken, zo gaan we hier het gebruik van XSLT [Hoofdstuk 5.] in het kader van onze thesistekst bespreken. Deze transformaties gebruiken wij om uitgaande van de XML bestanden van onze tekst een navigeerbare versie van onze tekst in HTML aan te kunnen bieden. We gaan slechts een klein gedeelte van onze transformaties behandelen. Alleen al deze stylesheets uitprinten zou resulteren in vijftien tot twintig bladzijden.

We gaan in dit hoofdstuk twee concrete problemen aanhalen waar we geconfronteerd mee zijn bij het maken van onze stylesheets. Deze problemen zullen we in het kort beschrijven, waarna we aan de hand van voorbeeldtemplates uit onze stylesheets bespreken hoe we het probleem hebben aangepakt en opgelost. De oplossing die bij bespreken is **onze oplossing** voor deze problemen, dit is **niet noodzakelijk** de **beste oplossing**.

6.1. Navigatie in HMTL

Toen we de eerste versie van onze stylesheets maakten hadden we er nooit aan durven denken dat onze tekst zo uitgebreid ging worden. Dit komt onder andere door de hoeveelheid aan onderwerpen die we behandeld hebben en de soms ingewikkelde constructies die de werkgroepen van W3C hebben bedacht, maar die wij met veel voorbeelden geprobeerd hebben uit te leggen. Zoals gezegd hadden we nooit aan gedacht dat onze tekst deze afmetingen zou aannemen en vonden we het niet zo een slecht idee om één grote HTML pagina te maken met onze volledige tekst. Eerst zonder inhoudstafel, vervolgens met inhoudstafel. Dit ging goed totdat de gegenereerde HTML pagina enkele honderden kilobytes groot werd. Sommige browsers begonnen zich te verslikken in de tabellen, die wij onder andere in de inhoudstafel gebruikten, en in de grootte van het bestand.

Om deze problemen aan te pakken besloten we een navigeerbare HTML versie te maken. Het idee was om dit als volgt op te bouwen:

- De bezoeker van de pagina krijgt eerst een lijst van de hoofdstukken en de appendices te zien. Indien hij/zij een bepaald hoofdstuk of een bepaalde appendix wil lezen, dan moet er maar gewoon een link aangeklikt worden.
- Wanneer er gekozen wordt voor een hoofdstuk of een appendix dan wordt de keuze aan de hand van URL parameters die deel uitmaken van de gevolgde link, doorgegeven aan Cocoon 2.
- Op basis van de waarde van de ontvangen parameters kunnen wij in de stylesheets bepalen welk hoofdstuk of appendix we moeten gaan verwerken.
- De bezoeker krijgt dan een overzicht van ver verschillende secties in dat hoofdstuk. Indien aanwezig, wordt ook de beknopte beschrijving van het hoofdstuk of de appendix getoond.
- De bezoeker kiest vervolgens voor een bepaalde sectie en krijgt dan de inhoud van deze sectie te zien. Van andere secties die kunnen beschouwd als kinderen van de geselecteerde sectie, wordt de inhoud niet getoond, maar geven we alleen een link.

Op deze manier wilden we een site bouwen met een eenvoudige navigatie om in onze tekst te bladeren. We maken gebruik van de **generate-id()** [Sectie 5.4.4.1., Tabel 5.4.] om voor onze hoofdstukken, appendices en secties een unieke identifier te genereren. Het zijn deze identifiers die we via URL parameters doorgeven aan de server. Op basis van deze URL parameters kan dan bepaald worden wat er moet gebeuren.

We gaan eerst behandelen hoe de initiële pagina opgebouwd wordt. Daarna gaan we

stapsgewijs wijzigingen doorvoeren totdat we ons uiteindelijke resultaat bekomen.

6.1.1. Opbouw initiële pagina

We geven hier nu de templates die zorgen voor de opbouw van de openingspagina:

Voorbeeld 6.1. Stylesheets voor openingspagina

```
<xsl:template match="THESIS">
   <html>
      <head>
         <title> Thesis Erwin Hermans en Carolien Coenen </title>
      </head>
      <body bgcolor="#FFFFFF" text="#000000">
         <xsl:apply-templates select="CHAPTER" mode="TOC"/>
         <xsl:apply-templates select="APPENDIX" mode="TOC"/>
      </body>
   </html>
</xsl:template>
<xsl:template match="CHAPTER" mode="TOC">
   <H4>
      <a>
         <xsl:attribute name="href"> ?chapter=
            <xsl:value-of select="generate-id()"/>
         </xsl:attribute>
         <xsl:number level="multiple" count="CHAPTER" format="1. "/>
         <xsl:value-of select="TITLE"/>
      </a>
   </H4>
</xsl:template>
<xsl:template match="APPENDIX" mode="TOC">
   <H4>
      <a>
         <xsl:attribute name="href"> ?appendix=
            <xsl:value-of select="generate-id()"/>
         </xsl:attribute>
         <xsl:number level="multiple" count="APPENDIX" format="A.1. "/>
         <xsl:value-of select="TITLE"/>
      </a>
   </H4>
</xsl:template>
```

We hebben hier een aantal vrij eenvoudige templates. Met deze templates genereren we een lijstje van links naar de respectieve hoofdstukken en appendices. Wat opvalt is dat we in de waarde voor het **href** attribuut van het **a** element alleen de URL parameters vermelden. Dit is nodig voor de manier waarop wij willen werken, maar hoe komt het dat dit werkt voor elke browser?

Om deze vraag te beantwoorden is het nodig dat we eventjes in de sitemap [Sectie 9.2.3.2.3.] kijken. In onze sitemap hebben we de volgende ingang:

Voorbeeld 6.2.

```
<map:match pattern="">
   <map:redirect-to uri="cocoon:/index.html"/>
</map:match>
```

De sitemap waar deze ingang in staat zorgt bij ons voor het afhandelen van de URI-ruimte die begint met *http://localhost:8080/cocoon/tekst/thesis/*. We werken in de sitemap met een interne omleiding (redirect). De URI *cocoon:/index.html* is een bijzondere URI. Het gedeelte **cocoon:/** vertelt aan Cocoon 2 dat de behandeling van een aanvraag voor dit patroon

behandeld eigenlijk moet gebeuren door de pipeline waar in het match-patroon **index.html** staat. Dit werkt in elke browser vermits de afhandeling van de omleiding in Cocoon 2 zelf gebeurt. Het is ook mogelijk om te werken met een omleiding die door de browser moet geïnitieerd worden. Het volstaat om gewoon **cocoon:/** weg te laten. Maar het probleem bij sommige browsers (bijvoorbeeld Netscape Communicator 4.7x) is dat ze dan de URL parameters niet meer meegeven, waardoor onze oplossing niet meer zou werken.

6.1.2. Verwerken van URL parameters

De parameter die we nodig hebben om te weten voor welk hoofdstuk of appendix we de inhoudstafel van de secties moeten genereren wordt nu wel doorgegeven, maar hoe kunnen we die gebruiken in onze stylesheets? In [Sectie 9.2.3.2.3.4.] bespreken we wat er moet gebeuren opdat de XSLT processor de parameters doorgeeft aan de stylesheets. We mogen er dus van uitgaan dat we aan de waarden van deze parameters kunnen in onze stylesheets.

Maar hoe vragen we die waarden nu op in onze stylesheets? Om dit doel te realiseren kunnen we het element **xsl:param** gebruiken als rechtstreeks kind van het **xsl:stylesheet** element.

Voorbeeld 6.3.

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:param name="chapter"/>
     <xsl:param name="appendix"/>
     ...

  </xsl:stylesheet>
```

De XSLT processor zorgt ervoor dat de waarde van bijvoorbeeld de URL parameter **chapter** nu in onze stylesheets gekend is en gekoppeld aan de parameter **$chapter**. We kunnen nu onze stylesheets aanpassen zodat we met de waarde van deze parameters rekening kunnen houden. Onze templates krijgen nu de volgende vorm:

Voorbeeld 6.4.

```
<!--Template voor THESIS blijft ongewijzigd-->
  <xsl:template match="CHAPTER" mode="TOC">
     <H4>
        <a>
           <xsl:attribute name="href"> ?chapter=
              <xsl:value-of select="generate-id()"/>
           </xsl:attribute>
           <xsl:number level="multiple" count="CHAPTER" format="1. "/>
           <xsl:value-of select="TITLE"/>
        </a>
     </H4>
     <xsl:if test="$chapter=generate-id(.)">
        <table border="0">
           <xsl:apply-templates select="SUMMARY"/>
           <xsl:apply-templates select="SECTION" mode="TOC"/>
        </table>
     </xsl:if>
  </xsl:template>
  <xsl:template match="CHAPTER//SECTION" mode="TOC">
     <table>
        <tr>
           <td width="30"/>
           <td>
```

```
            <a>
               <xsl:attribute name="href"> ?chapter=
                  <xsl:value-of select="$chapter"/>
               &amp;section=
                  <xsl:value-of select="generate-id(.)"/>
               </xsl:attribute>
               <xsl:number level="multiple" count="CHAPTER|SECTION"
               format="1.1.1. "/>
               <xsl:value-of select="TITLE"/>
            </a>
            <xsl:apply-templates select="SECTION" mode="TOC"/>
         </td>
      </tr>
   </table>
</xsl:template>
<xsl:template match="APPENDIX" mode="TOC">
   <H4>
      <a>
         <xsl:attribute name="href"> ?appendix=
            <xsl:value-of select="generate-id()"/>
         </xsl:attribute>
         <xsl:number level="multiple" count="APPENDIX" format="A.1. "/>
         <xsl:value-of select="TITLE"/>
      </a>
   </H4>
   <xsl:if test="$appendix=generate-id(.)">
      <table border="0">
         <xsl:apply-templates select="SUMMARY"/>
         <xsl:apply-templates select="SECTION" mode="TOC"/>
      </table>
   </xsl:if>
</xsl:template>
<xsl:template match="APPENDIX//SECTION" mode="TOC">
   <table>
      <tr>
         <td width="30"/>
         <td>
            <a>
               <xsl:attribute name="href"> ?chapter=
                  <xsl:value-of select="$appendix"/>
               &amp;section=
                  <xsl:value-of select="generate-id(.)"/>
               </xsl:attribute>
               <xsl:number level="multiple" count="APPENDIX|SECTION"
               format="A.1.1.1. "/>
               <xsl:value-of select="TITLE"/>
            </a>
            <xsl:apply-templates select="SECTION" mode="TOC"/>
         </td>
      </tr>
   </table>
</xsl:template>
```

We hebben een kleine wijziging aangebracht bij de templates voor de **CHAPTER** en **APPENDIX** elementen. Na het genereren van de uitvoer voor de titel van het hoofdstuk controleren we of de unieke identifier die gegenereerd wordt voor dit hoofdstuk dezelfde is als de waarde van de parameter **$chapter**. Indien dat het geval is, dan gaan we voor de verschillende secties van onze tekst op een soortgelijke manier te werk als voor de hoofdstukken. We plaatsen een link waarvan het **href** attribuut nu een bijkomende parameter **section** heeft.

De gegenereerde identifiers zijn uniek in die zin dat geen enkel ander knooppunt uit de XML boom van dat document dezelfde identifier zal krijgen. Maar deze identifiers worden volgens een bepaald algoritme berekend. Dit algoritme zorgt ervoor dat deze identifiers dezelfde waarde hebben als het document voor de tweede, derde, ... keer opgevraagd wordt. Wel

belangrijk om te vermelden is dat, indien er wijzigingen gebeuren aan een stuk tekst, het heel goed mogelijk is dat een aantal identifiers veranderen.

Merk op dat de templates voor hoofdstukken en appendices vrij gelijkaardig zijn. Deze templates kunnen geparametriseerd worden [Sectie 5.3.1.11.3.] . We hebben dit bij deze voorbeelden bewust niet gedaan om de voorbeelden niet nodeloos ingewikkeld te maken en om aan te tonen dat de templates veel op elkaar lijken.

We willen hier toch nog een probleem vermelden met Microsoft Internet Explorer 4.0 (versie **4.72.3110.4**) onder Microsoft Windows voor het doorgeven van de parameters. Deze browser maakt van de parameters die doorgegeven moeten worden als **?chapter=N10134&section=N10319** het volgende: **?chapter=N10134§ion=N10319**. Dit komt waarschijnlijk omdat deze browser het gedeelte **&sect** van de parameters interpreteert als een entiteit, wat totaal verkeerd is. Recentere versies van deze browsers hebben daar geen last van.

6.1.3. Tonen van de juiste sectie

We weten nu hoe we het juiste hoofdstuk of de juiste appendix kunnen selecteren, maar hoe zit het nu met de secties? We introduceren als kind van het **xsl:stylesheet** element een nieuw **xsl:param** element:

Voorbeeld 6.5.

```
...
 <xsl:param name="section"/>
...
```

De template voor een **SECTION** element wordt dan op de volgende manier geherdefinieerd:

Voorbeeld 6.6.

```
<xsl:template match="CHAPTER//SECTION" mode="TOC">
   <table>
      <tr>
         <td width="30"/>
         <td>
            <xsl:if test="not($section=generate-id(.))">
               <a>
                  <xsl:attribute name="href"> ?chapter=
                     <xsl:value-of select="$chapter"/>
                  &amp;section=
                     <xsl:value-of select="generate-id(.)"/>
                  </xsl:attribute>
                  <xsl:number level="multiple" count="CHAPTER|SECTION"
                  format="1.1.1. "/>
                  <xsl:value-of select="TITLE"/>
               </a>
               <xsl:apply-templates select="SECTION" mode="TOC"/>
            </xsl:if>
            <xsl:if test="$section=generate-id(.)">
               <xsl:apply-templates select="."/>
            </xsl:if>
         </td>
      </tr>
   </table>
```

```
</xsl:template>
<xsl:template match="CHAPTER//SECTION">
   <H2>
      <xsl:value-of select="TITLE"/>
   </H2>
   <xsl:apply-templates select="PARA | SECTION" mode="TOC"/>
</xsl:template>
```

In de template in de mode **TOC** controleren we of we de gewenste sectie tegenkomen. Indien we de gewenste sectie tegenkomen passen we de normale templates toe, in het andere geval moet het gedrag hetzelfde blijven. Merk op dat we bij de normale template voor het **SECTION** element ook de kindelementen oproepen met als mode **TOC**. De template voor het **PARA** element in de mode **TOC** roept gewoon de normale template voor het **PARA** element aan. Dit is een elegante hack voor een probleem dat wij hadden.

Het probleem is dat, indien we alleen de **PARA** elementen selecteren, de inhoud van deze elementen **voor** de titels van de **SECTION** elementen komt te staan, ook al staan ze in het **originele** bestand **tussen** of **na** de **SECTION** elementen. Het is niet mogelijk om op een elegantere manier aan te geven dat de templates op de kindelementen moeten uitgevoerd worden, maar afhankelijk van het kindelement in een andere mode. Het is dus wel mogelijk, maar deze oplossing is in onze ogen nog slechter. Deze oplossing bestaat erin alle kindelementen te overlopen met behulp van de **xsl:for-each** constructie. We zouden dan met behulp van de functie **local-name()** de naam van de elementen moeten gaan opvragen en, afhankelijk van deze naam, de templates in de juiste mode toepassen.

6.1.4. Besluit

Deze manier van werken laat ons toe om toch met een vrij grote tekst te werken maar het de browsers niet te lastig te maken met grote pagina's. Ook is het zo veel handiger om een bepaalde sectie te lezen. Met een kleine aanpassing aan de templates is het mogelijk om de tekst bijvoorbeeld hoofdstuk per hoofdstuk op te vragen, voor wie dat liever heeft. En met nog een kleine aanpassing kan dan uiteindelijk terug de volledige tekst weergegeven worden (en hebben we terug de originele stylesheets natuurlijk).

6.2. Referenties in de tekst

We hebben lange tijd gedacht over een systeem om interne referenties op te nemen in onze tekst. Omdat onze thesis over XML en XSLT gaat hebben we geprobeerd om met behulp van XPath [Sectie 5.4.] paden tot een oplossing te komen. Om dan naar de vijfde sectie van het tweede hoofdstuk te verwijzen zouden we bijvoorbeeld **/CHAPTER[2]/SECTION[5]** als locatiepad kunnen gebruiken. In onze thesis gaf dat een probleem. We hadden namelijk al een aantal hoofdstukken geschreven, maar de volgorde van deze hoofdstukken lag nog niet vast. Het zou ook kunnen zijn dat nog enkele secties van plaats zouden veranderen. Deze flexibiliteit wilden wij niet opgeven en dat is de reden waarom we geen referenties van die aard konden gebruiken. XPath is dus wel geschikt voor vaststaande structuren, maar niet voor dynamische, veranderende structuren, zoals onze tekst.

Dit is ook de reden waarom we de nummering van de verschillende delen niet konden gebruiken. Deze nummering gebeurt automatisch en verandert dus indien een sectie of een hoofdstuk van plaats veranderd. Dit blijkt ook bij de XSLT van Docbook een probleem op te leveren [NMDB].

We hebben dan nog eventjes overwogen om onze referentie op te bouwen uit de titel van een hoofdstuk en de titel van de sectie in dat hoofdstuk waar we naar willen verwijzen. Dit levert echt problemen op omdat het best mogelijk is dat er in één hoofdstuk twee secties met dezelfde titel zijn.

Tenslotte zijn we tot de nu door ons gebruikte oplossing gekomen. We hebben een attribuut **id** ingevoerd waarvan de waarde uniek moet zijn over de hele tekst bekeken. Hiervoor moeten wij ons aan bepaalde regels houden. Een identifier is uit verschillende delen opgebouwd, gescheiden door _ (underscores). Het eerste deel van de identifier moet een onderscheid maken tussen de verschillende hoofdstukken en appendices. Dit eerste deel komt overeen met ofwel de bestandsnaam ofwel de directorynaam van de directory waar het bestand zich in bevindt. Het tweede deel moet dan alleen uniek zijn voor dat hoofdstuk. Meestal is het tweede deel een letterwoord dat de titel van de sectie kort samenvat, maar om naar een XML voorbeeld te wijzen zouden we als tweede deel **xml_vb_naamvb** kunnen gebruiken. Wij hebben afgesproken om dit zo te doen en om een soortgelijke conventie te gebruiken voor afbeeldingen (**fig**), Java voorbeelden (**java**),... Dit gebeurt analoog voor de elementen die een identifier kunnen hebben maar hier niet vernoemd zijn.

Wanneer we in onze tekst hebben:

Voorbeeld 6.7.

```
<CHAPTER id="xsl">
   <TITLE> XSL(T) </TITLE>
   ...

   <SECTION id="xsl_xpath">
      <TITLE> XPath </TITLE>
      ...

   </SECTION>
   ...

</CHAPTER>
```

Dan kunnen we op de volgende manier een verwijzing opnemen in onze tekst:

Voorbeeld 6.8.

```
<PARA>
   ...

en hiervoor verwijzen wij naar
   <REFERENCE> xsl_xpath </REFERENCE>
   ...

</PARA>
```

Met behulp van XSLT transformaties is het dan de bedoeling dit om te referentie om te zetten naar bijvoorbeeld [Sectie 3.4] (fictieve nummering). We zullen hier het voorbeeld geven van onze transformaties waarmee we dit oplossen:

Voorbeeld 6.9.

```
...
```

```
 <xsl:key name="REFIDS" match="CHAPTER | APPENDIX | SECTION"
 use="generate-id(.)"/>
...

 <xsl:template match="REFERENCE">
    <xsl:variable name="refid">
       <xsl:value-of select="."/>
    </xsl:variable>
    <xsl:variable name="chapter">
       <xsl:value-of
       select="generate-id(//CHAPTER[descendant-or-self::*[@id=$refid]])"/>
    </xsl:variable>
    ...

    <xsl:choose>
       <xsl:when test="not($chapter='')">
          <xsl:variable name="section">
             <xsl:value-of
             select="generate-id(//CHAPTER//SECTION[@id=$refid])"/>
          </xsl:variable>
          ...

          <xsl:choose>
             <xsl:when test="not($section='')">
                <xsl:call-template name="sectref">
                   <xsl:with-param name="refid">
                      <xsl:value-of select="$refid"/>
                   </xsl:with-param>
                   <xsl:with-param name="chapter">
                      <xsl:value-of select="$chapter"/>
                   </xsl:with-param>
                   <xsl:with-param name="section">
                      <xsl:value-of select="$section"/>
                   </xsl:with-param>
                </xsl:call-template>
             </xsl:when>
             ...

          </xsl:choose>
       </xsl:when>
       ...

    </xsl:choose>
 </xsl:template>
...

 <xsl:template name="sectref">
    <xsl:param name="refid"/>
    <xsl:param name="chapter"/>
    <xsl:param name="section"/>
    <a>
       <xsl:attribute name="href"> #
          <xsl:value-of select="$refid"/>
       ?chapter=
          <xsl:value-of select="$chapter"/>
       &section=
          <xsl:value-of select="$section"/>
       </xsl:attribute>
       <xsl:for-each select="key('REFIDS', $section)"> [Sectie
          <xsl:number level="multiple" count="CHAPTER | SECTION"
          format="1.1.1. "/>
       ] </xsl:for-each>
    </a>
 </xsl:template>
...
```

We hebben in dit voorbeeld alleen de templates gegeven die nodig zijn voor een referentie naar een sectie te verwerken. De andere templates zijn allemaal vrij analoog.

Dit is slecht één van de mogelijkheden om dit probleem op te lossen. Een andere mogelijkheid bestaat erin intensief gebruik te maken van sleutels met behulp van het **xsl:key** element. We zouden sleutels kunnen maken waarin we een opzoeking kunnen verrichten op de waarden van de verschillende **id** attributen. Op die manier kunnen we onmiddellijk opzoeken bij welk element een bepaald id hoort en moeten we geen ingewikkelde XPath expressies bedenken. We geven deze oplossing hier omdat dit onze eerste oplossing was voor dit probleem.

We maken hier eerst een key aan waarin we de unieke gegenereerde identifiers voor hoofdstukken, appendices en secties bijhouden. We maken ook een variabele **$refid** aan die als waarde de inhoud van het **REFERENCE** element krijgt. Vervolgens gaan we opzoeken in welk hoofdstuk een **id** attribuut met die waarde is gedeclareerd. We doen ook nog andere opzoekingen, maar die vermelden we niet omdat we niet relevant zijn voor dit voorbeeld.

Vervolgens testen we indien de variabele **$chapter** niet leeg is. We weten op dit moment dat we in een hoofdstuk zitten en niet in een appendix. Vervolgens gaan we opzoeken met welke sectie de waarde van de opgegeven identifier overeenkomt. We kunnen ook nog een aantal andere opzoekingen voor de elementen die ook een **id** attribuut kunnen hebben.

Indien we effectief een sectie vinden die aan deze voorwaarden voldoet, dan roepen we een geparametriseerde template op die de naam **sectref** draagt en drie parameters definieert. Deze template is verantwoordelijk voor het genereren van de HTML-code voor de link. Merk op dat we in de template **sectref** gebruik maken van de **xsl:for-each** functie. Vermits de waarde van **$section** een unieke waarde is, zal dit maar één keer uitgevoerd worden.

Op deze manier slagen we erin om links te leggen naar de juiste plaatsen van onze tekst. We moeten hiervoor een klein beetje zelfdiscipline opbrengen om ervoor te zorgen dat de door ons gecreëerde waarden van het **id** attribuut uniek zijn. Anders valt deze oplossing als een kaartenhuisje in elkaar. Maar het gebruik van deze "identifiers" is de beste (en misschien ook de enige mogelijke) oplossing die we gevonden hebben. De implementatie in onze stylesheets is misschien nog niet optimaal zoals uit de voorbeelden blijkt, maar we merken nog elke dag dat we verbeteringen kunnen aanbrengen aan deze stylesheets. Dit komt onder andere door het feit dat er zoveel mogelijkheden zijn, dat je al een tijdje met XSLT moet gewerkt hebben om het volledige potentieel te ontdekken en te benutten.

7. XSL-FO

Zoals we reeds eerder aangehaald hebben, bestaat de eXtensible Stylesheet Language (XSL) eigenlijk uit twee aparte talen. De transformatietaal XSL Transformations hebben we reeds in een voorgaand hoofstuk besproken [Hoofdstuk 5.] . Het doel van dit hoofdstuk is het behandelen van de formatteringstaal XSL Formatting Objects.

Wij hebben XSL-FO bestudeerd omdat wij onze thesistekst in XML formaat wilden omzetten naar het PDF formaat [PLPDF]. Het is niet mogelijk deze omzetting rechtstreeks te doen. om tot een volwaardig PDF document te komen zijn er twee stappen nodig. Eerst wordt via XSLT transformaties een omzetting gemaakt van de XML documenten naar XSL vormgevingsobjecten. Vervolgens wordt het nieuw gegenereerde XML document dat opgebouwd is uit XSL vormgevingsobjecten door FOP (Apache's Formatting Objects Processor) [FOP] omgezet naar het PDF formaat.

We beginnen dit hoofdstuk met kort uit te leggen wat we juist onder XSL-FO verstaan en we geven een kort overzicht van de ontstaansgeschiedenis. Vervolgens behandelen wij op een vrij theoretische manier de twee fundamentele processen waaruit het verwerken van een stylesheet bestaat: de transformatie en de vormgeving van het XML document. Voordat we overgaan op het effectief beschrijven van alle vormgevingsobjecten en hun respectievelijke vormgevingseigenschappen, bespreken we eerst nog enkele belangrijke concepten van vormgeving, die in de rest van het hoofdstuk gebruikt zullen worden.

7.1. Wat is XSL-FO?

XSLT en XSL-FO zijn twee formaten die voldoen aan de XML syntax en die elk hun eigen Document Type Definitions (DTD) hebben . Aangezien XSL-FO en XSLT beiden gebruik maken van een XML syntax, is elk XSL-FO of XSLT document een well-formed XML document. XSLT en XSL-FO zijn twee volkomen onafhankelijke talen.

eXtensible Stylesheet Language (XSL) is de officiële naam van de W3C specificatie die een manier beschrijft om vormgevingsinformatie aan XML documenten te verbinden [TRXSL]. Om verwarring met XSLT te vermijden, wordt deze specificatie vaak aangeduid met XSL-FO. FO is de afkorting voor Formatting Objects.

Voor dit hoofdstuk baseerden wij ons op [ERHXSLFO], [KLFO], [DTDWFO] en [TRXSL].

Een gewoon XML document beschrijft gegevens op een ordelijke manier met als enige mogelijke vormgeving de indentatie van de tags. De (audio-)visuele gegevensvoorstelling wordt overgelaten aan de applicaties zelf. Als je meer controle wilt over hoe je document eruit moet zien, dan kom je op het terrein van XSL-FO terecht. XSL-FO is een gespecialiseerde XML woordenschat ontworpen om het uitzicht van een document te beschrijven.

Ontwerpers gebruiken XSL stylesheets om weer te geven hoe gestructureerde inhoud voorgesteld moet worden. Anders gezegd, ze bepalen de stijl, de lay-out, de paginatie, de kleuren, enzovoort die het uitvoerdocument krijgt in een presentatiemedium zoals een venster van een webbrowser, een WAP-GSM, een verslag of een boek.

Merk op dat in de XSL specificatie van het W3C [TRXSL] geen officiële DTD aanwezig is om FO documenten te kunnen valideren. Er is echter wel een experimentele FO DTD gemaakt door RenderX [RX]. Deze DTD kan gebruikt worden voor documentvalidatie.

7.2. Korte geschiedenis van XSL-FO

Voor deze sectie gebruikten wij [KLFO] als bron.

Eén van de basisideeën waarop gestructureerde talen zoals SGML [RCSGML] en XML [TRXML] gebaseerd zijn, is het scheiden van inhoud en presentatie. In de 15 jaar na het goedkeuren van SGML als ISO standaard zijn er verschillende grote pogingen geweest om een bijhorende standaard te ontwikkelen die gebruikers zou toelaten vormgevingsinformatie te associëren met gestructureerde documenten.

Eén van de eerste inspanningen voor standaardisatie resulteerde in de Formatting Output Specification Instance (FOSI). FOSI is ontwikkeld door het defensiedepartement van de Verenigde Staten van Amerika en werd als standaard aanvaard door de bedrijven Arbortext [AT] en Datalogics [DL] om stylesheets voor SGML documenten te definiëren. FOSI definieert een SGML woordenschat om vormgevingskarakteristieken te verbinden aan elementen in een bepaalde context. Dit fundamentele principe is ook het basisidee achter al de andere vormgevingstalen. Bij XSL gebeurt dit echter op een iets andere manier, zoals we later zullen zien.

De volgende grote poging om het probleem op te lossen was de ISO standaard DSSSL (Document Style Semantics and Specification Language) [DSSSL]. DSSSL bestaat uit zowel een transformatie- als een vormgevingsgedeelte. Tegen de tijd dat DSSSL goedgekeurd werd, waren het Internet en HTML [TRHTML] aan hun opmars begonnen. De SGML gemeenschap reageerde hierop door XML te ontwikkelen. HTML heeft een ingebouwde semantiek voor vormgeving. Aangezien XML, zoals SGML, geen enkele vormgevingssemantiek impliceert, kan XML niet gebruikt worden voor presentatie zonder een methode om vormgevingssemantiek te verbinden aan de elementen. Dit gegeven leidde al snel tot de geboorte van de XSL werkgroep binnen het W3C.

In het begin was de XSL werkgroep samengesteld uit veel van de mensen die DSSSL ontworpen hadden. Veel initiële versies van de specificatie waren dan ook zwaar beïnvloed door de ideeën van DSSSL, maar maakten gebruik van een XML syntax in plaats van een Scheme syntax [SCHEME]. Net zoals dit bij DSSSL het geval was (maar niet bij FOSI en CSS (Cascading Style Sheets) [TRCSS]), was het XSL voorstel gebaseerd op een verwerkingsmodel in twee stappen: transformatie van het originele document en vormgeving van het getransformeerde document. Dit is een cruciaal punt in de geschiedenis van XSL. Dit feit leidde al snel tot de afsplitsing van het transformatie-aspect in de XSLT specificatie, waarvoor het gemakkelijker was een consensus te bereiken. Tegelijkertijd werd ook het werk aan de algemene XSL specificatie verder gezet en de set van vormgevingseigenschappen evolueerde in de richting van de CSS woordenschat.

7.3. Een transformatieproces in twee stappen

Voor deze sectie hebben wij ons gebaseerd op de W3C Recommendation [TRXSL]. De afbeeldingen in deze sectie zijn ook afkomstig uit de W3C Recommendation *http://www.w3.org/Consortium/Legal/copyright-documents-19990405*.

Voordat we verder gaan met het uiteenzetten van XSL-FO is het belangrijk goed te begrijpen hoe het transformatieproces van een XML document naar een voorstelling geschikt voor een welbepaald medium (browser, WAP-GSM, enzovoort) gebeurt. Het transformatieproces wordt beschreven in een stylesheet. Deze stylesheet gebruikt XSLT om een XML document te transformeren naar een nieuw XML document dat de XSL-FO presentatiewoordenschat

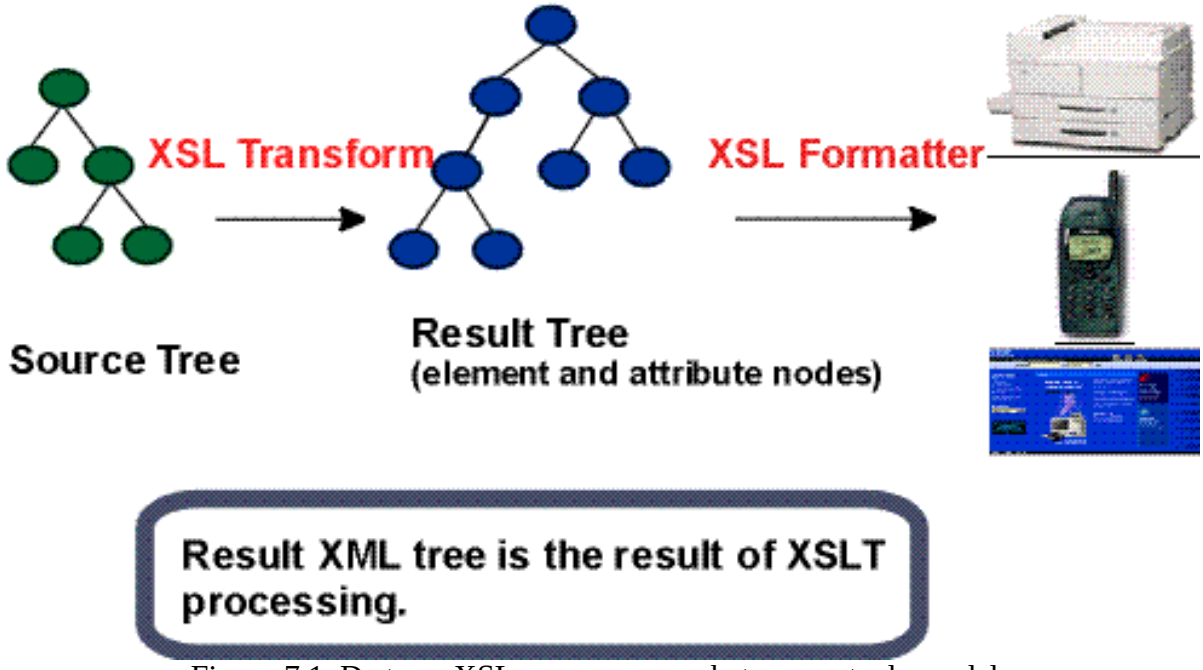
gebruikt. In de toekomst zullen webbrowsers misschien in staat zijn om gegevens die markup met XSL vormgevingsobjecten bevatten rechtstreeks weer te geven. Momenteel is er echter nog altijd een aanvullende stap nodig waarbij het uitvoerdocument verder getransformeerd moet worden in een ander formaat zoals Adobe's PDF [AAR]. In ons geval wordt die aanvullende stap uitgevoerd door FOP [FOP]. We onderscheiden dus twee processen: de transformatie en de vormgeving.

Een XSL processor neemt een XML document en een XSL stylesheet en produceert vervolgens de voorstelling van het XML brondocument, die door de ontwerper in de stylesheets vastgelegd is. Er zijn twee fasen aan dit representatieproces. Eerst wordt er vertrekkende van de XML bronboom een resulterende boom opgebouwd (deze boom kan vormgevingsobjecten bevatten, maar dit is niet noodzakelijk). Vervolgens wordt de resulterende boom geïnterpreteerd om het vormgegeven resultaat geschikt te maken voor weergave op een welbepaald medium. Det eerste fase wordt boomtransformatie (tree transformation) genoemd en de tweede fase wordt vormgeving (formatting) genoemd. Het vormgevingsproces wordt uitgevoerd door de formatter.

De boomtransformatie maakt het mogelijk om voor de resulterende boom een totaal andere structuur dan die van de bronboom op te bouwen. Zo is het bijvoorbeeld mogelijk een inhoudstafel toe te voegen op basis van een selectie van elementen uit het brondocument [Sectie 5.3.1.6.] . Of je kan bijvoorbeeld brongegevens sorteren op naam [Sectie 5.3.1.10.] en ordenen in een tabel. Bij het opbouwen van de resulterende boom voegt het boomtransformatieproces door middel van vormgevingsobjecten ook informatie toe die nodig is om die resulterende boom een bepaalde vormgeving mee te geven.

Vormgeving wordt mogelijk gemaakt door vormgevingssemantiek op te nemen in de resulterende boom. De manier van vormgeven wordt uitgedrukt aan de hand van een verzameling klassen van vormgevingsobjecten (formatting objects) [Sectie 7.5., Tabel 7.1.] . De knooppunten van de resulterende boom zijn vormgevingsobjecten. De klassen van vormgevingsobjecten geven typografische abstracties weer zoals pagina's [Sectie 7.6.3.] , paragrafen, tabellen [Sectie 7.11.] , enzovoort. Een fijnere controle over de voorstelling van deze abstracties wordt geleverd door een set van vormgevingseigenschappen [Sectie 7.5.1.] , zoals bijvoorbeeld, de controle over de indentatie [Sectie 7.15.3.3.] , de ruimte tussen letters [Sectie 7.15.5.1.] en woorden [Sectie 7.15.5.2.] , controle over de koppeltekens [Sectie 7.15.3.2.] , enzovoort. In XSL-FO leveren de klassen van de vormgevingsobjecten en vormgevingseigenschappen de woordenschat om de voorstellingswijze uit te drukken.

Het XSL-FO uitdrukkingsmodel dat we zojuist kort beschreven hebben, is een conceptueel model. Een implementatie van dit model is niet verplicht om deze twee onderdelen als afzonderlijke processen te beschrijven. Implementaties mogen het brondocument verwerken op eender welke manier zolang ze maar hetzelfde resultaat bekomen als de verwerking aan de hand van het conceptueel model zou opleveren. Het diagram hieronder geeft het conceptueel model grafisch weer:



Figuur 7.1. De twee XSL processen van het conceptuele model

Copyright © 15 October 2001 World Wide Web Consortium

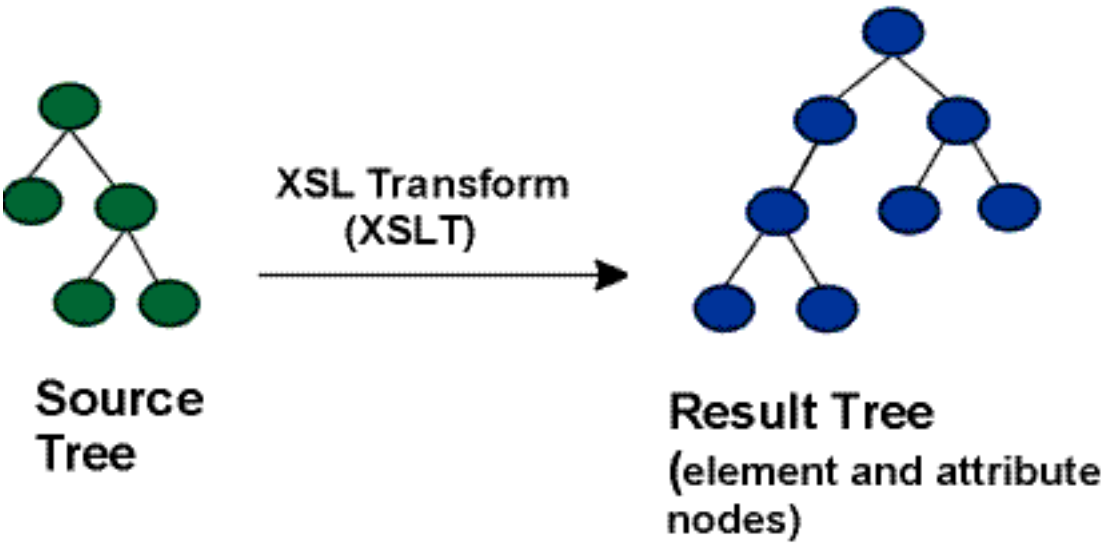
In de volgende sectie gaan we dieper in op beide (conceptuele) processen: de transformatie en de vormgeving.

7.3.1. Transformatie

De boomtransformatie bouwt de resulterende boom op. De objecten van de resulterende boom, ook wel element- en attributenboom genoemd, behoren hoofdzakelijk tot de namespace van **vormgevingsobjecten** (formatting objects) [Sectie 7.4.] . In deze boom worden vormgevingsobjecten voorgesteld als XML elementen waarvan de eigenschappen weergegeven worden door een set van XML attribuut-waarde paren. De inhoud van het vormgevingsobject is de inhoud van het XML element. De boomtransformatie wordt gedefinieerd in de W3C XSLT Recommendation [TRXSLT] en hebben wij al uitgebreid behandeld in het hoofdstuk over XSLT [Hoofdstuk 5.] .

De XSL stylesheet wordt gebruikt in de boomtransformatie. Een stylesheet bevat een aantal constructieregels om de boom op te bouwen. De boomconstructieregels bestaan, zoals we in het hoofdstuk over XSLT uitvoerig behandeld hebben, uit twee delen: een patroon dat gematcht wordt tegen elementen in de bronboom [Sectie 5.3.1.2.1.] en een template die een deel van de resulterende boom opbouwt [Sectie 5.3.1.2.] . Dit mechanisme zorgt ervoor dat een stylesheet toepasbaar is op een zeer brede waaier van documenten die gelijkaardige boomstructuren hebben.

In onderstaande tekening geven we een schematische voorstelling van het XSLT transformatieproces.



Figuur 7.2. De boomtransformatie
Copyright © 15 October 2001 World Wide Web Consortium

In principe is het niet noodzakelijk om het XSLT gedeelte van XSL te gebruiken om een XML document te creëren dat gebruik maakt van de vormgevingswoordenschat. Eender welke tool, zelfs een gewone teksteditor, mag gebruikt worden om zo'n document te creëren. In veel gevallen echter, zal het proces wel gebruik maken van XSLT. In dit geval bestaat het document met de vormgevingsobjecten niet als een blijvende voorstelling (bijvoorbeeld een bestand), maar alleen als een voorstelling in het geheugen. Die voorstelling kan een Document Object Model (DOM) boom [TRDOM] of een stroom van Simple API for XML (SAX) events [SAX] zijn.

7.3.2. Vormgeving

Vormgeving interpreteert de resulterende boom (bekeken als een boom van vormgevingsobjecten) om die voorstelling te leveren die bedoeld is door de ontwerper van de stylesheet waaruit de element- en attributenboom is opgebouwd. Deze elementen en attributen behoren tot de namespace *http://www.w3.org/1999/XSL/Format* die wij in onze tekst met het voorvoegsel **fo** associëren [Sectie 7.4.] .

De woordenschat van de vormgevingsobjecten [Sectie 7.5., Tabel 7.1.] die ondersteund worden door XSL-FO (dit is dus de verzameling van **fo:** elementtypes) is de verzameling van typografische abstracties die toegankelijk zijn voor de ontwerper. Elk vormgevingsobject stelt een specificatie voor die een deel van de opdeling van de pagina's, de lay-out en de stijlinformatie bevat. Als resultaat van het vormgeven van de hele resulterende boom zal deze specificatie toegepast worden op de inhoud van de vormgevingsobjecten. Elke klasse van vormgevingsobjecten stelt een bepaald soort van vormgevingsgedrag voor. Bijvoorbeeld, de klasse van vormgevingsobjecten van blokniveau [Sectie 7.7.1.] stelt het breken van de inhoud van een paragraaf in lijnen voor. Andere delen van de specificatie kunnen afkomstig zijn van andere vormgevingsobjecten. Het vormgeven van een paragraaf (een blokvormgevingsobject) hangt bijvoorbeeld af van zowel de specificatie van eigenschappen van het blokvormgevingsobject [Sectie 7.5.1.] als van de specificatie van de lay-outstructuur waarin het blok geplaatst wordt door de formatter.

De eigenschappen die geassocieerd worden met een bepaald vormgevingsobject controleren

de vormgeving van dat object. Een aantal van de eigenschappen, zoals bijvoorbeeld **color** [Sectie 7.15.4.1.] , specificeren rechtstreeks het vormgevingsresultaat. Andere eigenschappen, zoals bijvoorbeeld **space-before** [Sectie 7.15.6.4.] , leggen alleen de verzameling van mogelijke vormgevingsresultaten beperkingen op, zonder expliciet een bepaald vormgevingsresultaat te specificeren. De formatter kan dan keuzes maken tussen een aantal mogelijke vormgevingen op basis van criteria zoals een mooi eindresultaat.

Vormgeving bestaat uit het opbouwen van een boom bestaande uit geometrische gebieden die de gebiedsboom genoemd wordt. De geometrische gebieden worden geplaatst op een opeenvolging van één of meer pagina's. Een browser gebruikt bijvoorbeeld maar één enkele pagina. Een pdf bestand bestaat gewoonlijk uit meerdere pagina's. Elk geometrisch gebied heeft een plaats op de pagina, een specificatie van wat er in dat gebied moet getoond worden en kan een achtergrond [Sectie 7.15.6.1.] hebben, begrenzingen en opvulling (padding) [Sectie 7.15.6.4.] . Het vormgeven van één enkel leesteken bijvoorbeeld genereert een gebied dat groot genoeg is om het symbool te bevatten. Deze gebieden kunnen in elkaar genest worden. Het symbool kan bijvoorbeeld geplaatst worden binnen een lijn. Die lijn kan in een blok zitten en dat blok kan op zijn beurt in een pagina zitten.

Rendering neemt de gebiedsboom (het abstracte model van de presentatie, in termen van pagina's en hun collectie van gebieden) en zorgt ervoor dat een presentatie verschijnt op het relevante medium, zoals een venster van een browser of een computerscherm. De juiste betekenis van rendering wordt in dit hoofdstuk niet in detail beschreven.

7.3.3. Enkele belangrijke concepten van vormgeving

In deze sectie worden enkele concepten van het algemene vormgevingsproces beschreven. Wij beschrijven deze begrippen hier omdat zij doorheen de rest van het hoofdstuk gebruikt zullen worden.

Vormgeving is het proces dat het resultaat van een XSL transformatie moet omzetten naar een bruikbare vorm voor de lezer of luisteraar. Het vormgevingsmodel dat XSL-FO gebruikt, bestaat uit het opbouwen van een gebiedsboom (area tree). Een gebiedsboom is een geordende boom die geometrische informatie bevat over de plaatsing van elk symbool, elke vorm en elke afbeelding in het document, samen met informatie over de spatiëring en andere informatie over de rendering. Deze informatie wordt beschreven door kenmerken. Deze kenmerken verhouden zich tot gebieden zoals eigenschappen zich verhouden tot vormgevingsobjecten. Wij gaan niet dieper in op de exacte eigenschappen van de gebiedsboom, omdat het hier over een abstract model gaat, dat niet hoeft geïmplementeerd te worden door de formatter. Voor meer informatie over het gebiedsmodel (area model) verwijzen wij naar [AREA].

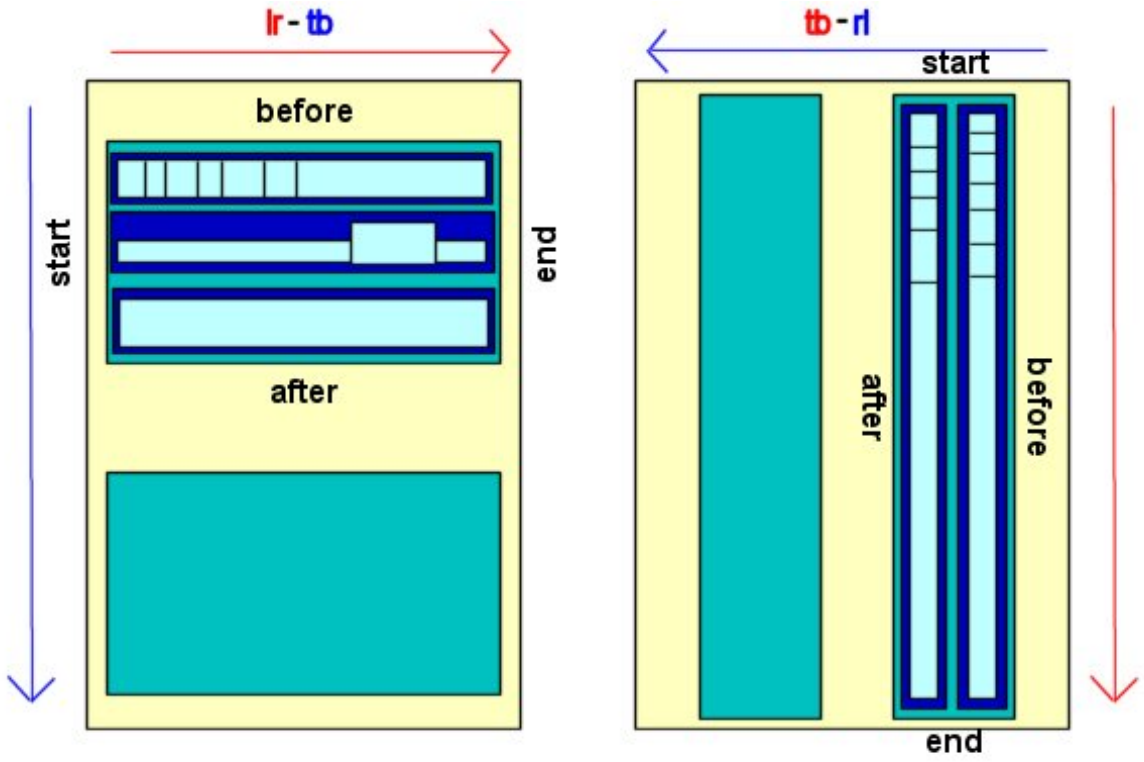
Vormgevingsobjecten [Sectie 7.5., Tabel 7.1.] zijn elementen die in de boom van vormgevingsobjecten zitten. De namen van vormgevingsobjecten behoren tot de XSL-FO namespace [TRXSLNA]. Wij bespreken deze namespace in [Sectie 7.4.] . Een vormgevingsobject hoort bij een klasse van vormgevingsobjecten die geïdentificeerd worden door de naam van het object. Het vormgevingsgedrag van elke klasse van vormgevingsobjecten wordt beschreven in termen van de gebieden die gecreëerd worden door een vormgevingsobject van die klasse en hoe die gebieden zich verhouden tot andere gebieden gecreëerd door ander vormgevingsobjecten.

Sommige vormgevingsobjecten zijn van blokniveau (**block-level**) [Sectie 7.7.1.] , andere zijn van in-regelniveau (**inline-level**) [Sectie 7.7.2.] . Dit verwijst naar de verschillende types van gebieden die de vormgevingsobjecten genereren. Deze gebieden verwijzen op hun beurt naar

hun default plaatsingsmethodes. In-regelgebieden (**inline-area**), bijvoorbeeld een symboolgebied, worden verzameld in lijnen (**lines**) en de richting waarin deze in-regelgebieden (**inline-areas**) opgestapeld worden, is de **inline-progression-direction**. Lijnen zijn dan weer een soort van blokgebied (**block-area**). Blokgebieden worden gestapeld in de richting loodrecht op de **inline-progression-direction**. Deze richting wordt de **block-progression-direction** genoemd.

In westerse schrijfsystemen is de **block-progression-direction** van boven naar onder en de **inline-progression-direction** van links naar rechts. Deze specificatie maakt het mogelijk ook andere schrijfsystemen (van rechts naar links bijvoorbeeld) te gebruiken, doordat er gewerkt wordt met de termen **block** en **inline** in plaats van absolute indicatoren als vertikaal en horizontaal. Er wordt ook zoveel mogelijk gebruik gemaakt van relatief gespecificeerde richtingen waar dat mogelijk is: **before** en **after** voor de **block-progression-direction**, **start** en **end** voor de **inline-progression-direction**. Deze relatief gespecificeerde richtingen kunnen een aanvulling zijn op of gebruikt worden in de plaats van absoluut gespecificeerde richtingen zoals **top**, **bottom**, **left** en **right**. Ze worden geïnterpreteerd aan de hand van de waardes van de **writing-mode** eigenschap. De **writing-mode** kan bijvoorbeeld de code **lr-tb** (left-to-right - top-to-bottom) of **tb-rl** (top-to-bottom - right-to-left) zijn. Deze code specificeert de **inline-progression-direction** en de **block-progression-direction**. **lr-tb** wil bijvoorbeeld zeggen dat de tekst van links naar rechts en van boven naar onder geschreven wordt. In het vervolg van deze tekst wijden wij nog een aparte sectie aan het **writing-mode** attribuut [Sectie 7.15.6.8.] .

In onderstaande figuur worden deze concepten visueel voorgesteld. De rode pijl stelt de **inline-progression-direction** voor, de blauwe de **block-progression-direction**. Links wordt de **writing-mode** van een westers schrift geïllustreerd, rechts die van het Chinees schrift.



Figuur 7.3. Illustratie van het writing-mode attribuut

Centraal in dit vormgevingsmodel staat verfijning. Dit is een rekenkundig proces dat de specificatie van eigenschappen gebaseerd op de attribuutwaardes van de vormgevingsobjecten in de resulterende XML-boom finaliseert. Verfijning houdt het volgende in:

- het doorgeven van de verschillende overgeërfde waardes of eigenschappen (zowel de impliciete waardes of eigenschappen als die welke een attribuutwaarde **inherit** hebben).
- het evalueren van uitdrukkingen in specificaties van waardes van eigenschappen naar actuele waardes, welke dan gebruikt worden om de waardes van de eigenschappen te bepalen
- het omzetten van relatieve getallen naar absolute getallen
- het opbouwen van bepaalde samengestelde eigenschappen van meer dan één attribuut

Sommige van deze operaties (in het bijzonder het evalueren van uitdrukkingen) hangen af van de kennis van de gebiedsboom. Verfijning is dus niet noodzakelijk een recht toe recht aan sequentiële procedure, er kan ook look-ahead of backtracking aan te pas komen.

Samengevat: vormgeving begint met het opbouwen van een gebiedsboom die gebieden met hun kenmerken bevat. Deze boom moet voldoen aan een aantal beperkingen gebaseerd op de informatie die vervat zit in de resulterende XML-boom (die elementknooppunten en attributen bevat). Conceptueel zijn er tussenliggende stappen om de boom van vormgevingsobjecten (die vormgevingsobjecten en hun eigenschappen bevat) op te bouwen en gebeuren er verfijningen. Deze stappen kunnen door elkaar lopen gedurende het opbouwen van de gebiedsboom.

7.4. XSL-FO namespace

De XSL-FO namespace heeft als URI *http://www.w3.org/1999/XSL/Format*. XSL processoren moeten gebruik maken van het XML namespace mechanisme [Sectie 2.4.] , [TRXMLNA] om elementen en attributen van deze namespace te herkennen. Het is niet toegestaan de XSL-FO namespace uit te breiden met bijkomende elementen of attributen. Elke nieuwe uitbreiding moet in een aparte namespace gedefinieerd worden.

In onze tekst zullen wij gebruik maken van het voorvoegsel **fo:** om te verwijzen naar elementen in de XSL-FO namespace. Het is echter perfect mogelijk om een ander voorvoegsel te gebruiken op voorwaarde dat er een namespace declaratie is die het voorvoegsel verbindt met de URI van de XSL-FO namespace.

Een element van de XSL-FO namespace kan attributen bevatten die niet tot de XSL-FO namespace behoren, vooropgesteld dat de gekwalificeerde naam van het attribuut een namespace URI heeft die niet leeg is. De aanwezigheid van zulke attributen mag het gedrag van de XSL elementen en functies niet veranderen. Bijgevolg mag een XSL processor zulke attributen altijd negeren en moet hij zulke attributen negeren zonder foutmelding als de processor de namespace URI niet herkent. Zulke attributen kunnen bijvoorbeeld zorgen voor unieke identifiers of kunnen hulpmiddelen voor documentatie zijn.

De conventies die gebruikt worden voor de namen van XSL-FO elementen, attributen en functies zijn de volgende: namen zijn allemaal in kleine letters, koppeltekens worden gebruikt om woorden van elkaar te scheiden en punten worden gebruikt om namen voor de componenten van complexe gegevenstypes van elkaar te scheiden. Afkortingen worden enkel gebruikt als ze voorkomen in de syntax van een verwante taal zoals XML of HTML.

7.5. XSL vormgevingsobjecten en hun eigenschappen

Het eerste fundamentele idee is dat XSL geen manier levert om rechtstreeks vormgevingseigenschappen aan elementen in een XML document te verbinden. In plaats daarvan verbindt het vormgevingseigenschappen aan vormgevingsobjecten.

XSL-FO levert een meer gesofistikeerd visueel vormgevingsmodel dan HTML [TRHTML] en CSS [TRCSS]. Een aantal voorbeelden van vormgeving die ondersteund wordt door XSL-FO, maar niet door HTML en CSS zijn onder andere tekst die van rechts naar links of van boven naar onder loopt, voetnoten, noten in de marge, paginanummering in kruisverwijzingen, ... CSS is allereerst bedoeld voor gebruik op het Web. XSL-FO is ontworpen voor een veel breder gebruik. Het moet bijvoorbeeld mogelijk zijn een XSL stylesheet te schrijven die vormgevingsobjecten gebruikt om een volledig gedrukt boek van lay-out te voorzien. Een andere stylesheet moet in staat zijn hetzelfde XML document om te zetten naar een website.

Er zijn drie soorten vormgevingsobjecten:

- vormgevingsobjecten die gebieden genereren (1)
- vormgevingsobjecten die gebieden teruggeven, maar ze niet genereren (2)
- vormgevingsobjecten die gebruikt worden in de generatie van gebieden (3)

De eerste en de tweede soort van vormgevingsobjecten worden stroomobjecten (flow objects) genoemd. De derde soort is ofwel een lay-outobject of een hulpobject.

Er zijn exact 56 XSL vormgevingsobject elementen. Deze behoren tot de reeds eerder besproken *http://www.w3.org/1999/XSL/Format* namespace [Sectie 7.4.] . De meeste van deze 56 elementen zijn verschillende soorten van rechthoekige gebieden. Het merendeel van de resterende elementen zijn containers voor rechthoekige gebieden.

We geven nu een alfabetisch overzicht van deze vormgevingsobjecten met telkens een korte uitleg. In sommige gevallen zal de uitleg een beetje abstract en moeilijk te vatten zijn. Wij hebben ervoor gekozen om in deze tekst eerst een algemene opsomming te geven van de verschillende mogelijkheden en later in de tekst terug te komen op de meer specifieke eigenschappen van deze vormgevingsobjecten.

Tabel 7.1. Overzicht van vormgevingsobjecten

Vormgevingsobject	**Korte beschrijving**
`fo:basic-link`	Het **fo:basic-link** vormgevingsobject wordt gebruikt om de bron (source) van een eenvoudige link weer te geven.
`fo:bidi-override`	Het **fo:bidi-override** in-regelvormgevingsobject wordt gebruikt daar waar het nodig is om de defaultrichting van het Unicode-bidirectional-algorithm [UCBA] te overschrijven. Dit kan nodig zijn voor verschillende (of geneste) in-regel vormgevingsobjecten die voorkomen in documenten met een gemengde taalinhoud.
`fo:block`	Het **fo:block** vormgevingsobject wordt gebruikt voor het vormgeven van paragrafen, titels, figuren, tabellen, enzovoort.
`fo:block-container`	Het **fo:block-container** stroomobject wordt gebruikt om een referentiegebied op blokniveau te genereren.
`fo:character`	Het **fo:character** stroomobject stelt een teken uit je invoerdocument voor dat gemapt wordt op een symbool voor de presentatie ervan.
`fo:color-profile`	Het **fo:color-profile** vormgevingsobject wordt gebruikt om een kleurprofiel te declareren voor een stylesheet.
`fo:conditional-page-master-reference`	Het **fo:conditional-page-master-reference** vormgevingsobject wordt gebruikt om een page-master te identificeren die

	gebruikt wordt wanneer aan de voorwaarden voor zijn gebruik voldaan zijn.
`fo:declarations`	Het **fo:declarations** vormgevingsobject wordt gebruikt om globale declaraties in een stylesheet te groeperen. Eén van die declaraties kan bijvoorbeeld het kleurprofiel zijn dat door het document wordt gebruikt.
`fo:external-graphic`	Het **fo:external-graphic** stroomobject wordt gebruikt voor grafische objecten waarvan de grafische gegevens zich niet in de resulterende boom in de **fo** namespace bevinden.
`fo:float`	Het **fo:float** vormgevingsobject heeft twee doeleinden. Het kan gebruikt worden bij de gewone plaatsing van inhoud, zodat sommige verwante inhoud vorm gegeven wordt in een apart gebied aan het begin van de pagina (of op een volgende pagina). Zo is deze inhoud beschikbaar voor de lezer zonder zich direct op te dringen. Aan de andere kant kan het **fo:float** vormgevingsobject gebruikt worden wanneer het de bedoeling is dat een gebied aan één kant drijft (to float) terwijl de gewone inhoud erlangs stroomt, wij denken hier bijvoorbeeld aan een afbeelding waarrond tekst geplaatst wordt.
`fo:flow`	De inhoud van het **fo:flow** vormgevingsobject is een opeenvolging van stroomobjecten die de stromende tekstinhoud leveren die verdeeld wordt over verschillende pagina's.
`fo:footnote`	Het **fo:footnote** object wordt gebruikt om een aanhaling van een voetnoot en de overeenkomstige voetnoot te produceren.
`fo:footnote-body`	Het **fo:footnote-body** element wordt gebruikt om de inhoud van de voetnoot te genereren.
`fo:initial-property-set`	Het **fo:initial-property-set** object specificeert vormgevingseigenschappen voor de eerste lijn van een **fo:block**.
`fo:inline`	Het **fo:inline** vormgevingsobject wordt meestal gebruikt om een deel van een tekst een bepaalde achtergrond te geven of om het te omsluiten met een kader.
`fo:inline-container`	Het **fo:inline-container** stroomobject wordt gebruikt om een in-regel referentiegebied te genereren.
`fo:instream-foreign-object`	Het **fo:instream-foreign-object** stroomobject wordt gebruikt voor een in-regel grafisch object of een ander "generisch" object waarvan de objectgegevens kinderen zijn van **fo:instream-foreign-object**.
`fo:layout-master-set`	Het **fo:layout-master-set** omvat alle masters die gebruikt worden in het document.
`fo:leader`	Het **fo:leader** vormgevingsobject wordt gebruikt om een leader (bijvoorbeeld een lijn van punten) te genereren die bestaat uit ofwel een lijn (rule) ofwel uit een rij van een zich herhalend teken of een zich cyclisch herhalend patroon van tekens. Zo'n leader kan gebruikt worden om twee tekstvormgevingsobjecten te verbinden.
`fo:list-block`	Het **fo:list-block** stroomobject wordt gebruikt om een lijst vorm te geven.
`fo:list-item`	Het **fo:list-item** vormgevingsobject bevat het label en het lichaam van een lijstitem.
`fo:list-item-body`	Het **fo:list-item-body** vormgevingsobject bevat de inhoud van het lichaam van een lijstitem.
`fo:list-item-label`	Het **fo:list-item-label** vormgevingsobject bevat de inhoud van het label van een lijstitem. Dit wordt gebruikt om het lichaam van een lijstitem te tellen, te identificeren of te ver-

	fraaien.
`fo:marker`	Het **fo:marker** object wordt gebruikt in combinatie met het **fo:retrieve-marker** object om lopende hoofdingen of footers te produceren.
`fo:multi-case`	Het **fo:multi-case** element wordt gebruikt binnen een **fo:multi-switch** element. Het bevat een aantal alternatieve deelbomen van vormgevingsobjecten waaruit de ouder **fo:multi-switch** er één zal kiezen om weer te geven en de overige zal verbergen.
`fo:multi-properties`	Het **fo:multi-properties** object wordt gebruikt om te wisselen tussen twee of meer sets van eigenschapen die geassocieerd zijn met een gegeven inhoud.
`fo:multi-property-set`	Het **fo:multi-property-set** element wordt gebruikt om een alternatieve set van vormgevingseigenschappen te specificeren die, afhankelijk van de status van de User Agent (bijvoorbeeld een Web browser), toegepast worden op de inhoud.
`fo:multi-switch`	Het **fo:multi-switch** object omvat de specificatie van alternatieve deelbomen van vormgevingsobjecten die elk in een **fo:multi-case** object zitten. Het controleert het wisselen (geactiveerd door **fo:multi-toggle**) van het ene alternatief naar het andere.
`fo:multi-toggle`	Het **fo:multi-toggle** vormgevingsobject wordt gebruikt binnen een **fo:multi-case** object om te wisselen van **fo:multi-case** object.
`fo:page-number`	Het **fo:page-number** vormgevingsobject wordt gebruikt om het huidige paginanummer voor te stellen.
`fo:page-number-citation`	Het **fo:page-number-citation** object wordt gebruikt om te verwijzen naar het paginanummer van de pagina die het eerste gebied bevat dat gegeven wordt door het vormgevingsobject waarnaar verwezen wordt.
`fo:page-sequence`	Het **fo:page-sequence** vormgevingsobject wordt gebruikt om te specificeren hoe een opeenvolging van pagina's binnen een document vormgegeven moet worden, bijvoorbeeld de verschillende pagina's van een hoofdstuk van een thesis. De inhoud van deze pagina's wordt geleverd door de stroomkinderen van het **fo:page-sequence** element.
`fo:page-sequence-master`	Het **fo:page-sequence-master** object specificeert opeenvolgingen van page-masters die gebruikt worden om een opeenvolging van verschillende pagina's te genereren.
`fo:region-after`	Deze regio definieert een rechthoekig kijkgebied (een viewport) dat zich situeert aan de na-zijde ("after") van de **fo:region-body** regio.
`fo:region-before`	Deze regio definieert een rechthoekig kijkgebied (een viewport) dat zich situeert aan de voor-zijde ("before") van de **fo:region-body** regio.
`fo:region-body`	Deze regio definieert een rechthoekig kijkgebied/referentie paar dat zich situeert in het midden van de **fo:simple-page-master**.
`fo:region-end`	Deze regio definieert een rechthoekig kijkgebied (een viewport) dat zich situeert aan de eind-zijde ("end") van de **fo:region-body** regio.
`fo:region-start`	Deze regio definieert een rechthoekig kijkgebied (een viewport) dat zich situeert aan de begin-zijde ("start") van de

	fo:region-body regio.
`fo:repeatable-page-master-alternatives`	Het **fo:repeatable-page-master-alternatives** element specificeert een opeenvolging van herhaalde voorkomens van een set van alternatieve page-masters. Het aantal herhalingen kan gelimiteerd zijn of mogelijk zelfs ongelimiteerd.
`fo:repeatable-page-master-reference`	Het **fo:repeatable-page-master-reference** specificeert een opeenvolging van herhaalde voorkomens van een single-page-master. Het aantal herhalingen kan gelimiteerd zijn of mogelijk zelfs ongelimiteerd.
`fo:retrieve-marker`	Het **fo:retrieve-marker** vormgevingsobject wordt gebruikt samen met **fo:marker** om lopende hoofdingen en footers te produceren.
`fo:root`	Het **fo:root** knooppunt is het wortelknooppunt van een XSL resulterende boom. Deze boom is opgebouwd uit vormgevingsobjecten.
`fo:simple-page-master`	Het **fo:simple-page-master** vormgevingsobject wordt gebruikt bij het genereren van de pagina's en specificeert de geometrie van de pagina. De pagina mag onderverdeeld worden in maximaal vijf regio's.
`fo:single-page-master-reference`	Het **fo:single-page-master-reference** vormgevingsobject specificeert een subsequentie die bestaat uit een enkele instantie van een enkelvoudige page-master.
`fo:static-content`	Het **fo:static-content** vormgevingsobject bevat een opeenvolging van of een boom van vormgevingsobjecten die voorgesteld moeten worden in een enkele regio of herhaald moeten worden in een gelijkaardig genaamde regio op één of meer pagina's in de **fo:page-sequence**. Dit object wordt meestal gebruikt om hoofdingen en footers te herhalen.
`fo:table`	Het **fo:table** stroomobject wordt gebruikt om een tabel vorm te geven.
`fo:table-and-caption`	Het **fo:table-and-caption** stroomobject wordt gebruikt om een tabel samen met zijn bijschrift vorm te geven.
`fo:table-body`	Het **fo:table-body** vormgevingsobject wordt gebruikt de inhoud van het lichaam van de tabel te bevatten.
`fo:table-caption`	Het **fo:table-caption** vormgevingsobject wordt gebruikt om vormgevingsobjecten van blokniveau te bevatten die het bijschrift van de tabel bevatten. Dit element wordt alleen gebruikt samen met **fo:table-and-caption**.
`fo:table-cell`	Het **fo:table-cell** vormgevingsobject wordt gebruikt om inhoud te groeperen die in een cel van een tabel geplaatst wordt.
`fo:table-column`	Het **fo:table-column** vormgevingsobject specificeert eigenschappen die van toepassing zijn op cellen van tabellen die dezelfde kolom en span hebben.
`fo:table-footer`	Het **fo:table-footer** vormgevingsobject wordt gebruikt om de inhoud van de footer van de tabel te bevatten.
`fo:table-header`	Het **fo:table-header** vormgevingsobject wordt gebruikt om de inhoud van een hoofding van een tabel te bevatten.
`fo:table-row`	Het **fo:table-row** vormgevingsobject wordt gebruikt om cellen van een tabel te groeperen in een rij.
`fo:title`	Het **fo:title** vormgevingsobject wordt gebruikt om een titel met een gegeven **fo:page-sequence** te associëren. Deze titel kan gebruikt worden door een interactieve User Agent om de pagina's te identificeren. De inhoud van het **fo:title**, element kan bijvoorbeeld getoond worden in een titelvenster.

`fo:wrapper` Het **fo:wrapper** vormgevingsobject wordt gebruikt om overgeërfde eigenschappen voor een groep van vormgevingsobjecten te specificeren. Het heeft geen bijkomde vormgevingssemantiek.

Het XSL vormgevingsmodel is gebaseerd op rechthoekige cellen, die gebieden genoemd worden. Deze gebieden kunnen tekst, lege ruimte, afbeeldingen of andere vormgevingsobjecten bevatten. Een gebied heeft begrenzingen en opvulling (padding) aan beide kanten, aangeduid door **space-before** en **space-after xslfo_margin**. Een XSL formatter leest de vormgevingsobjecten om te bepalen waar welke gebieden geplaatst moeten worden op de pagina. Veel vormgevingsobjecten produceren enkelvoudige gebieden, althans toch het merendeel van de tijd. Maar er zijn veel details, zoals het einde van een pagina, het plaatsen van koppeltekens, enzovoort waarmee rekening gehouden moet worden wanneer een potentieel oneindige hoeveelheid tekst op een eindige hoeveelheid ruimte moet afgebeeld worden. Daarom genereren sommige vormgevingsobjecten nu en dan meer dan één gebied.

Vormgevingsobjecten verschillen hoofdzakelijk in wat ze voorstellen. Bijvoorbeeld het **fo:list-item-label** vormgevingsobject is een cel die een bolletje, een getal of iets dergelijks plaatst voor een element van een lijst. Een **fo:list-item-body** vormgevingsobject is een cel die de tekst van het lijstitem bevat (zonder label). En het **fo:list-item** vormgevingsobject is een cel die zowel het **fo:list-item-label** als het **fo:list-item-body** vormgevingsobject bevat. [Sectie 7.10.]

Bij het verwerken wordt het document met de vormgevingsobjecten opgedeeld in pagina's. Het venster van een Web browser wordt meestal behandeld als één heel lange pagina. Een printformaat zal vaak uit vele individuele pagina's bestaan. Elke pagina bevat een zeker aantal gebieden. De vier hoofdgebieden in XSL zijn: **region** (regio), **block area** (blokgebied), **line area** (regelgebied) en **inline area** (in-regelgebied). Die gebieden vormen een zekere hiërarchie. Regio's kunnen blokgebieden bevatten. Blokgebieden kunnen andere blokgebieden, regelgebieden, in-regelgebieden en inhoud bevatten. Regelgebieden bevaten in-regelgebieden. In-regelgebieden bevatten andere in-regelgebieden en inhoud. We verduidelijken deze hiërarchie:

- Een **regio** is de container van het hoogste niveau in XSL-FO. De pagina van een boek bijvoorbeeld kan drie regio's bevatten: de header, het lichaam van de pagina zelf en de footer. Vormgevingsobjecten die regio's produceren zijn **fo:region-body**, **fo:region-before**, **fo:region-start**, **fo:region-end** en **fo:region-after**. [Sectie 7.6.2.1.]
- Een **blokgebied** stelt een blokniveau (block-level) element voor, bijvoorbeeld een paragraaf of een lijstelement. Alhoewel blokgebieden andere blokgebieden kunnen bevatten, moet er altijd een line break voor het begin en na het einde van elk blokgebied staan. Een blokgebied wordt in plaats van exact gepositioneerd te worden, sequentieel geplaatst in een gebied dat het blokgebied bevat. Als andere blokgebieden toegevoegd of weggenomen worden vóór of binnen het blokgebied, schuift de positie van het blokgebied op in de juiste richting om plaats te maken. Een blokgebied kan geparste tekens, in-regelgebieden, regelgebieden en andere blokgebieden die sequentieel gerangschikt worden in het bevattende blokgebied, bevatten. Vormgevingsobjecten die blokgebieden produceren zijn **fo:block**, **fo:table-and-caption** en **fo:list-block**. [Sectie 7.7.1.]
- Een **lijngebied** stelt een lijn met tekst binnen een blok voor. Bijvoorbeeld, elk van de lijnen in dit lijstelement is een lijngebied. Lijngebieden kunnen in-regelgebieden en in-regelspaties bevatten. Er zijn geen vormgevingsobjecten die lijngebieden produceren. Het vormgevingsmechanisme berekent de lijngebieden wanneer het beslist hoe de lijnen binnen het blokgebied gerangschikt worden.
- In-regel gebieden zijn een onderdeel van een lijn, zoals een enkel letterteken, een

referentie naar een voetnoot of een wiskundige vergelijking. In-regelgebieden kunnen andere in-regelgebieden en naakte tekst bevatten. Vormgevingsobjecten die in-regelgebieden produceren zijn **fo:character**, **fo:external-graphic**, **fo:inline**, **fo:instream-foreign-object**, **fo:leader** en **fo:page-number**. [Sectie 7.7.2.]

7.5.1. Vormgevingseigenschappen

Als we de verschillende **vormgevingsobjecten** in een XSL-FO document als een geheel bekijken, dan geven zij de volgorde weer waarin inhoud op pagina's geplaatst wordt. **Vormgevingseigenschappen** daarentegen specificeren de vormgevingsdetails zoals daar zijn grootte, positie, lettertype, kleur, enzovoort. Vormgevingseigenschappen worden voorgesteld als attributen van de individuele vormgevingsobject elementen.

Veel van deze eigenschappen zullen je bekend voorkomen als je al met CSS gewerkt hebt. Zo betekent bijvoorbeeld de **font-family** eigenschap van CSS hetzelfde als de **font-family** eigenschap van XSL [Sectie 7.15.4.2.] . De syntax voor het toekennen van waardes aan eigenschappen is echter anders bij XSL dan bij CSS. Als we bijvoorbeeld het **fo:block** willen weergeven in een zeker lettertype, dan krijgen we in CSS:

fo: block {font-family: 'New York', 'Times New Roman', serif}

In de XSL-FO wordt een **font-family** attribute opgenomen in de **fo:block** starttag:

Voorbeeld 7.1.

```
<fo:block font-family="'New York', 'Times New Roman', serif"/>
```

Alhoewel dit oppervlakkig verschillend is, zijn de stijlnaam (**font-family**) en de stijlwaarde (**'New York', 'Times New Roman', serif**) hetzelfde. De **font-family** eigenschap van CSS is gespecificeerd als een lijst van namen van lettertypes, geordend van eerste keuze naar laatste keuze, door komma's gescheiden. De **font-family** eigenschap van XSL-FO is gespecificeerd als een lijst van namen van lettertypes, geordend van eerste keuze naar laatste keuze, door komma's gescheiden. Zowel CSS als XSL-FO gebruiken namen van lettertypes die witruimte bevatten. Zowel CSS als XSL-FO begrijpen het **serif** sleutelwoord.

XSL vormgevingsobjecten ondersteunen ook veel eigenschappen die geen CSS equivalent hebben, zoals **destination-placement-offset**, **block-progression-dimension**, **character** en **hyphenation-keep**. Om ten volle gebruik te maken van XSL, is het belangrijk dat je deze eigenschappen leert gebruiken.

Hieronder geven we een overzicht van de meer dan tweehonderd standaard XSL-FO eigenschappen:

- absolute-position
- active-state
- alignment-adjust
- auto-restore
- azimuth
- background
- background-attachment
- background-color
- background-image

- background-position
- background-position-horizontal
- background-position-vertical
- background-repeat
- baseline-shift
- blank-or-not-blank
- block-progression-dimension
- border
- border-after-color
- border-after-precedence
- border-after-style
- border-after-width
- border-before-color
- border-before-precedence
- border-before-style
- border-before-width
- border-bottom
- border-bottom-color
- border-bottom-style
- border-bottom-width
- border-collapse
- border-color
- border-end-color
- border-end-precedence
- border-end-style
- border-end-width
- border-left
- border-left-color
- border-left-style
- border-left-width
- border-right
- border-right-color
- border-right-style
- border-right-width
- border-separation
- border-spacing
- border-start-color
- border-start-precedence
- border-start-style
- border-start-width
- border-style
- border-top
- border-top-color
- border-top-style
- border-top-width
- border-width
- bottom
- break-after
- break-before
- caption-side
- case-name
- case-title
- character
- clear

- clip
- color
- color-profile-name
- column-count
- column-gap
- column-number
- column-width
- content-height
- content-type
- content-width
- country
- cue
- cue-after
- cue-before
- destination-placement-offset
- direction
- display-align
- dominant-baseline
- elevation
- empty-cells
- end-indent
- ends-row
- extent
- external-destination
- float
- flow-name
- font
- font-family
- font-selection-strategy
- font-size
- font-size-adjust
- font-stretch
- font-style
- font-variant
- font-weight
- force-page-count
- format
- glyph-orientation-horizontal
- glyph-orientation-vertical
- grouping-separator
- grouping-size
- height
- hyphenate
- hyphenation-character
- hyphenation-keep
- hyphenation-ladder-count
- hyphenation-push-character-count
- hyphenation-remain-character-count
- id
- indicate-destination
- initial-page-number
- inline-progression-dimension
- internal-destination
- keep-together

- keep-with-next
- keep-with-previous
- language
- last-line-end-indent
- leader-alignment
- leader-length
- leader-pattern
- leader-pattern-width
- left
- letter-spacing
- letter-value
- linefeed-treatment
- line-height
- line-height-shift-adjustment
- line-stacking-strategy
- margin
- margin-bottom
- margin-left
- margin-right
- margin-top
- marker-class-name
- master-name
- master-reference
- max-height
- maximum-repeats
- max-width
- media-usage
- min-height
- min-width
- number-columns-repeated
- number-columns-spanned
- number-rows-spanned
- odd-or-even
- orphans
- overflow
- padding
- padding-after
- padding-before
- padding-bottom
- padding-end
- padding-left
- padding-right
- padding-start
- padding-top
- page-break-after
- page-break-before
- page-break-inside
- page-height
- page-position
- page-width
- pause
- pause-after
- pause-before
- pitch

- pitch-range
- play-during
- position
- precedence
- provisional-distance-between-starts
- provisional-label-separation
- reference-orientation
- ref-id
- region-name
- relative-align
- relative-position
- rendering-intent
- retrieve-boundary
- retrieve-class-name
- retrieve-position
- richness
- right
- role
- rule-style
- rule-thickness
- scaling
- scaling-method
- score-spaces
- script
- show-destination
- size
- source-document
- space-after
- space-before
- space-end
- space-start
- space-treatment
- span
- speak
- speak-header
- speak-numeral
- speak-punctuation
- speech-rate
- src
- start-indent
- starting-state
- starts-row
- stress
- suppress-at-line-break
- switch-to
- table-layout
- table-omit-footer-at-break
- table-omit-header-at-break
- target-presentation-content
- target-processing-context
- target-stylesheet
- text-align
- text-align-last
- text-altitude

- text-decoration
- text-depth
- text-indent
- text-shadow
- text-transform
- top
- treat-as-word-space
- unicode-bidi
- vertical-align
- visibility
- voice-family
- volume
- white-space
- white-space-collapse
- widows
- width
- word-spacing
- wrap-option
- writing-mode
- xml:lang
- z-index

7.5.2. Transformatie naar vormgevingsobjecten

XSL-FO is een volledige XML woordenschat om teksten op een pagina te rangschikken. Een XSL-FO document is gewoon een goedgevormd XML document dat gebruik maakt van deze woordenschat. Dat betekent dat een XSL-FO document een XML declaratie, een rootelement, kindelementen, enzovoort bevat. Het moet voldoen aan alle goedgevormdheidregels van een XML document voordat een formatter het document accepteert. [Sectie 2.3.] De conventie is dat een bestand dat XSL vormgevingsobjecten bevat de drie-letter extensie **.fob** of de twee-letter extensie **.fo** hebben. Een XSL-FO document kan echter ook een **.xml** achtervoegsel hebben omdat het natuurlijk ook een goedgevormd XML document is. Extensies zijn slechts conventies. Iedereen is vrij een extensie te kiezen, indien er al één gekozen wordt.

Hieronder vind je een eenvoudig document dat gebruikt maakt van XSL vormgevingsobjecten. De root van het document is **fo:root** [Sectie 7.6.1.] . Dit element bevat een **fo:layout-master-set** en een **fo:page-sequence** kindelement. Het **fo:layout-master-set** element bevat op zijn beurt een **fo:simple-page-master** kindelement. [Sectie 7.6.] Elk **fo:simple-page-master** element beschrijft de soort pagina waarop de inhoud geplaatst zal worden. [Sectie 7.6.2.] In dit eenvoudig voorbeeldje is dit slechts één heel eenvoudige pagina, maar meer complexe documenten kunnen verschillende page-masters hebben voor bijvoorbeeld de eerste en de laatste pagina, voor de voor- en achterzijde van een pagina, enzovoort. Elke page-master kan verschillende paginanummers, instellingen van kantlijnen, enzovoort definiëren.

De eigenlijke inhoud wordt aan de hand van het **fo:page-sequence** element geplaatst op kopies van de originele page-master. Het **fo:page-sequence** element heeft een **master-reference** attribuut dat specificeert welke page-master gebruikt moet worden. [Sectie 7.6.3.] Het **fo:flow** kindelement van **fo:page-sequence** bevat de eigenlijke inhoud die op de pagina's geplaatst moet worden. De inhoud hier bestaat uit twee **fo:block** kindelementen die elk een **font-size** eigenschap hebben die 12 points is, een **font-family** eigenschap die serif is

en een **line-height** eigenschap die 14 points is.

Voorbeeld 7.2. Een eenvoudig XSL-FO document

```
<?xml version="1.0"?>
 <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
       <fo:simple-page-master master-name="enige-page-master">
          <fo:region-body/>
       </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="enige-page-master">
       <fo:flow flow-name="xsl-region-body">
          <fo:block font-size="12pt" font-family="serif"
          line-height="14pt"> Eerste hoofdstuk </fo:block>
          <fo:block font-size="12pt" font-family="serif"
          line-height="14pt"> Tweede hoofdstuk </fo:block>
       </fo:flow>
    </fo:page-sequence>
 </fo:root>
```

Wij hebben dus net geïllustreerd dat het perfect mogelijk is zelf een XSL-FO document in te typen. Door dit te doen, verlies je wel alle voordelen van de scheiding van inhoud en voorstelling die XML zo hoog in het vaandel draagt. De normale procedure bestaat uit het schrijven van een XSLT stylesheet die een XML brondocument transformeert naar een XSL-FO document. Een formatteringsprogramma, de Formatting Objects Processor, genereert vervolgens de uiteindelijke presentatie naar het gekozen medium (web browser of een af te printen document).

Hieronder geven we een XSLT stylesheet die bovenstaand XSL-FO document zou kunnen opleveren.

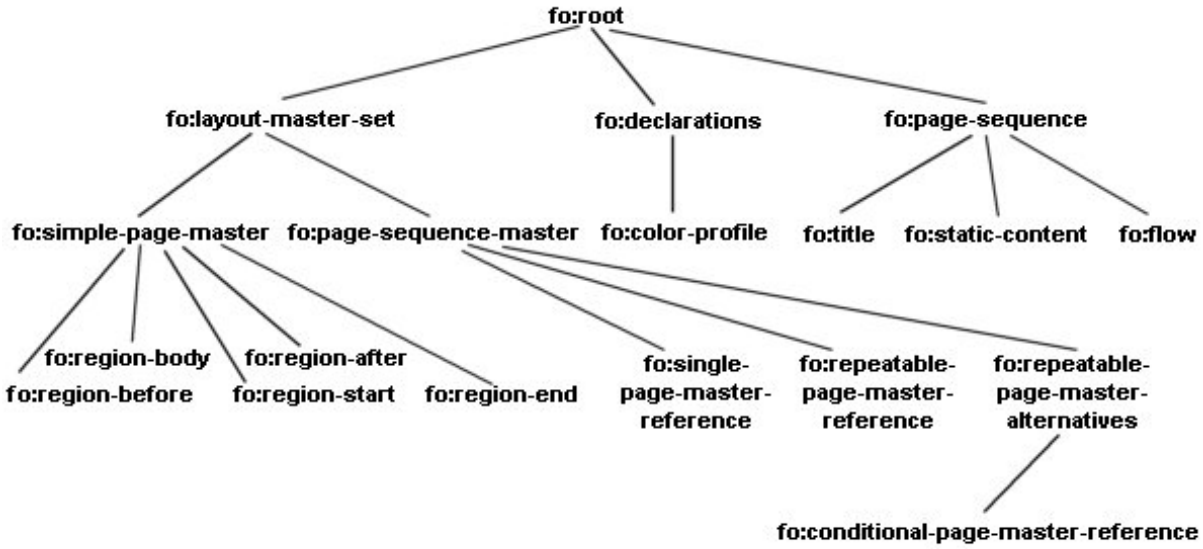
Voorbeeld 7.3. Een transformatie naar XSL vormgevingsobjecten

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <xsl:output indent="yes"/>
    <xsl:template match="THESIS">
       <fo:root xsmlns:fo="http://www.w3.org/1999/XSL/Format">
          <fo:layout-master-set>
             <fo:simple-page-master master-name="enige-page-master">
                <fo:region-body/>
             </fo:simple-page-master>
          </fo:layout-master-set>
          <fo:page-sequence master-reference="enige-page-master">
             <xsl:apply-templates select="CHAPTER"/>
          </fo:page-sequence>
       </fo:root>
    </xsl:template>
    <xsl:template match="CHAPTER">
       <fo:block font-size="12pt" font-family="serif" line-height="14pt">
          <xsl:value-of select="TITLE"/>
       </fo:block>
    </xsl:template>
 </xsl:stylesheet>
```

7.6. De lay-out van een pagina

Het rootelement van een document met vormgevingsobjecten is **fo:root**. Dit element bevat één **fo:layout-master-set** kindelement en één of meerdere **fo:page-sequence** kindelementen. De **fo:page-sequence** elementen kunnen inhoud bevatten, met name tekst en afbeeldingen die op een pagina geplaatst moeten worden. Het **fo:layout-master-set** element bevat templateregels voor de pagina's die gecreëerd zullen worden. Wanneer de formatter een XSL-FO document leest, creëert deze een pagina gebaseerd op één van de templates in de **fo:layout-master-set**. Vervolgens vult het deze pagina met inhoud van de **fo:page-sequence** elementen. Wanneer de eerste pagina opgevuld is, wordt een tweede pagina gecreëerd die gebaseerd is op een template van de **fo:layout-master-set** en wordt deze vervolgens weer gevuld met de inhoud. Dit proces wordt verder gezet totdat de formatter geen inhoud meer te verwerken heeft.

Voordat we overgaan naar de bespreking van de belangrijkste vormgevingsobjecten voor het bepalen van de lay-out van een pagina geven we een overzicht van de boomstructuur van deze vormgevingsobjecten. In deze boom zijn alle mogelijke objecten opgenomen, sommige objecten, zoals het **fo:declarations** object, zijn echter optioneel.



Figuur 7.4. Boomstructuur van de vormgevingsobjecten

7.6.1. Het rootelement

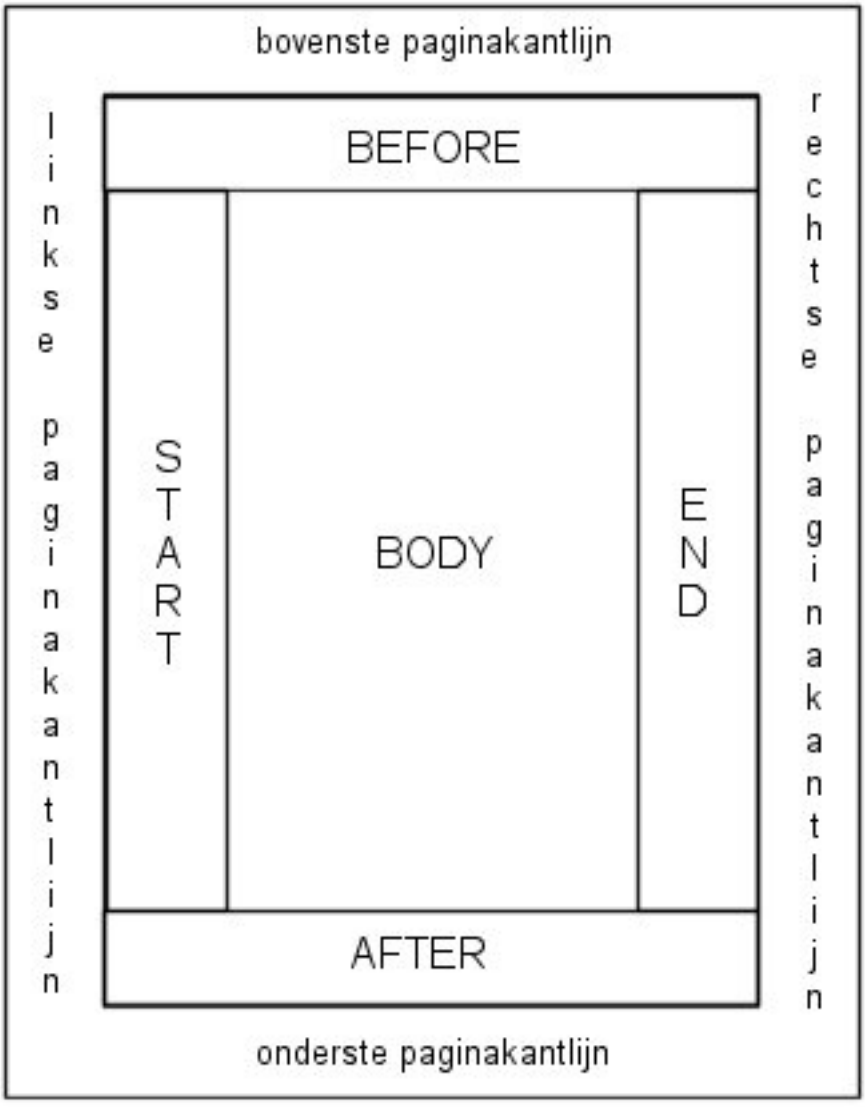
Het **fo:root** element heeft een **xmlns:fo** attribuut met als waarde *http://www.w3.org/1999/XSL/Format*. Het **fo:root** element bestaat alleen om de namespace te kunnen declareren en om het rootelement van een document te vormen. Het element zelf heeft geen direct gevolg op de vormgeving en de lay-out van de pagina's.

7.6.2. Het fo:simple-page-master element

De templates voor de pagina's worden page-masters genoemd. Page-masters zijn paginasjablonen die een algemene lay-out voor een pagina definiëren . Die lay-out houdt onder andere in: kantlijnen, de grootte van de hoofding, de footer en het lichaam van de pagina, enzovoort. Elke effectieve pagina in je uiteindelijke gegenereerde document is

gebaseerd op één page-master en erft bepaalde eigenschappen zoals kantlijnen en de grootte van de hoofding van deze page-master. XSL-FO 1.0 definieert één soort page-master, de **fo:simple-page-master**. Deze stelt een rechthoekige pagina voor. Het is waarschijnlijk dat in de toekomst andere soorten page-masters opgenomen zullen worden in XSL-FO. Deze zullen dan misschien zelfs pagina's die niet rechthoekig zijn, definiëren.

Het **fo:layout-master-set** element heeft één of meerdere **fo:simple-page-master** kindelementen die elk een verschillende page-master definiëren. Het **fo:simple-page-master** element specifieert de geometrie van de pagina. De pagina kan opgedeeld worden in vijf verschillende regio's: **region-body**, **region-before**, **region-after**, **region-start**, en **region-end**. Onderstaande figuur toont de schikking van deze onderdelen voor een pagina waarbij de **writing-mode** eigenschap van **fo:simple-page-master lr-tb** is. Merk op dat het **region-body** element eigenlijk de **region-before**, **region-after**, **region-start**, en **region-end** elementen bevat.



Figuur 7.5. Opdeling van een pagina in regio's

In het Nederlands en de meeste andere westerse talen komt **region-start** overeen met de linkerkant van de pagina en **region-end** met de rechterkant van de pagina. Voor talen zoals

het Arabisch die van rechts naar links geschreven worden is dit echter andersom. In bijna alle talen komt de **region-before** overeen met de hoofding en de **region-after** met de footer. Ook dit kan echter omgekeerd worden voor een taal die van onder naar boven geschreven zou worden.

7.6.2.1. Regio kindelementen

Om de grootte van de vijf verschillende regio's en de afstanden tussen de regio's onderling vast te leggen, moet je de volgende kindelementen toevoegen aan het **fo:simple-page-master** element:

- **fo:region-before**
- **fo:region-after**
- **fo:region-body**
- **fo:region-start**
- **fo:region-end**

De **fo:region-before** en **fo:region-after** elementen hebben allebei een **extent** attribuut dat de hoogte van deze regio's aangeeft. De breedte van deze regio's komt overeen met de breedte van de pagina min de linker- en de rechterkantlijn. De **fo:region-start** en **fo:region-end** elementen hebben ook een **extent** attribuut, maar in hun geval specificeert dit attribuut de breedte van deze regio's. Hun hoogte wordt gemeten van de onderkant van **fo:region-before** tot de bovenkant van **fo:region-after**. We willen er nogmaals op wijzen dat dit natuurlijk alleen geldt voor westerse talen.

Het **fo:region-body** element heeft geen **extent** attribuut. De afmeting van het lichaam is simpelweg alles wat zich binnen de kantlijnen van de pagina bevindt. Doordat de **region-body** de andere regio's overlapt, is het mogelijk dat wanneer je inhoud zet in al deze regio's, deze boven andere inhoud terecht zal komen. Om dit te vermijden moet je de linkerkantlijn van het lichaam breder maken dan het **extent** attribuut van **region-start**, de bovenste kantlijn van het lichaam moet groter zijn dan het **extent** attribuut van **region-before** aangeeft, enzovoort.

Elk van de vijf regio's van **fo:simple-page-master** kan opgevuld worden met inhoud van een **fo:flow** of een **fo:static-content** element tijdens de verwerking van het document. Het **fo:flow** en het **fo:static-content** element bevatten echter zelf geen inhoud. De elementen geven alleen maar de afmetingen weer van de container die de formatter zal bouwen om de inhoud in te plaatsen.

In het volgende voorbeeldje wordt er met behulp van het **fo:simple-page-master** element een pagina gecreëerd met **region-before** en **region-after** die allebei één centimeter bedragen. Door aan het **margin-top** en het **margin-bottom** attribuut van het **fo:region-body** element de juiste waardes mee te geven, zorgen we ervoor dat de **region-body** niet overlapt met de **region-before** en de **region-after**. Aangezien er geen **region-start** en **region-end** elementen zijn, zal de **region-body** zich over de breedte van de pagina uitstrekken (de kantlijnen niet meegerekend).

Voorbeeld 7.4. Het gebruik van region-before en region-after

```
<fo:simple-page-master master-name="illustratie">
   <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
   <fo:region-before extent="1cm"/>
   <fo:region-after extent="1cm"/>
</fo:simple-page-master>
```

Je kan je afvragen waarom de twee **margin** eigenschappen in bovenstaand voorbeeld er niet als volgt uitzien: **margin-before** en **margin-after**, aangezien de ontwerpers van XSL-FO zoveel moeite gedaan hebben om de verschillende regio's en aanverwanten relatief ten opzichte van elkaar te definiëren. De reden die voor deze beslissing opgegeven wordt, is compatibiliteit met CSS. Er is wel een alternatief voor handen voor deze twee eigenschappen, met name de equivalente (als de writing-mode op **lr-tb** staat) eigenschappen: **space-before** en **space-after**. Wij vinden het jammer dat er niet resoluut voor de (ons inziens) betere oplossing **space-before** en **space-after** gekozen is, in plaats van de halfslachtige situatie die nu bestaat in stand te houden. [Sectie 7.15.6.4.]

7.6.2.2. Eigenschappen

Het **fo:simple-page-master** element heeft drie hoofdattributen:

- het **master-name** attribuut: de naam waarmee **fo:page-sequence** elementen naar deze **fo:simple-page-master** zullen verwijzen
- het **page-height** attribuut: de hoogte van de pagina
- het **page-width** attribuut: de breedte van de pagina

Als er geen waardes voor **page-height** en **page-width** zijn opgegeven, dan kiest de formatter een aanvaardbare default voor beide attributen gebaseerd op het gebruikte medium.

Andere, minder belangrijke attributen zijn:

- de **margin-top**, **margin-bottom**, **margin-left** en **margin-right** attributen of kortweg het **margin** attribuut [Sectie 7.15.6.4.]
- het reeds meermaals ter sprake gekomen **writing-mode** attribuut [Sectie 7.3.3., Figuur 7.3.] , dat bepaalt in welke richting de tekst stroomt over de pagina, bijvoorbeeld van links naar rechts, van rechts naar links of van boven naar onder
- het **reference-orientation** attribuut dat specificeert of en hoeveel inhoud geroteerd moet worden; er kan alleen geroteerd worden over een hoek die een veelvoud van 90 graden is

In onderstaand voorbeeldje illustreren we bovenstaande attributen aan de hand van het **fo:simple-page-master** element [Sectie 7.6.2.] dat wij gebruiken voor het genereren van de inhoudstafel van de PDF-versie van onze thesistekst.

Voorbeeld 7.5. De attributen van fo:simple-page-master

```
<fo:simple-page-master master-name="toc" page-height="29.7cm"
page-width="21cm" margin-top="2cm" margin-bottom="2cm"
margin-left="2.5cm" margin-right="2.5cm">
   <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
   <fo:region-before extent="1cm"/>
   <fo:region-after extent="1cm"/>
</fo:simple-page-master>
```

7.6.3. Het fo:page-sequence element

Naast het **fo:layout-master-set** element bevat ieder XSL-FO document één of meerdere **fo:page-sequence** elementen. Elke pagina in deze sequentie heeft een geassocieerde page-master waarin gedefinieerd staat hoe de pagina eruit zal zien. De page-master die gebruikt wordt, wordt bepaald door het **master-reference** attribuut van het **fo:page-sequence**

element. De waarde van dit attribuut moet overeenkomen met de naam van een page-master in het **fo:layout-master-set** element.

In de voorbeelden die we tot nu toe behandeld hebben, was er altijd maar één page-master aanwezig. Het is echter niet ongebruikelijk om meerdere page-masters te hebben. In dat geval kunnen de page-masters gegroepeerd worden als onderdeel van een **fo:page-sequence-master** element. Wij komen hier nog op terug in de volgende sectie van deze tekst. [Sectie 7.6.4.] Je kan bijvoorbeeld een aparte page-master hebben voor de eerste pagina van elk hoofdstuk, een andere page-master voor al de volgende linkerpagina's en eentje voor al de rechterpagina's. Of je kan een **fo:simple-page-master** element [Sectie 7.6.2.] hebben voor een inhoudstafel, een tweede voor de gewone tekst en een derde voor de index, zoals dat het geval is bij de stylesheets die wij geschreven hebben voor de omzetting van onze thesistekst naar PDF. [Hoofdstuk 8.] In het laatste geval gebruik je voor zowel de inhoudstafel, de gewone tekst als de index een afzonderlijk **fo:page-sequence** element.

Elk **fo:page-sequence** element kan drie kindelementen bevatten. Deze kindelementen moeten voorkomen in de volgorde waarin ze hieronder beschreven worden. Een andere volgorde is niet toegestaan.

- Een optioneel **fo:title** element, dat in-regel inhoud bevat die gebruikt kan worden als de titel van het document. Deze inhoud wordt gewoonlijk geplaatst in de titelbalk van het browser venster zoals dit het geval is bij het <TITLE> element in HTML.
- Nul of meer **fo:static-content** elementen die de tekst bevatten die op elke pagina van het document herhaald wordt.
- Eén **fo:flow** element die de eigenlijke gegevens bevat die op de achtereenvolgende pagina's terecht zullen komen.

Het grootste verschil tussen een **fo:flow** en een **fo:static-content** element is dat tekst van de stroom uiteindelijk maar op één pagina van het document terecht zal komen, terwijl de statische inhoud wel op meerdere pagina's zal staan. Als je bijvoorbeeld kijkt naar de woorden die in deze thesistekst staan, dan zijn die stroominhoud: ze staan slechts op één pagina van onze thesis. In de PDF-versie van onze tekst staat er echter op elk blad een paginanummer dat voorafgegaan wordt door de tekst "p.". Zowel deze tekst als het paginanummer worden op elk blad herhaald.

7.6.3.1. Het fo:flow element

Het **fo:flow** element bevat de elementen in de volgorde waarin ze op de pagina geplaatst moeten worden. Als een pagina helemaal opgevuld is met elementen uit de stroom, dan wordt voor de nog resterende elementen in de stroom een nieuwe pagina gecreëerd aan de hand van de volgende page-master in het **fo:page-sequence-master** element. Als de page-master een **fo:simple-page-master** is dan wordt dezelfde pagina telkens opnieuw gegenereerd, zo vaak als nodig is om al de inhoud te bevatten.

De inhoud van het **fo:flow** element is samengesteld uit een opeenvolging van **fo:block**, **fo:block-container**, **fo:table-and-caption**, **fo:table** en **fo:list-block** elementen. Wij beperken ons in deze sectie tot het elementaire **fo:block** element. Je kan dit element vergelijken met het <DIV> element in HTML. We komen later nog terug op de andere blokniveau elementen die een stroom kan bevatten.

Het **fo:flow** element heeft een **flow-name** attribuut. Dit attribuut specificeert in welke van de vijf regio's van de pagina de inhoud van deze stroom zal geplaatst worden. De vijf toegestane waardes voor dit attribuut zijn:

- **xsl-region-body**
- **xsl-region-before**
- **xsl-region-after**
- **xsl-region-start**
- **xsl-region-end**

De stroom voor een hoofding zal bijvoorbeeld als stroomnaam de waarde **xsl-region-before** hebben. Een stroom voor het lichaam heeft de stroomnaam **xsl-region-body**. Een **fo:page-sequence** element kan maximaal één **fo:flow** element bevatten, overeenkomend met één van de vijf regio's van de pagina. In de regio waar de stroom zich bevindt mag zich geen statische inhoud bevinden.

Als voorbeeld geven we (een deel van) de stylesheet die wij gebruiken voor het opbouwen van de inhoudstafel voor de PDF-versie van onze thesistekst. De **fo:static-content** elementen hebben we weggelaten uit de stylesheet. We komen later nog terug op de exacte betekenis van de attributen van het **fo:block** element.

Voorbeeld 7.6. Gebruik van het fo:flow vormgevingsobject

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <xsl:output indent="yes"/>
    <xsl:template match="THESIS">
       <fo:root xsmlns:fo="http://www.w3.org/1999/XSL/Format">
          <fo:layout-master-set>
             <fo:simple-page-master master-name="toc" page-height="29.7"
             page-width="21cm" margin-top="2cm" margin-bottom="2cm"
             margin-left="2.5cm" margin-right="2.5cm">
                <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
                <fo:region-before extent="1cm"/>
                <fo:region-after extent="1cm"/>
             </fo:simple-page-master>
          </fo:layout-master-set>
          <fo:page-sequence master-reference="toc">
             <fo:flow flow-name="xsl-region-body">
                <fo:block break-before="page">
                   <xsl:apply-templates select="CHAPTER" mode="TOC"/>
                </fo:block>
                <fo:block break-before="page">
                   <xsl:apply-templates select="APPENDIX" mode="TOC"/>
                </fo:block>
             </fo:flow>
          </fo:page-sequence>
       </fo:root>
    </xsl:template>
 </xsl:stylesheet>
```

7.6.3.2. Het fo:static-content element

Het **fo:static-content** element bevat informatie die op elke pagina geplaatst moet worden. Dit element wordt bijvoorbeeld gebruikt om de titel van een boek op elke pagina van het boek te plaatsen. Statische inhoud kan aangepast worden, afhankelijk van de page-master die gebruikt wordt. Het is bijvoorbeeld mogelijk dat de titel van het hoofdstuk op de pagina's aan de linkerkant komt te staan en de titel van het boek op de pagina's aan de rechterkant. Het **fo:static-content** element kan ook gebruikt worden voor dingen zoals paginanummers, die op elk blad opnieuw berekend moeten worden. In dit geval is dus niet de tekst statisch, maar de

berekening die de tekst produceert. Het is niet verplicht **fo:static-content** elementen te gebruiken, maar als je ze gebruikt, moeten ze geplaatst worden vóór de **fo:flow** kindelementen van **fo:page-sequence**.

Het **fo:static-content** element heeft dezelfde attributen en inhoud als het **fo:flow** element. Een **fo:static-content** element kan in tegenstelling tot een **fo:flow** element zijn inhoud niet spreiden over verschillende pagina's indien dit nodig zou zijn. Daarom heeft dit element in het algemeen minder inhoud dat het stroomelement.

Een **fo:page-sequence** element kan vijf **fo:static-content** elementen bevatten, overeenkomend met de vijf regio's van de pagina.

In onderstaand voorbeeld hebben we de stylesheet voor de inhoudstafel van hierboven uitgebreid met een **fo:static-content** element dat de tekst "Inhoudstafel" aan elke pagina toevoegt.

Voorbeeld 7.7. Gebruik van het fo:static-content vormgevingsobject

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
     <xsl:output indent="yes"/>
     <xsl:template match="THESIS">
        <fo:root xsmlns:fo="http://www.w3.org/1999/XSL/Format">
           <fo:layout-master-set>
              <fo:simple-page-master master-name="toc" page-height="29.7"
              page-width="21cm" margin-top="2cm" margin-bottom="2cm"
              margin-left="2.5cm" margin-right="2.5cm">
                 <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
                 <fo:region-before extent="1cm"/>
                 <fo:region-after extent="1cm"/>
              </fo:simple-page-master>
           </fo:layout-master-set>
           <fo:page-sequence master-reference="toc">
              <fo:static-content flow-name="xsl-region-before">
                 <fo:block font-size="10pt" font-family="serif"
                 text-align="end"> Inhoudstafel </fo:block>
              </fo:static-content>
              <fo:flow flow-name="xsl-region-body">
                 <fo:block break-before="page">
                    <xsl:apply-templates select="CHAPTER" mode="TOC"/>
                 </fo:block>
                 <fo:block break-before="page">
                    <xsl:apply-templates select="APPENDIX" mode="TOC"/>
                 </fo:block>
              </fo:flow>
           </fo:page-sequence>
        </fo:root>
     </xsl:template>
  </xsl:stylesheet>
```

7.6.3.3. Paginanummering

Het **fo:page-sequence** element heeft acht optionele attributen die paginanummers definiëren voor de opeenvolging van pagina's. Die attributen zijn:

- **country**
- **language**
- **initial-page-number**

- **force-page-count**
- **format**
- **grouping-separator**
- **grouping-size**
- **letter-value**

Het **country** attribuut moet een ISO 3166 landcode [IS03166], [RFC1766] als waarde hebben.

Het **language** attribuut moet een ISO 693 taalcode [ISO693], [RFC1766] als waarde hebben. De waarde **en** wordt zo bijvoorbeeld gebruikt om de taal Engels aan het duiden en **us** om de Verenigde Staten van Amerika aan te duiden. Deze waardes zijn in feite hetzelfde als de waardes die het reeds eerder besproken **xml:lang** attribuut kan aannemen, met dit verschil dat de landcode en taalcode in twee afzonderlijke attributen geplaatst worden in plaats van in één attribuut. [Sectie 2.3.8.3.]

Het **initial-page-number** attribuut geeft het nummer van de eerste pagina in de opeenvolging van pagina's. De meest aannemelijke waarde voor dit attribuut is 1. Het kan ook een groter nummer zijn als de vorige pagina's tot een andere **fo:page-sequence** object behoren of eventueel zelfs tot een ander document. De waarde van het **initial-page-number** attribuut kan één van de volgende drie sleutelwoorden aannemen:

- **auto** is gelijk aan 1, tenzij pagina's van een voorgaande **fo:page-sequence** deze waarde verhoogd hebben. Dit is de defaultwaarde.
- **auto-odd** heeft dezelfde waarde als **auto**, maar voegt 1 toe als de beginwaarde een even nummer is. Dit wil zeggen dat er altijd begonnen zal worden met een oneven pagina.
- **auto-even** heeft dezelfde waarde als **auto**, maar voegt 1 toe als de beginwaarde een oneven nummer is. Dit wil zeggen dat er altijd begonnen zal worden met een even pagina.

Het **force-page-count** attribuut wordt gebruikt om ervoor te zorgen dat het document een even of oneven aantal pagina's heeft of om ervoor te zorgen dat het document eindigt op een even of oneven pagina. Dit is soms noodzakelijk voor gedrukte boeken. De waardes van het **force-page-count** attribuut kunnen één van de volgende zes sleutelwoorden zijn:

- **auto** zorgt ervoor dat de laatste pagina een oneven pagina is als het **initial-page-number** van de volgende **fo:page-sequence** even is. Als het **initial-page-number** van de volgende **fo:page-sequence** oneven is, zorgt het ervoor de dat laatste papina een even pagina is. Als er geen volgende **fo:page-sequence** is of als de volgende **fo:page-sequence** geen **initial-page-number** specificeert, dan worden er geen beperkingen opgelegd aan de laatste pagina.
- **even** zorgt ervoor dat het totale aantal pagina's even is. Er wordt een extra blanco pagina toegevoegd indien nodig om dit te bewerkstelligen.
- **odd** zorgt ervoor dat het totale aantal pagina's oneven is. Er wordt een extra blanco pagina toegevoegd indien nodig om dit te bewerkstelligen.
- **end-on-even** zorgt ervoor dat de laatste pagina een even paginanummer heeft. Er wordt een extra blanco pagina toegevoegd indien nodig om dit te bewerkstelligen.
- **end-on-odd** zorgt ervoor dat de laatste pagina een oneven paginanummer heeft. Er wordt een extra blanco pagina toegevoegd indien nodig om dit te bewerkstelligen.
- **no-force** stelt geen eisen aan het even of oneven zijn van het aantal pagina's.

De overblijvende vier attributen hebben exact dezelfde syntax en betekenis als wanneer zij gebruikt worden als attributen van het **xsl:number** XSLT element. Voor de bespreking van

deze attributen verwijzen we naar [Sectie 5.3.1.9.] .

Om op elke pagina van paginanummer te zzetten, kan gebruik gemaakt worden van het **fo:page-number** vormgevingsobject. Dit is een leeg in-regel element dat het nummer van de huidige pagina invoegt. De formatter is verantwoordelijk voor het bepalen van dat nummer. Dit element kan gebruik maken van de gewone vormgevingsattributen die je normaal bij een in-regel element aantreft. Deze attributen zijn bijvoorbeeld: **font-family** en **text-decoration**.

In onderstaand voorbeeld wordt de stylesheet van hierboven hernomen en uitgebreid. Dit voorbeeld illustreert het gebruik van **fo:static-content** en **fo:page-number** om het paginanummer in de footer van elke pagina te plaatsen:

Voorbeeld 7.8. Gebruik van het fo:page-number voor het toevoegen van een paginanummer

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
     <xsl:output indent="yes"/>
     <xsl:template match="THESIS">
        <fo:root xsmlns:fo="http://www.w3.org/1999/XSL/Format">
           <fo:layout-master-set>
              <fo:simple-page-master master-name="toc" page-height="29.7"
              page-width="21cm" margin-top="2cm" margin-bottom="2cm"
              margin-left="2.5cm" margin-right="2.5cm">
                 <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
                 <fo:region-before extent="1cm"/>
                 <fo:region-after extent="1cm"/>
              </fo:simple-page-master>
           </fo:layout-master-set>
           <fo:page-sequence master-reference="toc" initial-page-number="1"
           format="i">
              <fo:static-content flow-name="xsl-region-before">
                 <fo:block font-size="10pt" font-family="serif"
                 text-align="end"> Inhoudstafel </fo:block>
              </fo:static-content>
              <fo:static-content flow-name="xsl-region-after">
                 <fo:block font-size="10pt" font-family="Courier"
                 text-align="end">
                    <fo:page-number/>
                 </fo:block>
              </fo:static-content>
              <fo:flow flow-name="xsl-region-body">
                 <fo:block break-before="page">
                    <xsl:apply-templates select="CHAPTER" mode="TOC"/>
                 </fo:block>
                 <fo:block break-before="page">
                    <xsl:apply-templates select="APPENDIX" mode="TOC"/>
                 </fo:block>
              </fo:flow>
           </fo:page-sequence>
        </fo:root>
     </xsl:template>
  </xsl:stylesheet>
```

7.6.4. Het fo:page-sequence-master element

Elke pagina die door de formatter gecreëerd wordt, is geassocieerd met een page-master van de **fo:layout-master-set** waarin gedefinieerd wordt hoe die pagina eruit zal zien. Welke page-master hiervoor gebruikt wordt, wordt bepaald door het **master-reference** attribuut van het **fo:page-sequence** element [Sectie 7.6.3.] . Tot nu toe hebben we alleen met één enkel

fo:simple-master-page element [Sectie 7.6.2.] gewerkt, maar zoals we reeds eerder vermeld hebben, is het niet ongewoon om meer dan één page-master te hebben. Het voorbeeld waarbij er een page-master is voor de eerste pagina van elk hoofdstuk, een andere voor al de volgende linkerpagina's en nog een andere voor al de volgende rechterpagina's hebben we ook al aangehaald. In dat geval kunnen de verschillende page-masters gegroepeerd worden als onderdeel van een **fo:page-sequence-master** element.

Het **fo:page-sequence-master** element is een kind van het **fo:layout-master-set** element. Het **fo:page-sequence-master** element geeft de volgorde aan waarin bepaalde page-masters gebruikt worden om pagina's te genereren aan de hand van de volgende drie kindelementen:

- **fo:single-page-master-reference**
- **fo:repeatable-page-master-reference**
- **fo:repeatable-page-master-alternatives**

7.6.4.1. Het fo:single-page-master-reference element

Het **fo:single-page-master-reference** element heeft een **master-reference** attribuut dat aangeeft welke page-master gebruikt moet worden. Het **fo:single-page-master-reference** element zorgt ervoor dat er maar één pagina gegenereerd wordt waarop al de tekstinhoud moet terecht komen.

In het onderstaande voorbeeld bevat het **fo:layout-master-set** element een **fo:page-sequence-master** kindelement met als naam **tableofcontents** dat ervoor zorgt dat alle tekst geplaatst wordt op één enkele pagina die gegenereerd wordt door de page-master met als naam **toc**.

Voorbeeld 7.9. Gebruik van het page-sequence-master element

```
<fo:layout-master-set>
   <fo:simple-page-master master-name="toc" page-height="29.7"
   page-width="21cm" margin-top="2cm" margin-bottom="2cm"
   margin-left="2.5cm" margin-right="2.5cm">
      <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
      <fo:region-before extent="1cm"/>
      <fo:region-after extent="1cm"/>
   </fo:simple-page-master>
   <fo:page-sequence-master master-name="tableofcontents">
      <fo:single-page-master-reference master-reference="toc"/>
   </fo:page-sequence-master>
</fo:layout-master-set>
```

Het **fo:single-page-master** element laat dus slechts de creatie van één pagina toe. In theorie is het verkeerd als er meer inhoud aanwezig is dan er op deze ene pagina past. In de praktijk echter herhalen de meeste formatters in dit geval gewoon de laatste pagina die ze gebruikt hebben totdat ze genoeg pagina's gecreëerd hebben om alle inhoud te bevatten.

We bekijken nu een ander voorbeeld:

Voorbeeld 7.10. Het page-sequence-master element met meerdere fo:single-page-master-reference kindelementen

```
<fo:page-sequence-master master-name="tableofcontents">
   <fo:single-page-master-reference master-reference="toc"/>
   <fo:single-page-master-reference master-reference="anderemaster"/>
```

```
</fo:page-sequence-master>
```

Dit **fo:page-sequence-master** element kan twee pagina's genereren. De eerste pagina die aangemaakt wordt is gebaseerd op de **toc** page-master, de tweede is gebaseerd op de **anderemaster** page-master. Als de eerste pagina vol is, wordt een tweede pagina gecreëerd. Als deze pagina op zijn beurt opgevuld is met de inhoud, kan de formatter een fout signaleren of gewoon nog extra pagina's aanmaken aan de hand van de laatst gebruikte page-master, in dit geval de **anderemaster** page-master.

7.6.4.2. Het fo:repeatable-page-master-reference element

Natuurlijk weet je meestal niet op voorhand hoeveel pagina's je document juist zal bevatten. Het **fo:repeatable-page-master-reference** element zorgt ervoor dat er zoveel pagina's als nodig gecreëerd worden om al de inhoud te kunnen bevatten. Deze pagina's zijn allemaal gebaseerd op dezelfde page-master. Het **master-reference** attribuut geeft aan welke page-master hiervoor gebruikt wordt.

In onderstaand voorbeeld zullen er zoveel pagina's als nodig gecreëerd worden op basis van de **toc** page-master.

Voorbeeld 7.11. Gebruik van het fo:repeatable-page-master-reference element

```
<fo:page-sequence-master master-name="tableofcontents">
  <fo:repeatable-page-master-reference master-reference="toc"/>
</fo:page-sequence-master>
```

Je kan het **maximum-repeats** attribuut van het **fo:repeatable-page-master-reference** gebruiken om het aantal pagina's dAT gecreëerd zAL worden te beperken. In het onderstaande voorbeeld wordt dit geïllustreerd:

Voorbeeld 7.12. Het maximum-repeats attribuut

```
<fo:page-sequence-master master-name="tableofcontents">
  <fo:repeatable-page-master-reference master-reference="toc"
  maximum-repeats="10"/>
</fo:page-sequence-master>
```

Dit attribuut maakt het ook mogelijk om bijvoorbeeld één page-master te gebruiken voor de eerste twee pagina's, een andere voor de volgende vier, enzovoort.

7.6.4.3. Het fo:repeatable-page-master-alternatives element

Het **fo:repeatable-page-master-alternatives** element specificeert verschillende page-masters voor de eerste pagina, de even pagina's, de oneven pagina's, de blanco pagina's, de laatste even pagina en de laatste oneven pagina. Dit element is meer ontworpen voor een hoofdstuk van een gedrukt boek, waar de eerste en laatste pagina's, de even en de oneven pagina's traditioneel verschillende kantlijnen, hoofdingen en footers hebben.

Omdat een **fo:repeatable-page-master-alternatives** element naar meer dan één page-master moet kunnen verwijzen, kan het geen gebruik maken van het **master-reference** attribuut. In plaats daarvan heeft het **fo:repeatable-page-master-alternatives** element

fo:conditional-page-master-reference kindelementen. Deze kindelementen hebben een **master-reference** attribuut dat de page-master aanduidt die gebruikt zal worden als aan de opgegeven voorwaarde voldaan is. De voorwaarden worden bepaald door drie attributen:

- Het **page-position** attribuut heeft als waarde **first**, **last**, **rest** of **any**. Dit duidt aan dat de page-master waarnaar verwezen wordt, alleen toegepast mag worden op respectievelijk de eerste, de laatste, eender welke pagina behalve de eerste of op eender welke pagina.
- Het **odd-or-even** attribuut kan als waarde **odd**, **even** of **any** hebben, wat willen zeggen dat de meegegeven page-master alleen toegepast mag worden op respectievelijk de oneven, de even of op alle pagina's.
- Het **blank-or-not-blank** attribuut heeft als waarde **blank**, **not-blank** of **any** om aan te duiden dat de meegegeven page-master alleen toegepast mag worden op respectievelijk blanco pagina's, pagina's die inhoud bevatten of op alle pagina's.

In onderstaand voorbeeld wordt voor de eerste pagina de page-master **first_page** toegepast en op al de volgende pagina's de page-master **page**.

Voorbeeld 7.13. Gebruik van het fo:repeatable-page-master-alternatives element

```
<fo:page-sequence-master master-name="contents">
  <fo:repeatable-page-master-alternatives>
    <fo:conditional-page-master-reference page-position="first"
    master-reference="first_page"/>
    <fo:conditional-page-master-reference page-position="rest"
    master-reference="page"/>
  </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
```

Als de inhoud groter is dan wat door de eerste pagina bevat kan worden, wordt de rest op een tweede pagina geplaatst. Als de tweede pagina vol is, wordt een derde pagina gecreëerd. Er worden zoveel pagina's geconstrueerd als nodig zijn om alle inhoud te bevatten.

7.7. De inhoud

De inhoud (in tegenstelling tot markup) van een XSL-FO document is meestal tekst. Niet-XML inhoud zoals GIF en JPEG afbeeldingen kunnen in het document opgenomen worden op een manier gelijkaardig aan het <IMG> element in HTML. Andere vormen van XML inhoud, zoals MathML (Mathematical Markup Language) [TRMATHML] en SVG (Scaled Vector Graphics) [SVG] kunnen rechtstreeks opgenomen worden in het XSL-FO document.

De inhoud van een XSL-FO document kan in verschillende elementen zitten, zoals:

- vormgevingsobjecten van blokniveau
- in-regel vormgevingsobjecten
- tabelvormgevingsobjecten
- lijstvormgevingsobjecten
- uit-regel vormgevingsobjecten

Al deze verschillende soorten elementen zijn nakomelingen van een **fo:flow** of een **fo:static-content** element. Deze elementen worden **nooit** rechtstreeks onder een page-master of een page-sequence geplaatst. Wij wijzen erop dat de opdeling die hier gemaakt is, niet disjunct is. Zo is zijn de tabelvormgevingsobjecten **fo:table** en **fo:table-and-caption** en het lijstvormgevingsobject **fo:list-block** bijvoorbeeld ook een vormgevingsobject van blokniveau.

De reden dat wij hier de tabelvormgevingsobjecten en lijstvormgevingsobjecten apart bespreken, is dat de kindelementen van **fo:table-and-caption**, **fo:table** en **fo:list-block** niet éénduidig bij de vormgevingsobjecten van blokniveau, in-regel vormgevingsobjecten of uit-regel vormgevingsobjecten ondergebracht kunnen worden.

7.7.1. Vormgevingsobjecten van blokniveau

Een vormgevingsobject van blokniveau wordt getekend als een rechthoekig gebied. Dit gebied wordt door een line break en mogelijk nog extra witruimte van al de voorafgaande en volgende inhoud gescheiden. Blokken kunnen andere blokken bevatten. In dat geval worden de omvatte blokken ook gescheiden van de omvattende blok door een line break en eventueel witruimte.

De vormgevingsobjecten van blokniveau zijn:

- **fo:block**
- **fo:block-container**
- **fo:list-block**
- **fo:table**
- **fo:table-and-caption**

Het **fo:block** element in XSL-FO is equivalent aan **display: block** in CSS [TRCSS] of <DIV> in HTML [TRHTML]. Blokken kunnen opgenomen worden in **fo:flow** elementen [Sectie 7.6.3.1.] , **fo:static-content** elementen [Sectie 7.6.3.2.] of andere **fo:block** elementen. **fo:block** elementen kunnen op hun beurt zelf andere elementen van blokniveau en in-regel elementen bevatten. Ze kunnen ook gewone pure tekst bevatten. Een voorbeeldje:

Voorbeeld 7.14. Gebruik van het fo:block element

```
<fo:block> Inhoudstafel, pagina
   <fo:page-number/>
</fo:block>
```

Vormgevingsobjecten van blokniveau hebben attributen voor zowel tekstvormgevingseigenschappenals kenmerken van gebieden. De tekstvormgevingseigenschappen worden overgeërfd door elk kindelement van het blokelement, tenzij de eigenschap overschreven wordt.

7.7.2. In-regel vormgevingsobjecten

Een in-regel vormgevingsobject wordt ook getekend als een rechthoeking gebied dat tekst of ander in-regel gebieden kan bevatten. In-regel gebieden worden gewoonlijk gerangschikt in lijnen die voor westerse schrijfsystemen van links naar rechts lopen (de **inline-progression-direction**). Als een lijn vol is, wordt een nieuwe lijn begonnen die onder de vorige lijn terecht komt. De richting waarin nieuwe lijnen aangroeien is de **block-progression-direction**. Zoals we reeds eerder besproken hebben, hangt de manier waarop de in-regel elementen geplaatst worden af van de **writing-mode**. [Sectie 7.3.3., Figuur 7.3.]

De volgende vormgevingsobjecten zijn in-regel vormgevingsobjecten:

- **fo:bidi-override**
- **fo:character**

- **fo:external-graphic**
- **fo:initial-property-set**
- **fo:inline**
- **fo:inline-container**
- **fo:instream-foreign-object**
- **fo:leader**
- **fo:page-number**
- **fo:page-number-citation**

7.7.3. Tabelvormgevingsobjecten

De tabelvormgevingsobjecten zijn de XSL-FO equivalentEN van de CSS tabeleigenschappen [TRCSS]. Tabellen in XSL-FO werken op een iets natuurlijker manier dan in CSS. We komen later nog terug op het gebruik van tabellen. [Sectie 7.11.] Een individuele tabel is voor het merendeel een blokvormgevingsobject. De aparte onderdelen van een tabel vallen eigenlijk noch onder de in-regel elementen, noch onder de vormgevingsobjecten van blokniveau te klasseren. Het is wel mogelijk een volledige tabel te veranderen in een in-regel object door de tabel op te nemen in een **fo:inline-container** element.

Hieronder geven we de negen tabelvormgevingsobjecten:

- **fo:table**
- **fo:table-and-caption**
- **fo:table-body**
- **fo:table-caption**
- **fo:table-cell**
- **fo:table-column**
- **fo:table-footer**
- **fo:table-header**
- **fo:table-row**

Het rootelement van een tabel is ofwel een **fo:table** element ofwel een **fo:table-and-caption** element dat zowel een **fo:table** element als een **fo:table-caption** element bevat. Het **fo:table** element bevat een **fo:table-header** element, een **fo:table-body** element en een **fo:table-footer** element. Het **fo:table-body** element bevat **fo:table-row** elementen die opgedeeld zijn in **fo:table-cell** elementen. [Sectie 7.11.]

7.7.4. Lijstvormgevingsobjecten

Er zijn vier lijstvormgevingsobjecten:

- **fo:list-block**
- **fo:list-item**
- **fo:list-item-body**
- **fo:list-item-label**

Het rootelement van een lijst is een **fo:list-block** element dat verschillende **fo:list-item** kindelementen bevat. Een **fo:list-item** element bevat op zijn beurt een **fo:list-item-body** en een **fo:list-item-label** element. [Sectie 7.10.]

7.7.5. Uit-regel vormgevingsobjecten

Er zijn drie uit-regel vormgevingsobjecten:

- **fo:float**
- **fo:foot-note**
- **fo:foot-note-body**

Uit-regel vormgevingsobjecten "lenen" ruimte van bestaande in-regel of blokobjecten. Ze verschijnen op de uiteindelijke resulterende pagina niet noodzakelijk tussen dezelfde elementen als waartussen ze stonden in de invoerboom van vormgevingsobjecten.

7.8. Lijnen en leaders

Een lijn (rule) is een horizontale lijn van blokniveau die aan de tekst wordt toegevoegd. Het <HR> element in HTML produceert zo een horizontale lijn. Een leader is een lijn die loopt van de achterzijde van linksgealigneerde tekst tot de voorzijde van rechtsgealigneerde tekst; deze tekst moet zich op dezelfde regel bevinden. Een leader bestaat meestal uit een opeenvolging van punten, alhoewel het ook mogelijk is andere tekens te gebruiken. Leaders worden vaak gebruikt in inhoudstafels. De inhoudstafel van onze eigen PDF tekst [Sectie 8.1.] maakt ook gebruik van leaders om de ruimte tussen de titel van een sectie en het paginanummer op te vullen.

In XSL-FO worden zowel lijnen als leaders geproduceerd door het **fo:leader** element. Het **fo:leader** element is een inline element dat een leader voorstelt. Dit element kan gemakkelijk gebruikt worden om een lijn te creëren door het in een **fo:block** element te plaatsen.

Er zijn zes attributen die het uitzicht van een leader kunnen bepalen:

- Het **leader-alignment** attribuut kan als waarde **reference-area** of **page** hebben om aan te duiden dat de beginzijde van de leader gealigneerd moet worden met de beginzijde van het opgegeven item (referentiegebied of pagina). Dit attribuut kan ook als waarde **none** of **inherit** hebben.
- Het **leader-length** attribuut bepaalt de lengte van de leader, bvb **4in** of **6cm**
- Het **leader-pattern** attribuut kan de waardes **space**, **rule**, **dots**, **use-content** of **inherit** hebben. De **use-content** waarde betekent dat de leadertekens die gebruikt worden diegene zijn die in de inhoud van het **fo:leader** element staan. De betekenis van de andere waardes spreekt voor zich.
- Het **leader-pattern-width** attribuut geeft aan hoe breed het gebruikte patroon van de leader moet zijn. Dit attribuut kan als waarde een lengte met bijhorende lengtemaat hebben, bijvoorbeeld 2mm, of het sleutelwoord **use-font-metrics**. De waarde **use-font-metrics** duidt aan dat de breedte van het weerkerende patroon in de leader gewoon zijn normale breedte moet hebben. Indien nodig wordt er witruimte aan het patroon toegevoegd om het uit te rekken tot de gespecificeerde breedte.
- Het **rule-style** attribuut heeft dezelfde waardes als de CSS **border-style** eigenschap. De waardes zijn **none**, **dotted**, **dashed**, **solid**, **double**, **groove**, **ridge** en **inherit**.
- Het **rule-thickness** attribuut geeft de dikte weer van de horizontale lijn. De defaultwaarde is 1pt.

Er zijn nog een aantal andere eigenschappen die ook op leaders van toepassing zijn. Zo kan je bijvoorbeeld de **font-family** eigenschap gebruiken om het lettertype van het patroon van de leader te veranderen. Of de **color** eigenschap om de kleur van de leader te veranderen.

Een voorbeeldje:

Voorbeeld 7.15. Gebruik van het fo:leader element

```
<fo:block>
   <fo:leader leader-length="6cm" leader-pattern="rule"
   rule-thickness="2pt" color="red"/>
</fo:block>
```

7.9. Grafische objecten

XSL-FO levert twee elementen om afbeeldingen in een vormgegeven document op te nemen. Het **fo:external-graphic** element voegt een niet-XML grafisch object zoals een JPEG afbeelding toe. Het **fo:instream-foreign-object** element voegt een XML document toe dat geen XSL-FO document is, bijvoorbeeld een SVG afbeelding [SVG] of een MathML vergelijking [TRMATHML].

7.9.1. Het fo:external-graphic element

Het **fo:external-graphic** element levert een equivalent voor het <IMG> element in HTML. Het haalt een afbeelding van een (waarschijnlijk) niet-XML formaat op van een bepaalde URI. Het **fo:external-graphic** element is altijd een leeg element zonder kinderen. Het **src** attribuut bevat een URI die de plaats van de op te nemen afbeelding aangeeft.

Een **fo:external-graphic** element kan er dus als volgt uitzien: `<fo:external-graphic src="image.gif"/>` . Het is natuurlijk ook mogelijk om in plaats van relatieve, absolute URL's te gebruiken. Net zoals bij Web browsers en HTML, is er geen garantie dat een bepaald grafisch formaat door de formatter herkend en ondersteund wordt.

fo:external-graphic is een in-regel element. Het is mogelijk om de afbeelding van blokniveau te maken door het object op te nemen in een **fo:block** element, zoals hieronder geïllustreerd wordt.

Voorbeeld 7.16.

```
<fo:block>
   <fo:external-graphic src="image.gif"/>
</fo:block>
```

We hernemen ons voorbeeld, maar voegen nu aan de statische inhoud die op elke pagina in de hoofding terecht komt, de afbeelding toe, waarvan de URI in het **LOCATION** element zit.

Voorbeeld 7.17. Gebruik van het fo:external-graphic element

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <xsl:output indent="yes"/>
   <xsl:template match="THESIS">
     <fo:root xsmlns:fo="http://www.w3.org/1999/XSL/Format">
       <fo:layout-master-set>
         <fo:simple-page-master master-name="toc" page-height="29.7"
         page-width="21cm" margin-top="2cm" margin-bottom="2cm"
```

```
            margin-left="2.5cm" margin-right="2.5cm">
               <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
               <fo:region-before extent="1cm"/>
               <fo:region-after extent="1cm"/>
            </fo:simple-page-master>
         </fo:layout-master-set>
         <fo:page-sequence master-reference="toc" initial-page-number="1"
         format="i">
            <fo:static-content flow-name="xsl-region-before">
               <fo:block>
                  <fo:external-graphic>
                     <xsl:attribute name="src">
                        <xsl:value-of select="LOCATION"/>
                     </xsl:attribute>
                  </fo:external-graphic>
               Inhoudstafel </fo:block>
            </fo:static-content>
            <fo:static-content flow-name="xsl-region-after">
               <fo:block font-size="10pt" font-family="Courier"
               text-align="end">
                  <fo:page-number/>
               </fo:block>
            </fo:static-content>
            <fo:flow flow-name="xsl-region-body">
               <fo:block break-before="page">
                  <xsl:apply-templates select="CHAPTER" mode="TOC"/>
               </fo:block>
               <fo:block break-before="page">
                  <xsl:apply-templates select="APPENDIX" mode="TOC"/>
               </fo:block>
            </fo:flow>
         </fo:page-sequence>
      </fo:root>
   </xsl:template>
</xsl:stylesheet>
```

7.9.2. Het fo:instream-foreign-object element

Het **fo:instream-foreign-object** element voegt een grafisch object toe dat beschreven is in XML en dat rechtstreeks opgenomen wordt in het XSL-FO document. Een **fo:instream-foreign-object** element kan bijvoorbeeld een SVG afbeelding bevatten. De formatter zal deze afbeelding renderen in het resulterende document.

We hernemen het voorgaande voorbeeld maar voegen nu een SVG (Scaled Vector Graphics) afbeelding toe aan de hoofding. De SVG afbeelding die hier gebruikt wordt is een donkerblauwe driehoek. Hoe SVG afbeeldingen juist opgebouwd worden, is verder voor dit hoofdstuk van geen belang.

Voorbeeld 7.18. Gebruik van het fo:instream-foreign-object element

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <xsl:output indent="yes"/>
   <xsl:template match="THESIS">
      <fo:root xsmlns:fo="http://www.w3.org/1999/XSL/Format">
         <fo:layout-master-set>
            <fo:simple-page-master master-name="toc" page-height="29.7"
            page-width="21cm" margin-top="2cm" margin-bottom="2cm"
            margin-left="2.5cm" margin-right="2.5cm">
```

```
            <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
            <fo:region-before extent="1cm"/>
            <fo:region-after extent="1cm"/>
         </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-reference="toc" initial-page-number="1"
      format="i">
         <fo:static-content flow-name="xsl-region-before">
            <fo:block>
               <fo:instream-foreign-object>
                  <svg xmlns="http://www.w3.org/2000/svg" width="1.5cm"
                  height="1cm">
                     <polygon style="fill:#000099" points="0,31 18,0
                     36,31"/>
                  </svg>
               </fo:instream-foreign-object>
            Inhoudstafel </fo:block>
         </fo:static-content>
         <fo:static-content flow-name="xsl-region-after">
            <fo:block font-size="10pt" font-family="Courier"
            text-align="end">
               <fo:page-number/>
            </fo:block>
         </fo:static-content>
         <fo:flow flow-name="xsl-region-body">
            <fo:block break-before="page">
               <xsl:apply-templates select="CHAPTER" mode="TOC"/>
            </fo:block>
            <fo:block break-before="page">
               <xsl:apply-templates select="APPENDIX" mode="TOC"/>
            </fo:block>
         </fo:flow>
      </fo:page-sequence>
   </fo:root>
  </xsl:template>
</xsl:stylesheet>
```

Alhoewel lang niet alle formatters alle mogelijke grafische formaten ondersteunen, is dit toch een nuttige techniek. Vooral wanneer je wilt dat je XSLT stylesheets at runtime afbeeldingen genereren. Je zou bijvoorbeeld een XSLT stylesheet kunnen schrijven die mooi vormgegeven jaarlijkse rapporten opstelt, inclusief grafieken en afbeeldingen, en dit gewoon door een gedeelte van het invoerdocument naar XSL-FO te transformeren en een ander deel van het invoerdocument naar SVG.

7.9.3. Grafische eigenschappen

Het **fo:instream-foreign-object** element en het **fo:external-graphic** element hebben een aantal gemeenschappelijke eigenschappen die ontworpen zijn om te scaleren, te positioneren, te aligneren of om op andere wijze de uiterlijke kenmerken van de afbeelding op de pagina aan te passen. Wij behandelen deze eigenschappen in de secties hieronder.

7.9.3.1. Het content-type attribuut

Het **content-type** attribuut specificeert het type van de grafische afbeelding. Je kan aan dit attribuut een MIME media type [MIME] meegeven, zoals **image/jpg** of **image/svg-xml** door deze mediatypes vooraf te laten gaan door **content-type:**. Om bijvoorbeeld aan te geven dat het **fo:external-graphic** element naar een GIF afbeelding verwijst, schrijf je dit:

Voorbeeld 7.19.

```
<fo:external-graphic content-type="image/gif" src="image.gif"/>
```

Je kan het type ook aangeven door middel van een namespace voorvoegsel, door gebruik te maken van een waarde in de vorm **namespace-prefix**. Om bijvoorbeeld aan te duiden dat het **fo:instream-foreign-object** element een SVG afbeelding bevat, schrijf je:

Voorbeeld 7.20.

```
<fo:instream-foreign-object xmlns:svg="http://www.w3.org/2000/svg"
content-type="namespace-prefix:svg"/>
```

Het namespace voorvoegsel moet niet, zoals in bovenstaand voorbeeld, gedeclareerd worden bij het **fo:instream-foreign-object** element. Het moet gewoon ergens in de voorouders van dit element gedeclareerd worden.

7.9.3.2. Grootte

Het **height** en het **width** attribuut specificeren de vertikale en horizontale grootte van de rechthoek die op de pagina gereserveerd wordt om de afbeelding in te plaatsen. De waarde van deze attributen kan op het sleutelwoord **auto** gezet worden, in plaats van op een absolute lengte, om aan te geven dat de grootte van de afbeelding zelf gebruikt moet worden voor de rechthoek.

Het **content-height** en het **content-width** attribuut specificeert de vertikale en de horizontale grootte van de afbeelding zelf. Als één van deze attributen niet hetzelfde is als de meegegeven attributen **height** en **width**, dan moet de afbeelding gescaleerd worden.

7.9.3.3. Scalering

Het **scaling** attribuut kan de waarde **uniform** of **non-uniform** aannemen. Uniforme scalering behoudt de verhoudingen van de oorspronkelijke afbeelding als het gescaleerd wordt. Dit is de defaultwaarde van het attribuut. Niet-uniforme scalering kan de hoogte en de breedte met een verschillende factor scaleren, zodat de afbeelding vervormd wordt.

Je kan het algoritme waarmee de scalering gebeurt zelf kiezen door gebruik te maken van het **scaling-method** attribuut. Dit attribuut kan de waardes **integer-pixels**, **resample-any-method** of **auto**. Gehele scalering behoudt een gehele ratio tussen de originele en de gescaleerde afbeelding. De scaleringsratio is dan bijvoorbeeld 2:1 of 3:1. In de meeste gevallen zijn afbeeldingen die geheel gescaleerd worden kleiner dan afbeeldingen die gescaleerd worden volgens **resample-any-method**. Afbeeldingen die geheel gescaleerd worden hebben wel geen dithering (een techniek om gedigitaliseerde kleuren regelmatiger en meer diffuus te laten verlopen door aanpassing van de kleurpunten met tussenkleuren) nodig. De waarde **auto** laat de formatter kiezen hoe er gescaleerd zal worden.

Je kan grafische objecten ook een grote variëteit aan gemeenschappelijke eigenschappen van in-regel elementen toekennen. De eigenschappen zijn onder andere: kadereigenschappen, achtergrondeigenschappen, kantlijneigenschappen, enzovoort. Omdat grafische objecten niet gesplitst kunnen worden over meerdere pagina's gelden de normale brekingseigenschappen [Sectie 7.15.3.1.] hier niet, maar ze ondersteunen wel het **keep-with-next** en het **keep-with-previous** attribuut.

7.10. Lijsten

Het **fo:list-block** vormgevingsobject beschrijft een lijst op blokniveau. (Er bestaan geen in-regel lijsten.) Een lijst kan op verschillende manieren vormgegeven worden: door indentatie, door nummering of op een andere manier. Elk **fo:list-block** element bevat een aantal **fo:list-item** elementen. Elk **fo:list-item** element bevat op zijn beurt een **fo:list-item-label** en een **fo:list-item-body** element. Het **fo:list-item-label** element bevat het label (een nummer, een bullet, een sterretje,...) van het lijstitem. Het label bevat altijd een element van blokniveau. Het **fo:list-item-body** element bevat elementen van blokniveau die de eigenlijke inhoud van het lijstitem bevatten.

Een voorbeeld van een XSL-FO lijst:

Voorbeeld 7.21.

```
<fo:list-block>
   <fo:list-item>
      <fo:list-item-label>
         <fo:block> * </fo:block>
      </fo:list-item-label>
      <fo:list-item-body>
         <fo:block> Eerste lijstitem </fo:block>
      </fo:list-item-body>
   </fo:list-item>
   <fo:list-item>
      <fo:list-item-label>
         <fo:block> * </fo:block>
      </fo:list-item-label>
      <fo:list-item-body>
         <fo:block> Tweede lijstitem </fo:block>
      </fo:list-item-body>
   </fo:list-item>
</fo:list-block>
```

Het **fo:list-block** element heeft twee speciale attributen die controle uitoefenen op de vormgeving van de lijst:

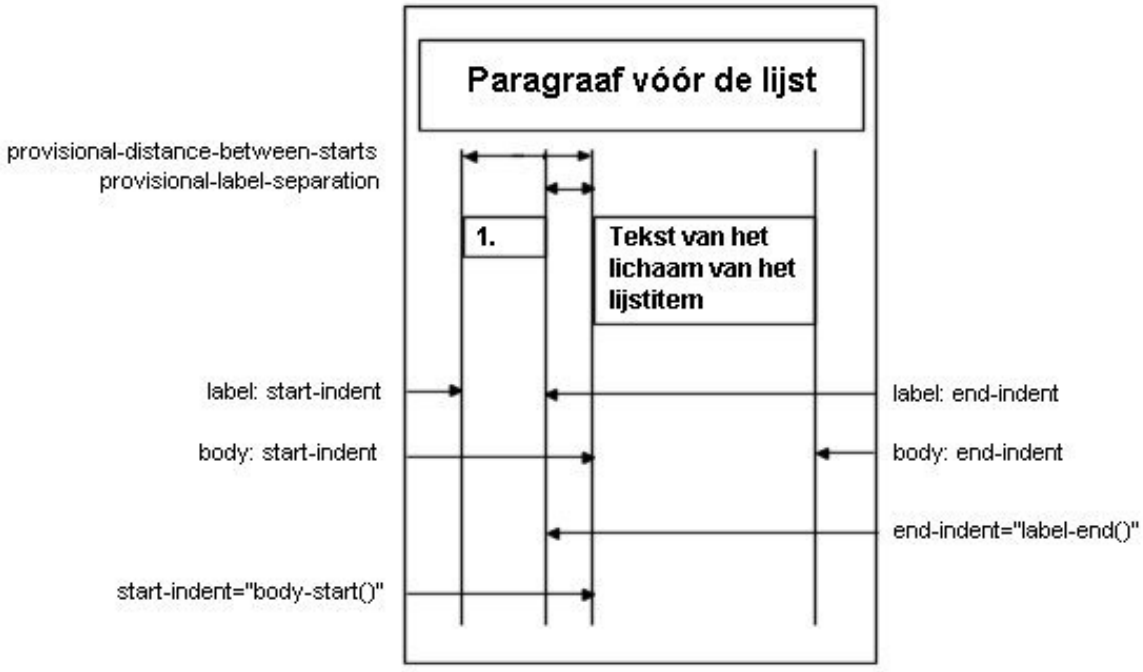
- Het **provisional-label-separation** attribuut geeft de afstand tussen het label en het lichaam van het lijstitem. De waarde van dit attribuut is een lengte met bijhorende lengtemaat.
- Het **provisional-distance-between-starts** attribuut geeft de afstand tussen het beginpunt van het label van het lijstitem en het beginpunt van het lichaam van het lijstitem. Deze afstand wordt uitgedrukt als een lengte met bijhorende lengtemaat.

Het **fo:list-block** element heeft nog een heel aantal andere eigenschappen, zoals eigenschappen voor achtergrond, kantlijnen, breking, opvulling, enzovoort. Het **fo:list-item** element heeft de standaard eigenschappen van elementen van blokniveau, zoals daar zijn: achtergronden, positie, opvulling, kantlijnen en kaders. Het **fo:list-item-label** en het **fo:list-item-body** element hebben alleen de eigenschappen **id** [Sectie 7.15.1.] en **keep-together** [Sectie 7.15.3.1.] . Wij komen later nog op deze eigenschappen terug. De rest van de vormgeving van deze beide elementen wordt bepaald door ofwel hun ouderelementen **fo:list-block** en **fo:list-item**, ofwel door de kindelementen die ze bevatten.

In HTML heeft een lijstitem altijd een zeker niveau van indentatie. In XSL-FO wordt er echter geen indentatie geïmpliceerd door de lijstitems. Als je wilt dat de lijstitems een indentatie krijgen, kan je gebruik maken van de **start-indent** en **end-indent** attributen [Sectie

7.15.3.3.] van de **fo:list-item-label** en **fo:list-item-body** elementen. De waarde van deze attributen is een lengte.

Het **end-indent** attribuut van het **fo:list-item-label** element krijgt vaak de XSL-FO functie **end-label()** mee als waarde. Deze functie geeft het eindpunt van het label terug als waarde. Omdat het lichaam van een lijstitem gewoonlijk op dezelfde lijn begint als het label van een lijstitem, wordt aan het **start-indent** attribuut van **fo:list-item-body** vaak de XSL-FO functie **body-start()** gegeven. Deze functie geeft de gecombineerde lengte van de waardes van **start-indent** en **provisional-distance-between-starts** terug. Een figuur om deze concepten te verduidelijken:



Figuur 7.6. Indentatie bij lijstobjecten

Het volgende voorbeeld is een deel van de templates die wij gebruiken om de inhoud van onze thesistekst te verwerken. Sommige attributen die in deze stylesheet gebruikt worden zullen pas verderop in deze tekst gebruikt worden. Wij willen hier vooral de aandacht vestigen op het gebruik van de lijstvormgevingsobjecten en hun attributen.

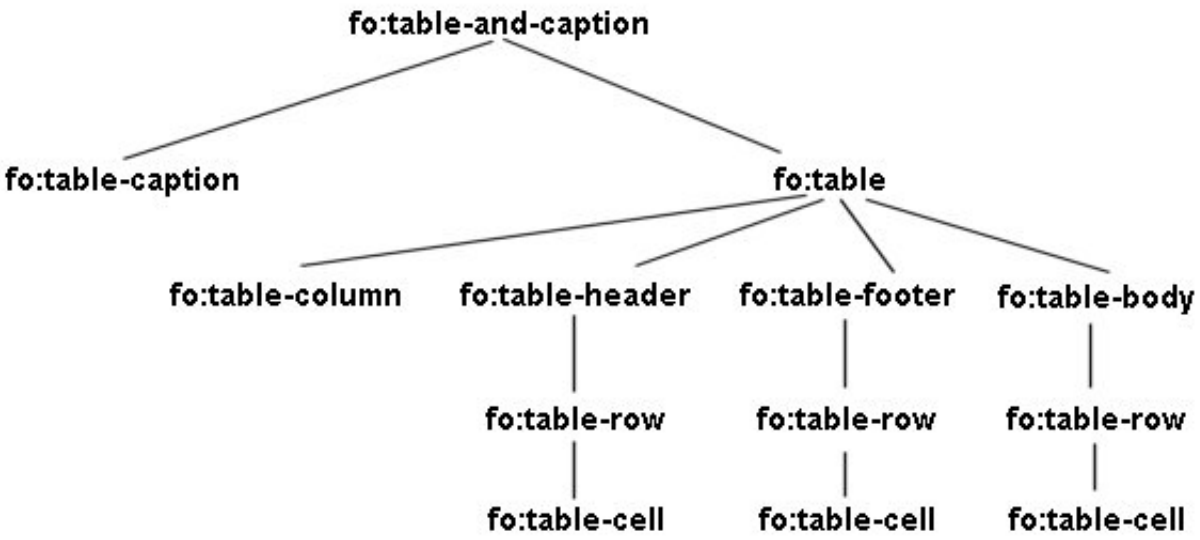
Voorbeeld 7.22. Gebruik van XSL-FO lijsten

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
     <xsl:output indent="yes"/>
     <xsl:template match="THESIS">
        <fo:root xsmlns:fo="http://www.w3.org/1999/XSL/Format">
           <fo:layout-master-set>
              <fo:simple-page-master master-name="toc" page-height="29.7"
              page-width="21cm" margin-top="2cm" margin-bottom="2cm"
              margin-left="2.5cm" margin-right="2.5cm">
                 <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
                 <fo:region-before extent="1cm"/>
                 <fo:region-after extent="1cm"/>
```

```
            </fo:simple-page-master>
          </fo:layout-master-set>
          <fo:page-sequence master-reference="content"
          initial-page-number="1">
            <fo:static-content flow-name="xsl-region-after">
              <fo:block font-size="10pt" font-family="Courier"
              text-align="end"> p.
                <fo:page-number/>
              </fo:block>
            </fo:static-content>
            <fo:flow flow-name="xsl-region-body">
              <xsl:apply-templates select="CHAPTER"/>
              <xsl:apply-templates select="APPENDIX"/>
              <xsl:apply-templates select="BIBLIOGRAPHY"/>
            </fo:flow>
          </fo:page-sequence>
        </fo:root>
      </xsl:template>
      <xsl:template match="CHAPTER">
        <fo:block font-size="20pt" break-before="page" font-family="serif"
        line-height="40pt" space-after="14pt" id="{generate-id()}">
          <xsl:number level="multiple" count="CHAPTER" format="1.1.1. "/>
          <xsl:value-of select="TITLE"/>
          <xsl:apply-templates select="SECTION"/>
        </fo:block>
      </xsl:template>
      <xsl:template match="CHAPTER//SECTION">
        <fo:block>
          <fo:block font-size="14pt" font-family="serif"
          line-height="20pt" space-after="14pt" id="{generate-id()}">
            <xsl:number level="multiple" count="CHAPTER|SECTION"
            format="1.1.1. "/>
            <xsl:value-of select="TITLE"/>
            <xsl:apply-templates select="PARA|SECTION"/>
          </fo:block>
        </fo:block>
      </xsl:template>
      <xsl:template match="PARA">
        <fo:block font-size="12pt" font-family="serif" line-height="14pt"
        space-after="14pt">
          <xsl:apply-templates/>
        </fo:block>
      </xsl:template>
      <xsl:template match="LIST">
        <fo:list-block space-after="14pt">
          <xsl:apply-templates select="LISTITEM"/>
        </fo:list-block>
      </xsl:template>
      <xsl:template match="LISTITEM">
        <fo:list-item>
          <fo:list-item-label end-indent="label-end()">
            <fo:block>
              <fo:inline font-family="Symbol"> &#183; </fo:inline>
            </fo:block>
          </fo:list-item-label>
          <fo:list-item-body start-indent="body-start()">
            <fo:block>
              <xsl:apply-templates/>
            </fo:block>
          </fo:list-item-body>
        </fo:list-item>
      </xsl:template>
    </xsl:stylesheet>
```

7.11. Tabellen

Het fundamentele tabelelement in XSL-FO is **fo:table-and-caption** element. Het **fo:table-and-caption** element is een vormgevingsobject van blokniveau dat een **fo:table** en een **fo:table-caption** element bevat. Als je tabel geen bijschrift nodig heeft, kan je gewoon een **fo:table** element gebruiken. De hiërarchie van de verschillende tabelelementen wordt hieronder grafisch voorgesteld.



Figuur 7.7. Hiërarchie van de tabelelementen

De opbouw van tabellen in XSL-FO lijkt heel sterk op die van HTML. Onderstaande tabel toont de overeenkomsten tussen HTML 4.0 tabelelementen en XSL vormgevingsobjecten:

Tabel 7.2. Tabellen in HTML versus Tabellen in XSL-FO

HTML element	XSL-FO vormgevingsobject
TABLE	fo:table-and-caption
no equivalent	fo:table
CAPTION	fo:table-caption
COL	fo:table-column
COLGROUP	no equivalent
THEAD	fo:table-header
TBODY	fo:table-body
TFOOT	fo:table-footer
TD	fo:table-cell
TR	fo:table-row

Elk **fo:table-and-caption** element bevat een optioneel **fo:table-caption** element en één **fo:table** element. Het bijschrift (caption) kan eender welk element van blokniveau bevatten. Default worden bijschriften vóór de tabel geplaatst. Dit kan echter aangepast worden door het **caption-side** attribuut van het **fo:table-and-caption** element één van de volgende waardes te geven:

- **before**
- **after**
- **start**
- **end**
- **top**

- **bottom**
- **left**
- **right**

In dit voorbeeldje wordt het bijschrift van de tabel onderaan geplaatst:

Voorbeeld 7.23. Het caption-side attribuut van het fo:table-and-caption element

```
<fo:table-and-caption caption-side="bottom">
   <fo:block font-size="12pt" font-family="serif"> Een tabel </fo:block>
   <fo:table>
      <!--de inhoud van deze tabel komt hier te staan-->
   </fo:table>
</fo:table-and-caption>
```

Het **fo:table** element bevat **fo:table-column** elementen, een optioneel **fo:table-header**, een optioneel **fo:table-footer** element en één of meerdere **fo:table-body** elementen. Het **fo:table-body** element wordt opgedeeld in **fo:table-row** elementen. Elk **fo:table-row** element wordt opgedeeld in **fo:table-cell** elementen. Het **fo:table-header** en het **fo:table-footer** element kunnen ofwel opgedeeld worden in **fo:table-cell** elementen ofwel in **fo:table-row** elementen.

Je kan ervoor zorgen dat cellen van een tabel meerdere rijen en kolommen overspannen, door het **number-columns-spanned** attribuut en/of het **number-rows-spanned** attribuut een gehele waarde mee te geven die het aantal rijen of kolommen weergeeft dat overspannen moet worden. Het optionele **column-number** geeft aan vanaf welke kolom het overspannen begint. De defaultwaarde voor dit attribuut is de huidige kolom. Het nummeren begint vanaf 1.

Het is ook mogelijk kaders rond delen van de tabel te tekenen door gebruik te maken van de normale kadereigenschappen. Het **empty-cells** attribuut heeft als waarde **show** of **hide**. **show** wordt gebruikt als kaders rond cellen die geen inhoud hebben getekend moeten worden, **hide** indien dit niet het geval is. De defaultwaarde van het **empty-cells** attribuut is **show**.

Als een lange tabel meerdere pagina's beslaat, wordt soms de hoofding en de footer op elke pagina herhaald. Je kan dit gedrag aanpassen met het **table-omit-header-at-break** attribuut en het **table-omit-header-at-break** attribuut van het **fo:table** element. De waarde **false** duidt aan dat de hoofding en de footer herhaald moeten worden op elke pagina. De waarde **true** duidt aan dat de hoofding en de footer niet herhaald moeten worden.

Het optionele **fo:table-column** element is een leeg element dat eigenschappen specificeert voor alle cellen in een bepaalde kolom. De cellen waarop dit element van toepassing is, worden geïdentificeerd door het **column-number** attribuut of door de positie van het **fo:table-column** element zelf. Het **fo:table-column** element bevat zelf geen cellen. Het **fo:table-column** element kan gebruikt worden om eigenschappen toe te kennen aan meer dan één opeenvolgende kolom door het **number-columns-spanned** attribuut een waarde groter dan 1 mee te geven. Het **fo:table-column** element wordt het vaakst gebruikt samen met het **column-width** attribuut. Dit attribuut heeft als waarde een lengte met bijhorende lengtemaat. De andere eigenschappen, zoals de standaardkader, de achtergrond, enzovoort kunnen echter ook aangepast worden.

In het volgende voorbeeld tonen de opbouw van de eerste twee rijen van bovenstaand voorbeeld [Sectie 7.11., Tabel 7.2.] :

Voorbeeld 7.24. Gebruik van de tabelelementen

```
<fo:table>
   <fo:table-column column-width="4.5cm"/>
   <fo:table-column column-width="11cm"/>
   <fo:table-header>
      <fo:table-cell>
         <fo:block font-weight="bold"> HTML element </fo:block>
      </fo:table-cell>
      <fo:table-cell>
         <fo:block font-weight="bold"> XSL-FO vormgevingsobject
         </fo:block>
      </fo:table-cell>
   </fo:table-header>
   <fo:table-body font-family="serif" padding="2pt">
      <fo:table-row>
         <fo:table-cell>
            <fo:block font-size="8pt" font-family="Courier"> TABLE
            </fo:block>
         </fo:table-cell>
         <fo:table-cell>
            <fo:block font-size="8pt" font-family="Courier">
            fo:table-and-caption </fo:block>
         </fo:table-cell>
      </fo:table-row>
      <fo:table-row>
         <fo:table-cell>
            <fo:block font-size="8pt" font-family="Courier"> no
            eauivalent </fo:block>
         </fo:table-cell>
         <fo:table-cell>
            <fo:block font-size="8pt" font-family="Courier"> fo:table
            </fo:block>
         </fo:table-cell>
      </fo:table-row>
   </fo:table-body>
</fo:table>
```

7.12. In-regel objecten

Het **fo:inline** element heeft geen speciaal effect op de lay-out van de pagina. Het is een element waaraan je makkelijk vormgevingsattributen zoals **font-style** [Sectie 7.15.4.2.] of **color** [Sectie 7.15.4.1.] kan verbinden om deze toe te passen op de inhoud van het **fo:inline** element. Het **fo:inline** vormgevingsobject is een element dat andere in-regel objecten samen groepeert. Het kan echter geen elementen van blokniveau bevatten. Een klein voorbeeldje:

Voorbeeld 7.25. Het fo:inline element en zijn attributen

```
<fo:static-content flow-name="xsl-region-after">
   <fo:block font-weight="bold" font-size="10pt" font-family="serif">
      <fo:inline font-style="italic" text-align="start"> XML en XSL, een
      studie </fo:inline>
      <fo:inline text-align="centered"> Pagina
         <fo:page-number/>
      </fo:inline>
      <fo:inline text-align="right"> Hoofdstuk 1: XML </fo:inline>
   </fo:block>
</fo:static-content>
```

7.13. Voetnoten

Het **fo:footnote** element creëert een voetnoot. Je plaatst het **fo:footnote** element in de stroom van de tekst daar waar de voetnootreferentie moet komen te staan. Het **fo:footnote** element bevat zowel de referentie als de tekst waarnaar gerefereerd wordt (de eigenlijke tekst van de voetnoot).

Het **fo:footnote** element heeft twee kinderen, die allebei verplicht aanwezig moeten zijn. Het eerste kind is een **fo:inline** vormgevingsobject dat de voetnootreferentie (bijvoorbeeld een sterretje of een nummer) oplevert. Het twee kind is een **fo:footnote-body** vormgevingsobject dat de inhoud (of het lichaam) van de voetnoot produceert. De formatter plaatst de voetnoottekst in de na-regio (region-after) (dit is meestal de footer) van de pagina.

In het onderstaande voorbeeld wordt een sterretje gebruikt als voetnootreferentie en wordt er gerefereerd naar de tekst "http://www.w3.org/TR/xsl/".

Voorbeeld 7.26. Het fo:footnote element

```
<fo:footnote>
   <fo:inline font-size="smaller" vertical-align="super"> * </fo:inline>
   <fo:footnote-body font-size="smaller">
      <fo:block font-style="italic"> http://www.w3.org/TR/xsl/
      </fo:block>
   </fo:footnote-body>
</fo:footnote>
```

7.14. Floats

Het **fo:float** element produceert een drijvende rechthoek die bevestigd wordt aan de bovenkant van de regio waarin dit element voorkomt. Het **fo:float** element wordt gewoonlijk gebruik voor grafische objecten, tabellen, grafieken of andere uit-regel inhoud die ergens op de pagina terecht moet komen, maar waarvan de exacte locate niet bijzonder belangrijk is.

In onderstaand voorbeeldje bevat het **fo:float** element een grafisch object:

Voorbeeld 7.27. Het fo:float element

```
<fo:block> Een overzicht van de boomstructuur van deze
vormgevingsobjecten
   <fo:float float="before">
      <fo:external-graphic src="images/PageTree.jpg" height="567px"
      width="281px"/>
      <fo:block font-family="Helvetica, sans">
         <fo:inline font-weight="bold"> Figuur 3.6 </fo:inline>
      Boomstructuur van de vormgevingsobjecten </fo:block>
   </fo:float>
vind je in figuur 3.6. </fo:block>
```

De formatter probeert het grafische object ergens op dezelfde pagina als de inhoud die rond het **fo:float** object staat te plaatsen. Het is echter mogelijk dat de formatter niet altijd plaats vindt op de pagina waar die inhoud komt te staan. Als dit het geval is, wordt het object op een volgende pagina geplaatst. De formatter is dus binnen bepaalde limieten vrij het object eender waar op de pagina te plaatsen.

De waarde van het **float** attribuut duidt aan aan welke kant van de pagina het **fo:float** object

drijft. Het kan de volgende waardes aannemen: **before**, **start**, **end**, **left**, **right**, **none** of **inherit**. De waarden **left**, **right** en **none** zijn afkomstig uit de CSS wereld. Volgens de technische aanbeveling zijn dit de enige waarden die het **float** attribuut kan aannemen. Wij hadden verwacht dat de waarde **after** ook bij dit lijstje zou staan. Waarom deze waarde ontbreekt, is een vraag die waarschijnlijk alleen de schrijvers van de technische aanbeveling kunnen beantwoorden.

Het **clear** attribuut kan toegekend worden aan elementen die zich in de buurt van het drijvende object bevinden. Dit attribuut geeft aan of deze elementen langs de kant van het object drijven of dat ze onder het drijvende object terecht zullen komen. Dit attribuut kan de volgende waardes aannemen:

- **start**: de beginzijde van het object mag niet grenzen aan een drijvend object
- **end**: de eindzijde van het object mag niet grenzen aan een drijvend object
- **left**: de linkerzijde van het object mag niet grenzen aan een drijvend object
- **right**: de rechterzijde van het object mag niet grenzen aan een drijvend object
- **both**: noch de linker- noch de rechterzijde van het object mag grenzen aan een drijvenobject
- **none**: er worden geen beperkingen opgelegd aan het object
- **inherit**: de beperkingen voor het **clear** attribuut worden geërfd van een ouderelement

7.15. Vormgevingseigenschappen

Vormgevingsobjecten alleen zeggen relatief weinig over hoe de inhoud vormgegeven moet worden. Het enige wat ze eigenlijk doen is de inhoud in abstracte containers stoppen, die geplaatst worden op bepaalde delen van een pagina. Het zijn de attributen die aan de vormgevingsobjecten toegekend worden die bepalen hoe de inhoud van deze dozen er uiteindelijk uit zal zien.

Zoals we eerder al vermeld hebben, zijn er meer dan tweehonderd verschillende vormgevingseigenschappen. [Sectie 7.5.1.] Niet alle eigenschappen kunnen toegepast worden op alle vormgevingsobjecten. Zo is het bijvoorbeeld niet echt nuttig om een **font-style** attribuut aan een **fo:external-graphic** toe te kennen. Eigenschappen die horen bij specifieke vormgevingsobjecten hebben we reeds bij deze vormgevingsobjecten besproken. De meeste eigenschappen kunnen echter wel toegepast worden op meer dan één soort vormgevingsobjecten. Als een eigenschap gemeenschappelijk is aan meerdere vormgevingsobjecten, heeft deze eigenschap dezelfde syntax en dezelfde betekenis voor al deze objecten.

Veel van de XSL-FO eigenschappen zijn gelijkaardig aan CSS [TRCSS] eigenschappen. De mogelijke waardes van de CSS **font-family** eigenschap zijn bijvoorbeeld hetzelfde als deze van een XSL-FO **font-family** attribuut. Ook de betekenis van deze eigenschap is dezelfde.

7.15.1. De id eigenschap

De **id** eigenschap kan toegepast worden op eender welk element. Deze eigenschap is een XML ID-type attribuut. De waarde van deze eigenschap moet dus een XML naam zijn die uniek is binnen de stylesheet en binnen het resulterende vormgevingsdocument. Deze laatste vereiste is lastiger omdat het mogelijk is dat één templateregel in de stylesheet honderden elementen genereert in het uitvoerdocument. De **generate-id()** functie van XSLT kan dit probleem oplossen. [Sectie 5.4.4.1., Tabel 5.4.]

7.15.2. De taal eigenschap

De **language** eigenschap specificeert de taal van de inhoud in een **fo:block** element of een **fo:character** element. In het algemeen is de waarde van deze eigenschap een ISO 693 taalcode [ISO693] zoals **en** of **nl**. De waarde van dit attribuut kan ook het sleutelwoord **none** of **use-document** zijn. Dit laatste sleutelwoord wil zeggen dat de taal van het invoerdocument zoals dat gespecificeerd wordt door het **xml:lang** attribuut gebruikt moet worden. [Sectie 2.3.8.3.]

Een voorbeeldje:

Voorbeeld 7.28. Het language attribuut en het id attribuut element

```
<fo:block>
   <fo:float id="a124" language="nl"> Dit is een zin geschreven in de
   Nederlandse taal. </fo:float>
</fo:block>
```

Alhoewel de **language** eigenschap geen direct effect heeft op de vormgeving, kan het een indirect effect hebben, als de formatter zijn lay-outalgoritmes kiest afhankelijk van de taal. De formatter gebruikt bijvoorbeeld een andere default **writing-mode** voor het Chinees dan voor het Engels. Wat op zijn beurt dan weer invloed heeft op het bepalen van de begin- en eindregio's en de **inline-progression-direction** en de **block-progression-direction**. [Sectie 7.3.3., Figuur 7.3.]

7.15.3. Blokeigenschappen

Blokeigenschappen zijn eigenschappen die normaal van toepassing zijn op een hele blok tekst (bijvoorbeeld een paragraaf). Zo is indentatie bijvoorbeeld een blokeigenschap omdat je wel een blok tekst kan indenteren, maar niet één enkel woord.

7.15.3.1. Brekingseigenschappen

De brekingseigenschappen specificeren waar pagina breaks toegelaten zijn en waar niet. Hieronder geven wij de brekingseigenschappen:

- **keep-with-next**
- **keep-with-previous**
- **keep-together**
- **break-before**
- **break-after**
- **inhibit-line-breaks**
- **page-break-before**
- **page-break-after**
- **page-break-inside**

Het **keep-with-next** attribuut bepaalt hoeveel inspanningen de formatter zal doen om het vormgevingsobject waar dit attribuut bijhoort op dezelfde pagina te houden als het volgende vormgevingsobject.

Het **keep-with-previous** attribuut bepaalt hoeveel inspanningen de formatter zal doen om het

vormgevingsobject waar dit attribuut bijhoort op dezelfde pagina te houden als het voorgaande vormgevingsobject.

Het **keep-together** attribuut bepaalt op zijn beurt hoeveel inspanningen de formatter zal doen om de volledige inhoud van het vormgevingsobject waar dit attribuut bijhoort op eenzelfde pagina te houden.

Deze regels zijn niet rigide, want het is natuurlijk altijd mogelijk dat een vormgevingsobject gewoon te groot is om op één pagina te passen. Elk van deze eigenschappen kan als waarde een geheel getal krijgen dat de hoeveelheid inspanning weergeeft die gedaan moet worden om de objecten op dezelfde pagina te houden. Hoe groter het geheel getal, hoe groter de inspanning. Deze eigenschappen kunnen ook als waarde de sleutelwoorden **always** of **auto** hebben. **always** betekent dat de inspanning maximaal moet zijn, **auto** betekent dat de breaks geplaatst mogen worden waar het het beste uitkomt. Dit is de defaultwaarde.

De eigenschappen **break-before** en **break-after** daarentegen leggen een break op. **break-before** specificeert dat het eerste gebied dat gegenereerd wordt bij het verwerken van het vormgevingsobject waar het attribuut bijhoort, het eerste gebied zal zijn dat in een bepaalde context (pagina, kolom) geplaatst wordt.

break-after specificeert dat het eerste gebied dat gegenereerd wordt bij het verwerken van het volgende vormgevingsobject (als er zo één is), het eerste gebied zal zijn dat in een bepaalde context (pagina, kolom) geplaatst wordt.

Wat juist wordt gebroken, wordt bepaald door de waarde van de eigenschap. De attributen **break-before** en **break-after** kunnen de volgende waardes hebben:

- **column**: breekt de huidige kolom en gaat verder met de volgende kolom
- **page**: breekt de huidige pagina en gaat verder naar de volgende pagina
- **even-page**: breekt de huidige pagina en gaat verder naar de volgende pagina met een even nummer; voegt een blanco pagina toe als de huidige pagina zelf een pagina met een even nummer is
- **odd-page**: breekt de huidige pagina en gaat verder naar de volgende pagina met een oneven nummer; voegt een blanco pagina toe als de huidige pagina zelf een pagina met een oneven nummer is
- **auto**: laat de formatter zelf beslissen waar te breken; de defaultwaarde

In onderstaand voorbeeld wordt een inhoudstafel gegenereerd waarbij voor de opsomming van de hoofdstukken en voor de opsomming van de appendices een nieuwe bladzijde begonnen wordt. Voor een beter begrip van dit voorbeeld verwijzen wij naar [Sectie 8.1.] .

Voorbeeld 7.29. Het break-before attribuut

```
<fo:flow flow-name="xsl-region-body">
   <fo:block break-before="page">
      <xsl:apply-templates select="CHAPTER" mode="TOC"/>
   </fo:block>
   <fo:block break-before="page">
      <xsl:apply-templates select="APPENDIX" mode="TOC"/>
   </fo:block>
</fo:flow>
```

Het **inhibit-line-breaks** attribuut is een boolean die op **true** kan gezet worden om aan te duiden dat regelbrekingen en paginabrekingen niet toegestaan zijn.

XSL-FO definieert verder nog drie extra paginabrekingseigenschappen: **page-break-after**, **page-break-before** en **page-break-inside**. Deze zijn niet absoluut noodzakelijk omdat hun effect ook bereikt kan worden door de juiste combinatie van de keep en break eigenschappen. Om bijvoorbeeld hetzelfde effect te krijgen als bij **page-break-after**, zet je **break-after** op **page** en **keep-with-previous** op **auto**.

7.15.3.2. Eigenschappen voor het gebruik van koppeltekens

De eigenschappen voor het gebruik van koppeltekens bepalen waar het splitsen van woorden toegestaan is en hoe deze opsplitsing moet gebeuren. Deze eigenschappen gelden alleen voor zachte of "optionele" koppeltekens, zoals diegene die soms gebruikt worden om lange woorden aan het eind van een lijn te breken. Deze eigenschappen zijn niet van toepassing op harde koppeltekens zoals die bijvoorbeeld gebruikt worden in het Nederlandse woord "West-Vlaanderen". Het is wel mogelijk dat de harde koppeltekens een invloed hebben op de toelaatbare plaatsen van zachte koppeltekens.

Er zijn zes eigenschappen voor het gebruik van koppeltekens:

- Het **hyphenate** attribuut laat automatische plaatsing van koppeltekens enkel toe als dit attribuut de waarde **true** heeft
- Het **hyphenation-character** attribuut geeft aan welk Unicode teken [UNICODE] gebruikt wordt om de woorden te splitsen, bijvoorbeeld een liggend streepje.
- Het **hyphenation-keep** attribuut specificeert waar en wanneer opsplitsing van woorden toegestaan is. De waarde van dit attribuut kan de volgende vier sleutelwoorden aannemen: **column**, **none**, **page** en **inherit**. De defaultwaarde staat op niet splitsen.
- Het **hyphenation-ladder-count** attribuut heeft als waarde een niet-negatief geheel getal dat het maximale aantal opeenvolgende lijnen die eindigen op een koppelteken specificeert.
- Het **hyphenation-push-character-count** attribuut heeft als waarde een niet-negatief geheel getal dat het minimale aantal tekens specificeert die moeten volgen op een automatisch toegevoegd koppelteken.
- Het **hyphenation-remain-character-count** attribuut heeft als waarde een niet-negatief geheel getal dat het minimale aantal tekens specificeert die moeten voorafgaan aan een automatisch toegevoegd koppelteken.

Hieronder geven we een eenvoudig voorbeeldje van het gebruik van deze eigenschappen:

Voorbeeld 7.30. Het toevoegen van koppeltekens

```
<fo:block hyphenate="true" hyphenation-character="-"
hyphenation-keep="page" hyphenation-ladder-count="1"
hyphenation-push-character-count="4"
hyphenation-remain-character-count="4"> Tekst waarvoor deze
eigenschappen gelden </fo:block>
```

XSL-FO specificeert geen splitsingsalgoritme voor het toevoegen van zachte koppeltekens. Zelfs als de eigenschappen splitsing toestaan, hangt het nog steeds volledig af van de formatter om te bepalen hoe een woord gesplitst moet worden. Het is ook mogelijk dat er gewoon geen poging ondernomen wordt om de woorden te splitsen.

7.15.3.3. Indentatie-eigenschappen

De indentatie-eigenschappen specificeren hoe ver lijnen geïndenteerd moeten worden vanaf de begrenzingen van de tekst. Er zijn vier indentatie-eigenschappen:

- De **start-indent** eigenschap plaatst alle lijnen van een tekst op een bepaalde afstand van de beginzijde van de tekst. In het Nederlands is dit de linkerzijde van de tekst.
- De **end-indent** eigenschap plaatst alle lijnen van een tekst op een bepaalde afstand van de eindzijde van de tekst. In het Nederlands is dit de rechterzijde.
- De **text-indent** eigenschap indenteert alleen de eerste lijn van de tekst vanaf de beginzijde van die tekst.
- De **last-line-end-indent** eigenschap indenteert alleen de laatste lijn van de tekst vanaf de eindzijde van die tekst.

De waardes van deze attributen worden gegeven als een lengte met bijhorende lengtemaat. In onderstaand voorbeeld wordt de eerste lijn van het tekstblok met 1 cm geïndenteerd.

Voorbeeld 7.31. Het text-indent attribuut

```
<fo:block text-indent="1cm"> De eerste regel van deze tekst wordt
geïndenteerd met 1cm. </fo:block>
```

Een blok tekst waarvan alle lijnen aan het begin en aan het einde geïndenteerd worden, ziet er als volgt uit:

Voorbeeld 7.32. Het start-indent en het end-indent attribuut

```
<fo:block start-indent="1cm" end-indent="1cm"> Alle regels van deze
tekst worden aan het begin en op het einde geïndenteerd met 1cm.
</fo:block>
```

De waarde van het **text-indent** attribuut wordt toegevoegd aan de waarde van het **start-indent** attribuut om de totale indentatie van de eerste lijn te verkrijgen. Het is dus perfect mogelijk om een negatieve waarde aan het **text-indent** attribuut toe te kennen, waardoor de eerste lijn minder geïndenteerd wordt dan de rest van het tekstblok. Zo'n constructie noemen we een hangende indentatie.

7.15.4. Eigenschappen van tekens

Tekeneigenschappen beschrijven de eigenschappen van individuele tekens (characters). Ze worden toegepast op elementen zoals **fo:block** en **fo:inline** die tekens bevatten. Deze eigenschappen beschrijven de kleur, het lettertype en andere gelijkaardige eigenschappen.

7.15.4.1. Het color attribuut

Het **color** attribuut bepaalt de voorgrondkleur van de inhoud. Deze eigenschap maakt gebruik van dezelfde syntax als de CSS **color** eigenschap [TRCSS].

In onderstaand voorbeeldje wordt de tekst in het blauw weergegeven:

Voorbeeld 7.33. Het color attribuut

```
<fo:inline color="#0000FF"> Deze tekst heeft een blauw kleurtje.
</fo:inline>
```

Zoals uit bovenstaand voorbeeld blijkt, worden kleuren op dezelfde manier gespecificeerd als in CSS [CSSCOLOR]. De kleuren worden voorgesteld als een hexadecimaal drietal in de vorm #RoodRoodGroenGroenBlauwBlauw of als één van de zestien kleuren die een naam gekregen hebben: **aqua**, **black**, **blue**, **fuchsia**, **gray**, **green**, **lime**, **maroon**, **navy**, **olive**, **purple**, **red**, **silver**, **teal**, **white** en **yellow**.

7.15.4.2. Lettertype-eigenschappen

Een vormgevingsobject dat tekst bevat kan heel veel lettertype-eigenschappen hebben. De meeste van deze eigenschappen zijn bekend uit CSS. Wij geven hieronder een opsomming:

- Het **font-family** attribuut heeft als waarde een lijst van namen van lettertypes in volgorde van voorkeur
- Het **font-size** attribuut heeft als waarde een lengte en een bijhorende lengtemaat
- Het **font-size-adjust** attribuut heeft als waarde de voorkeurratio tussen de x-hoogte en de grootte (size) van een lettertype. Deze ratio wordt gespecificeerd als een reëel getal of als **none**.
- Het **font-stretch** attribuut heeft als waarde de breedte (width) van een lettertype. Deze waarde kan één van de volgende sleutelwoorden zijn **ultra-condensed**, **extra-condensed**, **semi-condensed**, **narrower**, **normal**, **wider**, **semi-expanded**, **extra-expanded** of **ultra-expanded**.
- Het **font-style** attribuut specificeert de stijl van het lettertype. Dit attribuut kan als waarde één van de volgende sleutelwoorden hebben: **italic**, **normal**, **oblique**, **reverse-normal** en **reverse-oblique**.
- Het **font-variant** attribuut kan twee waardes aannemen **normal** of **smallecaps**. **smallecaps** wil zeggen dat de kleine letters in de tekst er hetzelfde uitzien als de hoofdletters, alleen iets kleiner.
- Het **font-weight** attribuut geeft de dikte van de letters weer. Dit attribuut kan volgende waardes aannemen: **lighter**, **100**, **200**, **300**, **normal** (is hetzelfde als **400**), **500**, **600**, **bold** (is hetzelfde als **700**), **800**, **900** of **bolder**

7.15.4.3. Teksteigenschappen

Teksteigenschappen passen een bepaalde stijl toe op een tekst die min of meer onafhankelijk zijn van het gekozen lettertype. Deze eigenschappen zijn:

- **text-transform**
- **text-shadow**
- **text-decoration**
- **score-spaces**

Het **text-transform** attribuut definieert of en waar er hoofdletters in de tekst moeten staan. Deze eigenschap is identiek aan de CSS eigenschap **text-transform**. Dit attribuut kan vier verschillende waardes aannemen:

- **none**: verandert niets aan de tekst; dit is de defaultwaarde
- **capitalize**: zorgt ervoor dat de eerste letter van elk woord een hoofdletter is en dat al de volgende letters kleine letters zijn
- **uppercase**: maakt van alle letters hoofdletters

- **lowercase**: maakt van alle letters kleine letters

Deze eigenschap is taalspecifiek. Het Chinees en het Hebreeuws hebben bijvoorbeeld geen gescheiden hoofdletters en kleine letters. De formatter kan deze eigenschappen dan ook perfect negeren als deze toegepast worden op niet-westerse teksten.

De **text-shadow** eigenschap voegt schaduw toe aan de tekst. Dit is gelijkaardig aan een achtergrondkleur toevoegen, met dit verschil dat de schaduw zich bevestigt aan de tekst zelf in plaats van aan de rechthoek die de tekst bevat. De waarde van het **text-shadow** attribuut kan het sleutelwoord **none** zijn, een kleur met een naam of een RGB kleur [CSSCOLOR].

Het **text-decoration** attribuut is gelijkaardig aan de CSS **text-decoration** eigenschap. Het attribuut heeft vijf mogelijke waardes:
- **none**: geen decoratie; dit is de defaultwaarde
- **underline**: onderlijnt de tekst
- **overline**: plaatst een lijn boven de tekst
- **line-through**: elke regel tekst wordt doorstreept
- **blink**: de tekst wisselt af tussen zichtbaar en onzichtbaar; het is geen vereiste dat deze waarde ondersteund wordt door de User Agent

In aanvulling op de vijf waardes die al bekend zijn vanuit CSS, heeft XSL-FO nog vier waardes die de decoratie die geërfd wordt van een ouderelement afzetten.
- **no-underline**
- **no-overline**
- **no-line-through**
- **no-blink**

Het **score-spaces** attribuut bepaalt of witruimte één of ander soort van lijn (onder, boven of door het midden) meekrijgt. Dit attribuut kan twee waardes hebben **true** of **false**. Als de waarde bijvoorbeeld op **true** staat en het **text-decoration** attribuut op **underline**, wordt de witruimte in een tekst onderlijnd.

7.15.5. Eigenschappen van zinnen

Eigenschappen van zinnen zijn toepasbaar op een groep van tekens. Dit wil zeggen dat deze eigenschappen alleen zin hebben wanneer we het hebben over meer dan één letter tegelijkertijd. Deze eigenschappen kunnen betrekking hebben over de hoeveelheid ruimte tussen letters of woorden.

7.15.5.1. De letter-spacing eigenschap

De hoeveelheid ruimte tussen twee tekens is geen absoluut getal. De meeste formatters passen de ruimte tussen de letters aan op basis van wat lokaal nodig is om een gebalanceerde tekst te verkrijgen. Kwalitatief hoogstaande lettertypes gebruiken ook verschillende hoeveelheden ruimte tussen verschillende symbolen. Het is echter wel mogelijk tekst wat ruimer of strakker te maken, gebruik makende van deze eigenschappen.

Het **letter-spacing** attribuut voegt extra ruimte toe tussen elk paar van symbolen. De waarde van dit attribuut is een lengte met bijhorende lengtemaat die de gewenste hoeveelheid extra

ruimte weergeeft. Een voorbeeldje:

Voorbeeld 7.34. Het letter-spacing attribuut

```
<fo:block letter-spacing="3px"> Aan de ruimtes tussen de letters van
deze tekst worden drie pixels extra toegevoegd. </fo:block>
```

De lengte van dit attribuut mag ook negatief zijn om op die manier een strakkere tekst te krijgen. In het algemeen leggen formatters limieten op aan de hoeveelheid extra ruimte die verwijderd of toegevoegd mag worden aan de ruimte tussen twee letters.

7.15.5.2. De word-spacing eigenschap

Het **word-spacing** attribuut past de hoeveelheid ruimte tussen woorden aan. Het gedrag van dit attribuut is gelijkaardig aan dat van het **letter-spacing** attribuut. De waarde van dit attribuut is een lengte met bijhorende lengtemaat die de hoeveelheid extra toe te voegen ruimte tussen twee woorden aangeeft.

Een voorbeeldje:

Voorbeeld 7.35. Het word-spacing attribuut

```
<fo:block word-spacing="0.2cm"> Aan de ruimtes tussen de woorden van
deze tekst wordt 0.2cm extra toegevoegd. </fo:block>
```

7.15.5.3. Eigenschappen voor tussenregelruimtes

De XSL-FO formatter verdeelt blokgebieden in lijngebieden. Het is niet mogelijk rechtstreeks vanuit XSL-FO lijngebieden te creëren. Je kan echter wel de spatiëring tussen de lijngebieden beïnvloeden aan de hand van de volgende vijf eigenschappen:

- Het **line-height** attribuut geeft de minimale hoogte van een lijn aan.
- Het **line-height-shift-adjustment** kan twee waardes aannemen: **consider-shifts** en **disregard-shifts**. De eerste waarde geeft aan dat sub- en superscripten de hoogte van een lijn moeten vergroten, de tweede waarde geeft aan dat dit niet moet gebeuren.
- Het **line-stacking-strategy** attribuut kan drie waardes aannemen: **line-height**, **font-height** of **max-height**. **line-heigth** komt overeen met de CSS **line-heigth** eigenschap en is de defaultwaarde. **font-height** maakt de lijn even hoog als de hoogte van het lettertype opgeteld met **text-depth** en **text-altitude**. **max-height** zorgt ervoor dat de lijn de maximaal toegestane hoogte krijgt. Hoe deze juist berekend wordt is niet belangrijk.
- Het **text-depth** attribuut specificeert hoeveel vertikale ruimte er na elke lijn toegevoegd wordt. De waarde van dit attribuut kan ook het sleutelwoord **use-font-metrics** zijn om aan te geven dat dit afhankelijk is van het gebruikte lettertype. Dit is ook de defaultwaarde.
- Het **text-altitude** attribuut heeft als waarde een lengte met bijhorende lengtemaat die de minimaal toe te voegen vertikale ruimte specificeert die toegevoegd wordt voor elke lijn. Dit attribuut kan ook als waarde het sleutelwoord **use-font-metrics** hebben. Dit is ook de defaultwaarde.

De lijnhoogte hangt ook in grote mate af van de grootte van het lettertype waarin de tekst geschreven is. Grotere lettertypes hebben vanzelfsprekend hogere lijnen nodig. Een voorbeeldje:

Voorbeeld 7.36. Eigenschappen voor tussenregelruimtes

```
<fo:block font-size="12pt" line-height="24pt"> De ruimte tussen de
lijnen van deze tekst is dubbel zo groot als de hoogte van de letters
van deze tekst. </fo:block>
```

7.15.5.4. Eigenschappen voor het aligneren van tekst

De **text-align** en **text-align-last** attributen specificeren hoe de in-regel inhoud gealigneerd wordt binnen de bevattende rechthoek. Het **text-align-last** attribuut maakt het mogelijk een verschillende waarde te specificeren voor de laatste regel in een blok. Dit is vooral belangrijk voor gebalanceerde tekst, waar de laatste regel vaak niet genoeg woorden heeft om op een aantrekkelijke manier gebalanceerd te worden. Hieronder vind je een opsomming van de mogelijke waardes die deze attributen kunnen aannemen. Het **text-align-last** attribuut heeft nog één extra waarde buiten deze acht, met name **relative**. Een relatief gealigneerde laatste regel zal op dezelfde manier gealigneerd worden als de andere regels tenzij **text-align** als waarde **justify** heeft. In dat geval zal de laatste regel gealigneerd worden met de start-zijde van de rechthoek.

- **start**: linksgealigneerd in het geval van talen zoals het Nederlands
- **center**: gecentreerd
- **end**: rechtsgealigneerd in het geval van westerse talen
- **justify**: extra ruimte wordt toegevoegd waar nodig om de regel de rechthoek van links naar rechts te laten opvullen
- **left**: aligneer met de linkerzijde van de pagina, ongeacht de schrijfrichting
- **right**: aligneer met de rechterzijde van de pagina, ongeacht de schrijfrichting
- **inside**: aligneer met de binnenzijde van de pagina, dit wil zeggen met de rechterzijde op de linkse pagina van een dubbele pagina of met de linkerzijde van de rechtse pagina van een dubbele pagina
- **outside**: aligneer met de buitenzijde van de pagina, dit wil zeggen met de linkerzijde op de linkse pagina van een dubbele pagina of met de rechterzijde van de rechtse pagina van een dubbele pagina.

7.15.5.5. White space eigenschappen

Het **space-treatment** attribuut specificeert wat de formatter moet doen met witruimte die nog aanwezig is na de transformatie van het originele brondocument naar het document van vormgevingsobjecten. Het kan de waardes **preserve** (de defaultwaarde) of **ignore** aannemen. Als je dit attribuut op **ignore** zet, wordt leidende (leading) en afsluitende (trailing) witruimte weggegooid.

Het **white-space-collapse** attribuut kan op de defaultwaarde **true** of op **false** gezet worden. Als deze waarde op **true** gezet wordt, dan worden opeenvolgende witruimte tekens vervangen door één enkele witruimte. In het andere geval worden deze witruimte tekens niet veranderd.

Het **wrap-option** attribuut bepaalt hoe tekst die te lang is om op één regel te passen, behandeld wordt. Deze eigenschap kan de defaultwaarde **wrap** of de waarde **no-wrap** hebben. Als deze waarde op **wrap** gezet wordt, mag de formatter lijnbrekingen toevoegen

waar nodig om de tekst op de pagina te plaatsen.

7.15.6. Eigenschappen van gebieden

Eigenschappen van gebieden worden toegepast op rechthoeken. Deze kunnen ofwel van blokniveau zijn ofwel van in-regel niveau. Deze rechthoeken hebben allemaal een achtergrond, kantlijnen, kaders, padding en een grootte.

7.15.6.1. Achtergrondeigenschappen

De achtergrondeigenschappen zijn identiek aan de CSS achtergrondeigenschappen. Wij geven de vijf eigenschappen hieronder:

- Het **background-color** attribuut specificeert de kleur van de achtergrond van de rechthoek. De waarde van dit attribuut is ofwel een kleur met een naam, ofwel een RGB kleur [CSSCOLOR] ofwel het sleutelwoord **transparent**
- Het **background-image** geeft de URI van een afbeelding die als achtergrond gebruikt wordt. Deze waarde kan ook het sleutelwoord **none** zijn.
- Het **background-attachment** attribuut specificeert of de achtergrondafbeelding bevestigd wordt aan het venster waarin het document getoond wordt of aan het document zelf. De waarde van dit attribuut is één van de twee sleutelwoorden: **fixed** of **scroll**.
- Het **background-position** attribuut specificeert waar de achtergrondafbeelding geplaatst wordt in de rechthoek. Mogelijke waardes zijn onder andere: **center**, **left**, **right**, **bottom**, **middle**, **top** of een coördinaat.
- Het **background-repeat** attribuut specificeert of en hoe een achtergrondafbeelding in tegels opgebroken wordt als het kleiner is dan de bevattende rechthoek. Mogelijke waardes zijn **repeat**, **no-repeat**, **repeat-x** en **repeat-y**.

In onderstaand voorbeeld worden deze eigenschappen geïllustreerd. Het is niet erg nuttig om aan het **background-color** attribuut de waarde **black** mee te geven, aangezien er een achtergrondafbeelding meegegeven wordt.

Voorbeeld 7.37. Achtergrondeigenschappen

```
<fo:block background-image="images/background.jpg"
background-position="0,0" background-repeat="repeat"
background-color="black"> Tekst die in het resulterende document tegen
de gespecificeerde achtergrond zal verschijnen. </fo:block>
```

7.15.6.2. Kadereigenschappen

De kadereigenschppen beschrijven hoe de kader rond de rechthoek eruit moet zien. Deze eigenschappen zijn grotendeels dezelfde als de CSS kadereigenschappen [TRCSS]. Er zijn 31 kadereigenschappen:

- Eigenschappen die de kleur bepalen: **border-color**, **border-before-color**, **border-after-color**, **border-start-color**, **border-end-color**, **border-top-color**, **border-bottom-color**, **border-left-color** en **border-right-color**. De defaultkleur is **black**.
- Eigenschappen die de breedte bepalen: **border-width**, **border-before-width**, **border-after-width**, **border-start-width**, **border-end-width**, **border-top-width**,

border-bottom-width, **border-left-width** en **border-right-width**. De defaultbreedte is **medium**.
- Eigenschappen die de stijl bepalen: **border-style**, **border-before-style**, **border-after-style**, **border-start-style**, **border-end-style**, **border-top-style**, **border-bottom-style**, **border-left-style** en **border-right-style**. De defaultwaarde is **none**.
- Shorthand eigenschappen: **border**, **border-top**, **border-bottom**, **border-left**, **border-right**, **border-color**, **border-style** en **border-width**.

Een voorbeeldje:

Voorbeeld 7.38. Kadereigenschappen

```
<fo:block border-before-color="red" border-before-width="1px"
border-after-color="green" border-after-width="1px"
border-start-color="yellow" border-start-width="1px"
border-end-color="bleu" border-end-width="1px"> Deze tekst komt in het
resulterende document in een rood/groen/geel/blauwe kader van 1 pixel
breed terecht. </fo:block>
```

7.15.6.3. Paddingeigenschappen

De paddingeigenschappen specificeren de hoeveelheid ruimte tussen de kader van de rechthoek en de inhoud van de rechthoek. De paddingeigenschappen zijn voor het grootste deel hetzelfde als de CSS paddingeigenschappen. Er zijn echter een aantal extra eigenschappen. Elk van deze attributen heeft een lengte met een bijhorende lengtemaat als waarde. De eigenschappen zijn:
- **padding-after**
- **padding-before**
- **padding-bottom**
- **padding-end**
- **padding-left**
- **padding-start**
- **padding-right**
- **padding-top**

Een voorbeeldje:

Voorbeeld 7.39. Paddingeigenschappen

```
<fo:block padding-before="1cm" padding-after="1cm" padding-start="1cm"
padding-end="1cm"> Deze tekst heeft 1 centimeter padding aan elke zijde
van de rechthoek. </fo:block>
```

7.15.6.4. Kantlijneigenschappen

7.15.6.4.1. Kantlijneigenschappen voor blokken

Er zijn vijf kantlijneigenschappen voor blokken. De waardes van deze eigenschappen worden

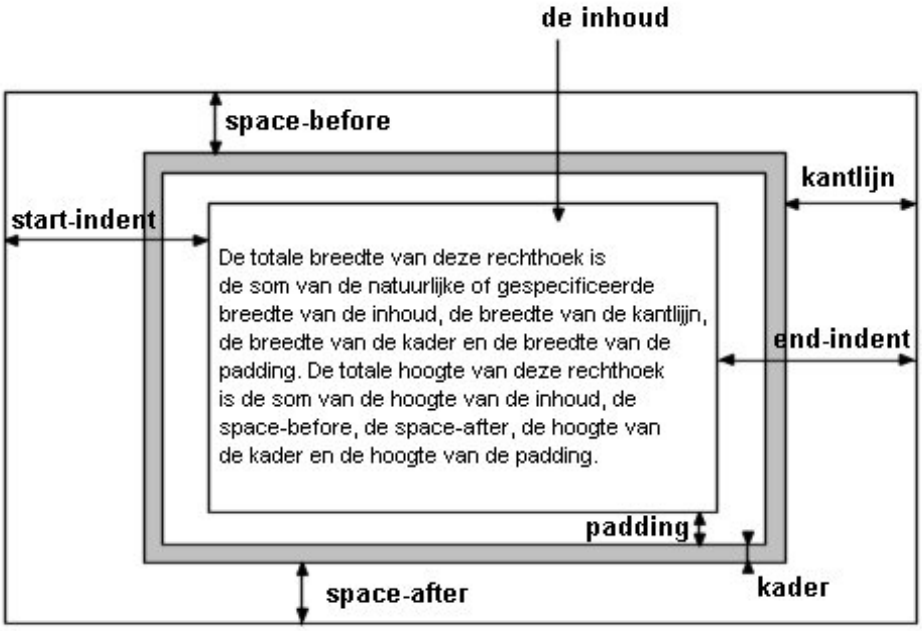
gegeven als een lengte. Deze eigenschappen zijn:

- **margin-top**
- **margin-bottom**
- **margin-left**
- **margin-right**
- **margin**

De enige bestaansreden van deze eigenschappen is compatibiliteit met CSS. In het algemeen is het beter dat je de volgende vier eigenschappen gebruikt, omdat deze, zoals wij reeds eerder aangehaald hebben, beter in het XSL-FO vormgevingsmodel passen:

- **space-before**
- **space-after**
- **start-indent**
- **end-indent**

De **space-before** en **space-after** eigenschappen zijn equivalent met respectievelijk de **margin-top** en **margin-bottom** eigenschappen, als de writing-mode op **lr-tb** staat. De **start-indent** eigenschap is equivalent met de som van **padding-left**, **border-left** en **margin-left-width**. De **end-indent** eigenschap is equivalent met de som van **padding-right**, **border-right-width** en **margin-ricgt**. Onderstaande figuur verduidelijkt deze concepten:



Figuur 7.8. Padding, indentatie, kaders , space-before, space-after

In tegenstelling tot de **margin**-eigenschappen, worden **space**-eigenschappen bepaald door meer dan één waarde. De ruimte-eigenschappen bevatten meer bepaald een minimale waarde, een voorkeurwaarde, een maximale waarde, een voorwaardelijkheid (conditionality) en een voorrangswaarde (precedence), in die volgorde. De formatter heeft zo wat meer vrijheid in het

bepalen van de pagina-lay-out. De formatter is vrij in het kiezen van eender welke hoeveelheid ruimte tussen de minimale en de maximale waarde om aan de beperkingen van de pagina te voldoen.

Elk van de **space**-waardes is een lengte met bijhorende lengtemaat. De voorwaardelijkheid heeft als waarde één van de twee sleutelwoorden **discard** of **retain**. De voorwaardelijkheid bepaalt wat er gebeurt met extra ruimte aan het eind van een regel. De defaultwaarde is **discard**. De voorrangswaarde bepaalt wat er gebeurt wanneer de **space-end** van één in-regel gebied in conflict komt met de **space-start** van het volgende in-regel gebied. (We behandelen deze twee eigenschappen in de volgende sectie.) In een conflictsituatie wint het gebied met de hoogste voorrangswaarde. De defaultvoorrangswaarde is 0. De vijf **space**-waardes worden van elkaar gescheiden door een punt-komma.

In onderstaand voorbeeld illustreren wij het gebruik van deze eigenschappen.

Voorbeeld 7.40. Kantlijneigenschappen

```
<fo:block space-before="0cm;0.5cm;1cm;discard;force"> In het ideale
geval voegt de formatter 0.5cm ruimte toe vóór dit element. Deze
toegevoegde ruimte kan echter variëren tussen 0cm en 1cm. De
voorwaardelijkheid staat op discard, wat wil zeggen dat extra ruimte op
het einde van de regel weggedaan wordt. Omdat de voorrangswaarde
tenslotte op force staat, worden conflicterende gebieden gewoon
overschreven. </fo:block>
```

7.15.6.4.2. Kantlijneigenschappen voor in-regel objecten

Er zijn twee kantlijneigenschappen die alleen van toepassing zijn op in-regelobjecten:

- **space-end**
- **space-start**

Deze attributen specificeren dat er voor of na het element extra ruimte toegevoegd moet worden. De eigenlijke ruimte die toegevoegd wordt, kan groter of kleiner zijn. Omdat deze extra ruimte geen onderdeel is van de rechthoek waarin het in-regel element staat, kan het gebeuren dat de space-end van één in-regel object in conflict komt met de space-start van een ander object. Wat er moet gebeuren in geval van conflicten, wordt bepaald door de voorrangswaarde van beide objecten.

7.15.6.5. Grootte-eigenschappen

Er zijn zes eigenschappen die de hoogte en de breedte van het inhoudsgebied van een rechthoek specificeren:

- **height**
- **width**
- **max-height**
- **max-width**
- **min-height**
- **min-width**

Deze eigenschappen specificeren **niet** de totale breedte en hoogte van de rechthoek: deze

afmetingen bevatten zoals uit de figuur [Sectie 7.15.6.4.1., Figuur 7.8.] hierboven blijkt, immers ook nog de kantlijnen, de padding en de kaders. Deze eigenschappen specificeren alleen de breedte en de hoogte van het inhoudsgebied. De waardes van deze attributen kunnen een lengte met bijhorende lengtemaat zijn of het sleutelwoord **auto**. Het sleutelwoord **auto** kiest de hoogte en de breedte van het inhoudsgebied gebaseerd op de hoeveelheid inhoud die zich in de rechthoek bevindt. De hoogte en de breedte zijn in geen enkel geval groter dan de waardes gespecificeerd door het **max-height** en het **max-width** attribuut. Ze kunnen ook niet kleiner zijn dan **min-height** en **min-width**.

Een voorbeeld:

Voorbeeld 7.41. Grootte-eigenschappen

```
<fo:block height="4cm" width="4cm"> Inhoud die terecht komt in een
inhoudsgebied van 4 op 4 centimeter. </fo:block>
```

7.15.6.6. De overflow eigenschap

De **overflow** eigenschap bepaalt wat er gebeurt wanneer er te veel inhoud is om binnen een rechthoek met een gespecificeerde grootte te passen. Dit kan ofwel een expliciete specificatie zijn die gebruik maakt van de grootte-eigenschappen ofwel een impliciete specificatie gebaseerd op de paginagrootte of andere beperkingen. De verschillende mogelijke waardes voor dit attribuut worden voorgesteld door de sleutelwoorden die wij hieronder opsommen:

- **auto**: Als er overflow optreedt, worden scrollbars gebruikt, anders niet. Als er geen scrollbars aanwezig zijn (bijvoorbeeld op een af te drukken pagina), voeg dan een nieuwe pagina toe in het geval van stroominhoud en signaleer een fout in het geval van statische inhoud. Dit is de defaultwaarde.
- **hidden**: De inhoud die buiten de rechthoek terecht komt, wordt niet getoond.
- **scroll**: Scrollbars worden aan de rechthoek bevestigd zodat de lezer kan scrollen naar de resterende inhoud.
- **visible**: De volledige inhoud wordt getoond. Indien nodig worden de grootte-beperkingen van de rechthoek aangepast.
- **error-if-overflow**: De formatter moet stoppen en een foutboodschap weergeven indien inhoud niet binnen zijn rechthoek past.
- **paginate**: Als het object waarbij overflow optreedt een pagina is, wordt een nieuwe pagina gecreëerd om de overgebleven inhoud te bevatten.

7.15.6.7. De reference-orientation eigenschap

De **reference-orientation** eigenschap maakt het mogelijk de inhoud van een rechthoek te roteren relatief ten opzichte van de normale oriëntatie van de inhoud. De enige toegelaten waardes voor dit attribuut zijn waardes die vanaf -270 graden met 90 graden toenemen in tegenwijzerzin, dus -270, -180, -90, 0, 90, 180 en 270.

Een voorbeeldje:

Voorbeeld 7.42. Grootte-eigenschappen

```
<fo:block reference-orientation="90"> Inhoud die negentig graden
geroteerd wordt in tegenwijzerzin. </fo:block>
```

7.15.6.8. De writing-mode eigenschap

De **writing-mode** eigenschap hebben we reeds eerder behandeld. [Sectie 7.3.3., Figuur 7.3.] Deze eigenschap specificeert de richting van de tekst in de bevattende rechthoek. Dit attribuut heeft belangrijke gevolgen voor de ordening van de vormgevingsobjecten in de rechthoek.

De **writing-mode** eigenschap specificeert de **writing-mode** voor een gebied. Dit attribuut kan één van de volgende sleutelwoordwaardes hebben:

- **bt-lr**: van onder naar boven, van links naar rechts
- **bt-rl**: van onder naar boven, van rechts naar links
- **lr-bt**: van links naar rechts, van onder naar boven
- **lr-tb**: van links naar rechts: van boven naar onder
- **rl-bt**: van rechts naar links, van onder naar boven
- **rl-tb**: van rechts naar links, van boven naar onder
- **tb-lr**: van boven naar onder, van links naar rechts
- **tb-rl**: van boven naar onder, van rechts naar links

7.15.6.9. Weduwen en wezen

Een weduwe is een enkele regel of een paragraaf aan de bovenkant van een pagina. Een wees is een enkele regel of een paragraaf aan de onderkant van de pagina. Bij drukwerk wordt meestal geprobeerd om deze eenzame regels of paragrafen op te schuiven naar een volgende of een vorige pagina om het voorkomen van weduwen en wezen te vermijden. In het geval van weduwes probeert men de eenzame regels aan de bovenkant van de pagina te verhuizen naar de voorgaande pagina. In het geval van wezen probeert men de eenzame regels aan de onderkant van de pagina naar de volgende pagina te verhuizen.

Het is mogelijk het aantal regels vast te leggen die als ze alleen voorkomen, als een weduwe of een wees beschouwd worden. Dit doe je door de **widows** of de **orphans** eigenschap een gehele getalwaarde toe te kennen. Als je er bijvoorbeeld voor wilt zorgen dat elke gedeeltelijke paragraaf aan de bovenkant van een pagina minimaal uit drie regels moet bestaan zet je het **widows** attribuut op 3. Als je er bijvoorbeeld voor wilt zorgen dat elke gedeeltelijke paragraaf aan de onderkant van een pagina minimaal uit drie regels moet bestaan zet je het **orphans** attribuut op 2.

Voorbeeld 7.43. Gebruik van de widows en orphans eigenschappen

```
<fo:simple-page-master master-name="name" widows="3" orphans="2"
page-height="29.7" page-width="21cm">
   <fo:region-body/>
</fo:simple-page-master>
```

Zoals de aandachtige lezer misschien heeft opgemerkt, zijn er in onze tekst weduwen en wezen aanwezig die maar uit één regel bestaan. Wij hebben deze eigenschappen wel juist ingesteld in onze stylesheets maar de Formatting Objects Processor dit wij gebruiken [FOP], ondersteunt deze eigenschap nog niet.

7.16. Conclusie

Zoals uit dit hoofdstuk blijkt biedt XSL-FO ontzettend veel mogelijkheden op het gebied van vormgeving. Deze lange waslijst van mogelijkheden, zorgt echter ook voor een vrij hoge drempel. Je moet heel veel objecten en eigenschappen kennen om de lay-out van je tekst exact te krijgen zoals jij het wilt. XSL-FO is, zoals we zelf ondervonden hebben, ook niet altijd makkelijk in het gebruik. De tools die de omzetting van XSL-FO document naar een ander medium doen, bieden ofwel een zeer beperkte ondersteuning met de nodige bugs ofwel een vrij volledige ondersteuning, maar daar betaal je dan ook voor. Nog een nadeel is de compatibiliteit met CSS die wordt nagestreefd. Een voorbeeld van de (semantische) conflicten die zo kunnen onstaan is aan bod gekomen toen we het over kantlijneigenschappen hadden.

8. Gebruik van XSL-FO in onze tekst

Zoals we in [Hoofdstuk 6.] het gebruik van XSLT transformaties in onze tekst hebben besproken, zullen we in dit hoofdstuk het gebruik van XSL-FO [Hoofdstuk 7.] in het kader van onze thesistekst bespreken. Zoals in het hoofdstuk over XSL-FO reeds aangehaald passen wij transformaties toe op onze tekst om ons XML bronbestand om te vormen tot FO bestand. Het is dit FO bestand dat vervolgens omgezet wordt naar PDF of PostScript. Merk op dat het FO bestand op zich meestal niet bestaat bij het genereren van een PDF bestand. Het is slechts een tussenvorm die alleen in het geheugen bestaat. Wij vinden het echter makkelijker om over het FO bestand te spreken dan over de stroom van SAX events of de in het geheugen opgebouwde boom (afhankelijk van de implementatie).

In dit hoofdstuk gaan we kort toelichten hoe we een inhoudstafel genereren voor onze tekst. Zoals uit het XML Schema van onze tekst [Hoofdstuk 4.] blijkt, hebben we in ons XML formaat geen voorzieningen getroffen voor een inhoudstafel. Dit is helemaal niet nodig, aangezien deze inhoudstafel automatisch gegenereerd wordt.

Voor het genereren van het PDF bestand van onze tekst, uitgaande van het FO bestand, hebben we de tool FOP [FOP] gebruikt. Omdat deze tool nogal eens vaak last krijgt van **java.lang.OutOfMemoryError**s, hebben we onze stylesheets aangepast om dit te vermijden. We zullen éé van deze aanpassingen bespreken en daarna bespreken we ook nog enkele tekortkomingen van FOP.

Voor deze bespreking gebruiken wij [FOP], [FOPUSML] en onze eigen ervaringen als bron.

8.1. Genereren van de inhoudstafel

Zoals reeds meerdere malen aangehaald hebben, laten we onze inhoudstafel automatisch genereren. Om dit te realiseren maken we gebruik van het feit dat je een template in meerdere modes [Sectie 5.3.1.6.] kan gebruiken. We overlopen eerst de juiste templates in de mode **TOC** (Table Of Contents). Hiermee bouwen we onze inhoudstafel op. Nadat we onze inhoudstafel hebben opgebouwd, passen we de templates zonder mode toe om de eigenlijk inhoud te genereren.

Om onze inhoudstafel te genereren maken we gebruik van de volgende templates:

Voorbeeld 8.1. Templates voor inhoudstafel

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
     <xsl:output indent="yes"/>
     <xsl:template match="THESIS">
        <fo:root xsmlns:fo="http://www.w3.org/1999/XSL/Format">
           <fo:layout-master-set>
              <fo:simple-page-master master-name="toc" page-height="29.7"
              page-width="21cm" margin-top="2cm" margin-bottom="2cm"
              margin-left="2.5cm" margin-right="2.5cm">
                 <fo:region-body margin-top="1cm" margin-bottom="1cm"/>
                 <fo:region-before extent="1cm"/>
                 <fo:region-after extent="1cm"/>
              </fo:simple-page-master>
           </fo:layout-master-set>
           <fo:page-sequence master-reference="toc" initial-page-number="1"
           format="i">
```

```
            <fo:static-content flow-name="xsl-region-before">
               <fo:block font-size="10pt" font-family="serif"
               text-align="end"> Inhoudstafel </fo:block>
            </fo:static-content>
            <fo:static-content flow-name="xsl-region-after">
               <fo:block font-size="10pt" font-family="serif"
               text-align="end">
                  <fo:page-number/>
               </fo:block>
            </fo:static-content>
            <fo:flow flow-name="xsl-region-body">
               <fo:block break-before="page">
                  <xsl:apply-templates select="CHAPTER" mode="TOC"/>
               </fo:block>
               <fo:block break-before="page">
                  <xsl:apply-templates select="APPENDIX" mode="TOC"/>
               </fo:block>
            </fo:flow>
         </fo:page-sequence>
      </fo:root>
   </xsl:template>
   ...

</xsl:stylesheet>
```

In de template voor het root element van onze tekst definiëren we de **page-master** die we gaan gebruiken voor de inhoudstafel. We gebruiken deze page-master in de **page-sequence** voor onze inhoudstafel. Om onze inhoudstafel te nummeren gebruiken we kleine Romeinse cijfers (i, ii, ...) en we willen dat er genummerd wordt vanaf 1.

We plaatsen de tekst **Inhoudstafel** bovenaan rechts van elke bladzijde van de inhoudstafel. Onderaan rechts van elke bladzijde plaatsen we het paginanummer van de bladzijde in de inhoudstafel. De eigenlijke inhoudstafel komt in de regio **xsl-region-body** en deze inhoud kan stromen (**fo:flow**) over meerdere bladzijden. We geven ook nog aan dat zowel de inhoudstafel van de hoofdstukken als die van de appendices op een nieuw blad moet beginnen.

De templates die de titels van de hoofdstukken, appendices en secties in de inhoud plaatsen zien er als volgt uit:

Voorbeeld 8.2.

```
   <xsl:template match="CHAPTER" mode="TOC">
      <fo:block font-size="20pt" font-family="serif" line-height="24pt"
      space-before="4pt" space-after="4pt">
         <xsl:number level="multiple" count="CHAPTER" format="1. "/>
         <xsl:value-of select="TITLE"/>
         <fo:leader leader-pattern="dots"/>
         <fo:page-number-citation ref-id="{generate-id()}"/>
         <fo:block text-align="end">
            <xsl:apply-templates select="SECTION" mode="TOC"/>
         </fo:block>
      </fo:block>
   </xsl:template>
   <xsl:template match="CHAPTER//SECTION" mode="TOC">
      <fo:block font-size="12pt" font-family="serif" line-height="14pt"
      space-before="1pt" space-after="1pt">
         <xsl:number level="multiple" count="CHAPTER|SECTION"
         format="1.1.1. "/>
         <xsl:value-of select="TITLE"/>
         <fo:leader leader-pattern="dots"/>
         <fo:page-number-citation ref-id="{generate-id()}"/>
         <xsl:apply-templates select="SECTION" mode="TOC"/>
      </fo:block>
```

```
</xsl:template>
<xsl:template match="APPENDIX" mode="TOC">
   <fo:block font-size="20pt" font-family="serif" line-height="24pt"
   space-before="4pt" space-after="4pt">
      <xsl:text> Appendix  </xsl:text>
      <xsl:number level="multiple" count="APPENDIX" format="A.1. "/>
      <xsl:value-of select="TITLE"/>
      <fo:leader leader-pattern="dots"/>
      <fo:page-number-citation ref-id="{generate-id()}"/>
      <fo:block text-align="end">
         <xsl:apply-templates select="SECTION" mode="TOC"/>
      </fo:block>
   </fo:block>
</xsl:template>
<xsl:template match="APPENDIX//SECTION" mode="TOC">
   <fo:block font-size="12pt" font-family="serif" line-height="14pt"
   space-before="1pt" space-after="1pt">
      <xsl:number level="multiple" count="APPENDIX|SECTION"
      format="A.1.1.1. "/>
      <xsl:value-of select="TITLE"/>
      <fo:leader leader-pattern="dots"/>
      <fo:page-number-citation ref-id="{generate-id()}"/>
      <xsl:apply-templates select="SECTION" mode="TOC"/>
   </fo:block>
</xsl:template>
```

Deze templates maken gebruik van het **fo:page-number-citation** element dat verantwoordelijk is voor het plaatsen van paginanummer. Het paginanummer dat geplaatst wordt is het nummer van de pagina waarop het element begint dat als waarde van het **id** attribuut dezelfde waarde heeft als het **ref-id** attribuut van het **fo:page-number-citation** element.

We geven hier de template voor de **CHAPTER** elementen in de normale mode:

Voorbeeld 8.3. CHAPTER-template in normale mode

```
<xsl:template match="CHAPTER">
   <fo:block font-size="20pt" font-family="serif" line-height="40pt"
   space-after="14pt" id="{generate-id()}">
      <xsl:number level="multiple" count="CHAPTER" format="1. "/>
      <xsl:text>   </xsl:text>
      <xsl:value-of select="TITLE"/>
      <xsl:apply-templates select="SUMMARY | SECTION"/>
   </fo:block>
</xsl:template>
```

Het **fo:block** element dat ingevoegd wordt door deze template krijgt een unieke identifier mee in het **id** attribuut. Het is naar deze waarde dat verwezen wordt door middel van het **ref-id** attribuut bij het **fo:page-number-citation** element. Op basis van de pagina in de paginasequentie waar het **fo:block** element terechtkomt, kan een paginanummer ingevoegd worden in de inhoudstafel.

Deze templates zorgen voor een mooie inhoudstafel en in casu ook voor de inhoudstafel van deze thesis. Dat is het beste praktische voorbeeld dat we kunnen geven. Deze templates genereren een tussenvorm die voor ons door FOP omgezet naar een PDF-bestand.

8.2. Gebruik van page-sequences

Voor de inhoud maakten we in eerste instantie gebruik van één **fo:page-sequence** element dat

één **fo:flow** element bevat. Binnen dit **fo:flow** plaatsten we dan de containers (**fo:block**) met de inhoud. Dit **fo:page-sequence** had de volgende vorm:

Voorbeeld 8.4. fo:page-sequence voor de inhoud

```
...
 <fo:page-sequence master-reference="toc" initial-page-number="1">
   <fo:static-content flow-name="xsl-region-after">
     <fo:block font-size="10pt" font-family="serif" text-align="end">
     p.
       <fo:page-number/>
     </fo:block>
   </fo:static-content>
   <fo:flow flow-name="xsl-region-body">
     <xsl:apply-templates select="CHAPTER"/>
     <xsl:apply-templates select="APPENDIX"/>
     <xsl:apply-templates select="BIBLIOGRAPHY"/>
     <xsl:apply-templates select="READINGS"/>
   </fo:flow>
 </fo:page-sequence>
...
```

De templates voor de verschillende onderdelen zijn verantwoordelijk voor het plaatsen van de **fo:block** elementen die de inhoud bevatten. De inhoud mag niet rechtstreeks in een **fo:flow** element geplaatst worden.

Het probleem met FOP is echter dat het **fo:page-sequence** element, net als de kindelementen, als Java objecten in het geheugen worden bijgehouden totdat het element wordt afgesloten. Pas indien het **fo:page-sequence** element wordt afgesloten worden de Java objecten omgezet naar PDF. Je voelt het probleem al komen: omdat elk element als een Java object wordt bijgehouden totdat de omzetting naar PDF kan gebeuren, zal het wel (meer dan) eens voorvallen dat het beschikbare geheugen opgebruikt is en de bewerking moet worden afgebroken.

De oplossing die voor FOP wordt voorgesteld is het opbreken van één groot **fo:page-sequence** element in meerdere kleine **fo:page-sequence** elementen. Met een groot element bedoelen we hier een element met veel inhoud. Hierdoor kunnen de objecten die overeenkomen met de **fo:page-sequence** elementen veel sneller omgezet worden in PDF. Dit zorgt ook voor een lagere geheugenbelasting. Het **fo:page-sequence** element staat niet meer, zoals in het bovenstaande voorbeeld, rond de verschillende **xsl:apply-templates** elementen. In plaats daarvan krijgen we de volgende situatie:

Voorbeeld 8.5. Gewijzigde stylesheet

```
...
 <xsl:apply-templates select="CHAPTER"/>
 <xsl:apply-templates select="APPENDIX"/>
 <xsl:apply-templates select="BIBLIOGRAPHY"/>
 <xsl:apply-templates select="READINGS"/>
...
 <xsl:template match="CHAPTER">
   <fo:page-sequence master-reference="toc">
     <xsl:if test="not(preceding-sibling::CHAPTER)">
       <xsl:attribute initial-page-number="1"> 1 </xsl:attribute>
     </xsl:if>
     <fo:static-content flow-name="xsl-region-after">
       <fo:block font-size="10pt" font-family="serif" text-align="end">
```

```
          p.
             <fo:page-number/>
          </fo:block>
       </fo:static-content>
       <fo:flow flow-name="xsl-region-body">
          <fo:block font-size="20pt" font-family="serif"
          line-height="40pt" space-after="14pt" id="{generate-id()}">
             <xsl:number level="multiple" count="CHAPTER" format="1. "/>
             <xsl:text>   </xsl:text>
             <xsl:value-of select="TITLE"/>
             <xsl:apply-templates select="SUMMARY | SECTION"/>
          </fo:block>
       </fo:flow>
    </fo:page-sequence>
 </xsl:template>
...
```

We hebben hier alleen de template voor het element **CHAPTER** gegeven, dit omdat de andere templates vrij analoog zijn. Zoals eerder gezegd krijgen we nu **fo:page-sequence** elementen die kleiner zijn en eerder naar PDF omgezet kunnen worden. Praktische testen leerden ons dat de geheugeneisen inderdaad minder waren door deze aanpassingen. We hebben deze testen uitgevoerd zonder voorwaartse referenties [Sectie 8.3.] omdat deze referenties een negatieve invloed hebben op onze conclusies.

Het enige waar we totaal niet gelukkig mee zijn is het feit dat we nu een **if** test moeten uitvoeren om te weten wanneer (bij het eerste hoofdstuk) de nummering van de bladzijden vanaf 1 moeten laten beginnen. De enige reden dat we dit zo moesten doen is dus omwille van FOP.

8.3. Tekortkomingen van FOP

Zoals reeds enkele malen aangehaald maken wij gebruik van de gratis open source tool **FOP** . Deze tool is nog in volle ontwikkeling en draagt op dit moment als versienummer **0.20.3** wat er op duidt dat er nog veel werk aan de winkel is. Op dit moment concentreren de inspanningen van de ontwikkelaars zich vooral op het herwerken van de interne architectuur om een aantal praktische problemen op te lossen. We gaan hier een aantal van deze praktische problemen bespreken, die wij zelf zijn tegengekomen.

Zoals in [Sectie 8.2.] aangehaald levert het gebruik een groot **fo:page-sequence** element problemen op. Dit moest opgelost worden dit dit element op te splitsen. De ontwikkelaars proberen dit probleem op te lossen door pagina's zo snel mogelijk te renderen in plaats van ze bij te houden als Java objecten. Indien er nog referenties aanwezig, zoals de verwijzingen naar de paginanummers in een inhoudstafel, dan kan dit naderhand nog opgelost worden. We hopen dat de ontwikkelaars in hun opzet slagen, maar voorlopig moeten we dus nog een work-around toepassen.

FOP heeft ook nog een probleem met voorwaartse referenties. Een voorbeeld van een voorwaartse referentie is het opnemen van de paginanummers in de inhoudstafel zoals wij dat doen. In de inhoudstafel verwijzen wij naar elementen die nog ergens in een paginasequentie moeten opgenomen worden. Het probleem hiermee is dat FOP de pagina's die deze referenties bevatten (en wij vermoeden ook de volgende pagina's) als Java objecten in het geheugen houdt totdat alle referenties geresolved zijn. Er wordt aangeraden om de paginasequenties om te wisselen zodat de inhoudstafel op het einde komt en niet in het begin. We hebben dit geprobeerd, maar dit leverde een totaal verknoeide lay-out op.

Sommige eigenschappen waren ook op een andere manier geïmplementeerd dan wij verwachten. We halen hier het voorbeeld van de eigenschap **start-indent** aan. Volgens [TRXSLSI] geldt voor **start-indent** het volgende: *"For each block-area generated by this formatting object, specifies the distance from the start-edge of the content-rectangle of the containing reference-area to the start-edge of the content-rectangle of that block-area. "* Wij geven aan deze uitleg de volgende interpretatie. Stel dat we twee **fo:block** elementen hebben die allebei de eigenschap **start-indent** specificeren met een waarde van **1cm**. Ook veronderstellen we dat het een **fo:block** element een kindelement is van het andere **fo:block** element. Onze verwachting was dat het omvattende **fo:block** element een indentatie zou hebben van één cm. Dit was het geval. Maar we hadden ook verwacht dat het **fo:block** twee cm zou zijn ingesprongen, dit is één cm ten opzichte van zijn ouderelement. Dit bleek niet het geval te zijn. We hebben dit opgelost door dit in de broncode aan te passen. Omdat we niet de tijd hadden om ook nog eens de broncode van FOP uitgebreid te bestuderen is dit een "hack" in de broncode in plaats van elegante code.

Daarnaast hadden we ook een aantal problemen met het invoegen van afbeeldingen. We hebben overal locaties gebruikt relatief ten opzichte van ons hoofddocument. Om een ons onbekende reden werkte dit niet. De oplossing hiervoor was om met absolute locaties te werken. De absolute locatie staat slechts op één plaats, namelijk in de stylesheet die zorgt voor de transformatie van de **IMAGE** elementen. Moest de absolute locatie veranderen dan moet er slechts op één plaats iets veranderd worden en zal het dan terug werken.

Daarna hadden we het probleem dat onze afbeeldingen steeds uitgerekt werden zodat ze de volledige breedte van het blad in beslag namen. De oplossing hiervoor was dat we ook de breedte en/of hoogte moesten specificeren in onze XML bestanden. We vonden het spijtig dat dit de enige oplossing was, want op die manier voerden we toch nog lay-out informatie toe aan ons XML bronbestand.

Ook zijn er een aantal, in onze ogen, belangrijke eigenschappen nog niet geïmplementeerd. Zo is het meer dan eens gebleken dat de laatste regel van een paragraaf op een ander blad viel. We hebben wel gespecificeerd met behulp van de **orphans** en **widows** eigenschappen [Sectie 7.15.6.9.] dat er minstens drie regels samen genomen moeten worden, maar dat is nog niet geïmplementeerd. Ook andere eigenschappen zoals **keep-together** zijn niet voorzien. Het zou dus goed kunnen dat de titel van een sectie op een bepaalde bladzijde staat terwijl de inhoud pas op de volgende bladzijde begint. Ook dit ligt aan FOP en niet aan de door ons geschreven stylesheets.

Daarnaast zijn er nog een aantal kleinere ergernissen die echter met de overstap van versie **0.20.2** naar versie **0.20.3** opgelost zijn. We denken hierbij aan gebieden die elkaar overlapten in de uitvoer, terwijl dit totaal niet mocht. Daarnaast wordt er af en toe ook overtollige witruimte ingevoerd, zoals de aandachtige lezers misschien al gemerkt zullen hebben bij het bestuderen van de voorbeeldjes.

8.4. Besluit

Met de combinatie van XSLT en XSL-FO is het mogelijk om een XML document te transformeren naar de beschrijving van een mooi uitziend geheel. Deze beschrijving (het FO bestand) kan dan met tools omgezet worden naar bestandsformaten zoals PDF en PostScript. Wij hebben gebruik gemaakt van FOP [FOP], omdat dit een open source tool is en omdat FOP, volgens onze zoektochten, van alle gratis tools, ondanks het ontbreken van de implementatie van een aantal eigenschappen, het meest gevorderd is. Er zijn wel commerciële tools beschikbaar, zoals XEP van RenderX [RX], maar daar wordt de prijs niet vermeld op de website. Een andere tool, de XSL Formatter van Antennahouse [APFSXSLF] met een optie

voor PDF uitvoer kost tussen de **$2000** en **$4500**, afhankelijk van de opties en het aantal licenties. We hebben van deze laatste tool een evaluatieversie geprobeerd, maar deze evaluatieversie kreeg ons FO bestand niet verwerkt, terwijl FOP daar wel in slaagde.

Het nadeel aan XSL-FO is dat de instapdrempel nogal hoog ligt. Om op een goede manier te kunnen werken met de XSL-FO woordenschat moet je eerst een hele hoop terminilogie onder de knie krijgen, zoals ook uit het eerste gedeelte van het hoofdstuk over XSL-FO [Hoofdstuk 7.] blijkt. We zijn er wel van overtuigd dat er een toekomst is weggelegd voor XSL-FO, maar dan zou de instapdrempel wel wat lager mogen liggen. We denken bijvoorbeeld aan programma's waarin je de pagina-lay-out eenvoudig op een grafische manier kan specificeren en dat dit programma dan de taak van het creëeren van de juiste templates op zich neemt.

9. Cocoon XML publishing framework

In dit hoofdstuk besturen we de beide versie van Cocoon. We hebben eerst wat praktijkervaring opgedaan met Cocoon 1, maar we zijn daarna overgeschakeld op Cocoon 2. Zoals uit dit hoofdstuk moet blijken heeft Cocoon 2 een aantal voordelen op Cocoon 1. Enerzijds komt dit door een totaal andere architectuur en anderzijds door het feit dat Cocoon 1 ontstaan is als proof-of-concept. Uiteindelijk kwamen hierdoor een aantal ontwerpbeperkingen tevoorschijn. Die heeft men trachten op te lossen door met een totaal nieuw ontwerp te beginnen en dit leidde tot de ontwikkeling van Cocoon 2.

Voor de volledigheid willen we ook nog vermelden dat de ontwikkeling van Cocoon 1 inmiddels is stopgezet. Versie 1.8.2 is de laatste versie van Cocoon 1.

9.1. Cocoon

9.1.1. Wat is Cocoon?

Cocoon ([COCOON1]) maakt deel uit van de XML projecten van de Apache Software Foundation ([APXMLP]), bekend als makers van de gelijknamige webserver. Om de makers te citeren: *"Cocoon is a 100% pure Java publishing framework that relies on new W3C technologies (such as DOM, XML, and XSL) to provide web content."* Het doel van Cocoon is webdocumenten op een radicaal andere manier dan de klassieke web publishing oplossingen aan te bieden, zonder dat de gebruiker eigenlijk enig verschil merkt, en dit vanuit de volgende filosofie: *"The new Cocoon paradigm is based on the fact that document content, style and logic are often created by different individuals or working groups."* Het doel van Cocoon is om deze 3 lagen te scheiden, net zoals het een (van de) doel(en) is van XML om inhoud en opmaak te scheiden. Met behulp van de aangehaalde technieken zoals XML en XSL(T) is dat inderdaad mogelijk. De inhoud en opmaak kunnen in 2 verschillende XML-documenten ondergebracht worden, en vervolgens kan je XSL transformaties schrijven om deze documenten samen te voegen en bijvoorbeeld HTML uitvoer te genereren. Maar er zijn ook vele andere uitvoermogelijkheden, zoals PDF, SVG, ...

9.1.2. Werken met Cocoon

9.1.2.1. Cocoon installeren

Voor een beschrijving van de installatie verwijzen wij naar [Appendix C.] .

Hoe het installeren van Cocoon verloopt is niet belangrijk voor het verdere verloop van de tekst. Wel belangrijk om te weten is dat Cocoon geïntegreerd kan worden met elke webserver die servlets (overeenkomstig de Servlet 2.x specificatie) ondersteunt. Op de installatie-pagina van Cocoon 1 ([C1INST]) zijn instructies te vinden om Cocoon 1 te integreren met het gros van de webservers die momenteel op de markt zijn. Na het (meestal) noodzakelijke herstarten van de webserver, is Cocoon 1 beschikbaar om XML inhoud te verwerken op aanvraag van een bezoeker. Typisch zijn dit bestanden met de extensie **xml**, maar een vereiste is dit niet: het maakt het gewoon eenvoudiger om deze bestanden te herkennen. Het is wel zo dat in het configuratiebestand van de webserver duidelijk gemaakt moet worden welke bestanden door Cocoon 1 behandeld moeten worden. Eens alles juist geconfigureerd en geïnitialiseerd is, kan er begonnen worden met het aanbieden van XML bestanden, die ofwel letterlijk ofwel verwerkt tot een ander formaat naar de bezoeker gestuurd worden.

9.1.2.2. Werking van Cocoon nader toegelicht

Eens Cocoon correct geïnstalleerd is, kunnen XML documenten aangeboden worden, op voorwaarde natuurlijk dat ze op een locatie staan die door de webserver gekend is en door de webserver mag aangeboden worden. Om een voorbeeldje te geven hoe zo een XML document er uit kan zien geven wij hier een klein voorbeeldje:

Voorbeeld 9.1. hello.xml

```
<?xml version="1.0"?>
<?xml-stylesheet href="hello-html.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
 <page>
    <title> Hello </title>
    <content>
       <paragraph> This is my first Cocoon page! </paragraph>
    </content>
 </page>
```

Dit voorbeeldje bevat een aantal processing instructions die zeggen wat er door Cocoon 1 gedaan moet worden met dit document. De eerste processing instruction zegt gewoon met welke versie van XML we te maken hebben. Het volgende waar Cocoon 1 op gaat letten zijn processing instructions van het type **cocoon-process**. Alhoewel er in dit document slechts één processing instruction staat, mogen er in de praktijk meerdere **cocoon-process** instructies staan, die Cocoon 1 allemaal in een bepaalde volgorde zal afhandelen. In dit geval wordt er tegen Cocoon 1 gezegd dat er op dit document een XSLT transformatie moet toegepast worden met behulp van het attribuut **type** (**type**="xslt"). In welk bestand deze transformaties zich bevinden is terug te vinden via de processing instruction **xml-stylesheet** ([TRXMLSS]). Voor dit XML document wil de auteur dat de instructies in het bestand **hello-html.xsl** toegepast worden om de uitvoer te genereren die naar de bezoeker moet gestuurd worden. Laten we dit xsl bestand ook even bekijken om te zien of daar ook Cocoon-specifieke instructies moeten in opgenomen worden.

Voorbeeld 9.2. hello-html.xsl

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="page">
       <xsl:processing-instruction name="cocoon-format"> type="text/html"
       </xsl:processing-instruction>
       <html>
          <head>
             <title>
                <xsl-value-of select="title"/>
             </title>
          </head>
          <body bgcolor="#ffffff">
             <xsl:apply-templates/>
          </body>
       </html>
    </xsl:template>
    <xsl:template match="title">
       <h1 align="center">
          <xsl:apply-templates/>
       </h1>
    </xsl:template>
    <xsl:template match="paragraph">
       <p align="center">
```

```
          <i>
            <xsl:apply-templates/>
          </i>
        </p>
      </xsl:template>
    </xsl:stylesheet>
```

In dit bestand zien we de XML declaratie, en zoals in elk XML document is er één hoofdelement, in casu **xsl:stylesheet**. Ook herkennen we hier de verschillende templates die dienen om het XML document om te zetten naar (in dit geval) een HTML document. Wel belangrijk om op te merken, is dat er in de template voor het hoofdelement van het XML document, met name **page**, als eerste een XSLT instructie staat die ervoor moet zorgen dat er een processing instruction met de naam **cocoon-format** en als bijkomende inhoud **type="text/html"** tussen de processing instruction tekens moet geplaatst worden. Om nuttig te zijn voor Cocoon moet deze processing instruction ofwel doorgestuurd worden voordat er ook maar iets anders dat deel uitmaakt van de uiteindelijke uitvoer doorgestuurd wordt, ofwel moet deze instructie niet vermeld worden. Dit verdient wat verdere uitleg.

9.1.2.2.1. Uitleg over de processing instruction cocoon-format

Aandachtige lezers zullen zich misschien afvragen waarom er niet van de XSLT instructie **xsl:output** [Sectie 5.3.1.15.1.] gebruikt gemaakt wordt, in plaats van de processing instruction **cocoon-format**. Het antwoord hierop is terug te vinden in [C1FAQXO]. Het antwoord luidt: *" The Cocoon project doesn't implement the xsl:output feature for XSLT because we believe it breaks the separation of concerns and doesn't match the internal Cocoon architecture. "* Waarom dit het **separation of concerns** principe zou schaden weten wij niet. Cocoon doet immers beroep op Cocoon-specifieke processing instructions in de XML documenten en in de XSLT bestanden en dat is in onze ogen toch ook een serieuze inbreuk op dit principe. We vinden dat gedeelte van de uitleg geen goede motivering. Over de interne architectuur van Cocoon kunnen wij ons niet uitspreken, maar we vinden het straf dat bepaalde delen van een standaard niet geïmplementeerd zouden zijn alleen maar omwille van het feit dat het niet strookt met de interne architectuur van de applicatie. Dit is ons standpunt hieromtrent.

Cocoon steunt op de processing instruction **cocoon-format** om te bepalen wat het uitvoerformaat is. Zoals hierboven gezegd, moet deze processing instruction niet vermeld worden; in dit geval gebruikt Cocoon de standaard waarde die gedefinieerd is in het configuratiebestand van Cocoon. In dit configuratiebestand zijn ook de andere mogelijke waarden gedefinieerd die het attribuut **type** bij deze processing instruction kan hebben. Afhankelijk van de waarde van dit attribuut stuurt Cocoon een DOCTYPE definitie mee, of een vermelding van het MIME-type van het document. Dit is ook allemaal gedefinieerd in het configuratiebestand. We geven hier een ingekorte versie van het betreffende gedeelte van het configuratiebestand.

```
########################################
# XML Formatters                       #
########################################

# This is used when no <?cocoon?> PI is present in the XSL to indicate
# which MIME type to associate to the document.
# NOTE: this type must be present in the map below.
formatter.default = text/html

# These are used when the <?cocoon-format type="xxx/yyy"?> PI is present
# The syntax for this is
#   formatter.type.xxx/yyy = full.class.name

# Full configurable formatters
```

```
################################

formatter.type.text/html        = org.apache.cocoon.formatter.HTMLFormatter
formatter.type.text/html/loose  = org.apache.cocoon.formatter.HTMLFormatter
formatter.type.text/xhtml       = org.apache.cocoon.formatter.XHTMLFormatter
formatter.type.text/xslfo       = org.apache.cocoon.formatter.FO2PDFFormatter

# You can modify the formatter's behavior by adding the following configurations
# for each formatter you want to specify. Note that even if two formatters
# share the same class, they will be seen as different entities, accessed
# only by their types.
#
#   formatter.[type].MIME-type      = [formatter MIME type]
#   formatter.[type].encoding       = [encoding type]
#   formatter.[type].doctype-public = [public identifier]
#   formatter.[type].doctype-system = [system identifier]
#   formatter.[type].preserve-space = [whether to preserve space or not]
#   formatter.[type].line-width     = [page width, wrapping column]
#   formatter.[type].indent         = [number of spaces for tag indenting]

# HTML 4.0 (strict)
formatter.text/html.doctype-public = -//W3C//DTD HTML 4.0//EN
formatter.text/html.doctype-system = http://www.w3.org/TR/REC-html40/strict.dtd

# XHTML 1.0 (strict)
formatter.text/xhtml.doctype-public = -//W3C//DTD XHTML 1.0 Strict//EN
formatter.text/xhtml.doctype-system = xhtml1-strict.dtd

# PDF
formatter.text/xslfo.MIME-type = application/pdf

# HTML 4.0 (transitional)
formatter.text/html/loose.doctype-public = -//W3C//DTD HTML 4.0 Transitional//EN
formatter.text/html/loose.doctype-system =
http://www.w3.org/TR/REC-html40/loose.dtd
formatter.text/html/loose.preserve-space = true
formatter.text/html/loose.encoding = UTF-8
formatter.text/html/loose.indent = 1
formatter.text/html/loose.line-width = 120
formatter.text/html/loose.MIME-type = text/html
```

Nu is het zo dat wij geschreven hebben dat deze processing instruction (cocoon-format) als eerste aan de uitvoerstroom moet verschijnen, maar wat gebeurt er als dit niet zo is? Geeft Cocoon dan een foutmelding of hoe wordt dit afgehandeld?

9.1.2.2.1.1. Locatie van de verwerkingsinstructies

Indien deze processing instruction door Cocoon moet verwerkt worden, moet deze als eerste in de uitvoerstroom verschijnen. Gebeurt dit niet, dan zal Cocoon deze processing instruction gewoon negeren. Deze processing instruction zal dan deel uitmaken van het uiteindelijke document zoals dit afgeleverd wordt aan de client. Dit is meestal een ongewenst resultaat, maar waarom dit dan door Cocoon niet verwerkt wordt, hebben we op dit moment nog niet helemaal doorgrond.

Dat dit ongewenst kan zijn, willen we aantonen met het volgende voorbeeldje. We geven hier als voorbeeld een HTML document dat gegenereerd is met behulp van Cocoon, maar waar we de processing instruction in het xsl bestand tussen het afsluiten van het **head** element en het begin van het **body** element hebben geplaatst. De gegenereerde uitvoer zag er als volgt uit:

Voorbeeld 9.3.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html40/strict.dtd">
```

```
  <html>
    <head>
      <title> Hello </title>
    </head>
    <?cocoon-format type="text/html"?>
    <body>
      <h1 align="center"> Hello </h1>
      <p align="center">
        <i> This is my first Cocoon page! </i>
      </p>
    </body>
  </html>
<!--This page was served in 58 milliseconds by Cocoon 1.8.2 -->
```

Hier is dat gelukkig nog niet zo erg, omdat we met een tekstformaat te maken hebben en (bijna) alle browsers de elementen die ze niet kennen, negeren. Erger is het als deze tekst midden in de uitvoer van een ander formaat zoals PDF verschijnt. Het feit dat deze extra tekst in het uiteindelijk PDF bestand verschijnt, kan genoeg zijn om ervoor te zorgen dat het PDF bestand door geen enkele PDF viewer correct kan geopend worden. Het is duidelijk dat deze situatie ongewenst is.

9.1.2.3. Geavanceerde mogelijkheden

9.1.2.3.1. Output op basis van een browser agent

Vermits de meeste browsers die gebruikt worden de dag van vandaag nog niet overweg kunnen met het verwerken van XML/XSLT, is het meestal aangewezen om een HTML-bestand te genereren en dit naar de client door te sturen. Maar het is algemeen geweten dat de meeste browsers het tonen van de HTML-primitieven nogal eens op een totaal andere manier interpreteren/implementeren. In plaats van één groot HTML-bestand te genereren waarin met alles rekening gehouden wordt, of gewoon een minimale HTML-pagina te genereren of zich slechts op één browser-versie te richten, is het mogelijk om een XSLT stylesheet te laten toepassen op de XML-bron op basis van de browser agent van de bezoeker van de site.

Cocoon biedt dus de mogelijkheid om de uitvoer aan te passen aan de browser waarmee de site bezocht wordt. Hoe gaan we hiervoor in Cocoon nu te werk? Zoals eerder aangehaald wordt bij Cocoon 1 de toe te passen XSLT stylesheet gespecificeerd in het XML bronbestand. De processing instruction heeft daarvoor twee attributen, href en type. Om aan te geven welke stylesheet gebruikt moet worden, komt er een derde attribuut, **media**, bij. Op die manier krijgen we de volgende regels aan het begin van een XML bronbestand:

Voorbeeld 9.4.

```
<?xml version="1.0"?>
<?xml-stylesheet href="hello.xsl" type="text/xsl"?>
<?xml-stylesheet href="hello-text.xsl" type="text/xsl" media="lynx"?>
<?xml-stylesheet href="hello-msie.xsl" type="text/xsl" media="explorer"?>
```

Standaard worden de regels uit hello.xsl toegepast op dit XML bronbestand. Wordt de site echter bezocht met lynx, dan past Cocoon hello-text.xsl toe en indien de site bezocht wordt met explorer, dan wordt hello-msie.xsl toegepast. Maar hoe weet Cocoon nu als er bij media de waarde 'lynx' of 'explorer' staat, met welke User Agent (UA) dit precies overeenkomt? Dat is allemaal gespecificeerd in het configuratiebestand.

De waarden die opgegeven zijn voor het attribuut media (hier lynx en explorer) worden gedefinieerd (geassocieerd) met een bepaalde stringwaarde uit de UA-string. Dit gebeurt in het configuratiebestand van Cocoon. Hierin wordt een alias gedefinieerd voor een bepaalde set van UA's, en die alias komt overeen met een bepaalde string-waarde. Cocoon controleert of die string-waarde voorkomt in de "user-Agent" HTTP header, en weet op basis daarvan met welke alias rekening dient gehouden te worden. Het configuratiegedeelte specifiek voor deze problematiek ziet er als volgt uit:

```
########################################
# User Agents (Browsers)               #
########################################

# NOTE: numbers indicate the search order. This is VERY VERY IMPORTANT since
# some words may be found in more than one browser description. (MSIE is
# presented as "Mozilla/4.0 (Compatible; MSIE 4.01; ...")
#
# For example, the "explorer=MSIE" tag indicates that the XSL stylesheet
# associated to the media type "explorer" should be mapped to those browsers
# that have the string "MSIE" in their "user-Agent" HTTP header.

browser.0 = explorer=MSIE
browser.1 = pocketexplorer=MSPIE
browser.2 = handweb=HandHTTP
browser.3 = avantgo=AvantGo
browser.4 = imode=DoCoMo
browser.5 = opera=Opera
browser.6 = lynx=Lynx
browser.7 = java=Java
browser.8 = wap=Nokia
browser.9 = wap=UP
browser.10 = wap=Wapalizer
browser.11 = mozilla5=Mozilla/5
browser.12 = mozilla5=Netscape6/
browser.13 = netscape=Mozilla
```

Indien er geen standaard stylesheet voorzien is om toe te passen en de pagina wordt met een onbekende browser (onbekende UA-string) bezocht, dan genereert Cocoon de foutmelding hieronder weergegeven. Er is in dit geval geen mechanisme voorzien om de bezoeker (een gedeelte van) de inhoud te tonen.

```
 Cocoon 1.8.2

                                                    Error found handling the
request.

org.apache.cocoon.processor.ProcessorException: Could not associate stylesheet to
document:  no matching stylesheet for: lynx
        at
        at
        at
        at
        at
        at
        at java.lang.Thread.run(Thread.java:475)

   Warning: this page has been dynamically generated.

                                              Copyright (c) 1999-2001 The Apache XML
Project.
                                                          All rights reserved.
```

9.1.2.3.2. Dynamisch gegenereerde inhoud

Een website publiceren is heel beperkt als je niet de mogelijkheid hebt om inhoud dynamisch te genereren op basis van bepaalde parameters. Dit is een mogelijkheid die wij niet zelf

hebben onderzocht, maar we zullen hier een korte beschrijving geven van de mogelijkheden, zonder er evenwel diep op in te gaan. De reden hiervoor is dat we dit in de praktijk niet hebben aangewend, maar dit toch willen meegeven om aan te tonen dat er met Cocoon meer mogelijk is.

In Cocoon is het mogelijk om dynamisch XML te genereren. De belangrijkste manier om dit te doen is via XSP [SMRRWDXSP], [XSPP]. XSP staat voor eXtensible Server Pages en valt te vergelijken met andere scripting-talen zoals PHP [PHP] en ASP [ASP], met dien verstande dat de taal die gebruikt wordt voor het scripten Java is. We zullen hier een voorbeeldje met wat uitleg verschaffen, dat naast nog veel meer info kan teruggevonden worden op [XSPP].

Voorbeeld 9.5. timeofday.xsp

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="sample.xsl" type="text/xsl"?>
 <xsp:page language="java"
 xmlns:xsp="http://www.apache.org/1999/XSP/Core">
    <page title="Time of Day">
      <!--Define a variable to hold the time of day-->
      <xsp:logic> Date now = new Date(); </xsp:logic>
      <p> To the best of my knowledge, it's now
        <!--Substitute time of day here-->
        <xsp:expr> now </xsp:expr>
      </p>
    </page>
 </xsp:page>
```

Aan de processing instructions in dit document is te zien dat er door Cocoon 2 stappen moeten gedaan worden om dit document te verwerken. Ten eerste moeten de XSP-instructies verwerkt worden en daarna moeten de XSLT instructies die in sample.xsl gespecificeerd zijn, verwerkt worden. In dit eenvoudige voorbeeldje wordt een nieuwe variabele **now** aangemaakt, die van het type **Date** is. Dit gebeurt binnen logic-tags die tot de gedefinieerde namespace xsp behoren. Verderop in dit document wordt dan aangegeven dat de waarde van de expressie **now** in de uitvoer moet ingevoerd worden. Dit wordt aangegeven met het **xsp:expr** element. Met nog het daarna toepassen van de XSLT instructies wordt de volgende HTML-pagina bekomen:

Voorbeeld 9.6. timeofday.html

```
<html>
   <head>
      <title> Time of Day </title>
   </head>
   <body>
      <h3 style="color: navy; text-align: center"/>
      <p> To the best of my knowledge, it's now Thu Dec 23 20:11:18 PST
      1999 </p>
   </body>
</html>
```

9.1.2.3.3. Andere mogelijkheden

Naast de hier vermelde mogelijkheden, output op basis van een browser agent en dynamisch gegenereerde inhoud, ondersteunt Cocoon 1 ook nog het gebruik van een SQL en een LDAP

processor (geen van deze twee hebben wij gebruikt of onderzocht), om er enkele te noemen. Ook is het mogelijk om op basis van één brondocument meerdere uitvoerformaten te genereren. Voor elk uitvoerformaat moeten dan wel specifieke XSLT templates gedefinieerd zijn. Een andere mogelijkheid is om Cocoon 1 verder uit te breiden met zelf geschreven componenten. We hebben gewoon een aantal mogelijkheden aangehaald om aan te tonen dat Cocoon 1 veel meer kan dan alleen XML omzetten tot HTML en we hebben hierbij de mogelijkheden besproken die voor ons het interessantst leken.

9.1.3. Conclusie

Cocoon 1 is een framework dat kan gebruikt worden voor publicatie op het web, en maakt daarbij gebruik van enkele recente W3C technologieën, zoals daar zijn XML, XSLT, DOM, ... Alhoewel zo een framework er toch voornamelijk op gericht is om HTML pagina's beschikbaar te maken aan de bezoekers, is het ook mogelijk om de XML bestanden zonder verdere bewerking naar de bezoeker door te sturen. Daarnaast zijn er nog een aantal uitgebreide mogelijkheden, zoals het dynamisch genereren van inhoud, stylesheet selectie op basis van de UA, ...Het is echt een heel uitgebreid framework waar wij tot nu toe slechts een gedeelte van het potentieel hebben bestudeerd, maar toch hebben wij enkele bedenkingen bij dit framework.

Deze bedenkingen slaan dan vooral op sommige inconsistenties betreffende het 'separation of concerns'-principe. Hierbij denken wij aan het vermelden van de te gebruiken stylesheet in het bron-document, maar vooral aan de Cocoon-specifieke processing instructions die in het bron-document moeten opgenomen worden. Dit is vooral storend, gelet op het feit dat de ontwikkelaars ervoor hebben gekozen xsl:output niet te ondersteunen omdat dit het 'separation of concerns'-principe zou schenden.

Dat de te gebruiken stylesheet in het brondocument moet vermeld worden met het relatieve pad ten opzichte van het brondocument heeft tijdens het werken met Cocoon 1 aanleiding gegeven tot heel wat frustraties. Terwijl deze keuze misschien nog te verdedigen valt voor een site met hooguit 2 à 3 pagina's, is dit heel moeilijk werkbaar indien je een site hebt met een tiental pagina's in geneste directories die allemaal met dezelfde stylesheet bewerkt moeten worden. Het kopiëren van de stylesheet in de verschillende directories is geen oplossing vanwege de redundantie en het gebruik van symbolic links om de verwijzing naar de stylesheets eenvoudig te houden is een platformafhankelijke oplossing. Bij het aanmaken van een pagina gaat het verwijzen naar ../../../stylesheets/index.xsl meestal nog goed, maar er wordt gemakkelijk één pagina over het hoofd gezien indien de locatie aangepast moet worden (cfr. verwijzingen naar CSS documenten in HTML-pagina's). Hierbij dachten wij dat het misschien beter zou zijn om een bepaald bestand te voorzien waarin vermeld wordt welke stylesheet op welk brondocument (mogelijkheid van te werken met wildcards, bepaalde kenmerken) moet worden toegepast, en dat dit bestand dan zou gelden voor alle subdirectories (indien het in een bepaalde subdirectory niet overschreven wordt door andere regels). Dit zou een gedeelte van het sitebeheer uit de bronbestanden halen en zo de flexibiliteit van de site ten goede kunnen komen.

Over deze bedenking en de bedenking in verband met de processing instructions verwijzen we naar [C1FAQCP] waar we het volgende terugvinden: *"I think that using Processing Instructions to "chain" document layers somehow violates the context separation since I would like to be able to place style-related information in sessions or request parameters. What do you think about this? "*
You are right, processing instruction reaction breaks the context separation and it is, in the final analysis, the wrong approach. To follow a complete "model, view, controller" design pattern, one should be able to associate a different processing chain for each requested URI

and for every possible request state (with request parameters, session parameters and environment parameters).

The proposed solution (as you can read in the Cocoon2 outline) is to have a site map where site managers decide what processing chain to apply to each possible request. This somehow follows the mod_rewrite model in the Apache Web Server, but rather than URL rewriting, the site map allows site designers to control the behavior of their documents in one place without having to modify every single reactive processing instruction in each source file.

So, you've been warned: the PIs will go away, current functionality will remain but the processing management will be abstracted one layer up.

Voor ons is het duidelijk dat deze bedenkingen bij een groot gedeelte van de Cocoon-gemeenschap aanwezig waren en dat er met deze bedenkingen rekening is gehouden bij het ontwerp van Cocoon 2. Maar dat is dan ook een groot verschil tussen Cocoon 1 en Cocoon 2, nl. het feit dat Cocoon 1 geboren is als een "proof-of-concept", terwijl Cocoon 2 steunt op de ervaringen in verband met Cocoon 1 en een daarmee gepaard gaand weldoordacht ontwerp. Uit het feit dat Cocoon 1 onstaan is als "proof-of-concept" spruiten ook onze bedenkingen voort. Het is namelijk zo dat een aantal uitbreidingen niet op een andere manier konden geïmplementeerd worden omwille van vroeger genomen design beslissingen. Bij het nemen van die beslissingen bestonden deze uitbreidingen nog niet en werd er geen rekening mee gehouden. Dit heeft men bij Cocoon 2 anders gedaan.

Cocoon 1 is wel degelijk een bruikbaar framework voor publicatie op het web, maar door bepaalde beslissingen die genomen zijn wordt het sitebeheer bemoeilijkt en is de scheiding van de belangen niet overal consistent doorgevoerd. Ons inziens hangt de bruikbaarheid van Cocoon 1 als framework vooral af van de complexiteit en de grootte van de sites die er mee beheerd moeten worden omwille van de aangehaalde redenen. Dit wordt volgens ons vooral veroorzaakt doordat er in elke pagina moet verwezen worden naar stylesheet die gebruikt moet worden. Is het een uitgebreide site met veel pagina's waar eigenlijk elke keer dezelfde keten van processing instructions in staat, dan is de kans reëel dat er hier en daar een foutje insluipt.

9.2. Cocoon 2

9.2.1. Inleiding

Na het onverwachte succes van Cocoon 1 en het daardoor te voorschijn komen van enkele serieuze beperkingen, beslisten de ontwikkelaars om een nieuwe en verbeterde versie te maken. Deze nieuwe versie zou deze beperkingen moeten opheffen en voor deze nieuwe versie werd het hele ontwerp herbedacht en kreeg dit framework ook een nieuw logo van de maker van Cocoon, Stefano Mazzocchi. Meer over de verschillen tussen Cocoon 1 en Cocoon 2 is terug te vinden in de sectie **Vergelijking met Cocoon 1** [Sectie 9.2.4.] .

Cocoon 2 ([COCOON2]) is bedoeld om, zoals de makers het uitdrukken, de mogelijkheid te bieden om tegelijkertijd meerdere 100MB documenten te verwerken in een JVM met slechts een paar MB heapsize. Hierdoor is het zorgvuldig gebruik van het geheugen en het op elkaar afstellen van de interne componenten één van de belangrijkste oogpunten van het project. In plaats van met een API te werken die vereist dat het hele document in het geheugen geplaatst is (zoals DOM [TRDOM], alhoewel sommige DOM implementaties dit anders aanpakken), werd er een eventgebaseerde API gekozen, met name SAX [SAX], [Sectie 12.2.] , die

gebaseerd is op het principe van de inversiecontrole.

Zonder nu reeds al te diep in te gaan op de interne werking van Cocoon 2 dient het toch even vermeld te worden dat door het eventgebaseerde model het mogelijk is dat de document generatoren events genereren, die in verschillende verwerkingsstappen afgehandeld worden en uiteindelijk een uitvoerstroom geven. Op de website van Cocoon 2 worden vijf punten vermeld waardoor dit model een serieuze invloed heeft op de performantie en het geheugengebruik. Deze zijn:

- **Incrementele verwerking** - de bedoeling is dat de client al een antwoord kan beginnen ontvangen voordat alle verwerkingsstappen volledig doorlopen zijn
- **Verminderd geheugengebruik** - de incrementele verwerking zorgt ervoor dat de XML events onmiddellijk in de uitvoerstroom kunnen getransformeerd worden, zonder de nood te hebben om de resultaten van elke verwerkingsstap volledig in het geheugen op te slaan
- **Makkelijkere schaalbaarheid** - dit hangt samen met het verminderde geheugengebruik, dat meerdere gelijktijdige verwerkingen toelaat en dus ook een zekere scaleerbaarheid als de belasting stijgt
- **Beter optimaliseerbaar code-model** - makkelijkere identificatie van **hotspots** die zorgen voor een betere optimalisatie van de code van Cocoon 2, dit wil zeggen dat de JVM de bytecode naar native code vertaalt nadat een stuk code een bepaald aantal maal is uitgevoerd
- **Verminderde garbage collection** - de meeste DOM implementaties vereisen drie tot vijf keer de originele grootte van het document in het geheugen. Door een andere API te nemen, wordt de garbage verminderd en dit zal dan ook ten goede komen aan de performantie en de scaleerbaarheid

Dit zijn zowat de voornaamste, maar zeker niet de enige, bedenkingen die vooraf gemaakt zijn bij het ontwerp van Cocoon 2. Op een aantal voorafgaande analyses komen we later in dit hoofdstuk nog terug, andere komen helemaal niet meer aan bod. Het is niet het doel van dit hoofdstuk om het volledige ontwerp van Cocoon 2 te bestuderen, maar eerder een algemeen idee te geven welke mogelijkheden Cocoon 2 biedt en niet biedt, hoe Cocoon 2 werkt, en dit steunend op ervaring die wij hebben opgedaan door met Cocoon 2 te leren werken. Op dit moment gebruiken wij Cocoon 2 ook om de teksten van onze thesis vanuit XML om te zetten naar HTML [Hoofdstuk 6.] en PDF [Hoofdstuk 8.] .

Voor het gedeelte over Cocoon 2 hebben wij onder andere beroep gedaan op [COCOON2] en [INTROC2].

9.2.2. Installatie van Cocoon 2

Voor onze ervaringen met het installeren van Cocoon 2 verwijzen wij naar [Appendix D.] .

9.2.3. Werking van Cocoon 2

Cocoon 2 werkt op de achtergrond, maar ook op het voorplan totaal anders dan Cocoon 1. Dit heeft verschillende oorzaken:

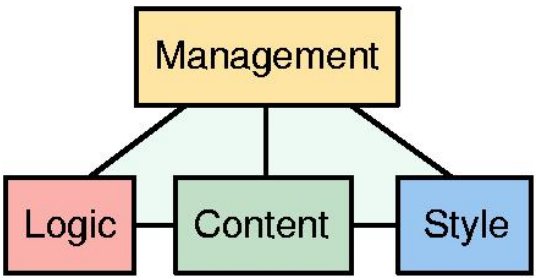
- Cocoon 1 is ontstaan als proof of concept
- Cocoon 1 maakt gebruik van passieve (boomgebaseerde) APIs, Cocoon 2 gebruikt actieve (eventgebaseerde) APIs
- aanzienlijke ontwerpbeperkingen in Cocoon 1
- onderschatting van het mogelijke succes van Cocoon 1

Dit om maar een paar oorzaken te noemen. Zoals in de inleiding reeds vermeld werd, is er om verschillende redenen gekozen om voornamelijk met de SAX API te werken, die in tegenstelling tot DOM, eventgebaseerd is. Tussen de verschillende componenten van Cocoon 2 wordt er gecommuniceerd met behulp van SAX events, en er ontstaat als het ware een pipeline van SAX events. Dit eventgebaseerde model heeft niet alleen een invloed gehad op de algemene architectuur van Cocoon 2, maar heeft er ook voor gezorgd dat de interne componenten zoals XSLT verwerking en PDF formattering herbekeken dienden te worden. Deze componenten bestaan ook als aparte projecten die deel uitmaken van de XML projecten van Apache. Er is dan ook een nauwe samenhang van deze projecten met Cocoon 2.

9.2.3.1. De filosofie van Cocoon 2

De filosofie van Cocoon 2 verdient het om nader bekeken te worden. Voor het beheren van een website wordt er gebruik gemaakt van een **sitemap** [Sectie 9.2.3.2.3.] . Een sitemap is een document dat zich op een bepaalde plaats bevindt en dit document vertegenwoordigt een centraal beheer voor de administratie van een website, inclusief het onderhouden van de URI ruimte. Omdat het mogelijk moet kunnen zijn om het overzicht over dit document te bewaren, maar tegelijktertijd nog veel verschillende websites te kunnen aanbieden, bestaat er ook de mogelijkheid om dit document op te splitsen in een aantal kleinere sitemaps, waarbij die opsplitsing meestal samenvalt met de verschillende (delen van) websites.

In het hart van Cocoon 2 wordt er gewerkt met pipeline mapping techniek, en dit vervangt het reactor patroon dat in Cocoon 1 gebruikt werd. Om verschillende technische redenen, maar vooral om een bepaalde sleutelbeperking (geen centraal punt van beheer) werd het reactor patroon overboord gegooid. In plaats daarvan heeft men gekozen voor een pipeline-structuur. Dit is vooral gebaseerd op het feit dat het aantal contracten, die betrekking hebben op het beheer van de website, vrij beperkt zijn, zelfs voor grote website. Ook de vaststelling dat het aantal contracten vrij beperkt blijft, ook al neemt de omvang van de site nog toe, is hierbij vrij belangrijk. Met contracten wordt in de context van Cocoon bedoeld de samenhang die er is tussen de veschillende logische gedeelten van een gestructureerd en coherent gedistribueerd informatiesysteem. Cocoon maakt daarvoor gebruik van een **pyramide model van webcontracten**.



Figuur 9.1. pyramide model van webcontracten

Copyright © 1999-2002 The Apache Software Foundation. All rights reserved.

Er zijn hier 4 onderdelen te onderscheiden in dit model:

- **Management** (Management) - deze mensen bepalen wat de site moet bevatten, hoe de site er uit moet zien en wat het gedrag is van de site
- **Inhoud** (Content) - deze mensen zijn verantwoordelijk voor het verzorgen van de inhoud van de site.
- **Logica** (Logic) - de verantwoordelijken voor de integratie van databases en generatie van dynamische inhoud in de site
- **Stijl** (Style) - de verantwoordelijken voor de presentatie van informatie, de look & feel, de tekeningen op de site en het onderhoud

Hierbij wordt er uitgegaan van vijf contracten die bestaan tussen deze vier groepen. Deze contracten zijn:

- management - inhoud
- management - logica
- management - stijl
- logica - inhoud
- inhoud - stijl

Hierbij zou het moeten opvallen dat er geen **logica - stijl** contract is. Een van de doelen van Cocoon is om te voorzien in software en richtlijnen om zo'n contract te kunnen vermijden. Het is vooral het concept van de **sitemap** [Sectie 9.2.3.2.3.] dat hierin een heel belangrijke rol speelt.

9.2.3.2. Cocoon 2 intern

Om de interne werking van Cocoon 2 uit te leggen, gaan we eerst enkele gebruikte concepten uitleggen. Deze concepten spelen een sleutelrol in de Cocoon 2 architectuur. Deze concepten zijn de pipeline, componenten en de sitemap.

9.2.3.2.1. Pipeline

De verwerking van een XML document kan in verschillende, niet overlappende stappen opgebroken worden. In de meest algemene vorm kan je deze stappen opdelen in de invoer, enkele verwerkingsstappen en daarna de uitvoer. De combinatie van deze stappen beschrijft een verwerkingspipeline. Tussen de verschillende stappen wordt gecommuniceerd met behulp van SAX events [SAX], [Sectie 12.2.] , waarbij de uitvoer van de ene stap de invoer van de volgende stap is.

Dit is eigenlijk een eenvoudig concept, aangezien het vrij makkelijk is om een moeilijke taak in een serie van kleine, gemakkelijke taken op te splitsen. Iedereen die vertrouwd is met UNIX en Linux weet wel hoe hij/zij een klein aantal algemene programma's (zoals ls, grep, find, sort) achter elkaar kan inschakelen om bepaalde taken te verwezenlijken. Elk programma vervult zijn specifieke taak maar is toch heel algemeen geschreven. De verschillende componenten van Cocoon 2 zijn op dezelfde manier opgevat. Een component die instaat voor een bepaalde stap in een pipeline wordt zo algemeen mogelijk geschreven zodat deze component ook gemakkelijk in andere projecten kan hergebruikt worden.

9.2.3.2.2. Componenten

Cocoon 2 bevat een aantal componenten (en hun implementaties) waarmee een pipeline te modelleren valt. Deze componenten kunnen in verschillende groepen opgedeeld worden, zoals een generator om de invoer te produceren, of een serializer die voor de uitvoer zorgt. Een pipeline bestaat dus uit de aaneenschakeling van een aantal componenten.

De componenten die standaard bij Cocoon 2 zitten, kunnen onderverdeeld worden in drie groepen: invoer, verwerking en uitvoer. Een pipeline bevat minstens één component uit de invoergroep en één component uit de uitvoergroep, enkele uitzonderingen buiten beschouwing gelaten.

De **invoergroep** bestaat uit generators en readers. Een generator staat in voor het lezen van een databron en deze data in de pipeline te brengen als een reeks van SAX events. Een SAXParser is de eenvoudigste generator, maar eender welke databron die in een serie SAX

events kan omgezet worden, kan als basis dienen voor een generator. De meest gebruikte generators zijn:

- FileGenerator: leest een XML bestand van het lokale bestandensysteem of van het web en genereert SAX events
- HTMLGenerator: leest een HTML pagina (lokaal of van het web), zet deze om tot well-formed HTML (ook bekend als XHTML [TRXHTML]) en genereert overeenkomstige SAX events
- DirectoryGenerator: leest het lokale bestandssysteem om in een directory listing te kunnen voorzien

Naast de generators, hebben we ook nog de readers om voor invoer te zorgen. Een reader is een speciaal geval in de pipeline. Een reader genereert geen SAX events, het is een component die zich zelfs niet bewust is van XML. Deze component dient om bepaalde resources rechtstreeks naar de uitvoerstroom te kopiëren en wordt meestal gebruikt om afbeeldingen of CSS stylesheets aan te bieden. Een reader is dus een pipeline op zich: deze zorgt zelf voor het lezen van de invoerdata en het serialiseren naar de uitvoer.

Bij de **verwerkers** kunnen we twee groepen onderscheiden, met name de voorwaardelijke verwerkers en de onvoorwaardelijke verwerkers. De voorwaardelijke verwerkers kunnen nog eens onderverdeeld worden in matchers en selectors. De onvoorwaardelijke kunnen op hun beurt onderverdeeld worden in transformers en actions. We zullen in deze paragraaf kort aanhalen waarvoor deze vier groepen dienen. **Matchers** zijn de eenvoudigste van de twee voorwaardelijke componenten en zijn te vergelijken met een if statement. Indien een bepaalde voorwaarde voldaan is, dan wordt een bepaalde pipeline of een gedeelte daarvan uitgevoerd. **Selectors** zijn te vergelijken met if-then-else statements. Ze worden vooral gebruikt wanneer er verschillende opties beschikbaar zijn, en typisch worden ze ingezet om voorwaardelijke secties te maken binnen een pipeline, terwijl matchers gebruikt worden om te bepalen wanneer een bepaalde pipeline moet uitgevoerd worden. Bij selectors gebruikt men meestal een soort opsomming van de mogelijke waarden, terwijl bij matchers met wildcards of met reguliere expressies kan gewerkt worden. Binnen een bepaalde pipeline kan er bijvoorbeeld gekozen worden om de uitvoer aan te passen aan de browser van de bezoeker. Om dit te verwezenlijken kan je met een selector op basis van de User-Agent bepalen welke XSLT transformatie moet toegepast worden. **Transformers** zijn onvoorwaardelijke verwerkers en zijn misschien wel de belangrijkste componenten in de hele pipeline. Een transformer is een component die SAX events aan de invoer verwacht, daar bepaalde bewerkingen op uitvoert, en dan terug SAX events genereert. Je kan ze zien als een component die SAX events wijzigt op het moment dat ze door deze component vloeien. De XSLT transformer is de meest gebruikte en dient om XSLT transformaties toe te passen op de invoer. De resultaten van deze transformatie worden dan aan het vervolg van de pipeline aangeboden. Het kan ook zijn dat het hoofddocument in verschillende kleinere documenten is gesplitst, je kan dan de xinclude transformatie toepassen om hiervan één groot document te maken vooraleer er andere transformaties op los te laten. Tenslotte hebben we nog de **actions**. Deze dienen om een bepaald dynamisch gedrag in de pipeline te kunnen invoeren. Ze worden vooral gebruikt bij database-updates, validatie van formulieren, bijhouden van sessiestatus, enz. Of de pipeline verder uitgevoerd wordt (of toch dat gedeelte waar de action betrekking op heeft) hangt af van het resultaat van de action. Is deze succesvol, dan zal er verdergegaan worden, maar als deze gefaald heeft, dan wordt de uitvoering van dat gedeelte afgebroken.

De serializers, oftewel de componenten in de **uitvoergroep**, zijn de eindpunten van de pipelines. Deze componenten nemen een stroom van SAX events en vertalen deze naar een geschikt formaat voor het antwoord aan de client. De SAX events zijn afkomstig van de rest van de pipeline, en dat kan ofwel rechtstreeks van de generator zijn ofwel van een verwerker. Het uitvoerformaat is afhankelijk van de gebruikte serializer. Er bestaan serializers om de uitvoer om te zetten naar XML (de eenvoudigste), HTML, PDF, PostScript, JPEG, PNG, ...

Wij beschouwen dit, zoals zo veel mensen, als de echte kracht van het Cocoon framework, om vanuit één bron van XML inhoud en enkele verwerkingsstappen verschillende formaten aan te bieden.

Dit klinkt allemaal heel mooi, maar er moet ook iets zijn waarin deze pipelines gespecificeerd worden. In Cocoon 2 wordt dit bijgehouden in een document dat de sitemap genoemd wordt. In dit document worden ook de componenten die gebruikt worden voor die site gespecificeerd. In de volgende sectie zullen we de sitemap uitgebreid bespreken.

9.2.3.2.3. Sitemap

Voor deze bespreking hebben we een beroep gedaan op [C2SITEMAP] en de eigen ervaringen die we opgedaan hebben. Toen we begonnen te werken met Cocoon 2 was er nog geen documentatie over die sitemap.

Als een client een request uitvoert, dan moet de juiste pipeline gevonden worden om die request af te handelen, en deze moet dan uitgevoerd worden om het antwoord voor de client te kunnen produceren. Dit moet ergens gespecificeerd zijn, en bij Cocoon 2 gebeurt dat in een document dat de sitemap genoemd wordt. Hierin wordt beschreven welke componenten een bepaalde pipeline bepalen. Deze componenten worden ook in deze sitemap gedefinieerd. De sitemap is dus het document dat deze twee functies vervult en op die manier een centrale rol speelt in het website management met behulp van Cocoon 2.

De sitemap is een XML configuratiebestand en heeft dus een welbepaalde structuur. De standaard Cocoon sitemap, *sitemap.xmap*, kan in de Cocoon Web applicatie directory teruggevonden worden, in ons geval *$TOMCAT_HOME/webapps/cocoon/sitemap.xmap*. Een sitemap is een XML document waarvan de elementen in de naamruimte *http://apache.org/cocoon/sitemap/1.0* zitten. Een sitemap bevat twee grote onderdelen, met name map:components en map:pipelines, en heeft verder de volgende structuur:

Voorbeeld 9.7.

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
   <map:components>
      <!--component declarations-->
      <map:generators/>
      <map:readers/>
      <map:transformers/>
      <map:actions/>
      <map:serializers/>
      <map:actions/>
      <map:matchers/>
      <map:selectors/>
   </map:components>
   <map:pipelines>
      <!--pipeline definitions-->
   </map:pipelines>
</map:sitemap>
```

Het is duidelijk dat de definities van alle generators binnen map:generators gezet worden en dat als kindelement van map:generators, met andere woorden deze definities worden gegroepeerd. In de volgende paragrafen zullen we wat dieper ingaan op het declareren van componenten en daarna op het definiëren van pipelines.

9.2.3.2.3.1. Componenten: algemeen

Algemeen kunnen we een groep van componenten met het volgende XML fragment beschrijven:

Voorbeeld 9.8.

```
<map:component-types default="component-name">
   <map:component-type name="component-name" src="implementation">
      <!--component-type specific parameters-->
   </map:component-type>
</map:component-types>
```

- **component-types** is de naam voor een specifieke groep componenten, bijvoorbeeld generators, serializers, matchers, ... **Component-type** is dan het enkelvoud hiervan.
- De waarde van het naam-attribuut van elke component moet uniek zijn. Dit omdat deze namen als sleutel gebruikt worden verderop in de sitemap om naar een bepaalde component te kunnen verwijzen.
- Ook moet met behulp van het src attribuut aangegeven worden welke klasse de implementatie van deze component realiseert. Het is mogelijk dat verschillende componenten dezelfde implementatie delen, maar dat die componenten gedeclareerd zijn met een verschillende naam.
- Bij het element component-types kan een default aangegeven worden. Deze default wordt gebruikt wanneer er naar een component van een bepaald type wordt verwezen zonder een specifieke component op te geven.
- Tenslotte hebben we nog de parameters specifiek voor deze component. Deze parameters worden vanuit de sitemap doorgegeven aan de component. De elementen in de parameter zijn specifiek voor elke component.

Alhoewel er een groot aantal componenten bij Cocoon 2 geleverd worden, kan het goed zijn dat je voor een bepaalde toepassing zelf een component moet implementeren. Mits het implementeren van de juiste interfaces en/of overerven van de juiste klassen is dit mogelijk. Cocoon 2 kan dus uitgebreid worden met niet-meegeleverde componenten.

In de volgende secties geven we een overzicht van de definities van een aantal componenten in de sitemap. Hoe deze componenten dan in een sitemap kunnen aangewend worden, tonen we in de sectie over pipelines [Sectie 9.2.3.2.3.10.] .

9.2.3.2.3.2. Componenten: generators

Een generator neemt XML van een bepaalde bron inhoud en genereert op basis van die XML inhoud SAX events, en initialiseert daarmee de verwerking van de pipeline. De definitie van een lijst van generators ziet er als volgt uit:

Voorbeeld 9.9.

```
<map:generators default="file">
   <map:generator name="file"
   src="org.apache.cocoon.generation.FileGenerator"/>
   <map:generator name="directory"
   src="org.apache.cocoon.generation.DirectoryGenerator"/>
   <map:generator name="html"
   src="org.apache.cocoon.generation.HTMLGenerator"/>
</map:generators>
```

Er worden in dit geval drie generators gedefinieerd die kunnen gebruikt worden in de pipeline. Indien je een generator wil gebruiken in een pipeline is het mogelijk om aan te

geven welke van de drie je wenst te gebruiken. Het default-attribuut geeft aan welke generator gebruikt moet worden indien in de pipeline een generator wordt gebruikt, maar er niet wordt aangegeven omn welke generator het gaat. Meer hierover in de sectie over de pipelines [Sectie 9.2.3.2.3.10.] .

9.2.3.2.3.3. Componenten: readers

Een reader wordt gebruikt om bepaalde resources rechtstreeks naar de uitvoerstroom te kopiëren. Maar een reader kan ook gebruikt worden om de uitvoer van een bepaalde bron aan het begin van een pipeline aan te leggen. Let wel: een reader genereert ook in dit geval geen SAX events, maar stuurt gewoon een stroom van bytes naar de pipeline.

Voorbeeld 9.10.

```
<map:readers default="resource">
   <map:reader name="resource"
   src="org.apache.cocoon.reading.ResourceReader"/>
   <map:reader name="jspreader"
   src="org.apache.cocoon.reading.JSPReader"/>
</map:readers>
```

In dit voorbeeld worden twee readers gedefinieerd. De reader met de naam resource kan gebruikt worden om statische bestanden, zoals afbeeldingen of CSS-bestanden, aan te bieden. De jspreader dient om de uitvoer van een JSP pagina in een pipeline in te brengen.

9.2.3.2.3.4. Componenten: transformers

Een transformer staat in een pipeline tussen invoer en uitvoer. Deze component heeft als invoer en als uitvoer SAX events, maar in de component zelf wordt er eventueel een bewerking op uitgevoerd.

Voorbeeld 9.11.

```
<map:transformers default="xslt">
   <map:transformer name="xslt"
   src="org.apache.cocoon.transformation.TraxTransformer">
      <use-request-parameters> true </use-request-parameters>
   </map:transformer>
   <map:transformer name="xinclude"
   src="org.apache.cocoon.transformation.XIncludeTransformer"/>
</map:transformers>
```

De eerste transformer, met als naam xslt, wordt als standaardtransformer gedefinieerd. Deze transformer dient om XSLT transformaties toe te passen op een XML bron (in casu een stroom van SAX events). Wat vooral aan de definitie van deze transformer opvalt, is dat er een kindelement is, genaamd use-request-parameters en dat als inhoud true heeft. Dit element wordt gebruikt om aan de transformer door te geven dat er met de parameters die meegegeven werden aan de request (bijvoorbeeld parameters die via de URL werden meegegeven), rekening gehouden moet worden bij het toepassen van de transformaties. Dit heeft natuurlijk alleen effect als de parameters ook effectief worden gebruikt bij de transformaties. Met de tweede transformer is het mogelijk om in het hoofddocument naar verschillende subdocumenten te verwijzen. Wordt deze transformer dan toegepast op het hoofddocument, dan zal deze transformer de include-statements gaan verwerken en deze statements vervangen door het document waarnaar ze verwijzen, dit gebeurt helaas niet recursief. Een voorbeeldje hiervan zullen we geven in de sectie over pipelines [Sectie 9.2.3.2.3.10.] .

9.2.3.2.3.5. Componenten: actions

Actions dienen om runtime parameters te manipuleren op basis van de request en de applicatietoestand. Om het eenvoudiger te zeggen: actions dienen om bepaalde beslissingen te nemen in een pipeline, en die beslissingen zijn gebaseerd op de doorgegeven parameters, de request of de toestand waarin de (web-)toepassing zich op dat moment bevindt. De ontwerpers van Cocoon hebben gekozen om Actions in te voeren om zo hun beslissingsmodel eenvoudiger te maken, alhoewel het volgens hen ook op andere manier mogelijk is, maar dat zou de sitemap echter onleesbaar en onbegrijpbaar maken volgens hen. Omdat we zelf geen gebruik gemaakt hebben van Actions en ze ook niet bestudeerd hebben, gaan we hier niet dieper op in.

9.2.3.2.3.6. Componenten: serializers

Een serializer transformeert SAX events naar een binaire of een karakterstroom die dient om aan de client doorgegeven te worden. Een serializer is de laatste component in een pipeline.

Voorbeeld 9.12.

```
<map:serializers default="html">
   <map:serializer name="html" mime-type="text/html"
   src="org.apache.cocoon.serialization.HTMLSerializer"/>
   <map:serializer name="xml" mime-type="text/xml"
   src="org.apache.cocoon.serialization.XMLSerializer"/>
   <map:serializer name="fo2pdf" mime-type="application/pdf"
   src="org.apache.cocoon.serialization.FOPSerializer"/>
   <map:serializer name="fo2ps" mime-type="application/postscript"
   src="org.apache.cocoon.serialization.FOPSerializer"/>
   <map:serializer name="svg2jpeg" mime-type="image/jpeg"
   src="org.apache.cocoon.serialization.SVGSerializer"/>
</map:serializers>
```

In dit blok worden vijf serializers gedefinieerd. De eerste zet een stroom van SAX events om in een HTML pagina, de tweede zorgt voor de omzetting in een XML bestand. De derde serializer zet een stroom van formatting objects (ook aangeleverd met behulp van SAX events) om in een pdf bestand. Deze serializer kan, zoals blijkt uit de vierde serializer, ook gebruikt worden om een PostScript bestand te genereren. De laatste serializer dient om een SVG bestand (Scalable Vector Graphics [SVG] - een XML toepassing voor vectorgrafieken) om te zetten in een JPEG bestand.

9.2.3.2.3.7. Componenten: matchers

Een matcher zorgt voor het matchen van een bepaald patroon. Dit wordt vooral gebruikt bij het selecteren van de juiste pipeline.

Voorbeeld 9.13.

```
<map:matchers default="wildcard">
   <map:matcher name="wildcard"
   src="org.apache.cocoon.matching.WildcardURIMatcher"/>
   <map:matcher name="regexp"
   src="org.apache.cocoon.matching.RegexpURIMatcher"/>
</map:matchers>
```

We hebben hier twee matchers gedefinieerd. De eerste matcher laat ons toe om URIs te matchen waarbij in de te matchen URI gebruik gemaakt kan worden van wildcards. Bij de

tweede matcher kunnen we gebruik maken van reguliere expressies om de te matchen URI te omschrijven.

9.2.3.2.3.8. Componenten: selectors

Een selector wordt gebruikt om conditionele logica (if-then-else of switch) te kunnen gebruiken in de sitemap. De functionaliteit is in zijn meest eenvoudige vorm terug te brengen tot het evalueren van een booleaanse uitdrukking.

Voorbeeld 9.14.

```
<map:selectors default="browser">
   <map:selector name="browser"
   src="org.apache.cocoon.selection.BrowserSelector">
      <!--NOTE: The appearance indicates the search order. This is very
      important since some words may be found in more than one browser
      description-->
      <browser name="explorer" useragent="MSIE"/>
      <browser name="opera" useragent="Opera"/>
      <browser name="lynx" useragent="Lynx"/>
      <browser name="mozilla5" useragent="Mozilla/5"/>
      <browser name="mozilla5" useragent="Netscape6/"/>
      <browser name="netscape" useragent="mozilla"/>
   </map:selector>
</map:selectors>
```

We hebben hier maar één selector gedefinieerd, met name browser, maar er bestaan nog verschillende andere selectors, die we hier niet vermelden. Met behulp van deze selector is het mogelijk om de pipeline aan te passen aan de browser waarmee de site bezocht wordt. Wanneer je met een selector bijvoorbeeld wilt aangeven dat een bepaald gedeelte voor explorer is, dan gaat de selector in de User-Agent string die de browser doorstuurt, nakijken of MSIE deel uitmaakt van die string. Indien MSIE in de User-Agent string wordt gevonden, geeft dit een positief resultaat en zal het desbetreffende gedeelte uitgevoerd worden, anders niet. De volgorde van de opgegeven mappings tussen de beschrijvende string en de string waarnaar gezocht moet worden, bepalen ook de zoekvolgorde. Dit wil zeggen dat voor mozilla5 er eerst gezocht wordt naar Mozilla/5 in de User-Agent string en als dat geen positief resultaat oplevert, zal er naar Netscape6/ gezocht worden. Dit gedrag is volledig analoog aan de afhandeling zoals die in Cocoon 1 gebeurt [Sectie 9.1.2.3.1.] . Een voorbeeld van het gebruik zullen we geven in de sectie over pipelines [Sectie 9.2.3.2.3.10.] .

9.2.3.2.3.9. Componenten: definitie van de gebruikte componenten

Om het overzicht te bewaren, herhalen we hier nog eens alle componenten die we in de vorige secties gedefinieerd hebben. Op die manier is het ook mogelijk om in één oogopslag te zien welke componenten default gebruikt worden, indien in de pipeline niet wordt aangegeven welke component gebruikt moet worden. Dit is vooral belangrijk bij het gedeelte over pipelines [Sectie 9.2.3.2.3.10.] .

Voorbeeld 9.15.

```
<map:generators default="file">
   <map:generator name="file"
   src="org.apache.cocoon.generation.FileGenerator"/>
   <map:generator name="directory"
   src="org.apache.cocoon.generation.DirectoryGenerator"/>
```

```
    <map:generator name="html"
    src="org.apache.cocoon.generation.HTMLGenerator"/>
  </map:generators>
  <map:readers default="resource">
    <map:reader name="resource"
    src="org.apache.cocoon.reading.ResourceReader"/>
    <map:reader name="jspreader"
    src="org.apache.cocoon.reading.JSPReader"/>
  </map:readers>
  <map:transformers default="xslt">
    <map:transformer name="xslt"
    src="org.apache.cocoon.transformation.TraxTransformer">
      <use-request-parameters> true </use-request-parameters>
    </map:transformer>
    <map:transformer name="xinclude"
    src="org.apache.cocoon.transformation.XIncludeTransformer"/>
  </map:transformers>
  <map:serializers default="html">
    <map:serializer name="html" mime-type="text/html"
    src="org.apache.cocoon.reading.serialization.HTMLSerializer"/>
    <map:serializer name="xml" mime-type="text/xml"
    src="org.apache.cocoon.reading.serialization.XMLSerializer"/>
    <map:serializer name="fo2pdf" mime-type="application/pdf"
    src="org.apache.cocoon.reading.serialization.FOPSerializer"/>
    <map:serializer name="fo2ps" mime-type="application/postscript"
    src="org.apache.cocoon.reading.serialization.FOPSerializer"/>
    <map:serializer name="svg2jpeg" mime-type="image/jpeg"
    src="org.apache.cocoon.serialization.SVGSerializer"/>
  </map:serializers>
  <map:matchers default="wildcard">
    <map:matcher name="wildcard"
    src="org.apache.cocoon.matching.WildcardURIMatcher"/>
    <map:matcher name="regexp"
    src="org.apache.cocoon.matching.RegexpURIMatcher"/>
  </map:matchers>
  <map:selectors default="browser">
    <map:selector name="browser"
    src="org.apache.cocoon.selection.BrowserSelector">
      <!--NOTE: The appearance indicates the search order. This is very
      important since some words may be found in more than one browser
      description-->
      <browser name="explorer" useragent="MSIE"/>
      <browser name="opera" useragent="Opera"/>
      <browser name="lynx" useragent="Lynx"/>
      <browser name="mozilla5" useragent="Mozilla/5"/>
      <browser name="mozilla5" useragent="Netscape6/"/>
      <browser name="netscape" useragent="mozilla"/>
    </map:selector>
  </map:selectors>
```

9.2.3.2.3.10. Pipelines

Voor deze sectie hebben we een beroep gedaan op [INTROC2], [C2DEVML] en onze eigen ervaringen.

Pipelines vormen het hart van Cocoon 2. Dit komt omdat Cocoon 2 dit model intensief gebruikt om documenten te serveren. Een XML document wordt door een pipeline, die uit verschillende transformatiestappen bestaat, geduwd. Elke pipeline begint met een generator, gevolgd door nul of meerdere transformers en eindigt met een serializer. Eén zo een pipeline wordt omsloten door een matcher. Zoals reeds eerder aangehaald dient een matcher om een bepaalde pipeline te selecteren. Binnen een pipeline kan ook een selector gebruikt worden om het gedrag van de pipeline op basis van bijvoorbeeld de gebruikte browser te beïnvloeden. De

verschillende componenten waaruit een pipeline kan bestaan hebben we reeds besproken; we gaan nu bekijken hoe we deze componenten in de pipeline kunnen aanwenden om documenten aan te bieden. We zullen eerst een voorbeeld geven van pipeline-definities die wij gebruiken, en daarna zullen we dat voorbeeld opbreken in kleinere stukken en deze stukken bespreken.

Voorbeeld 9.16.

```
<map:pipelines>
   <map:pipeline>
      <map:match pattern="">
         <map:redirect-to uri="index.html"/>
      </map:match>
      <map:match pattern="index.html">
         <map:generate src="index.xml"/>
         <map:transform type="xinclude"/>
         <map:transform src="stylesheets/html.xsl"/>
         <map:serialize/>
      </map:match>
      <map:match pattern="overview.html">
         <map:generate src="index.xml"/>
         <map:transform type="xinclude"/>
         <map:transform src="stylesheets/overview.xsl"/>
         <map:serialize/>
      </map:match>
      <map:match pattern="index.pdf">
         <map:generate src="index.xml"/>
         <map:transform type="xinclude"/>
         <map:transform src="stylesheets/fopdf.xsl"/>
         <map:serialize type="fo2pdf"/>
      </map:match>
      <map:match pattern="index.ps">
         <map:generate src="index.xml"/>
         <map:transform type="xinclude"/>
         <map:transform src="stylesheets/fopdf.xsl"/>
         <map:serialize type="fo2ps"/>
      </map:match>
      <map:match pattern="index.xml">
         <map:generate src="index.xml"/>
         <map:transform type="xinclude"/>
         <map:serialize type="xml"/>
      </map:match>
      <map:match pattern="index.fo">
         <map:generate src="index.xml"/>
         <map:transform type="xinclude"/>
         <map:transform src="stylesheets/fopdf.xsl"/>
         <map:serialize type="xml"/>
      </map:match>
      <map:match pattern="images/*.jpg">
         <map:generate src="images/{1}.svg"/>
         <map:serialize type="svg2jpeg"/>
      </map:match>
   </map:pipeline>
</map:pipelines>
```

Zoals eerder aangehaald dient het **map:pipelines** element om de pipelines te definiëren. Dit element heeft nul of meerdere **map:pipeline** elementen en dat zijn de enige kindelementen die dat element kan hebben. Het element **map:pipeline** heeft een optioneel (nul of één) kindelement **map:handle-errors** en nul of meerdere **map:match** elementen als kindelement. Dat is in het kort weergegeven hoe de top van de boom van dit gedeelte van de sitemap eruitziet.

Voor het voorbeeld dat wij hier gegeven hebben, hebben we één **map:pipelines** element, dat één **map:pipeline** element als kind heeft. Dit laatste element heeft verschillende **map:match**

elementen als kind. Het voorbeeld dat we hier gegeven hebben, is een subsitemap [Sectie 9.2.3.2.3.11.] waarin (in ons geval) de virtuele URI-ruimte *http://host/cocoon/tekst/thesis/* is gedefinieerd. Dus alles waar het eerste gedeelte van de URL *http://host/cocoon/tekst/thesis/* is en dat we willen aanbieden met behulp van Cocoon 2 wordt in dit voorbeeld gedefinieerd. We zullen nu wat dieper ingaan op dit voorbeeld.

Voorbeeld 9.17.

```
<map:match pattern="">
   <map:redirect-to uri="index.html"/>
</map:match>
```

Met dit gedeelte van de pipeline wordt aan Cocoon 2 verteld wat er moet gebeuren indien een request voor *http://host/cocoon/tekst/thesis/* (de waarde van het attribuut **pattern** is de lege string) moet afgehandeld worden. Het is de matcher die ervoor zorgt dat voor een bepaalde request de juiste pipeline gekozen wordt en dit op basis van het gespecificeerde patroon. Met het element **map:redirect-to** zeggen we dat Cocoon 2 een redirect antwoord moet sturen aan de client. Het attribuut **uri** definieert de URI waarnaar de client moet doorverwezen worden. In dit geval wordt de client doorverwezen naar **index.html** waardoor de URI waarnaar de client dan gaat vragen *http://host/cocoon/tekst/thesis/index.html* wordt. **Opmerking:** In het vervolg van de bespreking gaan we niet meer expliciet bij elke verwijzing naar een URI het gedeelte *http://host/cocoon/tekst/thesis/* herhalen, maar, indien niet anders vermeld wordt, maakt dit gedeelte deel uit van de URIs waarnaar we verwijzen.

Voorbeeld 9.18.

```
<map:match pattern="index.html">
   <map:generate src="index.xml"/>
   <map:transform type="xinclude"/>
   <map:transform src="stylesheets/html.xsl"/>
   <map:serialize/>
</map:match>
```

In dit gedeelte wordt de pipeline beschreven voor een request van een client voor *index.html*. We hebben hier een pipeline die bestaat uit een generator, twee transformers en een serializer. De generator is het begin van de pipeline en heeft in dit geval één attribuut. Dit attribuut src is optioneel, denken we maar aan een generator van statusinformatie die geen src attribuut nodig heeft. Indien we de generator willen gebruiken die we als default hebben aangeduid bij het declareren van de componenten, moeten we dit niet aangeven bij het **map:generate** element. We willen in dit geval de FileGenerator, die als default is aangeduid, gebruiken. Moesten we toch een andere generator willen gebruiken, dan kunnen we dat aangeven met het **type** attribuut, dat als waarde de gedefinieerde naam (de waarde van het **name** attribuut bij de declaratie van die generator) van de generator heeft. Een ander attribuut dat we hier wel gebruikt hebben, is het **src** attribuut. Met de waarde van dit attribuut, in dit geval *index.xml*, geven we aan dat het bestand index.xml als bron moet dienen voor de generator om SAX events te genereren. Het bestand index.xml bevindt zich in het lokale bestandssysteem op dezelfde plaats als deze sitemap, de verwijzing die in het **src** attribuut gebruikt wordt, is dus een relatieve verwijzing. Absolute verwijzingen naar bestanden in het lokale bestandensysteem zijn ook mogelijk en een voorbeeld hiervan is (voor een UNIX bestandssysteem): *file:///home/erwin/public_html/thesis/tekst/index.xml*. Volgens de beschrijving van de FileGenerator moet het ook mogelijk zijn om XML bestanden van het web te halen en die als bron te gebruiken, door een URL te geven als waarde van het **src** attribuut, maar dat hebben we nog niet uitgeprobeerd.

De generator genereert dus SAX events die aan de volgende component in de pipeline worden doorgegeven. Als volgende component zien we een transformer, die gevolgd wordt door een andere transformer. De eerste transformer is een transformer die de naam xinclude draagt. Vooraleer we uitleggen waarom we deze transformer gebruiken, geven we eerst een korte uitleg over de fysieke structuur van onze tekst. Voor onze tekst hebben we één hoofdbestand, met name *index.xml*. Om de tekst overzichtelijk te houden hebben we per hoofdstuk een apart bestand gecreëerd. In het bestand *index.xml* verwijzen we volgens de [TRXINCL] standaard naar deze verschillende hoofdstukken. De XIncludeTransformer is een transformer die geschreven is om met deze include-statements om te gaan. Vooraleer we dus andere transformaties kunnen gaan toepassen op onze XML bron (in feite de SAX events), moeten deze statements eerst verwerkt worden. De xinclude transformer gaat deze statements verwerken. Het gevolg van de verwerking door deze transformer is dat in de volgende stap het juist lijkt alsof er met één groot document gewerkt wordt.

9.2.3.2.3.10.1. XInclude in de praktijk

We zullen hier een voorbeeldje geven van het gebruik van de [TRXINCL] standaard. Dit voorbeeldje is een gedeelte van het hoofdbestand van onze thesistekst. In onze thesis gebruiken we dit zodat we niet met één groot tekstbestand zouden zitten, maar we elk hoofdstuk in een apart bestand (of meerdere bestanden) kunnen opdelen.

Het hoofdbestand van onze thesis, *index.xml*, waarnaar wij verwijzen in onze sitemaps ziet er als volgt uit:

Voorbeeld 9.19. index.xml

```
<?xml version="1.0"?>
  <THESIS xmlns:xinclude="http://www.w3.org/2001/XInclude">
     <TITLE> Verkenning van XML en XSL - toepassing: gebruiksvriendelijke
     documentatie voor Java bestanden </TITLE>
     <TITLE language="en"> Exploration of XML and XSL - application: user
     friendly documentation for Java files </TITLE>
     ...

     <xinclude:include parse="xml" href="inleiding/inleiding.xml"/>
     <xinclude:include parse="xml" href="xml.xml"/>
     ...

     <xinclude:include parse="xml" href="bibliografie/bibliografie.xml"/>
  </THESIS>
```

We introduceren een nieuwe namespace bij het hoofdelement van dit bestand, met name *http://www.w3.org/2001/XInclude* en geven deze namespace de naam **xinclude**. Om te verwijzen naar de bestanden waar onze hoofdstukken in staan maken we gebruik van het **include** element dat tot deze namespace behoort. Wij gebruiken twee attributen van dit element. Het eerste attribuut, **parse** geeft aan hoe de resource, aangeduid via het **href** attribuut moet ingevoegd worden. Een waarde **xml** geeft aan dat de resource geparst moet worden als XML en samengevoegd moet worden met het verwijzende bestand om één geheel te vormen. Een waarde **text** geeft aan dat de resource ingevoegd moet worden als de samenstellende tekens. Het enige dat speciaal gebeurt is dat tekens die een speciale betekenis hebben in XML, zoals < en > omgezet worden naar entiteiten. Het parse attribuut is optioneel. Wanneer het niet wordt opgegeven moet een waarde **xml** verondersteld worden. Wij hebben het hier gebruikt om duidelijk aan te geven dat we willen dat de ingevoegde bestanden als XML worden behandeld.

Wanneer het parse attribuut de waarde xml heeft, moet het ingevoegde document recursief

behandeld worden. We bedoelen hiermee dat in het ingevoegde document ook gezocht moet worden naar **include** elementen die tot de namespace *http://www.w3.org/2001/XInclude* behoren en dan ook behandeld moeten worden. Lussen zijn niet toegelaten en het voorkomen van een lus resulteert in een fatale fout. De aanbeveling [TRXINCL] is nog veel uitgebreider, maar de bespreking die we hier gegeven hebben, volstaat voor het begrijpen van de verdere bespreking.

Praktisch gebeurt in Cocoon 2 het volgende:

- de xinclude transformer ontvangt SAX events
- SAX events die geen betrekking hebben op deze transformer zullen gewoon doorgegeven worden naar de volgende stap
- SAX events die wel betrekking hebben op deze transformer zullen als volgt behandeld worden
    - het gaat om een xinclude statement
    - de bron, waarnaar verwezen wordt vanuit een **include** element met behulp van het **href** attribuut, wordt opgehaald
    - afhankelijk van de waarde van het **parse** attribuut wordt het verwezen document behandeld als XML of als pure tekst, voor ons doel moet **parse** attribuut de waarde xml hebben
    - in dit geval verkrijgen we als eindresultaat een document waarin het include statement vervangen is door het verwezen document, maar dan in de vorm van SAX events

Dit klinkt allemaal mooi in theorie, maar in de praktijk hebben we gemerkt dat er van een recursieve behandeling geen sprake is. Tot onze grote spijt was het hierdoor niet mogelijk om een hoofdstuk in nog kleinere delen op te splitsen. Enkele weken voor de deadline hebben we een work-around gevonden voor dit probleem. In plaats van slechts éénmaal aan te geven dat het document door de XIncludeTransformer moet behandeld worden, kunnen we in de sitemap aangeven dat deze transformatie meerdere malen moet toegepast worden. We doen dit door de regel `<map:transform type="xinclude"/>` zo vaak te herhalen als nodig is. Het nadeel hieraan is dat je exact moet weten hoe diep deze elementen genest zijn.

Er is nog een ander nadeel verbonden aan deze aanpak. Het **href** attribuut geeft aan welke resource moet ingevoegd worden. Indien daarvoor absolute URIs gebruikt worden is er geen enkel probleem. Wanneer je echter relatieve URIs gaat gebruiken en je een bepaalde nestingsdiepte hebt, mag je echter niet vergeten dat de relatieve URI relatief is ten opzichte van de locatie van het hoofdbestand, waar de transformatie op toegepast wordt, althans bij de implementatie die deel uitmaakt van Cocoon 2. We hebben geen andere implementaties onderzocht. We zullen dit verduidelijken met een voorbeeldje. Stel dat je in een bepaalde directory de volgende structuur hebt opgebouwd:

- een directory *hoofdstuk1* met daarin de bestanden
    - *inleiding.xml*
    - *h1.xml*
    - *sectie1.xml*
    - *sectie2.xml*
    - *sectie3.xml*

- een bestand *book.xml*

In *book.xml* verwijzen we naar *hoofdstuk1/h1.xml*. Omdat de auteur van *h1.xml* het overzicht wil bewaren heeft hij/zij het hoofdstuk opgesplitst in een inleiding en drie secties. Het volledige boek genereren doen we door de juiste transformaties toe te passen op *book.xml*. Maar welke URI moet er in *h1.xml* gebruikt worden? We hebben de neiging om te zeggen dat

we gewoon *inleiding.xml*, *sectie1.xml* enzovoort mogen gebruiken, maar in de praktijk blijkt dit niet het geval te zijn. In *h1.xml* moeten we verwijzen naar *hoofdstuk1/inleiding.xml*, *hoofdstuk1/sectie1.xml* enzovoort. We vinden dit nogal verwarrend en eerlijk gezegd fout vermits de auteur van het document moet weten vanuit welke locatie zijn/haar hoofdstuk ingevoegd wordt en indien de tekst in een andere context herbruikt wordt veranderd moet worden terwijl dit niet nodig zou mogen zijn. De reden hiervoor is te vinden in [TRXMLBS4]. Wanneer we deze regels onderzoeken, vinden we dat de volgende regel in ons geval van toepassing is: *"2.The base URI is that of the encapsulating entity (message, document, or none)."*. Volgens ons is deze regel de oorzaak van heel wat dubbelzinnigheid. Welk document beschouw je als de inkapselende entiteit? Het hoofddocument *book.xml* of *hoofdstuk1/h1.xml*? Voor ons is het *hoofdstuk1/h1.xml* dat als inkapselende entiteit moet beschouwd worden, maar dat wordt blijkbaar niet zo gedaan. De reden hiervoor is dat in de huidige implementatie de XML inhoud gewoon wordt ingevoegd als XML, zonder de betekenis te onderzoeken. Volgens ons kan men dit probleem gemakkelijk oplossen door tijdens het invoegen het ingevoegde document te onderzoeken en indien er verwijzingen met relatieve URIs voorkomen, de relatieve URIs aan te passen. Door deze beperkingen hebben we een aantal opsplitsingen in onze tekst niet kunnen doorvoeren.

In theorie blijft de tekst op die manier goed beheerbaar, en blijft toch de eenvoud van werken behouden. Ook is het mogelijk om op die manier met twee personen tegelijkertijd aan de tekst te werken. Het nadeel is dat we nog geen implementaties hebben gevonden die dit ondersteunen en we dus creatief moeten zoeken naar work-arounds om in praktijk gedaan te krijgen wat in theorie mogelijk zou moeten zijn.

De volgende transformer in deze pipeline is de standaard transformer, in het geval van deze sitemap de XSLT transformer, die hier de naam xslt draagt. Zoals eerder gezegd moet dit niet opgegeven worden indien het om de als standaard gedefinieerde component gaat. Met het **src** attribuut geven we aan in welk bestand de transformaties zijn gedefinieerd die we willen toepassen op ons XML bestand. De XSLT transformer ontvangt de SAX events van de vorige stap en deze events vormen samen met de transformaties gedefinieerd in de aangegeven stylesheet de invoer van deze transformer. Intern worden de gedefinieerde transformaties toegepast op de SAX events die binnenkomen. Als uitvoer krijgen we een nieuwe stroom SAX events, een stroom die het resultaat is van het toepassen van de transformaties op de invoer. In deze pipeline gaat deze stroom van events naar een serializer.

De serializer is het eindpunt van deze pipeline. Het is de verantwoordelijkheid van de serializer om de SAX events die de uitvoer zijn van de vorige stap, om te zetten naar een bytestream of een characterstream die doorgestuurd kan worden naar de client. Dit element heeft hier geen attributen, wat dus wil zeggen dat we gebruik maken van de als standaard gedefinieerde serializer, in dit geval de HTMLSerializer. Cocoon 2 zal het document dan doorsturen naar de client en hierbij aan de client vertellen dat die zich aan een HTML document mag verwachten. Op die manier is de pipeline voltooid.

Voorbeeld 9.20.

```
<map:match pattern="overview.html">
   <map:generate src="index.xml"/>
   <map:transform type="xinclude"/>
   <map:transform src="stylesheets/overview.xsl"/>
   <map:serialize/>
</map:match>
```

Deze pipeline is exact dezelfde als de vorige pipeline, op twee kleine verschillen na. Deze pipeline zal uitgevoerd worden wanneer iemand het document opvraagt dat overeenkomt met

de URI *overview.html*. Het andere verschil is dat in deze pipeline andere transformaties toegepast worden, *overview.xsl* in plaats van *html.xsl*, wat hier ander eindresultaat zal opleveren. Voor de rest is deze pipeline identiek dezelfde aan de vorige pipeline, wat illustreert dat het heel gemakkelijk is om van dezelfde bron een heel ander eindresultaat te bekomen.

Voorbeeld 9.21.

```
<map:match pattern="index.pdf">
   <map:generate src="index.xml"/>
   <map:transform type="xinclude"/>
   <map:transform src="stylesheets/fopdf.xsl"/>
   <map:serialize type="fo2pdf"/>
</map:match>
<map:match pattern="index.ps">
   <map:generate src="index.xml"/>
   <map:transform type="xinclude"/>
   <map:transform src="stylesheets/fopdf.xsl"/>
   <map:serialize type="fo2ps"/>
</map:match>
<map:match pattern="index.fo">
   <map:generate src="index.xml"/>
   <map:transform type="xinclude"/>
   <map:transform src="stylesheets/fopdf.xsl"/>
   <map:serialize type="xml"/>
</map:match>
<map:match pattern="index.xml">
   <map:generate src="index.xml"/>
   <map:transform type="xinclude"/>
   <map:serialize type="xml"/>
</map:match>
```

We hebben hier vier verschillende pipelines die eigenlijk heel veel op elkaar gelijken. Vanuit één bronbestand genereren we met deze pipelines vier verschillende bestanden, met name een PDF versie, een PostScript versie, een FO versie en tenslotte een XML versie van de tekst. De XML versie bekomen we door na het toepassen van de xinclude transformatie de SAX events onmiddellijk door te spelen aan de XML serializer. Die is dan verantwoordelijk voor de omzetting van de stroom van SAX events in een XML bestand. De andere drie uitvoerformaten bekomen we telkens door dezelfde transformaties toe te passen, maar een andere serializer als eindpunt te kiezen. Uit deze voorbeelden mag het wel duidelijk zijn dat het heel gemakkelijk is om veel verschillende uitvoerformaten on-the-fly te genereren, terwijl er steeds maar één bronbestand is. Dit is een ideale manier om al deze bestanden consistent te houden maar toch een brede waaier aan bestandsformaten te kunnen aanbieden. Wijzigt namelijk het bronbestand, dan zal dit gereflecteerd worden in al deze uitvoerformaten.

Voorbeeld 9.22.

```
<map:match pattern="images/*.jpg">
   <map:generate src="images/{1}.svg"/>
   <map:serialize type="svg2jpeg"/>
</map:match>
```

Als laatste in deze voorbeeldsitemap hebben we een match element waarvan het **pattern** attribuut een wildcard bevat. De wildcard is in dit geval ***** en deze matcht elke string (ook de lege string), maar / (forward slash) mag geen deel uitmaken van deze string, i.e. *images/afbeelding1.jpg* zal wel gematcht worden, maar *images/afbeelding/1.jpg* zal niet gematcht worden. Om dit te verwezenlijken moet je ****** gebruiken. Deze wildcard zal zowel

voor *images/afbeelding1.jpg* als voor *images/afbeelding/1.jpg* matchen. Hetgeen gematcht wordt met de wildcard kan binnen de pipeline gebruikt worden, en dit door gebruik te maken van **{nummer van de wildcard}**. Het is zo dat er meerdere wildcards gebruikt kunnen worden, waarbij deze wildcards van links naar rechts worden genummerd en de meest linkse wildcard draagt het nummer 1.

We hebben nu van een aantal componenten voorbeelden gegeven, maar we hebben nog geen voorbeeld gegeven van het gebruik van een reader of van een selector. We zullen dat nu doen en we zullen met de reader beginnen.

Voorbeeld 9.23.

```
<map:match pattern="**/img/**.jpg">
   <map:read src="img/{2}.jpg" mime-type="image/jpg"/>
</map:match>
```

We hebben hier een reader geconfigureerd om de requests voor jpg-afbeeldingen af te handelen. In dit voorbeeld is ook het gebruik van meerdere wildcards te zien, maar het belangrijkste is het feit dat er bij een reader geen generator of serializer te zien is. Zoals reeds eerder gezegd is een reader eigenlijk een pipeline op zich. Deze reader zal de gevraagde afbeelding ophalen en doorsturen naar de client en dat is dan ook het enige wat er gebeurt. Op die manier kunnen ook resources aangeboden worden die geen bewerkingen dienen te ondergaan. Dit is ook het geval voor Cascading Style Sheets en tal van andere formaten.

Het doel van deze pipeline is dat er op eender welke diepte een bestand met de extensie jpg in een img directory wordt opgehaald uit een directory img, relatief ten opzichte van de locatie van de sitemap. In deze directory willen we het bestand ophalen op dezelfde diepte als het in de request onder de img directory ligt. Hiermee willen we aantonen dat het zeker niet zo is dat alle gematchte wildcards gebruikt moeten worden. Zo maken we hier geen gebruik van het patroon dat overeenkomt met de variabele **{1}**.

Voorbeeld 9.24.

```
<map:match pattern="cocoondemo/browsers.html">
   <map:generate src="cocoondemo/browsers.xml"/>
   <map:select type="browser">
      <map:when test="netscape">
         <map:transform src="cocoondemo/netscape.xsl"/>
      </map:when>
      <map:when test="lynx">
         <map:transform src="cocoondemo/lynx.xsl"/>
      </map:when>
      <map:otherwise>
         <map:transform src="cocoondemo/other.xsl"/>
      </map:otherwise>
   </map:select>
   <map:serialize/>
</map:match>
```

In dit voorbeeld is te zien hoe een gedeelte van de pipeline kan afhangen van een selector. We laten bij deze pipeline de transformaties, die toegepast moeten worden op de XML bron, afhangen van de browser waarmee deze pagina opgevraagd wordt.

Tot slot willen we nog vermelden dat de volgorde waarin de verschillende **map:match** elementen voorkomen wel degelijk van belang is. Als een bepaalde request op twee of meerdere manieren kan gematcht worden, dan zal Cocoon 2 kiezen voor de match die het

eerst gedefinieerd is. We zullen dit eventjes illustreren met twee voorbeeldjes waarbij het enige verschil bestaat uit de andere volgorde van de **map:match** blokken.

Voorbeeld 9.25.

```
<map:match pattern="**.html">
   <map:generate src="content/{1}.xml"/>
   <map:serialize/>
</map:match>
<map:match pattern="index.html">
   <map:generate src="index.xml"/>
   <map:serialize/>
</map:match>
```

Voorbeeld 9.26.

```
<map:match pattern="index.html">
   <map:generate src="index.xml"/>
   <map:serialize/>
</map:match>
<map:match pattern="**.html">
   <map:generate src="content/{1}.xml"/>
   <map:serialize/>
</map:match>
```

Indien er een request komt voor *index.html* zal dat in het eerste voorbeeldje altijd gebeuren via het patroon van het eerste **map:match** element. Er zal in dat geval nooit gebruik gemaakt worden van het patroon van het tweede **map:match** element. Indien je daar niet van op de hoogte bent, kan dit aanleiding geven tot heel wat zoektochten waar de fout zit. De fout die in het eerste voorbeeldje gemaakt is, is dat het meest algemene patroon om te matchen als eerste gedefinieerd is en het meest specifieke pas daarna. De correcte manier van werken vinden we terug in het tweede voorbeeldje, waar het meest specifieke patroon als eerste wordt gedefinieerd, gevolgd door een meer algemeen patroon.

9.2.3.2.3.11. Subsitemaps

Het is duidelijk dat er met een sitemap (zelfs na deze korte uitleg) heel wat mogelijk is. Met deze uitgebreidheid komt ook een nadeel om de hoek kijken. Indien je met een hele grote site werkt, of met verschillende kleine sites, wordt het beheren van deze sites moeilijker naarmate het aantal aan te bieden documenten groeit. Alles in één sitemap definiëren geeft een bijna onmogelijke opgave voor het onderhoud. Gelukkig hebben de ontwerpers van Cocoon 2 bij deze problematiek stilgestaan en heeft men de notie van subsitemaps ingevoerd. We geven hier eerst een voorbeeld waarin naar subsitemaps verwezen wordt en we zullen dit voorbeeld dan verder uitleggen.

Voorbeeld 9.27.

```
<map:pipelines>
   <map:pipeline>
      <map:match pattern="thesis/**">
         <map:mount uri-prefix="thesis" src="thesis/"
         check-reload="yes"/>
      </map:match>
      <map:match pattern="reviews/**">
         <map:mount uri-prefix="reviews" src="reviews/"
         check-reload="yes" reload-method="synchron"/>
```

```
        </map:match>
        <map:match pattern="softdoc/**">
           <map:mount uri-prefix="softdoc" src="softdoc/"
           check-reload="yes" reload-method="synchron"/>
        </map:match>
        <map:handle-errors type="404">
           <map:transform src="../stylesheets/error2html.xsl"/>
           <map:serialize/>
        </map:handle-errors>
     </map:pipeline>
     <map:pipeline>
        <map:match pattern="calendar/**">
           <map:mount uri-prefix="calendar" src="calendar/"
           check-reload="yes" reload-method="yes"/>
        </map:match>
        <map:handle-errors>
           <map:transform src="stylesheets/system/error2html.xsl"/>
           <map:serialize status-code="500"/>
        </map:handle-errors>
     </map:pipeline>
  </map:pipelines>
```

In dit voorbeeld heeft het **pipelines** element twee **pipeline** kindelementen. Voor de uitleg over het nut hiervan verwijzen we naar [Sectie 9.2.3.2.3.12.] . Binnen het eerste **pipeline** element worden drie subsitemaps gemount, waarvan we de eerste gaan nemen om onze uitleg te geven.

De sitemap waaruit deze voorbeelden komen, zorgt voor het afhandelen van de URIs die beginnen met *http://host/cocoon/tekst/*. Indien er binnen die ruimte een request is voor een URI die met *thesis/* begint, dan moet er gekeken worden naar het bestand sitemap.xmap in de map thesis/. Dat is de betekenis van het **map:mount** element en het **src** attribuut. Het element geeft aan dat voor de selectie van de juiste pipeline naar een andere sitemap moet gekeken worden. De waarde van het **src** attribuut geeft aan waar deze sitemap zich bevindt. Eindigt de waarde van het **src** attribuut met een /, dan plakt Cocoon 2 hier automatisch de bestandsnaam sitemap.xmap aan vast en gaat dan op zoek naar dat bestand. Eindigt deze waarde niet met een /, dan gaat Cocoon 2 gewoon op zoek naar dat bestand. Indien het bestand niet bestaat zal dit resulteren in een foutmelding of een blanco pagina (afhankelijk van de gebruikte versie van Cocoon 2).

In de sitemap die gaat aangesproken worden, zijn natuurlijk ook de patronen gespecificeerd waarop gematcht moet worden. Normaal gezien wordt als request URL bijvoorbeeld *thesis/index.html* doorgegeven. Maar om in de sitemap, die instaat voor de URI-ruimte *http://host/cocoon/tekst/thesis/*, niet steeds **thesis** te moeten vermelden in elk match pattern is het mogelijk om aan te geven dat Cocoon 2 een gedeelte van het begin van de request URL moet strippen. Dit wordt aangegeven met het **uri-prefix** attribuut. Cocoon 2 zal de waarde van dit attribuut controleren op een eindigende / en indien die niet aanwezig is zal die eraan toegevoegd worden. In het voorbeeld wil dit zeggen dat, alhoewel wij als waarde van het **uri-prefix** attribuut **thesis** hebben opgegeven, Cocoon 2 steeds van de request URI **thesis/** zal verwijderen en dan pas beginnen te matchen met de patronen gespecificeerd in de subsitemap.

Buiten de verplichte attributen **uri-prefix** en **src** zijn er nog twee optionele attributen voor het element **map:mount**, met name de attributen **check-reload** en **reload-method**. Met het **check-reload** attribuut wordt aangegeven of Cocoon 2 de wijzigingsdatum van de subsitemap moet controleren. Dit attribuut kan een waarde yes/true of no/false hebben. Indien er een wijziging is gebeurd, dan zal Cocoon 2 in dat geval, indien het attribuut de waarde yes of true heeft, de sitemap opnieuw inlezen. Standaard staat de waarde op no. Met het attribuut

reload-method kan je aangeven hoe dit moet gebeuren. Dit attribuut kan de waarden **asynchron** (de standaard waarde) of **synchron** aannemen. Indien de methode asynchron is, dan zal Cocoon 2 de sitemap wel gaan herladen en op de achtergrond gaan verwerken, maar zolang de verwerking nog niet volledig afgehandeld is, zal Cocoon 2 de requests afhandelen met de versie van de sitemap die in het geheugen zit. Is de methode synchron dan zal Cocoon 2, bij de detectie van een wijziging, eerst de subsitemap volledig gaan verwerken vooraleer requests gaan afgehandeld worden. Dit is ideaal voor testdoeleinden, terwijl de eerste methode eerder geschikt is voor een echte website (alhoewel hierover stevig gediscussieerd kan worden). In beide gevallen gaat Cocoon 2 de wijzigingen pas detecteren bij de eerste request die plaatsvindt nadat de wijzigingen zijn aangebracht en die betrekking heeft op die sitemap.

Wat we hier tot nu toe nog niet vermeld hebben maar zeker niet minder belangrijk is het feit dat in een subsitemap gebruik gemaakt kan worden van de componenten die op een hoger niveau gedefinieerd zijn. Dit wil niet zeggen dat er geen **map:components** element met zijn kindelementen meer moet gedefinieerd zijn in de subsitemap. In het volgende voorbeeldje geven we weer wat minimaal moet gedefinieerd zijn.

Voorbeeld 9.28.

```
<map:components>
   <map:generators default="file"/>
   <map:transformers default="xslt"/>
   <map:readers default="resource"/>
   <map:serializers default="html"/>
   <map:selectors default="browser"/>
   <map:matchers default="wildcard"/>
</map:components>
```

Zoals uit dit voorbeeld blijkt moeten de componenten die gebruikt kunnen worden niet meer geherdeclareerd worden en kunnen ze zelfs gebruikt worden met de namen die ze in een andere sitemap, die hoger staat in de hiërarchie, gekregen hebben. Wel moet er voor elk type van de hier aangegeven componenten herhaald worden welke als default geldt.

9.2.3.2.3.12. Meerdere pipeline elementen

Zoals in het voorbeeld van de subsitemaps duidelijk werd, is het mogelijk om meerdere **pipeline** elementen te hebben binnen één **pipelines** element. Dit lijkt misschien niet nuttig, vermits de echte pipeline eigenlijk binnen het **match** element gedefinieerd wordt. Om het nut van meerdere pipeline elementen aan te tonen willen we eerst vermelden dat, zoals in het voorbeeld van de subsitemaps, het mogelijk is een error-handler per pipeline te definiëren. Dit wil zeggen dat de fouten binnen het ene **pipeline** element anders kunnen afgehandeld worden dan fouten binnen een ander **pipeline** element. Dit is een eerste reden waarom het gebruik van meerdere pipeline elementen mogelijk gemaakt is.

Een andere reden om meerdere **pipeline** elementen te hebben binnen één **pipelines** element is dat zo een **pipeline** element een attribuut **internal-only** kan hebben. Met dit attribuut kan aangegeven worden dat alles wat binnen dat element gedefinieerd is, alleen intern voor Cocoon zichtbaar is. We zullen nu een voorbeeld geven van het gebruik van dit attribuut en hoe dit kan toegepast worden in de sitemap.

Voorbeeld 9.29.

```
<map:pipelines>
   <map:pipeline internal-only="true">
```

```
        <map:match pattern="index">
           <map:generate src="index.xml"/>
           <map:transform type="xinclude"/>
        </map:match>
     </map:pipeline>
     <map:pipeline>
        <map:match pattern="index.html">
           <map:generate src="cocoon:/index"/>
           <map:transform src="stylesheets/html.xsl"/>
           <map:serialize/>
        </map:match>
        <map:match pattern="index.pdf">
           <map:generate src="cocoon:/index"/>
           <map:transform src="stylesheets/fopdf.xsl"/>
           <map:serialize type="fo2pdf"/>
        </map:match>
     </map:pipeline>
  </map:pipelines>
```

In dit voorbeeld hebben we duidelijk twee **pipeline** elementen en bij één van deze **pipeline** elementen hebben we het attribuut **internal-only** de waarde true gegeven. Deze pipeline met relatieve URL *index* is niet bereikbaar van buitenaf, dit zou in dit geval trouwens geen zin hebben omdat deze pipeline niet eindigt met een serializer. In het andere **pipeline** element zijn er twee mogelijkheden voorzien om een bestand op te vragen. Als bron van de generator wordt daar met behulp van **cocoon:/** naar een pipeline van de huidige sitemap verwezen. Op die manier is het mogelijk om een gemeenschappelijk gedeelte van veel pipelines op een welbepaalde plaats te definiëren en er naar te verwijzen wanneer nodig. Om dan deze gedeeltelijke pipelines niet van buitenaf toegankelijk te maken kunnen ze binnen een **pipeline** element gedefinieerd worden waarvan het **internal-only** de waarde true heeft gekregen. Deze gedeeltelijke pipelines kunnen dan in de sitemaps gebruikt worden.

9.2.4. Vergelijking met Cocoon 1

We hebben de kans gehad om zowel met Cocoon 1 als met Cocoon 2 te kunnen werken en dit heeft ons een schat aan ervaring en begrip voor deze architecturen opgeleverd. Met behulp van deze ervaringen kunnen we deze twee frameworks op een aantal punten vergelijken. Het is niet de bedoeling om hier een volledige vergelijking te geven van deze twee frameworks, maar om een aantal vergelijkingspunten aan te geven die ons opgevallen zijn bij het gebruik. In de volgende paragrafen gaan we enkele verschilpunten aanhalen en die kort bespreken.

Het **integreren** van Cocoon 1 **met een webserver**, Apache in ons geval, is veel makkelijker dan de integratie van Cocoon 2 met dezelfde webserver. Beide frameworks worden bij installatie geïntegreerd met een servlet engine en deze engine moet geïntegreerd worden met de webserver. Bij Cocoon 1 moesten we slechts één regel toevoegen aan het configuratiebestand van Apache, deze webserver herstarten en de integratie was compleet. In het geval van Cocoon 2 moesten we voor de architectuur waar wij op werken zelf een module compileren die zorgt voor de integratie van de servlet engine in Apache. Er moesten een aantal aanpassingen gemaakt worden aan het compilatie-script en aan de configuratiebestanden vooraleer de integratie compleet was. Dit verschil in complexiteit van installatie wordt vooral veroorzaakt door de servlet engines, maar om als webserver gebruikt te kunnen worden, vereisen deze frameworks wel een integratie met zulke servlet engines.

Ook de **installatie** van Cocoon 1 (versie 1.8.2) zelf verliep iets makkelijker dan de installatie van Cocoon 2 (versie 2.0RC2). Het enige probleem bij Cocoon 1 was dat we zelf naar alle benodigde onderdelen hebben moeten zoeken, terwijl bij Cocoon 2 eigenlijk alles (op de

servlet engine na) meegeleverd wordt. Bij de aangegeven versie van Cocoon 2 waren nog een aantal andere problemen bij het compileren, die veroorzaakt werden door fouten in de installatiescripts (onder Linux) en er werd ook geen rekening gehouden met een beperkte omgevingsruimte (Windows), maar na wat speurwerk hebben we deze problemen toch kunnen oplossen. Op één van de computers waar we Cocoon 2 probeerden te draaien, bleven we onduidelijke foutmeldingen krijgen. De oorzaak van deze foutmeldingen was uiteindelijk terug te voeren naar een (kort weergegeven) **java.lang.UnsatisfiedLinkError : libawt.so: XpGetDocumentData: symbol not defined**. Na rondvragen bleek dit meer dan waarschijnlijk te liggen aan een verkeerde versie van de openmotif bibliotheek. Maar de versie die geïnstalleerd was (versie 2.1.30-3) was volgens dezelfde mensen een versie die recent genoeg was hiervoor. Na het upgraden van deze bibliotheek naar versie 2.1.30-8 bleek het probleem opgelost te zijn, maar het heeft toch enkele weken geduurd vooraleer we hier achter kwamen. Het lijkt misschien raar dat een webserver gebruik maakt van grafische bibliotheken, maar een onderdeel van de Cocoon 2 distributie, met name de batik bibliotheek, heeft (onder Linux) een connectie naar een Xserver nodig en vandaar kwam die foutmelding. Batik is een bibliotheek die zorgt voor de omzetting van svg naar jpeg en heeft hierdoor enkele grafische bibliotheken nodig. In versie 2.0.2 van Cocoon is batik een optionele component geworden en op dit moment staat ook een oplossing op de site indien je toch batik wil gebruiken en geen Xserver wil draaien. Dit is er echter pas gekomen nadat we met deze problemen geconfronteerd werden en ze opgelost hadden.

Ook in het **gebruik** is er een aanmerkelijk verschil tussen Cocoon 1 en Cocoon 2. Het sitebeheer in Cocoon 2 gebeurt aan de hand van sitemaps. Dat zorgt voor een makkelijke scaleerbaarheid en een eenvoudig onderhoud van de site. Ook staan er in de XML bestanden bij Cocoon 2 geen verwijzingen meer naar de te gebruiken stylesheets. Dat is allemaal verplaatst naar de sitemap. Het aanbieden van verschillende formaten wordt ook door de sitemap fel vereenvoudigd. Bij Cocoon 1 daarentegen wordt een gedeelte van de logica (welke transformaties, welk uitvoerformaat, ...) in het XML document zelf geplaatst met behulp van Processing instructions. Ook is het heel wat moeilijker, om niet te zeggen onmogelijk, om vanuit één brondocument verschillende uitvoerformaten te genereren. Het grote voordeel van Cocoon 2 tegenover Cocoon 1 is, volgens ons, dat er in de XML bestanden geen verwijzingen meer moeten staan naar de XSLT stylesheets en dat er ook niks meer over het uitvoerformaat dient gezegd te worden. Deze verwijzingen hebben ons problemen opgeleverd bij een, wat de structuur betreft, nogal ingewikkelde site. Zonder symbolic links of dataduplicatie (wat voor inconsistenties zorgt) was dit niet op te lossen. Het overschakelen naar Cocoon 2 was een hele verademing omdat op één centrale plaats kon gedeclareerd worden welke stylesheets op welke XML bestanden toegepast moeten worden en we deze verwijzingen konden laten vallen uit de XML bestanden. Het nadeel van Cocoon 2 is dan weer dat, indien geen voorzorgen worden genomen op het niveau van de webserver, er **altijd** een gedeelte *cocoon/* terug te vinden is in de URL. Dit is niet het geval bij Cocoon 1.

Over de **foutmeldingen** van Cocoon 2 waren we, in vergelijking met de foutmeldingen van Cocoon 1, niet echt te spreken. Alhoewel de foutmeldingen in beide gevallen nogal cryptisch waren, zijn ze bij Cocoon 1 toch iets duidelijker dan bij Cocoon 2. Op dit moment zijn de ontwikkelaars van Cocoon 2 bezig met het begrijpbaar maken van de foutmeldingen en proberen de exacte locatie van de fout aan te geven.

In onze testen bleek Cocoon 1 ook **sneller** te zijn dan Cocoon 2, maar dat is alleen zo bij de eerste requests, vermits Cocoon 2 dan nog heel wat werk te verrichten heeft. Maar omdat wij daarna de cache van Cocoon 2 hadden uitgeschakeld, bleef Cocoon 2 vanaf dan consistent trager dan Cocoon 1. Dit uitschakelen van de cache is enkel aan te raden wanneer je bezig bent met de ontwikkeling van de site en niet wanneer de site op het World Wide Web wordt losgelaten.

9.2.5. Conclusie

Cocoon 2 is een krachtig, flexibel XML publishing framework dat erin slaagt op een algemene manier aan web publishing te doen vertrekkende vanuit XML inhoud. Maar Cocoon 2 is nog lang niet volwassen, getuige daarvan de vele wijzigingen die nog zijn doorgevoerd tussen versie 2.0.1 en 2.0.2. Deze versies zijn wel al volwassen geworden, maar de ontwikkelaars beseffen dat er ook nog veel werk is, vooral op het gebied van documentatie. Zo hadden wij nog enkele vragen in verband met de sitemap. Deze vragen kwamen naar voren toen we de DTD van de sitemap hadden bekeken, maar deze DTD was ondertussen nogal verouderd zodat een aantal van onze vragen terecht waren, maar we er ons onnodig ongerust over hadden gemaakt. Vooral de sitemap is een heel krachtig concept, omdat zo het beheer van een site kan opgesplitst worden. Ondanks de doelstelling van het team om het met behulp van Cocoon 2 mogelijk te maken om grote documenten in een kleine heapsize te verwerken, is het ons niet gelukt dit te bevestigen. Volgens onze vaststellingen was dit te wijten aan het gebruik van FOP, dat veel geheugen vereist. De slotsom is dat Cocoon 2 een heel krachtig framework is dat nog niet volledig volwassen is, maar er toch niet ver vanaf is.

10. Gebruik van Cocoon 2

In dit hoofdstuk gaan we kort het gebruik van Cocoon 2 in het kader van onze thesis bespreken. We gaan het belangrijkste gedeelte van de door ons gebruikte sitemap behandelen. Op die manier kunnen we laten zien dat we veel verschillende uitvoerformaten kunnen aanbieden vertrekkende vanuit één bronbestand. Indien we het bronbestand aanpassen heeft dit een onmiddellijke weerslag op alle uitvoerformaten. Dit biedt het grote voordeel dat alle documenten op elk moment consistent blijven.

10.1. De sitemap

In onze configuratie bevindt de hoofdsitemap **sitemap.xmap** [Sectie 9.2.3.2.3.] zich in de directory $TOMCAT_HOME/webapps/cocoon/. De sitemap die wij willen gebruiken en alle bestanden van onze thesis staan echter in de directory /home/erwin/public_html/thesis/tekst/thesis/. Om dit op te lossen moeten we gebruik maken van het subsitemap-mechanisme [Sectie 9.2.3.2.3.11.] van Cocoon 2.

In de hoofdsitemap hebben we daartoe het volgende toegevoegd (onder het map:pipelines element zoals uit het voorbeeld blijkt):

Voorbeeld 10.1.

```
...
 <map:pipelines>
    <map:pipeline>
       ...

       <map:match pattern="thesis/**">
          <map:mount uri-prefix="thesis"
          src="file:///home/erwin/public_html/thesis/tekst/thesis/"
          check-reload="yes" reload-method="synchron"/>
       </map:match>
       <map:handle-errors type="404">
          <map:transform src="stylesheets/error2html.xsl"/>
          <map:serialize/>
       </map:handle-errors>
    </map:pipeline>
    ...

 </map:pipelines>
```

Op deze manier wordt het beheer van de URI-ruimte in *http://localhost:8080/cocoon/thesis/* afgehandeld door de sitemap in de directory /home/erwin/public_html/thesis/tekst/thesis/.

De sitemap die wij in de directory /home/erwin/public_html/thesis/tekst/thesis/ hebben opgesteld, heeft de volgende vorm:

Voorbeeld 10.2. sitemap.xmap

```
<?xml version="1.0"?>
 <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
    <map:components>
       <map:generators default="file"/>
       <map:transformers default="xslt"/>
       <map:readers default="resource"/>
       <map:serializers default="html"/>
       <map:selectors default="browser"/>
```

```
      <map:matchers default="wildcard"/>
    </map:components>
    <map:pipelines>
      <map:pipeline internal-only="true">
        <map:match pattern="index">
          <map:generate src="index.xml"/>
          <map:transform type="xinclude"/>
          <map:transform type="xinclude"/>
        </map:match>
      </map:pipeline>
      <map:pipeline>
        <map:match pattern="index.html">
          <map:generate src="cocoon:/index"/>
          <map:transform src="stylesheets/html.xsl"/>
          <map:serialize/>
        </map:match>
        <map:match pattern="toc.html">
          <map:generate src="cocoon:/index"/>
          <map:transform src="stylesheets/overview.xsl"/>
          <map:serialize/>
        </map:match>
        <map:match pattern="index.pdf">
          <map:generate src="cocoon:/index"/>
          <map:transform src="stylesheets/fopdf.xsl"/>
          <map:serialize type="fo2pdf"/>
        </map:match>
        <map:match pattern="index.fo">
          <map:generate src="cocoon:/index"/>
          <map:transform src="stylesheets/fopdf.xsl"/>
          <map:serialize type="xml"/>
        </map:match>
        ...

        <map:match pattern="images/**.svg">
          <map:generate src="images/{1}.svg"/>
          <map:serialize type="svg2jpeg"/>
        </map:match>
        <map:match pattern="images/**.gif">
          <map:read src="img/{1}.gif" mime-type="image/gif"/>
        </map:match>
        <map:match pattern="images/**.jpg">
          <map:read src="img/{1}.jpg" mime-type="image/jpeg"/>
        </map:match>
        ...

      </map:pipeline>
    </map:pipelines>
  </map:sitemap>
```

Bemerk dat alle hier gespecificeerde uitvoerbestanden (op de grafische formaten na) allemaal uit dezelfde bron gegenereerd worden en hoe we gebruik maken van een interne pipeline om deze bron te specificeren [Sectie 9.2.3.2.3.12.] . Voor uitleg over het gebruik van de XIncludeTransformer verwijzen we naar [Sectie 9.2.3.2.3.10.1.] . Door het toepassen van andere transformaties of door de keuze van een andere serializer [Sectie 9.2.3.2.3.6.] kunnen we het uitvoerformaat beïnvloeden.

We gebruiken ook drie soorten grafische formaten. De formaten GIF en JPEG worden gewoon doorgegeven, op deze formaten gebeuren geen bewerkingen. We maken ook gebruik van SVG. Een SVG-afbeelding zal omgezet worden naar een JPEG afbeelding die aan de client zal afgeleverd worden. Bemerk dat dit onderscheid gemaakt wordt op basis van de extensies. Het is (nog) niet mogelijk om het formaat automatisch te laten detecteren.

Deze sitemap is in feite vrij eenvoudig, maar het is een zeer krachtig instrument voor het

beheren van sites. De sitemap zorgt ervoor dat de XML bestanden niet meer "vervuilt" worden met verwijzingen naar stylesheets. Het is op die manier perfect mogelijk geworden om, vertrekkende van één bronbestand, tien verschillende einddocumenten af te leveren. Het resultaat hangt gewoonweg af van de toegepaste transformaties.

10.2. Valkuilen

Bij het gebruik van Cocoon 2 zijn we enkele valkuilen tegengekomen die te maken hebben met **caching**. De eerste valkuil had betrekking op het cachen van XML documenten terwijl de twee valkuil betrekking had op het cachen van de stylesheets. We zullen dit kort behandelen en de oplossing hiervoor aanreiken. Deze behandeling gaat over onze situatie en over het XML bestanden als bron. Het zou kunnen dat de caching bij andere bronnen op een andere manier gebeurd.

We hadden echter ook nog een probleem met de combinatie Tomcat/Cocoon 2. In het begin is het een paar keer gebeurd dat we de hoofdsitemap van Cocoon 2 hadden aangepast, maar dat deze wijzigingen blijkbaar door Cocoon 2 genegeerd werden, zelfs na het herhaalde malen opnieuw starten van Tomcat/Cocoon 2. Het probleem is dat Cocoon 2 een gecompileerde versie van de sitemap bijhoudt in de directory $TOMCAT_HOME/work/. Indien deze gecompileerde versie aanwezig is, dan wordt blijkbaar deze gecompileerde versie geladen in plaats van de hoofdsitemap. Ook wijzigingen in de configuratie waren door deze strategie niet zichtbaar voor Cocoon 2. Het leegmaken van deze directory bij wijzigingen in de configuratie en in de hoofdsitemap bleek dit op te lossen. Tomcat/Cocoon 2 moet wel herstart worden.

Het tweede probleem dat we tegenkwamen had te maken met het cachen van de bronbestanden. Na het aanbrengen van enkele wijzigingen trachtten we de HTML versie te herladen in onze browser. We kregen de wijzigingen echter niet te zien. Dit bleek te liggen aan de caching strategie van Cocoon 2. Eénmaal het bestand ingelezen is en het resultaat "berekend", wordt dit resultaat in het geheugen bijgehouden. Wanneer dan iemand deze pagina opvraagt wordt gekeken of het resultaat in het geheugen zit. Indien het resultaat in het geheugen zit, wordt dit resultaat doorgestuurd. Zit het resultaat niet in het geheugen, dan wordt dit "berekend". Indien het resultaat in het geheugen zit, wordt niet gekeken of het bronbestand ondertussen al gewijzigd is. Dit is zeer vervelend.

De oplossing hiervoor is het uitschakelen van de cache. Het configuratiebestand bevindt zich in de directory $TOMCAT_HOME/webapps/cocoon/WEB-INF/ en heet *cocoon.xconf*. Bij oudere versies van Cocoon 2 bevindt dit bestand zich in de directory $TOMCAT_HOME/webapps/cocoon/. Om de cache uit te schakelen moeten de volgende regels

```
<stream-pipeline
class="org.apache.cocoon.components.pipeline.CachingStreamPipeline"
  logger="core.stream-pipeline" pool-grow="4" pool-max="32" pool-min="8"/>
<event-pipeline class="org.apache.cocoon.components.pipeline.CachingEventPipeline"
  logger="core.event-pipeline" pool-grow="4" pool-max="32" pool-min="8"/>
```

gewijzigd worden in

```
<stream-pipeline
class="org.apache.cocoon.components.pipeline.NonCachingStreamPipeline"/>
<event-pipeline
class="org.apache.cocoon.components.pipeline.NonCachingEventPipeline"/>
```

Dit zorgt voor het uitschakelen van de cache. Na het herstarten van Cocoon 2 (en het leegmaken van de werkdirectory) bleek de cache uitgeschakeld en waren opgeslagen wijzigingen in het bronbestand onmiddellijk zichtbaar bij het herladen van de HTML pagina

of het PDF bestand.

Naarmate de thesis vorderde en er meer en meer elementen bijkwamen, werden de stylesheets opgesplitst. We hebben één hoofdstylesheet gemaakt die de andere stylesheets importeert. Deze hoofdstylesheet bevat de hoofdtemplate en eventueel enkele andere kleine templates. De andere stylesheets zijn opgesplitst per thema: zo is er één stylesheet voor de elementen die met Java te maken hebben, een andere stylesheet voor de elementen die met DTD's te maken hebben, ...

We vonden het eigenaardig dat, indien we aan een van deze gespecialiseerde stylesheets wijzigingen aanbrachten, dit niet gereflecteerd werd in de uitvoerbestanden. Dit was zeker eigenaardig, want we hadden de cache uitgeschakeld, dus daar kon het niet meer aan liggen. Indien we een wijziging aanbrachten in de hoofdstylesheet, dan waren ineens de wijzigingen in de andere stylesheets ook zichtbaar. Hierbij maakte het geen verschil uit of we **xsl:import** of **xsl:include** [Sectie 5.3.1.14.] gebruikten om te verwijzen naar de andere stylesheets. Het bleek dat dit een implementatiebeslissing was die door meerdere gebruikers vervelend werd gevonden [C2USML]. De enige oplossing hiervoor is de hoofdstylesheet wijzigen (door bijvoorbeeld het invoegen van een extra witregel) of Tomcat/Cocoon 2 herstarten. De ontwikkelaars denken erover na om dit mechanisme te wijzigen [C2DEVML].

10.3. Besluit

Cocoon 2 is een zeer bruikbaar framework en laat toe om het beheer van een site op te delen in verschillende kleinere delen (het concept van de sitemap en de subsitemap). Ook is Cocoon 2 bijzonder krachtig wanneer het aankomt op het aanbieden van verschillende formaten, uitgaande van één bronbestand. Deze formaten worden on-the-fly gecreëerd wat ervoor zorgt dat de verschillende formaten consistent zijn.

Naarmate de ontwikkeling van Cocoon 2 vordert, worden ook de foutmeldingen duidelijker. Indien je bij versie **2rc1a** nog gewoon de melding kreeg dat er "ergens" een element niet correct afgesloten was, krijg je bij versie **2.1.0-dev** ook het regelnummer waar de fout zich ongeveer situeert. Dit is al een hele verbetering en spaart soms uren zoekwerk uit.

Hét grote nadeel aan Cocoon 2 is (het gebrek aan) documentatie. Hierdoor was het zeer vaak nodig zelf in de code te duiken of gewoon rond te kijken in de verschillende andere bestanden in de Cocoon 2 directory. Na een beetje zoeken leverde dit vaak resultaat op, maar we vinden dat dit toch iets beter kan. Op dit moment is er op de mailinglist van de ontwikkelaars [C2DEVML] een discussie over hoe het documentatieproject voor Cocoon 2 het beste zou aangepakt worden. We hopen dat de resultaten hiervan snel merkbaar zullen zijn.

11. JDOM

In dit hoofdstuk behandelen wij JDOM, een vrij recente Application Programming Interface, speciaal ontworpen om XML gegevens te gebruiken en te manipuleren vanuit Java code. Wij hebben JDOM bestudeerd en vervolgens ingebouwd in het jnome project [JNOME] [Hoofdstuk 13.] . jnomedoc verwerkt Java code, bouwt daarvan een metamodel en genereert vervolgens XML documentatie voor deze code. De XML documentatie van jnomedoc wordt nu met behulp van JDOM gegenereerd.

11.1. Wat is JDOM?

JDOM [JDOM] is een nieuwe API (Application Programming Interface) om XML documenten te lezen, te schrijven en te bewerken vanuit Java [JAVA] code. Het is speciaal ontworpen voor Java ontwikkelaars om XML documenten en hun inhoud op een intuïtieve manier voor te stellen. JDOM is een Java toolkit die de snelle ontwikkeling van XML applicaties mogelijk maakt. Zoals de naam al laat uitschijnen, is JDOM een DOM-variant [Sectie 12.1.] , [TRDOM], maar dan ontworpen volgens de Java syntax en de Java semantiek. JDOM gedraagt zich zoals Java, het maakt gebruik van onder andere het overladen van methodes, Java **Collections** en Java's ingebouwde **String** ondersteuning. Bovendien houdt het de kosten om XML te gebruiken laag. Het is niet nodig dat JDOM gebruikers XML-experten zijn om toch produktief te zijn en hun werk gedaan te krijgen.

JDOM is ontworpen door Brett McLaughlin en Jason Hunter. Zij vonden dat het gebruik van XML intuïtief, gemakkelijk en produktief moest zijn voor Java ontwikkelaars. Begin 2000 werd JDOM ingehuldigd als een open source project onder een licentie gelijkaardig aan de Apache licentie [APACHE]. Deze licentie is één van de minst beperkende software licenties. Dit maakt het programmeurs mogelijk JDOM te gebruiken om eigen produkten te creëren zonder deze produkten zelf als open source vrij te moeten geven. Door de open source licentie kan JDOM profiteren van veel bijdragen, feedback en bug fixes van Java programmeurs en ontwikkelaars en kan het zich snel aanpassen aan nieuwe standaarden. In april van 2001 brachten Jason Hunter en Brett Mclaughin een beta vorm van hun API uit. Het einddoel is een volledige op het Java-platform gebaseerde oplossing te verkrijgen die toelaat XML gegevens te gebruiken en te manipuleren vanuit Java code. [WBHEDW], [JHBMJW]

11.2. Waarom JDOM en een vergelijking met bestaande API's

JDOM kan gebruikt worden als een alternatief voor het org.w3c.dom pakket om XML documenten te manipuleren. JDOM is echter **geen** vervanging voor DOM [Sectie 12.1.] , [TRDOM], in feite kunnen JDOM en DOM perfect naast elkaar bestaan. JDOM werkt samen met bestaande standaarden zoals SAX (Simple API for XML) [Sectie 12.2.] , [SAX] en DOM (Document Object Model). JDOM is echter meer dan een simpele abstractie van deze API's. JDOM gebruikt de beste concepten van bestaande API's en creëert een nieuwe set van klassen en interfaces. JDOM levert, volgens een JDOM gebruiker *"The interface I expected when I first looked at org.w3c.dom"*. JDOM kan lezen van bestaande DOM en SAX bronnen en kan uitvoer leveren aan DOM- en SAX-ontvangende componenten. Deze mogelijkheden laten JDOM toe naadloos samen te werken met bestaande programmacomponenten die gebruik maken van SAX of DOM.

11.2.1. DOM versus JDOM

Om beter te begrijpen waarom er nood is aan een alternatieve API, bekijken we nu de

ontwerpbeperkingen van het W3C DOM [TRDOML1]:

- **Taalonafhankelijk.** DOM is niet ontworpen met de taal Java in het achterhoofd. Deze manier van handelen behoudt een zeer gelijkaardige API voor de verschillende talen (dezelfde API ondersteunt bijvoorbeeld zowel Java, C++ als JavaScript). Dit maakt deze API moeilijker hanteerbaar voor programmeurs die gewend zijn aan het Java dialect. Een voorbeeldje ter verduidelijking: waar Java gebruik maakt van een ingebouwde klasse **String**, definieert de DOM specificatie zijn eigen klasse **Text**.
- **Strikte hiërarchie.** De DOM API volgt rechtstreeks uit de XML specificatie zelf. In XML is alles een knooppunt (node). Dus vind je in DOM een knooppuntgebaseerde interface, waarvan bijna elke klasse een uitbreiding is en een hele hoop methodes die een knooppunt als resultaat teruggeven. Dit is ongetwijfeld een zeer elegante manier van werken vanuit polymorfistisch standpunt, maar het is moeilijk om mee te werken in Java. Je krijgt dan expliciete downcasts van een knooppunt (node) naar een bepaald bladtype (leaf), hetgeen ingewikkelde en moeilijk leesbare code oplevert.
- **De interface als drijvende kracht.** De publieke DOM API bestaat uitsluitend uit interfaces (de enige uitzondering hierop is een Exception klasse). Het W3C is niet geïnteresseerd in het leveren van implementaties, enkel in het definiëren van interfaces. Dit betekent echter dat je als Java programmeur te maken krijgt met een hoge scheidingsgraad tussen je code en de XML objecten die je creëert, aangezien de W3C standaarden gebruik maken van generische factory klassen en heel flexibele maar minder directe patronen (patterns).

Voor de programmeur betekenen deze beperkingen een zware (zowel wat geheugengebruik als wat de grootte van de interface betreft) en onefficiënte API. De API is moeilijk om onder de knie te krijgen en vaak frustrerend om te gebruiken. Daarom is JDOM opgevat als een lichtgewicht API met als belangrijkste kenmerk dat hij Java-gebaseerd is. De JDOM principes zijn bijna het tegengestelde van de DOM principes:

- **Taalafhankelijk.** JDOM is Java platform afhankelijk. De API maakt gebruik van de in Java ingebouwde **String** ondersteuning. De API maakt ook gebruik van de Java 2 platform **Collection** klassen, zoals **List** en **Iterator**.
- **Geen hiërarchieën.** In JDOM is een element een instantie van de klasse **Element**. Een XML attribuut is een instantie van de klasse **Attribute** en een XML document zelf is een instantie van **Document**. Omdat deze allemaal verschillende concepten in XML voorstellen, wordt er altijd naar hen gerefereerd via hun eigen types, nooit via een amorfe "node" (knooppunt).
- **Klassen als drijvende kracht.** JDOM objecten zijn directe instanties van klassen zoals **Document**, **Element** en **Attribute**. Een nieuw object aanmaken is even gemakkelijk als het gebruik van de **new** operator in Java. Dit betekent ook dat er geen factory interfaces te configureren zijn. JDOM is daardoor out-of-the-box te gebruiken.

11.2.2. SAX versus JDOM

SAX [Sectie 12.2.] bouwt, in tegenstelling tot DOM, geen documentboom op in het geheugen. In plaats daarvan presenteert het een document als een opeenvolging van events. SAX meldt bijvoorbeeld elke keer wanneer het een begin- of eindtag tegenkomt. Deze benadering maakt van SAX een lichtgewicht API die heel goed is om snel documenten te lezen. Een event-gebaseerde kijk op een document is echter voor veel hedendaagse server-side en objectgeoriënteerde Java ontwerpers niet intuïtief genoeg. SAX laat ook niet toe het document aan te passen, althans niet op een intuïtieve manier, en het is niet mogelijk op een willekeurige plaats toegang tot het document te verkrijgen. In JDOM is dit wel mogelijk.

11.2.3. JDOM kort samengevat

JDOM probeert het beste van DOM en SAX te verenigen. Met andere woorden: JDOM is een lichtgewicht API die ontworpen is om snel te kunnen presteren zonder al te veel geheugen te gebruiken. JDOM levert ook een volledig zicht op het document en willekeurige toegang tot dit document. Het is echter niet nodig dat het hele document in het geheugen aanwezig is, zoals bij DOM. De JDOM API voorziet in toekomstige implementaties die informatie alleen maar in het geheugen laden als dit nodig is. JDOM ondersteunt ook eenvoudige documentmanipulatie met behulp van standaard constructoren en gewone **set** methodes.

11.3. JDOM syntax

We zullen hier aan de hand van een aantal voorbeelden uitleggen hoe er met JDOM gewerkt wordt. Eerst zullen we bespreken hoe een JDOM document gecreëerd wordt, vanuit een bestaande bron of programmatorisch. Vervolgens bespreken we de manipulatie van JDOM documenten om te besluiten met een beschrijving van de uitvoermogelijkheden van JDOM.

11.3.1. XML lezen van een bestaande bron

Er zijn verschillende manieren om XML in te lezen van een bestaande bron en om te zetten naar een JDOM document. De klassen die hiervoor verantwoordelijk zijn zitten in het pakket **org.jdom.input**. We zullen hier een voorbeeld bespreken waarbij we een XML bestand gaan inlezen en dit in het geheugen plaatsen. Het bestand heeft de naam *invoer.xml*. Er zijn twee manieren om dit te verwezenlijken. E;én van deze twee manieren is echter reeds als **deprecrated** gemarkeerd. De methode die aangeraden wordt, gebruikt de **SAXBuilder** klasse in het **org.jdom.input** pakket om een JDOM document in het geheugen op te bouwen. Zoals de naam van deze klasse al laat vermoeden wordt SAX gebruikt om de JDOM boom op te bouwen. We geven een voorbeeld van de aangeraden methode.

Voorbeeldcode 11.1

```
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import java.io.File;

...

try {
   SAXBuilder sb = new SAXBuilder();
    Document document = sb.build(new File("invoer.xml"));
} catch (JDOMException je) {
    je.printStrackTrace();
}
...
```

Dit is in feite een simpel voorbeeldje. Eerst maken we een nieuw SAXBuilder object aan en vervolgens bouwen we een nieuwe JDOM boom op voor het bestand *invoer.xml*. Er zijn nog veel andere bronnen van waaruit een JDOM boom met behulp van de SAXBuilder kan opgebouwd worden. Voor meer informatie daarover verwijzen wij naar [JDOMDOC].

Indien we in dit voorbeeldje **SAXBuilder** vervangen door **DOMBuilder** vervangen, dan gebruiken we methodes die **deprecated** zijn. De **DOMBuilder** klasse is vooral bedoeld om een JDOM boom op te bouwen vertrekkende van een bestaande DOM boom.

11.3.2. Opbouwen van een JDOM document

We gaan nu aan de hand van een uitgewerkt voorbeeld illustreren hoe je een JDOM document opbouwt. Als je JDOM wilt gebruiken in je eigen Java code, vergeet dan niet het package org.jdom.* te importeren. De exacte semantiek van de verschillende elementen in dit voorbeeld is niet zo belangrijk, aangezien dit voorbeeld enkel dient om uit te leggen hoe JDOM gebruikt wordt. Het zal blijken dat de volgende voorbeeldjes niet veel uitleg vereisen. Deze vaststelling geeft nogmaals aan hoe eenvoudig JDOM is in het gebruik.

Voorbeeld 11.1. package.xml

```
<?xml version="1.0" encoding="UTF-8"?>
 <resolvedPackage>
    <packageName unNamed="yes">
      <!--Het attribuut unNamed is toegevoegd om louter illustratieve
      redenen. In de echte package.xml die gegenereerd wordt door
      jnomedoc is dit attribuut een element-->
      <isResolved/>
    </packageName>
    <subPackages>
      <packageName>
        <name> jnome </name>
        <isResolved/>
      </packageName>
    </subPackages>
    <compilationunits/>
 </resolvedPackage>
```

11.3.2.1. Creatie van een document

We beginnen met het creëren van het wortelelement (rootelemen) en voegen dit vervolgens toe aan het document.

Voorbeeldcode 11.2 Creëren van een document

```
rootElement = new Element("resolvedPackage");
newDoc = new Document(rootElement);
```

Deze eerste stap creëert een nieuw **org.jdom.Element** rootElement met naam resolvedPackage. In de tweede stap wordt van dit element het wortelelement van het document gemaakt. Dit document is een instantie van **org.jdom.Document**, dat we hier de naam newDoc geven. Omdat een XML document altijd één enkel rootelemen moet hebben, krijgt de constructor van **Document** een **Element** mee als argument.

Er bestaat ook een publieke constructor van **Document** zonder argumenten. Zo'n document zonder argumenten is geen goedgevormd document; inspectoren zullen een **IllegalStateException** gooien zolang er geen rootelemen is toegevoegd. Deze constructor is voorzien omdat de makers van JDOM dit nuttig vonden voor het gebruik van JDOM in andere tools. Ook is het op die manier mogelijk eerst enkele processing instructions toe te voegen vooraleer het rootelement wordt toegevoegd.

11.3.2.2. Elementen en sub-Elementen

Vervolgens voegen we het **Element** packageName en zijn sub-**Element** isResolved toe.

Bemerk hoe het toevoegen op een zeer intuïve wijze gebeurt.

Voorbeeldcode 11.3 Toevoegen van elementen

```
packName = new Element("packageName");
resolved = new Element("isResolved");
packName.addContent(resolved);
rootElement.addContent(packName);
```

De methode **addContent** wordt gebruikt om een element inhoud te geven. Omdat **addContent** een **Element** teruggeeft (in plaats van gewoon een void methode te zijn), kunnen we het bovenstaande ook compacter beschrijven op volgende manier:

Voorbeeldcode 11.4 Toevoegen van elementen op een compactere manier

```
rootElement.addContent( ( new Element("packageName") ) .addContent(new
Element("isResolved")));
```

Beide werkwijzen hebben hetzelfde resultaat. Het hangt van de programmeur af welke vorm verkozen wordt. Alle methodes die zorgen voor een wijziging van de inhoud van een **Element** object, geven het gewijzigde **Element** object als resultaat terug. Deze methodes zijn de verschillende **addContent** methodes en de verschillende **set** methodes. Dit is gedaan om de geneste constructies van hierboven mogelijk te maken. De geneste constructies zijn gemakkelijk om enkele elementen met weinig inhoud toe te voegen, voor ingewikkelder constructies zorgt de eerste manier van werken voor meer leesbaare code.

Merk op: de volgorde waarin de elementen worden toegevoegd is ook de volgorde waarin de elementen in de uitvoer zullen verschijnen, tenzij deze volgorde gewijzigd is tussen het toevoegen van het element en het genereren van uitvoer.

11.3.2.3. Een attribuut toevoegen

Een attribuut toevoegen gebeurt als volgt:

Voorbeeldcode 11.5 Toevoegen van attributen

```
packName.setAttribute(new Attribute("unNamed","yes"));
```

Er bestaat nog een andere werkwijze om een attribuut aan een element toe te kennen, buiten het aanmaken van een **Attribute** object en dit als parameter door te geven. We kunnen gewoon de naam van het attribuut en de waarde meegeven aan de methode **setAttribute**. De code ziet er dan als volgt uit:

Voorbeeldcode 11.6 Toevoegen van attributen

```
packName.setAttribute("unNamed","yes");
```

11.3.2.4. Commentaar toevoegen

Commentaar kan ook heel eenvoudig toegevoegd worden, zoals onderstaand voorbeeld illustreert.

Voorbeeldcode 11.7 Toevoegen van commentaar

```
packName.addContent(new Comment("Het attribuut unNamed is toegevoegd om louter
illustratieve redenen. In de echte package.xml die gegenereerd wordt door
jnomedoc is dit attribuut een element"));
```

11.3.2.5. Processing instructions toevoegen

Ook voor processing instructions is er een klasse voorzien. De constructor van deze klasse neemt twee **String**s als argumenten. De eerste **String** is de naam van het doel van de processing instruction [Sectie 2.3.5.] en de tweede **String** bevat data voor de processing instruction, dit zijn attributen met hun waarden. In plaats van een string kan de tweede parameter ook van het type **java.util.Map** zijn, waar de data in de vorm van naam/waarde paren in zit. Om bijvoorbeeld de processing instruction **xml-stylesheet** met de nodige attributen toe te voegen aan een Document, kunnen we de volgende code gebruiken:

Voorbeeldcode 11.8

```
newDoc.addContent(new ProcessingInstruction("xml-stylesheet","type=\"text/xsl\"
href=\"stylesheet.xsl\""));
```

11.3.2.6. Volledig uitgewerkt voorbeeld

Ten slotte geven we de volledige code om het voorgaande voorbeeld op te bouwen met JDOM.

Voorbeeldcode 11.9 Volledige code

```
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.Attribute;
import org.jdom.Comment;

...

rootElement = new Element("resolvedPackage");
newDoc = new Document(rootElement);
packName = new Element("packageName");
packName.setAttribute(new Attribute("unNamed","yes"));
packName.addContent(new Comment("Het attribuut unNamed is toegevoegd om louter
illustratieve redenen. In de echte package.xml die gegenereerd wordt door
jnomedoc is dit attribuut een element"));
packName.addContent(new Element("isResolved"));
rootElement.addContent(packName);
rootElement.addContent( ( new Element("subPackages") ) .addContent( ( new
Element("packageName") ) .addContent( ( new Element("name") )
.addContent("jnome"),new Element("isResolved"))));
rootElement.addContent(new Element("compilationunits"));
...
```

We hebben hier ook de geneste manier van werken gebruikt om te laten zien hoe dit eruit zit voor ingewikkelde constructies. Je ziet dat het hier niet meer zo duidelijk is hoe de nesting precies in elkaar zit. Je kan dit verbeteren door een andere layout te geven aan de broncode, maar dan ga je de duidelijkheid van je code laten afhangen van de layout ervan. Zelf

verkiezen wij de meer uitgebreide manier van werken voor ingewikkelder constructies. Wij vinden dat deze werkwijze de duidelijkheid van je code alleen maar ten goede kan komen.

11.3.3. Manipuleren van een JDOM document

We zullen in dit gedeelte gaan bekijken hoe we kinderen van een element kunnen selecteren en hoe we een kind van een bepaald element kunnen verwijderen uit een document. Hoe het toevoegen van kinderen verloopt hebben we reeds gezien in [Sectie 11.3.2.2.] . De methodes die we gaan bespreken zijn allemaal gedefinieerd in de **org.jdom.Element** klasse.

Wanneer we alleen beschikken over een instantie van de **Document** klasse, moeten we eerst de methode **getRootElement()** toepassen om het rootelemen te bekomen. Deze methodes kunnen vervolgens op het rootelemen toegepast worden.

11.3.3.1. Selecteren van kinderen

Om een kind of meerdere kinderen van een element te selecteren zijn de methodes **getChild** en **getChildren** voorzien. We zullen hier de voornaamste mogelijkheden bespreken, maar voor een uitgebreid overzicht verwijzen we naar [JDOM].

We bespreken eerst de methode **getChild**. Deze methode heeft één parameter van het type **String**. Deze parameter wordt gebruikt om aan te geven dat we een kindelement met die lokale naam willen selecteren. Volgens de specificatie geeft deze methode het **eerste** kindelement met de gespecificeerde lokale naam terug, waarbij dit kindelement bovendien niet tot een bepaalde namespace behoort. Het is ook mogelijk om aan te geven dat we het eerste kind willen met de opgegeven lokale naam en behorend tot een bepaalde namespace. We geven hier nu een klein voorbeeldje van het gebruik, verderbouwend op de eerder opgebouwde code [Sectie 11.3.2.6.] :

Voorbeeldcode 11.10

```
...
Element elPackName = rootElement.getChild("packageName");
...
```

Het resultaat is een rechtstreeks kind van het rootelemen, en wel het eerste kind met de naam packageName.

Er is ook een methode **getChildren** voorzien. Het resultaat van deze methodes is een lijst van de rechtstreekse kinderen van het element waar deze methode op toegepast wordt. Het resultaat van deze methodes is steeds van het type **java.util.List**. Het is wel belangrijk op te merken dat deze lijst **"live"** is, dit wil zeggen dat wijzigingen die uitgevoerd worden op de verkregen lijst, weerspiegeld worden in de JDOM boom. We vermoeden dat dit gedaan is opdat wijzigingen in de onderlinge volgorde van de elementen door middel van bewerkingen op een lijst doorgevoerd kunnen worden .

De methode **getChildren** bestaat in een aantal vormen. De vorm zonder argumenten geeft een lijst terug van alle rechtstreekse kinderen, en dit als **Element** objecten die deel uitmaken van de teruggegeven lijst. Indien er geen kinderen zijn, is het resultaat een lege lijst. Het is ook mogelijk een bepaalde lokale naam mee te geven aan deze methode. Dit resulteert dan in een lijst van kindelementen met die naam. Als we een bepaalde lokale naam meegeven, kunnen we nog specifieker zijn door een namespace mee te geven waartoe alle elementen in de

resultaatset moeten behoren. Het is bij ons weten niet mogelijk om rechtstreeks alle kinderen die tot een bepaalde namespace behoren, op te vragen. Om alle kinderen van het rootelemen uit ons voorbeeld [Sectie 11.3.2.6.] op te vragen, kunnen we de volgende code gebruiken:

Voorbeeldcode 11.11

```
...

java.util.List kinderen = rootElement.getChildren();
...
```

11.3.3.2. Verwijderen van kinderen

Analoog aan de getChild/getChildren methodes besproken in [Sectie 11.3.3.1.] bestaan er ook **removeChild**/**removeChildren** methodes. De argumenten die kunnen meegegeven worden aan deze methodes zijn volledig analoog aan de argumenten van de getChild/getChildren methodes. Het resultaat van de **removeChild**/**removeChildren** methodes is een boolean die waar is indien er een kind verwijderd is en onwaar anders. Deze methodes verwijderen alleen de rechtstreekse kinderen. Wel is het zo dat een kleinkind van het huidige element ook (onrechtstreeks) verwijderd wordt als we de ouder van dat kleinkind, een kind van het huidige element, verwijderen.

Om het kind met de naam subPackages te verwijderen, gebruiken we de volgende regel code:

Voorbeeldcode 11.12

```
rootElement.removeChild("subPackages");
```

Om alle kindelementen te verwijderen, gebruiken we de volgende code:

Voorbeeldcode 11.13

```
rootElement.removeChildren();
```

11.3.4. Het genereren van uitvoer via JDOM

JDOM ondersteunt op dit moment drie uitvoermethoden. Deze methoden zijn **DOM** [Sectie 12.1.] , **SAX** [Sectie 12.2.] en **XML** [Hoofdstuk 2.] . De DOM uitvoermethode bouwt op basis van een bestaande JDOM boom een **org.w3c.dom.Document** op. De SAX uitvoermethode genereert SAX events. Tenslotte is er nog de XML uitvoermethode die goedgevormde XML schrijft naar een **Writer** of een **OutputStream**. We zullen het gebruik van deze drie uitvoermethoden illustreren aan de hand van kleine voorbeeldjes. In onze voorbeeldjes maken we weer gebruik van de variabelen uit [Sectie 11.3.2.6.] .

Voorbeeldcode 11.14 Uitvoer van een DOM document

```
...

DOMOutputter domOutputter = new DOMOutputter();
org.w3c.dom.Document DOMdoc = domOutputter.output(newDoc);
...
```

Voorbeeldcode 11.15 Uitvoer via SAX events

```
...

SAXOutputter saxOutputter = new SAXOutputter(myContentHandler);
saxOutputter.output(newDoc);
...
```

In bovenstaand voorbeeld is **myContentHandler** een instantie van een klasse die SAX events afhandelt. Voor meer informatie over SAX verwijzen we naar [Sectie 12.2.] .

Voorbeeldcode 11.16 De XML uitvoermethode

```
...

FileOutputStream out = new FileOutputStream(absPath);
XMLOutputter outputter = new XMLOutputter("\t",true);
outputter.output(newdoc,out);
out.flush();
out.close();
...
```

In bovenstaand voorbeeld maken we een **FileOutputStream** object aan waarnaar we de XML uitvoer willen schrijven. Vervolgens maken we een **XMLOutputter** object aan. Het eerste argument ("\t") is de indentatiestring. Deze string zal gebruikt worden om een kindelement verder te laten inspringen dan zijn ouderelement. In dit geval wordt er met tabs gewerkt. Dit heeft alleen effect als het tweede argument de waarde **true** heeft. Met het tweede argument wordt aangegeven of we newlines in de uitvoer willen of niet.

In een volgende stap wordt het Document **newDoc** naar de FileOutputStream **out** geschreven en wordt deze FileOutputStream afgesloten.

11.4. Conclusie

JDOM is een API om XML gegevens te gebruiken en te manipuleren vanuit Java code. Deze API is bijzonder gemakkelijk in het gebruik. Het beste bewijs hiervan is het feit dat we reeds goed met JDOM konden werken na het lezen van slechts één artikel [WBHEDW].

Het enige nadeel dat we kunnen vermelden is dat het nogal veel werk vraagt om een kind op een bepaalde diepte te selecteren. Of om alle kinderen, kleinkinderen, enzovoort van het rootelement te selecteren met een bepaalde naam. Er wordt echter volop gewerkt aan de ondersteuning voor XPath [Sectie 5.4.] , waardoor dit veel eenvoudiger zal worden.

Een andere troef is dat er aan de invoerkant zowel SAX, als DOM als bestanden aanvaard worden en dat het ook mogelijk is om de uitvoer in één van die formaten te genereren. Het zou wel eens heel goed kunnen dat JDOM, net zoals SAX, een de facto standaard wordt, maar dit is natuurlijk moeilijk te voorspellen.

12. XML API's

We hebben reeds een volledig hoofdstuk aan JDOM gewijd, maar dat is zeker niet de enige API (Application Programming Interface) om met XML documenten te werken. In dit hoofdstuk gaan we kort enkele andere API's bespreken die ook veel gebruikt worden en aan de hand van een kort voorbeeldje laten zien hoe ze typisch gebruikt worden. De API's die we gaan bespreken zijn DOM, SAX en JAXP. We zullen een kort beeld schetsen van de ontstaansgeschiedenis van deze API's. Ook zullen we enkele codevoorbeeldjes geven om de lezer een beter beeld te geven van deze API's. Bij DOM en JAXP zullen we dit redelijk kort houden, maar SAX behandelen we uitgebreider. De reden hiervoor is dat SAX een centrale rol speelt in Cocoon 2 [Sectie 9.2.] en we zelf met SAX hebben geëxperimenteerd om de werking te begrijpen. DOM en JAXP hebben we wel eens bekeken, maar we hebben er zelf niet rechtstreeks mee gewerkt.

Dit hoofdstuk is voor een gedeelte gebaseerd op [DMSJDMU] en [BMCONDOM].

12.1. DOM

Voor deze sectie hebben wij ons gebaseerd op [TRDOM], [TRDOMAC] en [TRDOMFAQ]

Het acroniem DOM staat voor Document Object Model. DOM is een technische aanbeveling (Technical Recommendation of TR) van het W3C. Het is een platform- en taalonafhankelijke interface die programma's en scripts moet toelaten om dynamisch toegang te krijgen tot de inhoud van een document en dan de inhoud, de structuur en de stijl aan te passen. Het document kan dan verder verwerkt worden en het resultaat zou naadloos moeten in te passen zijn in de aangeboden pagina. Het doel van DOM is om het voor de ontwikkelaars eenvoudig te maken om toegang te krijgen tot de componenten van een document en om inhoud, attributen en stijl te bewerken, toe te voegen of weg te laten. DOM zou moeten toelaten om hetzelfde programmamodel te gebruiken, onafhankelijk van de gebruikte taal, het platform, browser of server. Ook is er het opzet dat, indien je werkt met gestandaardiseerde API's, eender welke DOM implementatie kan gebruikt worden samen met eender welke DOM-gebaseerde toepassing. Alhoewel alle DOM implementaties interoperabel zouden moeten zijn, kunnen er toch significante verschillen zijn betreffende snelheid, geheugengebruik, grootte van de code, ... Omwille van deze verschillen kan het van pas komen om verschillende implementaties te kunnen uitproberen. Ook is men zich ten volle bewust van één groot nadeel, dat bij zo goed als alle gegeneraliseerde interfaces opduikt, met name dat de DOM oproepen gebruikt kunnen worden om een hele grote variatie aan problemen op te lossen, maar dat het mogelijk niet dé beste oplossing voor dat specifieke probleem is. Maar bij het W3C verwacht men dat dit nadeel niet opweegt tegen het voordeel van de interoperabiliteit. Een laatste opmerking willen we toch niet weerhouden: men is er zich ook ten volle van bewust dat een ontwikkelaar beroep zou willen doen op een andere API naast, of in plaats van DOM om bepaalde taken uit te voeren. Het is dus mogelijk dat de applicatie omwille van performantie eigen interfaces gebruikt, maar dat de communicatie met de buitenwereld via DOM kan gebeuren.

Zoals reeds eerder aangehaald voorziet het W3C alleen maar in een DOM specificatie en niet in een DOM implementatie of een DOM applicatie. Een DOM implementatie (host implementatie) is een stukje software dat een geparst XML of HTML document neemt en het beschikbaar stelt voor verwerking via de DOM interfaces. Een DOM applicatie is een ander stukje software dat "iets" doet met het document dat beschikbaar gesteld wordt door de implementatie. Omdat het hier over een taalonafhankelijke interface gaat kan iedereen die DOM wil implementeren kiezen in welke taal dit zal gebeuren. Het is dan ook noodzakelijk

dat de specificatie taalonafhankelijk is. Om dit laatste te realiseren heeft het W3C geopteerd om gebruik te maken van de Object Management Group Interface Definition Language (OMG IDL) omdat deze ontworpen was om taal- en implementatie-onafhankelijke interfaces te specificeren.

De gehele specificatie van DOM is niet in één document beschreven. De DOM werkgroep werkt met fases of levels. Twee van deze fases zijn reeds TRs, maar de derde fase is nog altijd een werkdocument.

- **DOM vereisten:** dit document bevat de vereisten voor de verschillende levels van DOM. Dit document wordt regelmatig aangepast om de vereisten van het laatste level te weerspiegelen.
- **DOM Level 0:** voor deze level is er geen W3C specificatie. Deze level is functioneel equivalent met hetgeen aanwezig is in Netscape Navigator 3.0 en Microsoft Internet Explorer 3.0, en de naam level 0 is dan ook maar als een informele referentie bedoeld [TRDOML0].
- **DOM Level 1:** voltooid in oktober 1998, voorziet ondersteuning voor XML 1.0 en HTML [TRDOML1]
- **DOM Level 2:** deze tweede level [TRDOML2] is voltooid in november 2000 en voorziet de volgende uitbreidingen op level 1:
    - XML 1.0 met namespaces
    - Cascading Style Sheets (CSS)
    - events (User Interface events en Boombewerkingsevents (Tree Manipulation events))
    - uitbreiding van de boombewerkingen ()

- **DOM Level 3:** deze level is momenteel nog onder ontwikkeling. Level 3 [TRDOML3] bevat naast level 2 ook:
    - voltooiing van ondersteuning voor XML 1.0
    - uitbreiding van user interface events
    - ondersteuning voor abstracte schema's (DTD, XML Schema, ...)
    - mogelijkheid om document in te lezen of op te slaan
    - XPath ondersteuning

Het belangrijkste om te onthouden is dat elk level uitgebreider wordt en dat er rekening gehouden wordt met de terugkoppeling van de gebruikers. Zo was er tot en met **level 2** een **bootstrap-probleem**: om initieel een DOM implementatie aan te spreken was er code nodig die specifiek was aan de gebruikte implementatie. Het is nogal duidelijk dat dit geen mooie oplossing is. Hiermee is rekening gehouden in **level 3** en er is een **DOMImplementationFactory** voorzien die juist dit probleem moet aanpakken. Een andere implementatie gebruiken zal dan mogelijk worden zonder je code te veranderen. **DOM level 3** is voorlopig nog een **W3C Working Draft** en nog geen **W3C recommendation**.

12.1.1. Voorbeeld: DOM boomstructuur

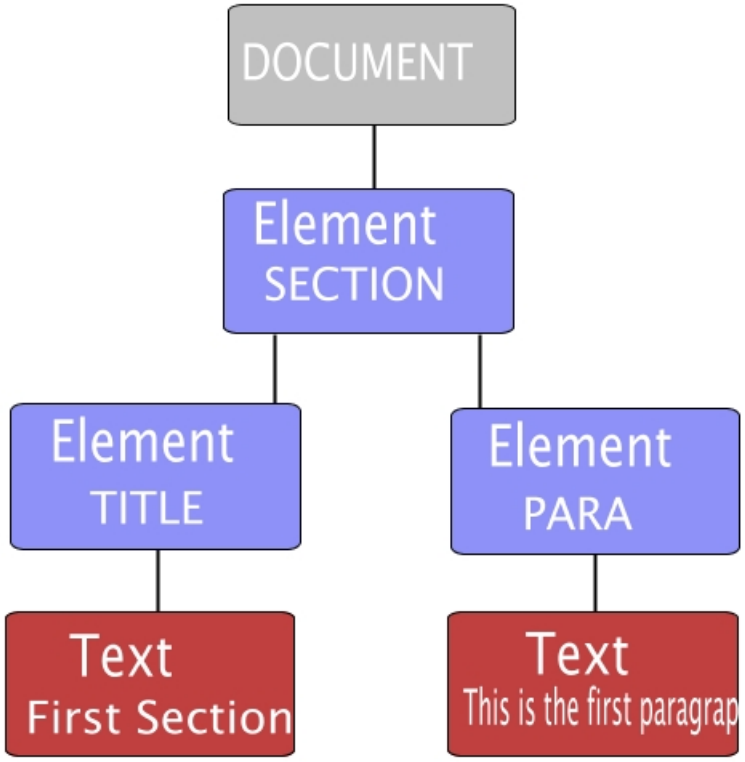
We gaan nu een kort voorbeeldje geven van het gebruik van DOM. Eerst zullen we een klein XML document met de overeenkomstige DOM boomstructuur geven. Vervolgens zullen we laten zien hoe we dit document kunnen aanspreken en bewerken.

Voorbeeld 12.1.

```
<?xml version="1.0"?>
```

```
<SECTION>
   <TITLE> First Section </TITLE>
   <PARA> This is the first paragraph </PARA>
</SECTION>
```

Dit document heeft een overeenkomstige DOM boomstructuur die er als volgt uitziet:



Figuur 12.1. DOM boomstructuur

Als wortel van de boomstructuur hebben we een **DOCUMENT** node, die op zijn beurt een **ELEMENT** node heeft als kind. Deze **ELEMENT** node stelt het hoofdelement van ons document voort, namelijk **SECTION**. We zien in de boomstructuur dus duidelijk de structuur van het XML document weerspiegeld worden. In DOM is het wel zo dat in de boomstructuur elke elementnode één (of meerdere) **TEXT**node(s) als kind heeft. Een **TEXT**node bevat de tekstuele inhoud van het element. Nu we kort hebben aangehaald hoe de boomstructuur van een XML document er voor DOM specifiek uitziet, gaan we een paar kleine voorbeeldjes bekijken van het gebruik van DOM in de praktijk.

Voorbeeldcode 12.1 Inlezen van een document

```
import org.w3c.dom.*;
import org.apache.xerces.parsers.DOMParser;

...

try {
   DOMParser parser = new DOMParser();
    parser.parse("test.xml");
    Document document = parser.getDocument();
} catch (Exception exc) {
}
...
```

In dit voorbeeld lezen we ons voorbeelddocument in (dat is opgeslaan onder de naam *index.xml*) en het geparste document wordt toegekend aan de variabele **document** die van het type **org.w3c.dom.Document** is.

Om het hoofdelement op te vragen kunnen we de volgende code gebruiken:

Voorbeeldcode 12.2

```
...

Element root = document.getDocumentElement();
System.out.println("Het hoofdelement is: " + root.getNodeName());
...
```

We vragen hier aan de variabele **document** het hoofdelement op. We krijgen een object van het type **org.w3c.dom.Element** terug en we noemen deze variabele **root**. Om de naam van dit element te weten te komen passen we de methode **getNodeName()** toe.

Om te weten hoeveel rechtstreekse kinderen het hoofdelement heeft, kunnen we het volgende stukje code toepassen:

Voorbeeldcode 12.3

```
...

NodeList children = root.getChildNodes();
System.out.println("Aantal kinderen: " + children.getLength());
...
```

Aan het element **root** vragen we met behulp van de methode **getChildNodes()** een lijst op van de rechtstreekse kinderen van dit element. Deze methode geeft een object terug van het type **org.w3c.dom.NodeList**.

Het is ook mogelijk om te itereren over de kinderen van een element.

Voorbeeldcode 12.4

```
...

for (Node child = root.getFirstChild(); child != null; child = child.getNextSibling()) {
    String childName = child.getNodeName();
    String childValue = child.getFirstChild().getNodeValue();
    System.out.println("Element: " + childName + " = " + childValue);
}
...
```

We maken gebruik van de **for** opdracht om te itereren over de kinderen van (in dit geval) het hoofdelement. Eerst vragen we met behulp van de methode **getFirstChild()** het eerste kind op. We krijgen van deze methode een object van het type **org.w3c.dom.Node** en noemen de variabele waarin we een referentie aan het huidige kind bijhouden **child**. Om het volgende kind op te vragen passen we op het huidige kind, gekend middels de variabele **child**, de methode **getNextSibling()** toe. Indien er geen volgend kind is, geeft deze methode **null** terug. Vandaar dat we het lichaam van de **for** opdracht slechts willen uitvoeren zolang **child** verschillend is van **null**.

In het lichaam van de **for** opdracht kennen we eerst de naam van het huidige kind toe aan de

variabele **childName**. Vervolgens gaan we de inhoud van dit element opvragen. Herinner dat in een DOM boomstructuur de inhoud van een element niet in de elementnode zit maar in een aparte tekstnode, die een kind is van de elementnode waar het bij hoort. Daarom moeten we eerst de methode **getFirstChild()** toepassen op het huidige kind en op het resultaat van deze methode (een tekstnode) kunnen we de methode **getNodeValue()** toepassen. Dit geldt alleen als er een tekstnode als eerste element van het huidige kind aanwezig is. Is bijvoorbeeld het eerste kind van het huidige kind een element dan verkrijgen we als resultaat van de methode **getNodeValue() null**. Door deze methodes recursief toe te passen kunnen we over alle elementen van het document itereren.

Hiermee willen wij deze korte bespreking van DOM besluiten. Buiten de besproken manieren om de DOM boomstructuur te ondervragen bestaan er ook nog methodes om de boomstructuur de manipuleren.

12.2. SAX

Voor de bespreking van SAX deden we een beroep op [SAX] en [USAX].

SAX oftewel de Simple API for XML is een ontwikkeling van de XML-DEV mailinglist. De release van SAX 1.0 dateert van 11 mei 1998, nog geen vijf maanden na de eerste voorstellen voor SAX. Een van de leden van deze mailinglist vond dat alle (XML) parser ontwikkelaars een gemeenschappelijke eventgebaseerde Java API zouden moeten ondersteunen, dit na veel kopzorgen door het ondersteunen van drie verschillende XML parsers, elk met hun eigen specifieke API. De eerste naam die aan deze API werd gegven was **YAXPAPI** en stond voor **Yet Another XML Parser API**. Na het starten van een discussie met twee andere ontwikkelaars van XML Parsers en het voeren van de ontwerpdiscussie op de publieke mailinglist werd er gestart met de ontwikkeling, die nog geen vijf maanden later resulteerde in SAX 1.0. De huidige release van SAX draagt het versienummer 2.0.1 (ook sax2 r2 genoemd).

Verdoken hebben we het verschil met DOM al aangehaald. Bij DOM is er sprake van boomstructuren, terwijl we bij SAX spreken over een eventgebaseerde API. Een **boomgebaseerde** API zoals DOM stelt intern een XML document voor als een boomstructuur, en laat een toepassing toe om te navigeren in deze boomstructuur. Een **eventgebaseerde** API daarentegen maakt gebruikt van het callback-mechanisme om de parse events te rapporteren aan de applicatie. In dit geval wordt meestal geen interne boomstructuur opgebouwd. De applicatie moet daarom handlers (bepaalde methodes) implementeren om de verschillende events af te handelen, zoals het afhandelen van events gebeurt voor een grafische user interface. SAX is een API die gebruik maakt van het eventmechanisme.

Een boomgebaseerde API kan handig zijn in vele omgevingen, maar dan moet je wel de hoge belasting op de systeembronnen erbij nemen, vooral bij grote documenten kan dit een belangrijk nadeel zijn. Vooral omdat veel applicaties zo een boom bijhouden in hun eigen streng getypeerde datastructuren eerder dan in een generieke boomstructuur. Een eventgebaseerde API daarentegen zal een (meestal) eenvoudigere toegang bieden tot het XML document en dit op een lager niveau. Het maakt het mogelijk om documenten te verwerken die groter zijn dan het aanwezige systeemgeheugen omdat het niet nodig is het volledige document in primair of secundair geheugen te hebben. Ook zal je je eigen datastructuren kunnen construeren door het gebruik van de callback event handlers. Het is het vermelden waard dat het mogelijk is om met behulp van een eventgebaseerde API een boom van het document (de zogenaamde parse tree) op te bouwen en om de boom, die in het geheugen zit, te doorlopen.

Stel dat we het volgende voorbeelddocument hebben:

Voorbeeld 12.2.

```
<?xml version="1.0"?>
  <SECTION>
     <TITLE> First Section </TITLE>
     <PARA> This is the first paragraph </PARA>
  </SECTION>
```

Dan zal deze structuur door een eventgebaseerde interface omgezet worden in een serie van lineaire events, voor dit document zou dit bijvoorbeeld kunnen zijn:

```
start document
start element: SECTION
characters: (whitespace)
start element: TITLE
characters: First Section
end element
characters: (whitespace)
start element: PARA
characters: This is the first paragraph
end element
characters: (whitespace)
end element
end document
```

Het zijn deze events waarvoor in de applicatie de event handlers moeten geschreven zijn en die een beroep doen op het callback-mechanisme om aan de applicatie doorgegeven te worden.

SAX wordt voornamelijk beschouwd als een read-only API, en dit heeft voornamelijk te maken met het eventgebaseerd zijn. Sommige bronnen zeggen dat het onmogelijk is met behulp van SAX wijzigingen aan te brengen, dat is overdreven, maar het is wel zeer moeilijk, en dit omdat er niet in het document genavigeerd kan worden. Het document wordt lineair ingelezen en events worden gecreëerd, maar het type van de events is afhankelijk van hetgeen tegengekomen wordt in het document. Er wordt (bijna) niets in het geheugen opgeslaan (tenzij in de applicatie zelf). Het is zeer moeilijk om de XML boom, de structuur van het document in dit geval, te manipuleren, want dit vereist het maken van een keten van filters en processors. Het is ook mogelijk om SAX te gebruiken voor uitvoer. De klasse die de events, die gegenereerd worden door de parser, opvangt, moet dan aan de hand van deze events een uitvoerstroom opbouwen.

Om dit betoog over SAX duidelijk te maken zullen we stap voor stap laten zien hoe een document met behulp van SAX kan worden ingelezen. Dit zullen we in vier stappen trachten duidelijk te maken:

- implementeren van een event handler
- creëren van een SAXParser
- associëren van de event handler met de SAXParser
- een document parsen

12.2.1. Implementatie van een event handler

Zoals reeds eerder aangehaald moet een applicatie die SAX wil gebruiken voorzien in een event handler. SAX voorziet in een basisklasse **DefaultHandler** waarvan de klassen in de applicatie kunnen overerven. De klasse **DefaultHandler** implementeert de methodes die in enkele interfaces gedefinieerd zijn. De implementatie van deze methodes beperkt zich echter tot het definiëren van de methodes, er is geen lichaam voorzien voor de methodes. Indien we

willen dat er ook effectief iets gebeurt als deze methodes door het eventmechanisme worden opgeroepen, dan zullen we het lichaam van die methodes zelf moeten implementeren.

Wanneer we de voorziene methodes bestuderen en daaruit de voor ons interessante methodes kiezen, dan bekomen we volgend lijstje van methodes om te implementeren:

- **startDocument()**: signaleert het begin van het document
- **endDocument()**: signaleert het einde van het document
- **startElement(String uri, String localName, String qName, Attributes attributes)**: geeft het begin van een element aan. De informatie die doorgegeven wordt is:
    - **String uri**: informatie omtrent de naamruimte van dit element
    - **String localName**: de eigenlijke naam van het element
    - **String qName**: de combinatie van alias van de naamruimte en de **localName**
    - **Attributes attributes**: de attributen van dit element (indien aanwezig)

- **endElement(String uri, String localName, String qName)**: signaleert het einde van het element dat bepaald wordt door de drie meegegeven argumenten
- **characters(char[] ch, int start, int length)**: bevat de eigenlijke data van een element. Merk op dat bij de argumenten van deze methode geen informatie zit omtrent het element waarvan dit de inhoud is. Is dit nodig, dan zal je die informatie zelf ergens moeten bijhouden. Deze methode bevat twee argumenten die het begin en het einde van de inhoud aangeven omdat in de praktijk **char[] ch** het volledige document bevat. Voor de inhoud van één element kan deze methode meerdere keren opgeroepen worden. Je mag daarom pas veronderstellen dat er geen inhoud meer komt nadat het **endElement** event is voorgekomen.
- **processingInstruction(String target, String data)**: **target** bevat de naam van de processing instruction en **data** bevat de rest, zijnde attributen of bijkomende data.

Niet elk document dat verwerkt zal worden zal foutloos zijn. Er zijn daarom voor foutafhandeling nog drie methodes voorzien:

- **warning (SAXParseException e)**: methode die waarschuwingen van de parser ontvangt
- **error (SAXParseException e)**: methode die waarschuwingen ontvangt omtrent fouten die de parser nog kan herstellen
- **fatalError (SAXParseException e)**: methode die informatie omtrent een fatale fout ontvangt. Wanneer deze methode opgeroepen wordt moet het normale verwerken van het document stopgezet worden vermits het document niet meer betrouwbaar is. Hierdoor kan het zijn dat de parser niet meer alle verdere fouten rapporteert.

De hierboven opgenoemde methodes kunnen ook nog een uitzondering gooien, met name **SAXException** en dat kan eender welke **SAXException** zijn, die mogelijk een wrapper is voor een andere uitzondering.

Implementeren we de hierboven opgesomde methodes, dan levert ons dit volgende code op (we printen gewoon alles wat we binnenkrijgen naar het standaarduitvoerkanaal):

Voorbeeldcode 12.5

```
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

public class ExampleSAXEventHandler extends DefaultHandler {
  public void startDocument () throws SAXException {
      System.out.println("Start of new document.");
  }
```

```
    public void endDocument () throws SAXException {
        System.out.println("End of document reached.");
    }

    public void startElement (String uri, String localName, String qName, Attributes
    attributes) throws SAXException {
        System.out.println("Start element");
        System.out.println("Namespace: " + uri);
        System.out.println("Local element name: " + localName);
        System.out.println("Fully qualified name: " + qName);
        for (int attNo = 0; attNo < attributes.getLength(); attNo++) {
            String attName = attributes.getLocalName(attNo);
            System.out.println("Attribute: " + attName + ": " + attributes.getValue(attName));
        }
    }

    public void endElement (String uri, String localName, String qName) throws SAXException {
        System.out.println("End element");
        System.out.println("Namespace: " + uri);
        System.out.println("Local element name: " + localName);
        System.out.println("Fully qualified name: " + qName);
    }

    public void characters (char[] ch, int start, int length) throws SAXException {
        System.out.println(new String(ch,start,length));
    }

    public void processingInstruction (String target, String data) throws SAXException {
        System.out.println("Processing instruction: " + target);
        System.out.println("PI data: " + data);
    }

    public void warning (SAXParseException e) throws SAXException {
        System.out.println("Problem parsing file: " + e.getMessage());
    }

    public void error (SAXParseException e) throws SAXException {
        System.out.println("Error parsing file: " + e.getMessage());
    }

    public void fatalError (SAXParseException e) throws SAXException {
        System.out.println("Fatal error parsing file: " + e.getMessage());
        System.out.println("Cannot continue parsing");
        System.exit(1);
    }

}
```

Deze klasse ExampleSAXEventHandler kan dienen als **ContentHandler** en **ErrorHandler** van een **SAXParser**. We gaan nu zien hoe we een **SAXParser** kunnen aanmaken om te gebruiken in een programma.

12.2.2. Creatie van een SAXParser

Vooraleer we onze klasse kunnen gebruiken, moeten we eerst weten hoe we een **SAXParser** aanmaken. In het jargon wordt een SAXParser ook wel **XMLReader** genoemd. Dit is niet helemaal correct vermits **XMLReader** een interface is en een **SAXParser** een concrete implementatie. Toch wordt dit hier en daar door elkaar gebruikt. Om het zo algemeen mogelijk te houden zullen wij spreken van een **XMLReader**. Op die manier voldoet elke klasse die deze interface implementeert aan onze beschrijving. Er zijn twee manieren om de **XMLReader** aan te maken. Ofwel gebeurt dit rechtstreeks ofwel via de **XMLReaderFactory**, die zijn informatie uit een Java omgevingsvariabele haalt.

Met rechtstreeks bedoelen we dat je weet welke klasse de eigenlijke parser implementeert. De **XMLReader** wordt dan op de volgende manier aangemaakt:

Voorbeeldcode 12.6

```
import org.xml.sax.XMLReader;

...

try {
   XMLReader xmlreader = new org.apache.xerces.parsers.SAXParser();
} catch (Exception e) {
}
...
```

De klasse **org.apache.xerces.parsers.SAXParser** is in dit geval onze parser. We weten precies welke klasse we willen gebruiken en we gebruiken deze informatie rechtstreeks in onze code. Willen we ooit veranderen van parser dan moeten we dit op alle plaatsen waar we deze constructor hebben gebruikt aanpassen.

Zoals reeds eerder gezegd kunnen we ook gebruik maken van de **XMLReaderFactory**, die zijn informatie haalt uit een Java omgevingsvariabele. Dit gebeurt door de applicatie op de volgende manier te starten (eventueel met bijkomende opties):

```
java -Dorg.xml.sax.driver=org.apache.xerces.parsers.SAXParser Programma
```

De **XMLReader** kunnen we dan op de volgende manier in onze code aanmaken:

Voorbeeldcode 12.7

```
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;

...

try {
    XMLReader xmlreader = XMLReaderFactory.createXMLReader();
} catch (Exception e) {
}
...
```

Op deze manier kunnen we, wanneer we willen, altijd van parser veranderen. Dit door **org.xml.sax.driver** een andere waarde te geven. We gaan nu bekijken hoe we de **XMLReader** duidelijk kunnen maken om een instantie van onze klasse te gebruiken als **ContentHandler** en **ErrorHandler**.

12.2.3. Associatie tussen SAXParser en event handler

Eénmaal we een **XMLReader** en een **ContentHandler** hebben, kunnen we aan de **XMLReader** doorgeven welk object als **ContentHandler** moet dienen. Omdat onze klasse **ExampleSAXEventHandler** hebben laten overerven van **DefaultHandler** kunnen we een instantie van onze klasse ook laten dienen als **ErrorHandler**, dit omdat de **DefaultHandler** onder andere de interfaces **ContentHandler** en **ErrorHandler** implementeert. Het leggen van de associatie gebeurt op de volgende manier:

Voorbeeldcode 12.8

```
...

ExampleSAXEventHandler exSAXEHandler = new ExampleSAXEventHandler();
```

```
xmlreader.setContentHandler(exSAXEHandler);
xmlreader.setErrorHandler(exSAXEHandler);
...
```

Op deze manier hebben we onze **XMLReader** verteld om een instantie van de klasse **ExampleSAXEventHandler** als **ContentHandler** en **ErrorHandler**.

12.2.4. Parsen van een document

Nu we alles in gereedheid hebben gebracht kunnen we overgaan tot parsen. We nemen daarom aan de er een bestand *test.xml* bestaat dat we eenvoudig kunnen aanspreken. De typische manier van werken in dat geval is als volgt:

Voorbeeldcode 12.9

```
import org.xml.sax.Inputsource;

...

InputSource source = new InputSource(test.xml);
xmlreader.parse(source);
...
```

Onze **XMLReader** zal nu de events genereren en deze signaleren aan onze klasse die als **ContentHandler** dient.

We hernemen nu het voorbeeldje dat we gebruikt hebben bij de uitleg over een **eventgebaseerde** manier van werken.

Voorbeeld 12.3.

```
<?xml version="1.0"?>
  <SECTION>
     <TITLE> First Section </TITLE>
     <PARA> This is the first paragraph </PARA>
  </SECTION>
```

Laten we daar ons programma op los, dan krijgen we de volgende uitvoer:

```
Start of new document.
Start element
Namespace:
Local element name: SECTION
Fully qualified name: SECTION


Start element
Namespace:
Local element name: TITLE
Fully qualified name: TITLE
Event: Characters: First Section
End element
Namespace:
Local element name: TITLE
Fully qualified name: TITLE

Start element
Namespace:
Local element name: PARA
Fully qualified name: PARA
```

```
This is the first paragraph
End element
Namespace:
Local element name: PARA
Fully qualified name: PARA

End element
Namespace:
Local element name: SECTION
Fully qualified name: SECTION
End of document reached.
```

Dit is misschien wat onduidelijk vanwege de dubbele boodschappen, daarom een klein woordje uitleg. Eerst wordt natuurlijk het begin van het document gesignaleerd. Vervolgens het begin van het element **SECTION**. Daarna krijgen we witruimte, dit omdat ook de witruimte wordt doorgegeven aan de methode **characters** in dit geval. Op het einde van het document lijkt het alsof er voor de tweede keer het begin van het element **SECTION** wordt gesignaleerd, maar hier wordt het einde van dit element gesignaleerd. Dit wordt aangegeven door de boodschap **End element** die de boodschap omtrent de namespace voorafgaat.

12.2.5. Slotbeschouwing

Zoals reeds vermeld in [Sectie 9.2.3.2.1.] , gebeurt de communicatie tussen de verschillende stappen in de pipeline met behulp van SAX events. Dit wil zeggen dat de XSLT processor (de componenten verantwoordelijk voor het uitvoeren van de transformaties) SAX events ontvangt. Maar een XSLT processor moet ook rekening kunnen houden met XPath [Sectie 5.4.] expressies, zoals **/CHAPTER//SECTION** en **../child::*[2]**. Het probleem met het eventgebaseerd model van SAX is dat je, als ontvanger van SAX events, niet kan vragen om de stroom nog eens te leveren. Het kan goed zijn dat, op het moment dat de XSLT processor de expressie **../child::*[2]** tegenkomt, het event voor dat knooppunt al is geweest. Hoe wordt dit dan concreet geïmplementeerd?

Om deze vraag te beantwoorden hebben we onderzocht hoe dit in Xalan [XALAN] wordt opgelost. Xalan is de XSLT processor die standaard bij Cocoon 2 zit. Het antwoord op onze vraag vinden we in [XALANDES]. Het antwoord is dat een boom wordt opgebouwd die overeenstemt met de ontvangen SAX events. De opgebouwde boomstructuur is geoptimaliseerd voor snelle transformaties. Het nadeel is dat nog altijd een volledige boom moet opgebouwd worden.

De ontwikkelaars zijn zich ten volle bewust van dit (grote) nadeel. Er zijn voorstellen om een statische analyse uit te voeren van de stylesheets waarmee het document getransformeerd kan worden. Op die manier zou het programma kunnen analyseren welke knooppunten, die nodig zijn voor de transformatie, in de boom moeten zitten. Tevens zouden deze knooppunten enkel in de boom zitten op het moment dat de transformatie moet uitgevoerd worden. Een alternatieve denkpiste is het analyseren van het XML Schema [schema] dat geassocieerd is met het te transformeren document. Zo zou het programma te weten kunnen komen welke elementen op welke plaats mogen voorkomen en de op te bouwen boom op die manier kunnen optimaliseren.

12.3. JAXP

Voor deze sectie hebben we een beroep gedaan op [JAXP], [JAXPFAQ], [JAXPSPEC], [BMAAJAXP] en [BMJRJAXP].

JAXP is een API van Sun en staat voor Java API for XML Parsing/Processing, waarvan de referentie-implementatie momenteel aan versie 1.1.2 zit. Strict genomen is JAXP wel een

API, maar het zou nauwkeuriger zijn om het een abstractielaag te noemen. JAXP alleen voorziet niet in parsing functionaliteit. Zonder SAX, DOM of een andere XML parsing API is het onmogelijk om met JAXP XML te parsen. JAXP voorziet niet in een nieuwe manier om XML te parsen en voegt ook niets nieuws toe aan SAX of DOM. Deze API is daarentegen bedoeld om het afhandelen van taken die met SAX en DOM nogal moeilijk zijn makkelijker te maken. Ook wil deze API voorzien in de mogelijkheid om taken, specifiek aan een bepaalde parser, zoals SAX en DOM, af te handelen op een manier, onafhankelijk van de implementatie van de parser (parser van een andere leverancier bijvoorbeeld). JAXP bestaat slechts uit zes klassen, waarvan er twee dienen voor foutenafhandeling. JAXP voorziet een uniforme manier om verschillende implementaties van parsers aan te spreken en om aan de resultaten te komen. Het is vermeldenswaard dat er geen recompilatie van de applicatie nodig is indien er van parserimplementatie gewisseld wordt, dit kan heel eenvoudig aangegeven worden door bijvoorbeeld een Java-systeem eigenschap een andere waarde te geven. Om het kort samen te vatten: JAXP definieert een abstractielaag zodat de SAX en DOM API's op een leverancier/implementatie-onafhankelijke manier kunnen aangesproken worden.

Vanwaar komt dan de verwarring dat JAPX wel een parser is? De distributie van de eerste versie (v1.0) werd geleverd met een parser van Sun. Deze parser (Crimson, tegenwoordig deel uitmakend van het XML project van Apache) maakt deel uit van de JAXP distributie, maar niet van de JAXP API. Dit eigenlijk om de eenvoudige reden dat JAXP out-of-the-box kan gebruikt worden, en dat heeft voor nogal wat verwarring gezorgd. Zoals in de vorige paragraaf aangehaald is de hele bedoeling van JAXP om onafhankelijkheid van de leverancier te bekomen. Wanneer je gebruik maakt van JAXP is het mogelijk om met dezelfde code gebruik te maken van de parser van Sun, ofwel van de Xerces XML Parser van Apache, of van de XML Parser van Oracle, om er maar enkele te noemen.

De huidige versie van JAXP ondersteunt SAX 2.0 en DOM Level 2. Het is zelfs gebleken dat de veranderingen tussen de verschillende versies van JAXP vrij transparant zijn gebleven voor de ontwikkelaar, hiermee bedoelen wij dat er nauwelijks code aangepast moest worden bij het omschakelen naar de nieuwere versies. Althans volgens Sun.

12.3.1. Aanmaken van parser via JAXP

JAXP definieert een mechanisme om een SAXParser of een DOMParser aan te spreken, zonder te weten welke implementatie je nu gebruikt. Maar je moet wel nog kiezen indien je een SAXParser of een DOMParser wilt. We gaan nu eerst behandelen hoe je met behulp van JAXP een SAXParser creëert en vervolgens bespreken we dit voor een DOMParser.

12.3.1.1. Aanmaken van een SAXParser

JAXP biedt de mogelijkheid om een SAXParser aan te maken zonder je te moeten bekommeren om wie de implementatie gemaakt heeft. JAXP maakt daarom gebruik van een **SAXParserFactory** klasse die de sleutel vormt om gemakkelijk van implementatie te kunnen veranderen. We geven nu eerst een voorbeeld en zullen dat vervolgens uitleggen.

Voorbeeldcode 12.10 Aanmaken SAXParser

```
// JAXP
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
// SAX
```

```
import org.xml.sax.SAXException;

...

try {
   // Instantie van SAXParserFactory opvragen
    SAXParserFactory factory = SAXParserFactory.newInstance();
   // SAXParser vragen aan de SAXParserFactory
    SAXParser parser = factory.newSAXParser();
   // Document test.xml parsen
    parser.parse("test.xml",new ExampleSAXEventHandler());
} catch (ParserConfigurationException pce) {
    System.out.println("De parser ondersteunt enkele features niet.");
} catch (FactoryConfigurationError fce pce) {
    System.out.println("Fout bij het verkrijgen van de SAXParserFactory.");
} catch (Exception e) {
    e.printStackTrace();
}
...
```

We importeren eerst de juiste klassen en interfaces vooraleer we JAXP gaan gebruiken. Het eerste wat we doen is een instantie van de **SAXParserFactory**. Vervolgens vragen we aan de **SAXParserFactory** om ons een **SAXParser** te leveren. Met behulp van deze parser kunnen we het bestand *test.xml* parsen. We gebruiken een instantie van **ExamepleSAXEventHandler** om de ontvangen events af te handelen. Merk op dat de **parse** methode twee argumenten meekrijgt. Het eerste argument geeft de te parsen bron aan en deze bron kan, zoals in het voorbeeld, een **String** zijn die de URI van de bron bevat, maar ook mogelijk zijn: **File**object, **InputSource** en **InputStream**. Het tweede argument geeft aan welk object voor de verwerking van de ontvangen events zorgt. Dit kan een object zijn van het type **DefaultHandler** of van het type **HandlerBase**. Het gebruik van objecten van het type **HandlerBase** wordt afgeraden omdat dit type in SAX 2.0 als verouderd is aangegeven.

De uitzonderingen **ParserConfigurationException** en **FactoryConfigurationError** zijn uitzonderingen specifiek aan JAXP. De uitzondering **ParserConfigurationException** wordt gegooid wanneer een bepaalde eigenschap niet beschikbaar is bij de gebruikte parser. De uitzondering **FactoryConfigurationError** wordt gegooid wanneer er een fout is met het verkrijgen van een instantie van **SAXParserFactory** of met het configureren hiervan.

Omdat **SAXParserFactory** in de JAXP-specificatie [JAXPSPEC] gedefinieerd is als een abstracte klasse moet je aangeven welke concrete implementatie moet gebruikt worden. De methode **newInstance()** gebruikt daarvoor de volgende procedure:

- de systeemvariabele **javax.xml.parsers.SAXParserFactory** gebruiken
- opzoeken in het bestand *lib/jaxp.properties* in de JRE directory welke implementatie moet gebruikt worden
- indien de Services API beschikbaar is, deze gebruiken om de klassenaam op te zoeken
- default **SAXParserFactory** instantie, afhankelijk van het gebruikte platform

Door een andere **SAXParserFactory** te configureren bepaal je welke parser JAXP in de achtergrond zal gebruiken.

12.3.1.2. Aanmaken van een DOMParser

We zullen nu een voorbeeld geven van het aanmaken en het gebruik van een DOMParser.

Voorbeeldcode 12.11 Aanmaken SAXParser

```
// JAXP
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
// DOM
import org.w3c.dom.Document;

...

try {
   // Instantie van DocumentBuilderFactory opvragen
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
   // DocumentBuilder vragen aan de DocumentBuilderFactory
    DocumentBuilder builder = factory.newDocumentBuilder();
   // Document test.xml parsen
    Document doc = builder.parse("test.xml");
} catch (ParserConfigurationException pce) {
    System.out.println("De parser ondersteunt enkele features niet.");
} catch (FactoryConfigurationError fce pce) {
    System.out.println("Fout bij het verkrijgen van de DocumentBuilderFactory.");
} catch (Exception e) {
    e.printStackTrace();
}
...
```

De manier van werken met de **DocumentBuilderFactory** is volledig analoog aan het werken met de **SAXParserFactory**. Het verschil zit hem hier in het feit dat de **parse** methode een **Document** teruggeeft en maar één argument heeft. Dit verschil laat zich verklaren door het verschil tussen DOM en SAX. DOM is een boomgebaseerde interface terwijl SAX een eventgebaseerde interface is.

JAXP biedt dus een uniforme manier aan om een parser (zij het SAX of DOM) te verkrijgen. Het is het doel van de **factory** om het vendor-specifieke voor het verkrijgen van een parser te verbergen. Op die manier kan er gemakkelijk voor de implementatie van een andere vendor gekozen worden zonder dat er iets aan de code voor het parsen van het document verandert.

13. jnome

jnome is een open source project aan de Faculteit Toegepaste Wetenschapen, departement Computerwetenschappen, van de Katholieke Universiteit Leuven. Dit project biedt een metamodel voor Java en is zelf geïmplementeerd in Java. Op basis van het metamodel werd een applicatie ontwikkeld met de naam **jnomedoc**, die documentatie genereert voor Java bronbestanden. Het metamodel en jnomedoc vormden de basis voor de eindverhandeling van Kristof Mertens en Nele Smeets [NSKMMMJ]. jnomedoc droeg toen nog de name SoFT, wat staat voor **So**urce **F**ile **T**ool.

jnomedoc is een programma dat een reeks Java bestanden inleest, ontleedt en er XML bestanden van genereert. Deze XML bestanden bieden een overzicht van de Java elementen uit de ingelezen bestanden samen met hun specificatie. Om de Java bestanden voor te kunnen stellen in het geheugen, werd een metamodel voor Java ontwikkeld. Dit model is onafhankelijk van de invoer- en uitvoerstructuur en daardoor herbruikbaar.

In dit hoofdstuk zullen we een korte bespreking geven van jnome en het metamodel dat de basis vormt van jnome. Vervolgens zullen we jnome situeren in onze thesis.

13.1. jnome: kort overzicht

In dit overzicht zullen we eerst kort een beeld schetsen van het opgebouwde metamodel voor Java. Het tweede deel van dit overzicht behandelt de werking van jnomedoc, die gebaseerd is op het opgebouwde metamodel. De bedoeling van dit korte overzicht is de lezer een idee te geven van het metamodel en de werking van jnomedoc, als kader voor ons eigen werk, maar we zullen zeker geen compleet beeld (kunnen) bieden.

13.1.1. Korte beschrijving van het metamodel voor Java

13.1.1.1. Aandachtspunten

Bij de opbouw van het metamodel van jnome is veel aandacht besteed aan het ontwikkelen van een nauwgezette modellering van de verschillende concepten uit de Java taal. Ook het vermijden van code vervuiling had een hoge prioriteit. De ontwikkelde Java code is uitvoerig gedocumenteerd met informele en formele beschrijvingen van precondities, postcondities en type invarianten, gebruik makend van (een uitbreiding van) de javadoc specificatie syntax [JAVADOC].

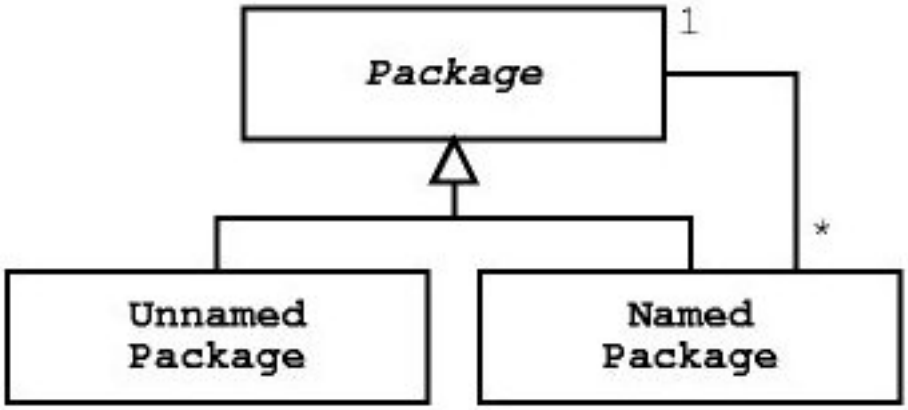
Het huidige metamodel biedt ondersteuning voor de volgende concepten: pakketten, compilation units, klassen, interfaces, primitieve types, array types, variabelen, methodes en documentatiecommentaar. In de volgende sectie wordt een korte beschrijving gegeven van de modellering van enkele van de bovenstaande concepten. De beschrijving is verre van volledig en is enkel bedoeld om de lezer een idee te geven van het bestaande metamodel en hem te helpen bij het begrijpen van de rest van de thesistekst.

Het metamodel biedt nog geen ondersteuning voor geneste klassen, de implementatie van methodes, initializatiecode en nog enkele gereserveerde woorden.

13.1.1.2. Het model

13.1.1.2.1. Pakketten

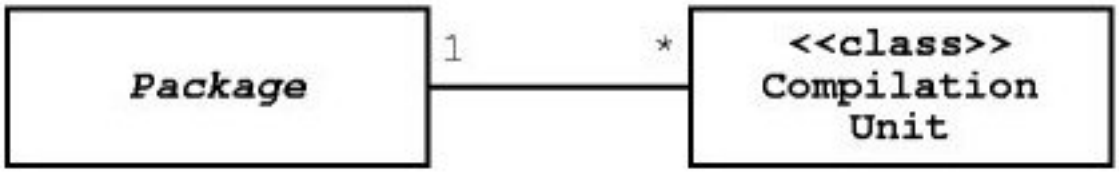
In het metamodel worden "gewone" pakketten, zoals **java** en **util**, voorgesteld door het type **NamedPackage**. Het verzuimpakket wordt gemodelleerd via het type **UnnamedPackage**. Zie onderstaande figuur.



Figuur 13.1. De pakkethiërarchie

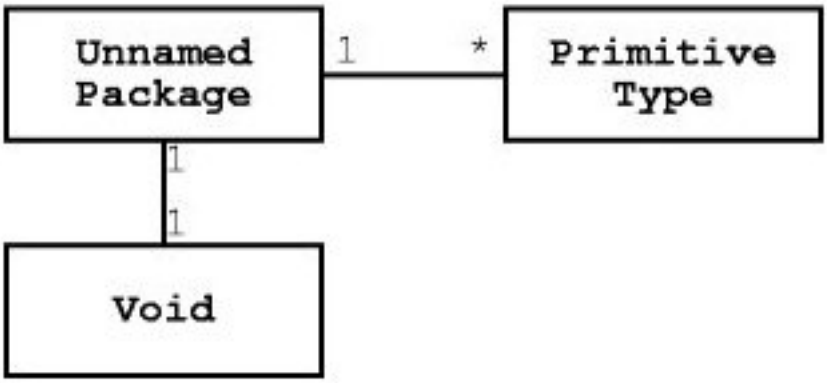
Top level pakketten worden in het verzuimpakket geplaatst, waardoor dit pakket de wortel vormt van de pakkethiërarchie en bijgevolg de wortel van de hele objectstructuur.

Elk pakket kan nul of meer compilation units bevatten.



Figuur 13.2. Pakket en compilation unit

Het verzuimpakket bevat niet alleen referenties naar de top level pakketten, maar ook naar elk van de primitieve types en naar het resultaattype void. Op deze manier kan afgedwongen worden dat er precies één object van de klasse **Void**, **Char**, **Byte**, **Short**, **Int**, enzovoort aanwezig is in de objectstructuur.

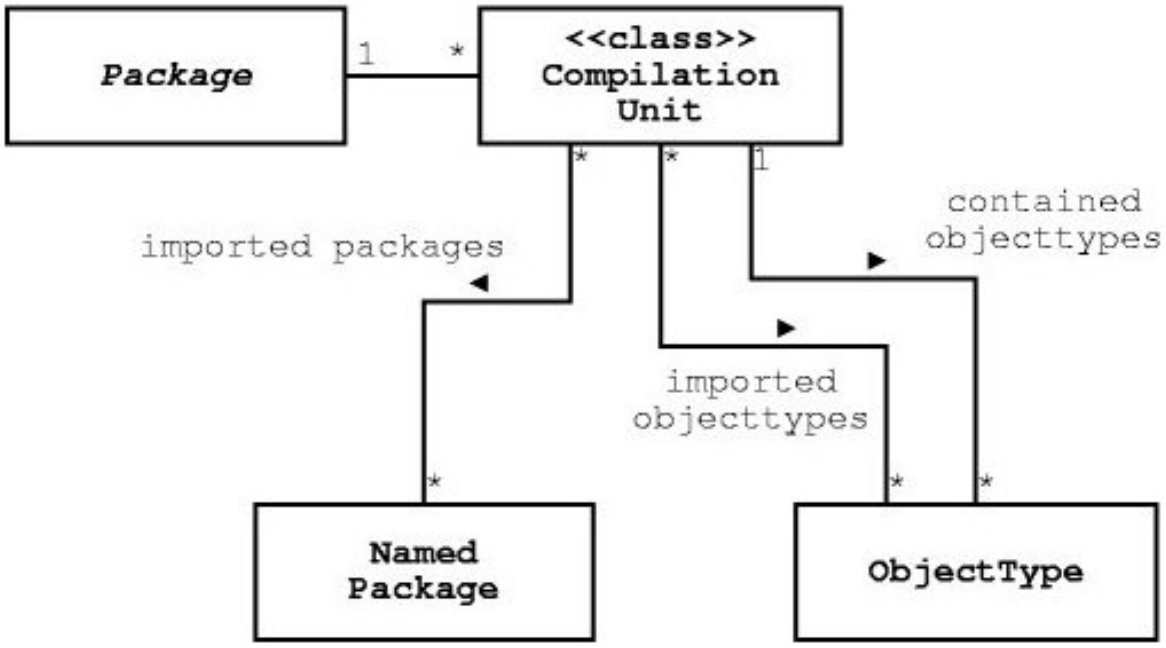


Figuur 13.3. Het verzuimpakket, de primitieve types en het resultaattype void

13.1.1.2.2. Compilation units

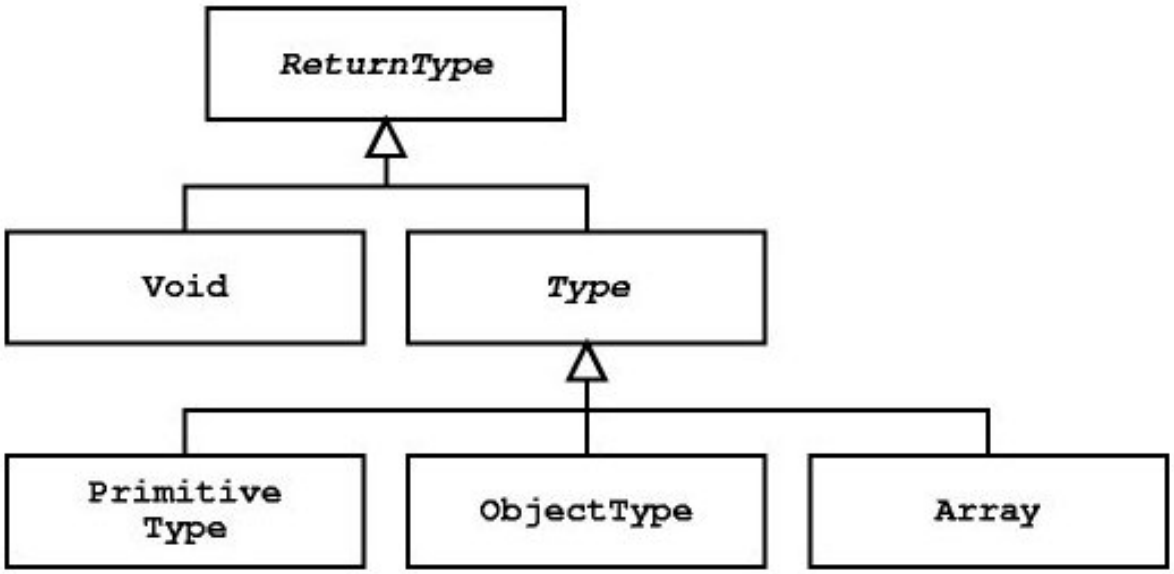
Onderstaande figuur toont hoe de drie delen van een compilation unit uitgedrukt worden in het metamodel. Deze drie delen zijn:

- pakket declaratie
- import statements
- type declaraties



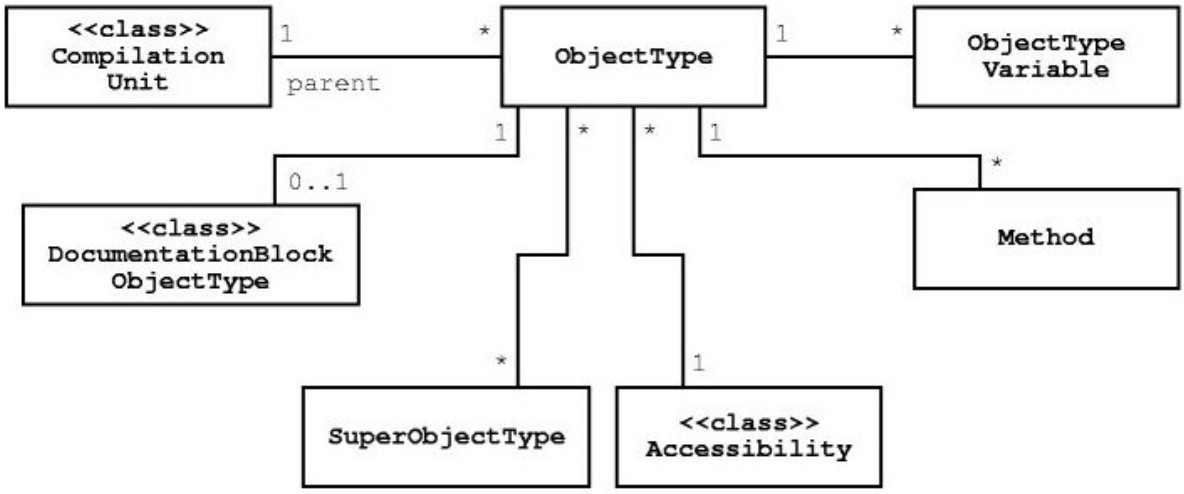
Figuur 13.4. De drie delen van de compilation unit

In Java zijn er twee soorten types: primitieve types en referentie types. De referentie types omvatten klassen, interfaces en array types. Verder is er ook het resultaattype void dat gebruikt wordt wanneer een methode geen resultaat teruggeeft. In jnome worden types gemodelleerd zoals in onderstaande figuur wordt aangegeven.



Figuur 13.5. Modelleren van de resultaattypes

We geven enkel wat meer uitleg over objecttypes; de andere types worden hier niet verder besproken. Een objecttype kan nul of meer variabele declaraties en methode declaraties bevatten. Verder kan een objecttype vergezeld worden van een documentatie blok en kan het nul of meer superklassen of superinterfaces hebben.



Figuur 13.6. Relaties van het objecttype

13.1.1.3. Variabelen/Methodes/Documentatie commentaar

Voor een gedetailleerde bespreking van het metamodel voor variabelen, methodes en documentatie commentaar verwijzen we naar [NSKMMMJ-6].

13.1.1.4. Unresolved elementen

In de praktijk wordt een objectstructuur opgebouwd voor een eindig aantal compilation units.

In dit geval is het mogelijk dat een bepaalde klasse behandeld wordt, terwijl zijn superklasse niet beschouwd wordt. Het is dan onmogelijk om een referentie te leggen van een klasse naar zijn superklasse. De problemen die opduiken door het afwezig zijn van bepaalde Java elementen in de objectstructuur worden expliciet gemodelleerd in het metamodel.

Zoals hierboven vermeld is de superklasse van een klasse niet altijd aanwezig in de objectstructuur. Hetzelfde geldt voor:

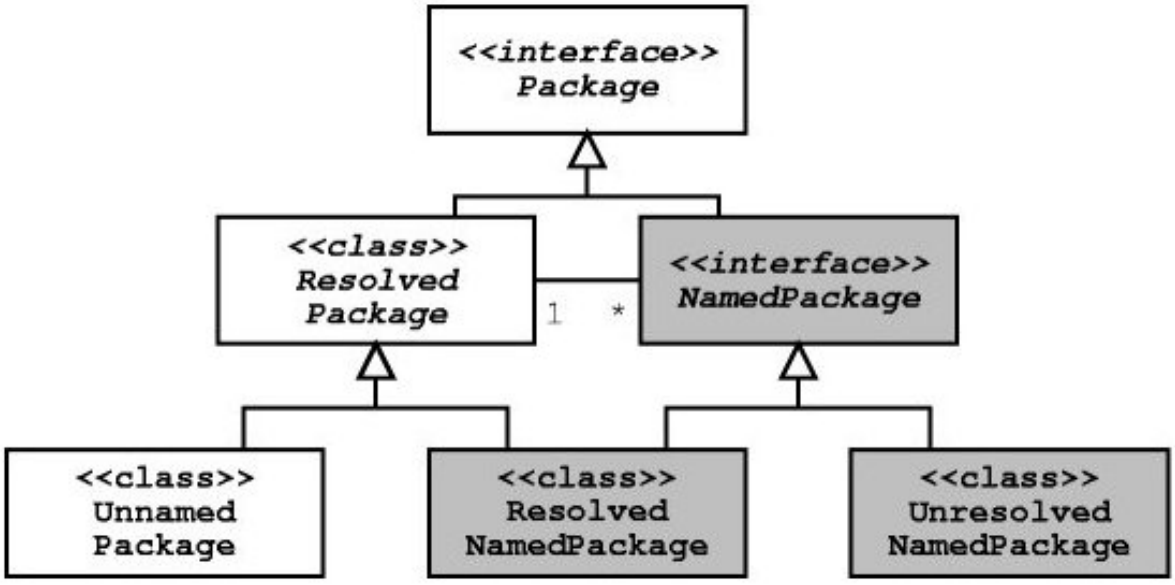
- een geïmporteerd pakket
- een geïmporteerd objecttype
- een superinterface
- een superklasse
- een klasse in de throws clause van een methode
- het resultaattype van een methode
- het type van een variabele

Bovenstaande opsomming is exhaustief voor het huidige metamodel. Ze zal echter moeten uitgebreid worden wanneer, bijvoorbeeld, ook methode-oproepen worden beschouwd.

Voor elk van bovenstaande elementen is een corresponderend resolved en unresolved element in het metamodel geïntroduceerd. Bijvoorbeeld, **ObjectType** heeft twee subtypes, namelijk **ResolvedObjectType** en **UnresolvedObjectType**.

Wanneer het niet gegarandeerd is dat een bepaald element aanwezig is in een objectstructuur, wordt het algemene type gebruikt, hetgeen betekent dat zowel het resolved als het unresolved type op die plaats mogelijk is. Bijvoorbeeld, het type van een variabele heeft statisch type **Type**. Het concrete object dat het type van een variabele voorstelt kan zowel van het type **ResolvedType** als van het type **UnresolvedType** zijn.

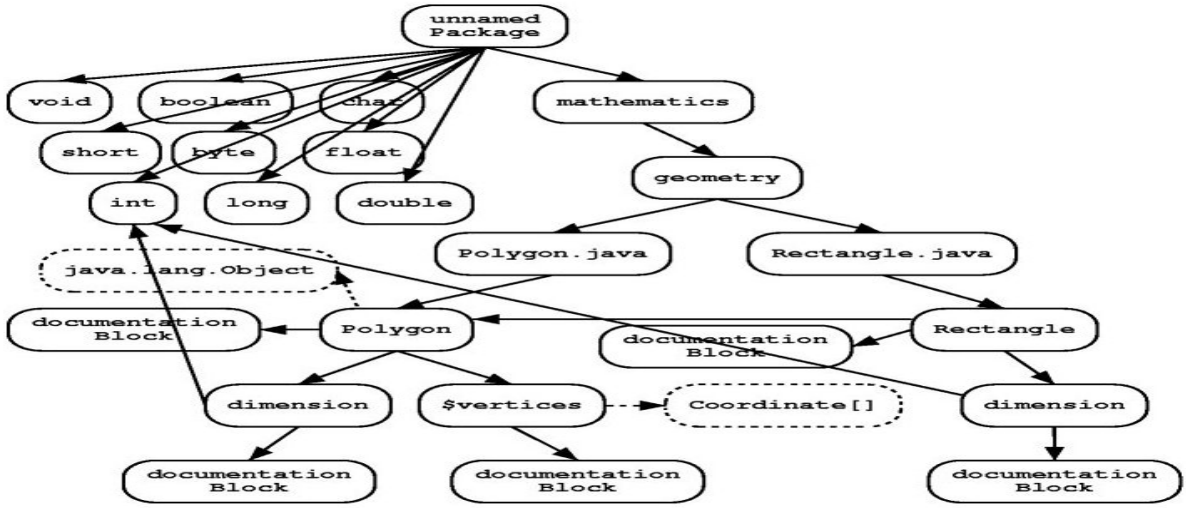
In de volgende figuur zie je het meta model voor unresolved pakketten. Een beschrijving van de andere unresolved elementen vind je in [NSKMMMJ-66].



Figuur 13.7. Het meta model voor unresolved pakketten

13.1.1.5. Een voorbeeld van een objectstructuur

Volgende figuur toont een voorbeeld van een instantie van het meta model. De objecten in stippellijn zijn unresolved.



Figuur 13.8. Instantie van het meta model

Voor nog meer uitleg over het metamodel verwijzen wij naar [NSKMMMJ-6].

13.1.2. Werking van jnomedoc

Het programma jnomedoc leest een reeks Java bestanden in, ontleedt deze en genereert tenslotte een aantal XML bestanden ervan. Die XML bestanden bieden een overzicht van de Java elementen uit de ingelezen bestanden samen met hun specificatie. Dit proces verloopt in drie fasen:

- in de eerste fase worden de Java bestanden geparst
- in de tweede fase wordt een Java object aangemaakt voor elke klasse, methode, variabele, ... in de geparste bestanden
- in de derde fase worden XML bestanden gegenereerd vanuit de opgebouwde objectstructuur

13.1.2.1. De drie fases

Aan de invoer krijgt jnome één of meerdere Java bestanden, die ontleed worden en waarvoor documentatie wordt gegenereerd. Dit proces bestaat uit drie fasen, die we al eerder hebben opgenoemd en die we nu kort zullen toelichten.

13.1.2.1.1. Eerste fase: lexen en parsen

In de eerste fase worden de gegeven Java bestanden geparst en omgezet in een boomstructuur, een AST (Abstract Syntax Tree of een abstract syntaxboom) [NSKMMMJ-35]. De nodes in de opgebouwde boom zijn allemaal van dezelfde (redelijk primitieve) klasse. De enige informatie die deze klassen bevatten, is de naam en het type van het token dat door die node wordt voorgesteld. De enige methodes die hierop kunnen worden toegepast zijn dan ook methodes om het type en de naam op te vragen, en enkele primitieve navigatie-methodes.

Voor deze informatie deden we een beroep op [NSKMMMJ-345].

Om de Java bestanden te kunnen parsen, moest men eerst een parser vinden voor Java bestanden. Er werd besloten om gebruik te maken van de parser generator ANTLR. Een parser generator krijgt als invoer een formele beschrijving van een grammatica en levert als uitvoer broncode voor een parser. Deze parser kan dan gebruikt worden voor het ontleden van bestanden die voldoen aan de gegeven grammatica.

In jnome wordt er gewerkt met twee verschillende **lexers** (lexical analyzer) en twee verschillende parsers. Hiervoor is gekozen omdat een Java programma ook uit twee delen bestaat, namelijk Java code en commentaarblokken. Zo zou bijvoorbeeld bij het gebruik van één enkele lexer dubbelzinnigheden kunnen optreden wanneer men de token definities van beide delen zou samenvoegen. Traditioneel lost men dit probleem op door met een lexer te werken die zich in verschillende toestanden kan bevinden. De ontwikkelaars van jnome hebben echter gekozen om met twee verschillende lexers te werken. Ten eerste omdat deze eenvoudiger te schrijven zijn, vermits er geen dubbelzinnigheden meer zijn en ten tweede omdat deze onderdelen dan eenvoudiger te hergebruiken zijn.

Net zoals voor de lexers zijn er twee **parsers**. Eén parser dient voor de analyse van de Java code, de andere voor de commentaarblokken. Deze parsers maken gebruik van dezelfde invoerstroom, net zoals de beide lexers eenzelfde invoerstroom gebruiken. Wanneer de parser voor de Java code het begin van een commentaarblok tegenkomt, dan draagt deze parser de controle over aan de parser voor commentaarblokken. De parser voor commentaarblokken heeft dan de controle totdat deze het einde van het commentaarblok tegenkomt en draagt de controle dan terug over aan de andere parser.

Tussen de lexers en de parsers zit nog een derde component, namelijk een **selector**. De selector dient om de twee stromen, gegenereerd door de twee lexers, samen te nemen tot één stroom die door de parsers gebruikt wordt. Op voorhand is aangegeven aan de selector wanneer deze een andere lexer moet selecteren om de invoerstroom te behandelen.

Omdat er geen grammica voorhanden was die commentaarblokken beschreef, hebben de auteurs van jnome zelf een grammatica hiervoor ontwikkeld. In jnome worden commentaarblokken geparst die zijn opgebouwd zoals de specificatieblokken uit de cursus IPS [SDB99]. Voor een gedetailleerde beschrijving van deze commentaarblokken verwijzen wij naar [NSKMMMJ-532]. Op deze manier is het ook mogelijk om, naast een AST voor de Java code, een AST voor documentatieblokken te krijgen.

Door het parsen van het volledige Java bestand wordt het vlakke bestand (een lineaire opeenvolging van tokens) omgezet in een boomstructuur die de syntactische structuur van het Java bestand weergeeft. Deze boom wordt Abstract Syntax Tree genoemd. De in jnome gecreëerde AST is homogeen. Dit wil zeggen dat alle nodes in de boom objecten zijn van dezelfde klasse. Zoals reeds eerder gezegd is deze klasse redelijk primitief en bevat deze slechts weinig informatie en biedt deze een beperkt aantal navigatie-methodes. De reden voor de keuze van een homogene AST in plaats van een heterogene AST, waar verschillende concepten worden voorgesteld door objecten van verschillende klassen, is beschreven in [NSKMMMJ-4643].

13.1.2.1.2. Tweede fase: opbouwen objectmodel

Het einddoel van de tweede fase is het bekomen van een goede, object-gerichte structuur in het geheugen voor de verschillende Java elementen die in de geparste bestanden voorkomen. Dit gebeurt in twee deelfasen en levert een instantie van het metamodel op.

In een **eerste deelfase**, ook de **acquire** fase genoemd, wordt, vanuit de bekomen ASTs, een Java object aangemaakt voor elke klasse, methode, variabele, ... in de geparste bestanden. Deze objecten worden opgenomen in een boom: elk pakket bevat referenties naar zijn subpakketten en naar de compilation units die in dat pakket zijn gedefinieerd, elke klasse bevat referenties naar zijn methodes en variabelen, ...

In een **tweede deelfase**, ook de **resolve** fase genoemd, worden dan (indien mogelijk) kruiselingse verwijzingen gelegd tussen de verschillende objecten: zo wordt vanuit elke variabele de referentie gelegd naar zijn type, vanuit een klasse naar zijn superklasse, ...

13.1.2.1.3. Derde fase: uitvoer genereren

Aan deze fase besteden we wat meer aandacht omdat we aan deze fase wijzigingen hebben aangebracht. We zullen eerst uitleggen hoe de uitvoer eerst gebeurde en vervolgens bespreken wij de wijzigingen die we aangebracht hebben. Wanneer we over objectmodel, model of metamodel spreken gaat het effectief over het metamodel. Indien we spreken over de objectstructuur dan bedoelen we de invulling van het model met objecten, een instantie van het objectmodel als het ware.

In de derde fase wordt uitvoer gegenereerd vanuit de gecreëerde objectstructuur. Er worden XML bestanden gegenereerd die allerlei informatie bevatten over de geparste Java bestanden: voor elke geparste klasse en interface wordt een XML bestand gemaakt, maar met daarin de fully qualified name van het objecttype en informatie over zijn methodes en variabelen. De gegenereerde XML bestanden worden met behulp van XSLT getransformeerd naar andere XML bestanden die met behulp van CSS (**C**ascading **S**tyle **S**heets) kunnen bekeken worden in een moderne browser. Vanuit elk XML bestand kan onmiddellijk overgegaan worden naar het XML bestand dat een gerelateerde klasse of interface beschrijft. Zo kan vanuit het XML bestand van een klasse overgegaan worden naar de superklasse, het type van een variabele of formele parameter.

Het genereren van de uitvoer vanuit de jnomedoc objectstructuur gebeurt met behulp van het **Worker** patroon. Door gebruik te maken van het **Worker** patroon wordt de code voor het genereren van uitvoer afgezonderd in aparte **Worker** klassen. Op deze manier wordt het metamodel niet vervuild met uitvoercode. Voor een verdere bespreking van dit patroon verwijzen wij naar [NSKMMMJ-913], [NSKMMMJ-92]. De Workers heten in het geval van jnome **Writer**s. Voor het aanmaken van deze writers wordt in het hoofdprogramma beroep gedaan op een **WriterFactory** [NSKMMMJ-93]. Het gebruik van het Factory patroon zorgt ervoor dat de code voor het aanmaken van de writers afgezonderd is. Ook is het mogelijk op basis van dit patroon een framework van workers, in dit geval writers, aan te maken. Door te wisselen van factory kan er een andere worker-structuur gekozen worden.

De inhoud van de gegenereerde XML bestanden reflecteert de structuur van verschillende Java elementen. De XML bestanden, die worden gegenereerd, zijn onder te verdelen in twee soorten. Een eerste soort beschrijft de **pakketten** en de tweede soort beschrijft **klassen** en **interfaces**.

Voor elk **pakket** in de verkregen objectstructuur wordt een XML bestand *package.xml* gegenereerd. Dit bestand bevat de fully qualified name van het pakket, samen met de namen van zijn subpakketten en compilation units. Voor elke compilation unit zijn ook de namen opgegeven van de objecttypes die in die compilation unit zijn gedefinieerd. Uiteraard kan enkel informatie gegeven worden over de compilation units die ook effectief mee geparst zijn.

Voor elke **klasse** en **interface** wordt een XML bestand gegenereerd met de naam van dit objecttype. Bijvoorbeeld voor een klasse met de naam **Object** (tevens het objecttype van deze klasse) wordt een XML bestand met de naam **Object** aangemaakt, eventueel met een bepaalde extensie. Dit bestand bevat allerlei informatie over het objecttype in kwestie: de modifiers, url, fully qualified name, methodes en variabelen. Verder worden alle methodes (in de geparste bestanden) vermeld die het bewuste objecttype als resultaattype hebben en de variabelen met dat objecttype als type. Voor een beschrijving van de exacte inhoud verwijzen we naar [NSKMMMJ-95].

De XML documenten bevatten voldoende informatie zodat verwijzingen tussen Java elementen in verschillende bestanden mogelijk zijn.

De gegenereerde XML bestanden bevatten een verwijzing naar een DTD die het XML document beschrijft alsook een verwijzing naar een XSLT stylesheet die dient om het XML bestand om te zetten naar een formaat dat door moderne browsers met behulp van CSS te tonen is. Al deze technieken worden in jnome gebruikt om uitvoer te genereren die voldoet aan de volgende criteria:

- data en opmaak worden volledig gescheiden gehouden
- kleine wijzigingen in de weergave zijn makkelijk en snel aan te brengen

Het eerste criterium zorgt ervoor dat de data in verschillende situaties kunnen gebruikt worden, situaties die niets met weergeven te maken hebben bijvoorbeeld. Het tweede criterium geeft de gebruikers de mogelijkheid om de weergave van documentatie aan te passen aan de persoonlijke voorkeur en dit op een eenvoudige en snelle manier.

Omdat XML beschouwd wordt als een uitbreidbare taal, in tegenstelling tot een vast omlijnde taal als HTML, gaat de controle op de correctheid verloren. Om een aantal testen op XML bestanden te kunnen uitvoeren wordt in jnome gebruik gemaakt van DTD. DTD wordt gebruikt om een bepaalde structuur op te leggen aan documenten. Maar DTD is beperkt. Zo ondersteunt DTD bijvoorbeeld geen datatypering en daardoor is het niet mogelijk aan te geven dat een bepaalde waarde van een bepaald type moet zijn. Deze DTD's weerspiegelen het objectmodel, maar kunnen bepaalde beperkingen niet accuraat weergeven.

13.2. jnome in onze thesis

Vermits jnome XML bestanden genereerde als uitvoer was de link met onze thesis snel gelegd. We hebben twee aspecten van jnome bekeken en herwerkt. Ten eerste hebben we het genereren van de uitvoer onder handen genomen en ervoor gezorgd dat de uitvoer met behulp van JDOM kon gebeuren [Sectie 13.2.1.] . Vervolgens hebben we de DTD's bekeken en de DTD's omgevormd naar een equivalent XML Schema. Deze DTD's weerspiegelen het opgebouwde metamodel [Sectie 13.1.1.] . We hebben dit gewoon genomen zoals het was, we hebben de DTD's niet herwerkt [Sectie 13.2.2.] . Tenslotte hebben we ook een eigen generator geschreven voor Cocoon 2 zodat de gegenereerde XML inhoud rechtstreeks aan Cocoon 2 kan afgeleverd worden [Sectie 13.2.3.] . Op die manier hebben we een tussenstap, met name het bewaren van de bestanden op schijf, geëlimineerd.

13.2.1. Uitvoer

Zoals in [Sectie 13.1.2.1.3.] uitgelegd, bestaat de uitvoer van jnomedoc uit een reeks XML documenten. Het genereren van deze XML documenten gebeurde door alle tekens letterlijk naar een eigen object **XMLPrintWriter** te schrijven. Dit object wordt geïnitialiseerd met een **OutputStream** die zal instaan voor het effectief wegschrijven van de documenten.

In de oude code wordt een XMLPrintWriter object aangemaakt. Dit object wordt aan alle **write** methodes doorgegeven en op dit object worden dan de methodes **print** en **println** toegepast om het XML document te genereren. Sommige **write** methodes roepen andere **write** methodes op en via deze aaneenschakeling van methodes ontstaat uiteindelijk het uitvoerdocument. De initialisatie van deze "keten" gebeurt als volgt:

Voorbeeldcode 13.1 ResolvedReturnTypeWriter.java

```
...

XMLPrintWriter writer = getWriterFactory().createXMLPrintWriter(relPath);
write(rrt,writer);
writer.close();
...
```

Eerst wordt via de **WriterFactory** een nieuw XMLPrintWriter object aangemaakt. De methode **createXMLPrintWriter** neemt een parameter **relPath**. Deze parameter bevat de relatieve locatie van het te creëren bestand. De locatie is relatief ten opzichte van de uitvoerdirectory die meegegeven wordt aan jnome [NSKMMMJ-212]. De methode **createXMLPrintWriter** maakt een correct geïnitialiseerde **OutputStream** aan en geeft vervolgens een XMLPrintWriter terug die met deze OutputStream is geïnitialiseerd.

Vervolgens wordt de **write** methode uitgevoerd, die hier twee parameters neemt. De parameter **rrt** is het object, in dit geval een ResolvedReturnType, waarvoor de documentatie gegenereerd gaat worden. **writer** is de tweede parameter en dit is de XMLPrintWriter die we in de vorige stap hebben aangemaakt. Alle informatie in verband met **rrt** zal naar deze writer geschreven worden.

Tenslotte moet, door de **close()** methode toe te passen op het object **writer**, nog duidelijk gemaakt worden dat de uitvoer volledig genereerd is en de uitvoerstroom afgesloten mag worden.

We geven nu twee voorbeelden van typische **write** methodes:

Voorbeeldcode 13.2

```
...

protected void write (ResolvedReturnType rrt, XMLPrintWriter writer) {
    // write the xml version
    writer.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
    writer.println();
    // open the returnType tag
    writer.println("<returnType>");
    writer.println();
    ...

    // write name and context of the return type
    writeNameContext(rrt,writer);
    // write the methods that return this objecttype
    writeMethodsReturningThisType(rrt,writer);
    ...

    // close the returnType tag
    writer.println("</returnType>");
    writer.println();
}

...
```

Voorbeeldcode 13.3

```
...

public void write (XMLPrintWriter writer) {
    writer.print("<url>");
    writer.print(getURL().toString());
    writer.println("</url>");
}

...
```

Het eerste van deze twee voorbeelden is in feite de methode die het begin is van de keten. De XML declaratie wordt eerst weggeschreven en vervolgens wordt het root element **returnType** geopend. Daarna worden er een aantal methodes opgeroepen die bepaalde inhoud wegschrijven. De methode **writeMethodsReturningThisType** maakt een nieuwe keten van **Writers** aan die informatie over een aantal methodes naar het object **writer** wegschrijven. Nadat deze methodes afgehandeld zijn wordt het root element gesloten.

In het tweede voorbeeldje wordt het element **url** met bepaalde inhoud toegevoegd.

We hebben voor het tweede voorbeeldje een korte methode genomen, maar deze volstaat in onze ogen om het volgende principe te illustreren. Zoals duidelijk te zien is in het voorbeeld is de programmeur verantwoordelijk voor het openen van het element (**<url>**), het toevoegen van de inhoud (**getURL().toString()**) en het afsluiten van het element (**</url>**). Voor deze methode is het nog eenvoudig om het overzicht te bewaren en ervoor te zorgen dat de uitvoer uit goedgevormde XML bestaat. Maar wanneer we grotere methodes hebben, zoals het eerste voorbeeldje, dan is het niet ondenkbaar dat er fouten gemaakt gaan worden. Het is heel goed mogelijk dat de programmeur per ongeluk één van de methodes, die nu nog binnen het openen en het sluiten van het **returnType** element staat, per ongeluk hier buiten zet. We krijgen dan een XML document met twee root elementen, wat niet toegestaan is. Deze fout zal pas ontdekt worden bij het genereren van de XML bestanden. Dit is nog een fout die makkelijk gevonden zal worden, maar er zijn ergere scenario's denkbaar, bijvoorbeeld een voorwaarste slash (/) die vergeten wordt bij het afsluiten van een element. Dit is een fout die gemakkelijk over het hoofd gezien wordt.

Zou het niet handiger en eenvoudiger zijn om elementen op een eenvoudige manier aan te maken en er dan gewoon de inhoud aan toe te kunnen voegen? De aangemaakte objecten zouden dan zelf moeten instaan voor het produceren van goedgevormde XML. Dit is precies wat JDOM doet. Eén van onze opdrachten was dan ook om jnome aan te passen zodat de opbouw en het uitschrijven van de documenten via JDOM verloopt. Zoals besproken wordt in het hoofdstuk over JDOM [Hoofdstuk 11.] moet de maker van het programma er niet meer zelf voor zorgen dat onder andere elementen correct worden afgesloten, of dat speciale tekens in de inhoud van elementen of de waarde van attributen omgezet worden naar hun respectieve entiteiten, althans de voorgedefinieerde entiteiten [Sectie 2.3.3., Tabel 2.1.] . Na het herschrijven krijgt het tweede voorbeeldje de volgende vorm:

Voorbeeldcode 13.4

```
...

public void write (Element element) {
    element.addContent( ( new Element("url") ) .addContent(getURL().toString()));
```

```
}

...
```

Het voordeel van deze vorm is dat het openen en het sluiten van het element niet meer handmatig gedaan moet worden door de programmeur. Ook bij het toevoegen van inhoud aan een element moet je geen voorzieningen treffen om tekens zoals < en > om te zetten in entiteiten. Dit gebeurt allemaal op de achtergrond.

Indien we het eerste voorbeeld herwerken krijgen we de volgende code:

Voorbeeldcode 13.5

```
...

protected void write (ResolvedReturnType rrt) {
    // create the root element returnType
    Element returnType = new Element("returnType");
    // create the Document
    // JDOM will output the xml
    // declaration itself
    Document doc = new Document(returnType);
    ...

    // write name and context of the return type
    writeNameContext(rrt,returnType);
    // write the methods that return this objecttype
    writeMethodsReturningThisType(rrt,returnType);
    ...

}

...
```

Het is duidelijk dat het herwerkte voorbeeld heel wat korter is dan de oorspronkelijke vorm. Ook bestaat nu de mogelijkheid niet meer om inhoud buiten het root element te plaatsen. Een ander verschilpunt is dat we nu een **element** doorgeven om de informatie aan toe te voegen in plaats van de writer. Dit zal zo zijn in alle write methodes. We dienen voor de duidelijkheid op te merken dat we bijna nooit het root element doorgeven. Het element dat doorgegeven wordt zal vrijwel altijd een element zijn dat zich lager in de boom bevindt. Het zal aan dat element zijn dat de inhoud wordt toegevoegd.

We hebben dan wel een JDOM document gecreëerd, maar we moeten er wel nog voor zorgen dat we dit naar een bestand geschreven krijgen. Dat gebeurt op de volgende manier:

Voorbeeldcode 13.6

```
...

FileOutputStream out = new FileOutputStream(absPath);
XMLOutputter outputter = new XMLOutputter("\t",true);
outputter.output(doc,out);
out.flush();
out.close();
...
```

Voor meer uitleg over dit stukje code methodes verwijzen wij naar [Sectie 11.3.4.] .

In het voorbeeld van de oude code zie je dat tweemaal de methode **print** gebruikt is en éénmaal de methode **println**. Hier wordt al bepaald hoe de uitvoer er uit gaat zien. Bij JDOM wordt deze beslissing uitgesteld tot het moment dat er een uitvoerstroom moet gegenereerd worden. Maar, naast het schrijven naar een uitvoerstroom, is het ook mogelijk om SAX events [Sectie 12.2.] te genereren of een DOM boom [Sectie 12.1.] op te bouwen.

Op deze manier hebben we de code van jnomedoc herwerkt en ervoor gezorgd dat de uitvoer gegenereerd wordt met behulp van JDOM. We vatten de voordelen van deze werkwijze ten opzichte van de vroegere werkwijze nog eens samen. Ten eerste moet de programmeur zich niet meer bekommeren om het XML document goedgevormd te houden, dit wordt automatisch door JDOM gedaan. Ten tweede zijn ook andere uitvoermogelijkheden aanwezig, maar dit kan in het originele project ook ingebouwd worden. We vinden dat het gebruik van JDOM het voor de programmeur een stuk eenvoudiger maakt om XML documenten te genereren.

13.2.2. Omzetting van de DTD's

13.2.2.1. Het vertrekpunt: de bestaande DTD's

De DTD's dienen om de structuur van de XML bestanden aan te geven. Deze structuur vloeit voor uit het opgebouwde metamodel [Sectie 13.1.1.] . De DTD's zijn geen exacte weergave van het metamodel, maar vormen er toch een weerspiegeling van. Op die manier wordt een bepaalde structuur, voortvloeiend uit het metamodel, via de DTD's, opgelegd aan de XML bestanden. Omdat een aantal beperkingen niet met behulp van DTD's kunnen opgelegd worden, zoals datatypering, werd besloten om deze DTD's om te zetten naar XML Schema.

Oorspronkelijk hadden we vier DTD's:

- **package.dtd**: bevat de structuur voor XML bestanden die pakketten beschrijven
- **void.dtd**: bevat de structuur voor XML bestanden die het resultaattype void beschrijven
- **primitivetype.dtd**: bevat de structuur voor XML bestanden die de primitieve types beschrijven
- **objecttype.dtd**: bevat de structuur voor XML bestanden die de objecttypes beschrijven

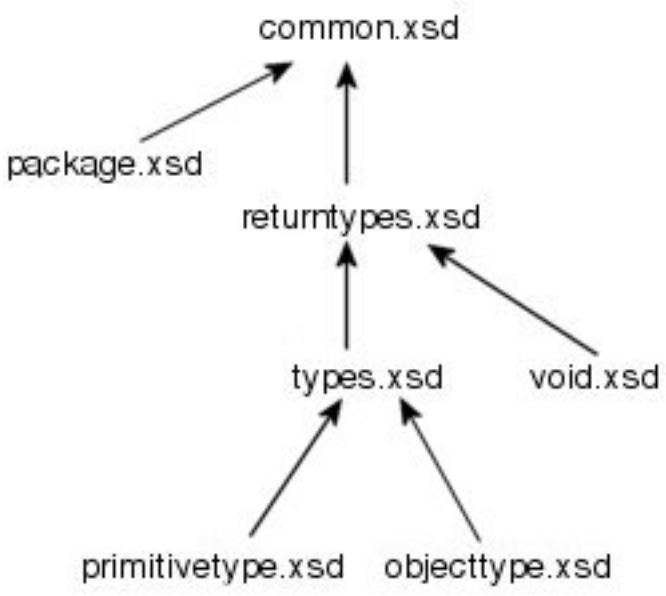
Deze DTD's hebben we onder handen genomen en omgezet naar XML Schema documenten.

13.2.2.2. Omzetting naar XML Schema

We gaan hier niet uitleggen hoe we de DTD's hebben omgezet naar een equivalent XML Schema. Dit wordt besproken in [Sectie 3.6.] , waar we **primitivetype.dtd** als voorbeeld hebben genomen te illustreren. Wel willen we nog eens vermelden dat we gewoon een omzetting hebben gerealiseerd en daarbij de mogelijkheden van de W3C Schema Language hebben bestudeerd. We hebben de bestaande DTD's hierbij niet geëvalueerd of getracht er veranderingen in aan te brengen. Wel hebben we geprobeerd alles zo streng mogelijk te typeren.

Na de omzetting van de DTD's naar XML Schema bekwamen we zeven XML Schema documenten. Dit komt omdat we zoveel mogelijk getracht hebben de gemeenschappelijke delen af te splitsen naar afzonderlijke schema's om op die manier een beter overzicht te kunnen bewaren over de schema's en duplicatie te vermijden. Op deze manieren moeten we niet telkens elk schema moesten gaan aanpassen mocht er toch iets gewijzigd moeten worden

en vermijden we inconsistenties. Dit leverde ons de volgende structuur van schema's op:



Figuur 13.9. XML Schema hiërarchie

We zullen deze schema's hier heel kort beschrijven:

- **common.xsd**: bevat een aantal definities die gemeenschappelijk zijn voor alle bestanden, zie ook [Sectie 3.4.1.] .
- **package.xsd**: importeert common.xsd en definieert de structuur voor XML bestanden die pakketten beschrijven.
- **returntypes.xsd**: importeert common.xsd en bevat daarnaast definities die gemeenschappelijk zijn voor types.xsd en void.xsd, zie ook [Sectie 3.4.2.] .
- **types.xsd**: importeert returntypes.xsd en bevat daarnaast een aantal definities die in objecttypes.xsd gebruikt worden.
- **void.xsd**: importeert returntypes.xsd en definieert de structuur voor XML bestanden die het resultaattype void beschrijven, zie ook [Sectie 3.4.3.] .
- **primitivetype.xsd**: importeert types.xsd en definieert de structuur voor XML bestanden die de primitieve types beschrijven.
- **objecttype.xsd**: importeert types.xsd en definieert de structuur voor XML bestanden die de objecttypes beschrijven.

13.2.3. jnome koppelen aan Cocoon 2

Een volgende stap was het aanleggen van de uitvoer van jnome aan de invoer van Cocoon 2 [Sectie 9.2.] . Hierdoor is er geen tussentijdse opslag van de gegenereerde XML bestanden op schijf nodig. Om dit te kunnen realiseren was er een verandering in de interne structuur van jnome nodig. In plaats van onmiddellijk alle bestanden uit te schrijven naar schijf, moeten we deze bestanden ergens in het geheugen bijhouden zodat een willekeurig document eenvoudig opgevraagd kan worden.

Zoals in [Sectie 13.1.2.1.2.] uitgelegd, zijn er bij het opbouwen van de objectstructuur twee fases, namelijk een **acquire**-fase en een **resolve**-fase. Het is het resultaat van deze

resolve-fase dat uitgeschreven wordt. Om ervoor te zorgen dat zowel het uitschrijven als het koppelen aan Cocoon 2 mogelijk blijft, moeten we na afloop van de twee opgenoemde fases de tussenresultaten bijhouden. Het resultaat zit dan wel in het geheugen, maar om ons doel op een eenvoudige manier te realiseren moest het mogelijk zijn om op een andere manier toegang te krijgen tot deze resultaten.

We wilden in feite dat we aan jnome een bestandsnaam met een relatief pad (relatieve bestandsnamen) kunnen geven en dat jnome dan als resultaat het gespecificeerde document teruggeeft. Om te vermijden dat bij elke opvraging steeds een groot deel van de objectstructuur moeten overlopen worden, wilden we een één-op-één koppeling hebben tussen de relatieve bestandsnamen en de te genereren uitvoer.

Het was duidelijk dat we hiervoor een bepaalde structuur nodig hadden die dit kon realiseren, e.g. een **Hashtable**. Ook logisch was dat we de relatieve bestandsnamen als sleutel namen in de Hashtable, maar met welke objecten gingen we deze sleutels associëren? We hadden hiervoor de keuze tussen de resulterende JDOM Document instanties en de objecten die het resultaat zijn van de resolve-fase. In het tweede geval moest het wel mogelijk zijn om de JDOM Document instanties on-the-fly te genereren. We hebben uiteindelijk voor de tweede oplossing gekozen omdat in dit geval er geen duplicatie van informatie in het geheugen nodig was en het ook vrij eenvoudig was om een JDOM Document instantie als resultaat op te vragen van de verschillende **write** methodes.

De aanpassingen aan de **write** methodes opdat deze methodes een JDOM Document instantie zouden teruggeven als resultaat in plaats van zelf voor de uitvoer te zorgen, waren miniem. Hoe het wegschrijven gebeurde hebben we reeds besproken in [Sectie 13.2.1.] en in het voorbeeld [Sectie 13.2.1., Voorbeeldcode 13.6.] . In plaats van dit stukje code konden we gewoon een returnstatement schrijven dat de JDOM Document instantie teruggaf. De signatuur van de methode werd overeenkomstig aangepast.

Wel moesten we er nog voor zorgen dat er de vooropgestelde één-op-één koppeling was tussen de relatieve bestandsnamen en de te genereren uitvoer. Eerst hebben we een extra klasse geïntroduceerd om deze koppelingen op een gemakkelijke manier te kunnen beheren. Vermits de write methodes recursief de gegenereerde objectstructuur volledig overliepen, hebben we een aantal methodes geïntroduceerd die op dezelfde manier de objectstructuur overlopen. Maar in plaats van uitvoer te genereren, zorgen deze methodes voor het leggen van de één-op-één koppelingen tussen de relatieve bestandsnamen en de objecten die de inhoud bevatten. Alhoewel de **write** en de methodes **voor het leggen van de koppelingen** beide de code bevatten voor het recursief overlopen van de objectstructuur, hebben we het gemeenschappelijke gedeelte niet afgezonderd. De enige reden hiervoor is tijdgebrek.

Omdat we weten dat het nogal moeilijk is deze theoretische beschouwingen te volgen, gaan we nu de code geven van de nieuwe klasse en hoe deze klasse gebruikt wordt doorheen jnome. Deze nieuwe klasse ziet er als volgt uit:

Voorbeeldcode 13.7 ResolvedObjectsCache.java

```
package org.jnome.util;

// import necessary java.util classes
import java.util.Enumeration;
import java.util.Hashtable;
// import other jnome classes
import org.jnome.mm.java.types.ResolvedReturnType;
import org.jnome.mm.java.packages.ResolvedPackage;

public class ResolvedObjectsCache {
```

```
/**
 The hashtable used for caching
*/
private Hashtable cache = null;

/**
 Default size of the hashtable we will use for caching
*/
private int defaultSize = 100;

/**
 Variable containing the sole instance of this class
*/
private ResolvedObjectsCache instance = null;

/**
 Private constructor to defeat instancing. This class is implemented according to the
 Singleton-pattern, so only one instance is allowed.
*/
private ResolvedObjectsCache () {
    setCache(new Hashtable(defaultSize));
}

/**
 Returns the one and only instance of ResolvedObjectsCache. This class is implemented
 according to the Singleton pattern.
   @return The instance of ResolvedObjectsCache.
*/
public ResolvedObjectsCache getInstance () {
  if (instance == null) {
        instance = new org.jnome.util.ResolvedObjectsCache();
  }
    return instance;
}

/**
   @param relativeURI The String that contains a relative URI to which the
   ResolvedReturnType will be mapped
   @param rrt A ResolvedReturnType that will be mapped to the given relativeURI
*/
public void addReturnType (String relativeURI, ResolvedReturnType rrt) {
    getCache().put(relativeURI,rrt);
}

/**
   @param relativeURI A String that contains a relativeURI for which we want to obtain the
   ResolvedReturnType that is mapped to it
   @return ResolvedReturnType that is mapped to the relativeURI
*/
public ResolvedReturnType getReturnType (String relativeURI) {
    return (ResolvedReturnType)getCache().get(relativeURI);
}

/**
   @param relativeURI The String that contains a relative URI to which the ResolvedPackage
   will be mapped
   @param rp A ResolvedPackage that will be mapped to the given relativeURI
*/
public void addResolvedPackage (String relativeURI, ResolvedPackage rp) {
    getCache().put(relativeURI,rp);
}

/**
   @param relativeURI A String that contains a relativeURI for which we want to obtain the
   ResolvedPackage that is mapped to it.
   @return ResolvedPackage that is mapped to the relativeURI
*/
public ResolvedPackage getResolvedPackage (String relativeURI) {
    return (ResolvedPackage)getCache().get(relativeURI);
}

/**
   @return Enumeration of the relativeURIs in this cache. Objects in this enumeration will
```

```
     have to be casted to String.
  */
  public Enumeration relativeURIs () {
      return getCache().keys();
  }

  private void setCache (Hashtable t) {
      cache = t;
  }

  private void getCache () {
      return cache;
  }

}
```

We hebben ervoor gekozen om deze klasse volgens het **singleton** patroon te implementeren omdat dit ons toelaat om er op een eenvoudige manier voor te zorgen dat er maar één instantie is van deze klasse. Ook is het vrij makkelijk om deze instantie op te vragen.

De code is vrij eenvoudig gehouden en weerspiegelt niet alle facetten die eigenlijk weerspiegeld zouden moeten worden. Zo hebben de we alleen een methode **addResolvedPackage** terwijl er in feite ook één **UnnamedPackage** is. Omdat er maar één object van het type UnnamedPackage is en UnnamedPackage een subklasse is van **ResolvedPackage** hebben we ervoor gekozen om dit zo te houden. Ook omdat dit onderscheid geweten moet zijn waar de gegevens worden opgevraagd, hebben we dit niet veranderd. Dit moet op dat niveau geweten zijn vermits er, om de uitvoer te genereren in deze gevallen, een andere klasse gebruikt moet worden voor de generatie van deze uitvoer.

Alhoewel het niet verwoordt is in de commentaar maken we gebruik van het feit dat er geen twee dezelfde bestandsnamen met hetzelfde relatieve pad zullen zijn. Dit zou er namelijk op wijzen dat er twee klassen met dezelfde naam in hetzelfde pakket zouden zitten, iets wat niet toegestaan is.

De methode die het leggen van de koppeling in gang zet ziet er als volgt uit:

Voorbeeldcode 13.8

```
...

public void populateCache () {
    ResolvedObjectsCache cache = ResolvedObjectsCache.getInstance();
    String relPath = getRelDir();
  if (relPath.equals("")) {
        relPath = "package" + "." + getWriterFactory().getPrimaryExtension();
  } else {
        relPath = relPath + File.separator + "package" + "." +
        getWriterFactory().getPrimaryExtension();
  }
    cache.addResolvedPackage(relPath,getPackage());
    populateCacheObjectTypes(getRelDir() + File.separator);
    populateCacheSubPackages();
    populateCacheAdditional();
}

...
```

De methodes die we hebben geïntroduceerd , hebben we ingevoerd in de **Writer** klassen, zodat ook dit volgens het **Worker** patroon verloopt en we het metamodel niet vervuilen.

Eerst wordt het correcte relatieve pad bepaald voor het object uit de objectstructuur dat zal toegevoegd worden, in dit geval het verzuimpakket. Vervolgens voegen we dit toe aan de cache, waarna alle objecttypes in dit pakket worden toegevoegd. Dit wordt ook toegepast voor alle subpakketten en dit gebeurt recursief. Tenslotte kan het zijn dat er nog additionele objecten moeten toegevoegd worden en dit gebeurt in een laatste stap in deze methode.

Nu konden we alle koppelingen genereren en vermits we alle sleutels via de methode **relativeURIs** konden opvragen aan de instantie van **ResolvedObjectsCache**, konden we de uitvoer ook op de klassieke manier, dit is naar bestanden op schijf, genereren. De **write** methodes waren immers aangepast zodat ze een JDOM Document instantie teruggaven en het meegeven van het correcte argument was reeds voorzien in deze methodes. Eens we een JDOM Document instantie hebben en we hebben het relatieve pad, dan is het uitschrijven kinderspel.

We willen nog vermelden dat het gebruik van de klasse **ResolvedObjectsCache** niet noodzakelijk is omdat uit het relatieve pad de FQN (Fully Qualified Name) kan afgeleid worden en omgekeerd. Dit is deels veroorzaakt door het feit dat we generator die we voor het Cocoon 2 project hebben geschreven [Appendix E.] zo algemeen mogelijk wilden houden. De applicatie die de XML data aflevert aan deze generator moet daarvoor in staat zijn om een (relatief) pad te kunnen mappen naar een resource. Hierdoor was ons denken reeds in een bepaalde richting gestuurd.

Vermits we nu zo ver stonden, konden we beginnen aan het schrijven van een component voor Cocoon 2 die de juiste bestanden aan jnome kon vragen. Over het schrijven van deze generator hebben we een Engelstalige handleiding geschreven, die terug te vinden is in [Appendix E.] . Dit document is, samen met de code van de klassen, overgemaakt aan de coördinatrice van het Cocoon 2 documentatieproject. Op dit moment is de Cocoon 2 developers gemeenschap nog aan een discussie bezig over hoe het documenteren het beste zou verlopen, maar we hebben er alle vertrouwen in dat dit document deel zal gaan uitmaken van de documentatie bij Cocoon 2. Ook is er een ontwikkelaar van het bedrijf Visitronics die gebruik heeft gemaakt van dit document om zelf een eigen generator te schrijven voor Cocoon 2.

Zoals uit de bespreking in [Appendix E.] blijkt, moeten we in jnome de interface **ServerFunctions** [Appendix sectie E.3.3.1., Voorbeeldcode E.6.] implementeren. We geven hier de geïmplementeerde functies:

Voorbeeldcode 13.9

```
...

public String sayHello () {
    return "<page>JnomeDoc RMI Server v1.0!</page>";
}

...

public String getResource (String file) {
    // Some useful objects
    ResolvedObjectsCache cache = ResolvedObjectsCache.getInstance();
    WriterFactory wf = getWriterFactory();
    Document pack;

  if (!file.endsWith("package.xml")) {
        pack = wf.createResolvedReturnTypeWriter().write(cache.getReturnType(file));
  } else if ( file.equals("package.xml") ) {
        pack =
        wf.createUnnamedPackageWriter((UnnamedPackage)cache.getResolvedPackage(file)).write();
  } else if ( file.endsWith("package.xml") ) {
```

```
        pack =
        wf.createResolvedNamedPackageWriter((ResolvedNamedPackage)cache.getResolvedPackage(file)).write();
   }
     XMLOutputter outputter = new XMLOutputter();
     return outputter.outputString(pack);
}

...
```

De code is volgens ons vrij duidelijk, maar we denken toch dat de testen bij het if-block wat meer uitleg verdienen. Deze testen dienen om te bepalen welke methodes moeten opgeroepen worden om de juiste uitvoer te genereren. De bestandsnaam *package.xml* wordt gebruikt voor de informatie over pakketten, inclusief het verzuimpakket. Indien het opgevraagde bestand niet eindigt op "package.xml", dan weten we dat we met een ResolvedReturnType te maken hebben en kunnen we de juiste writer oproepen. Vermits er voor het verzuimpakket een andere writer is dan voor de andere pakketten, moeten we nog het onderscheid maken tussen het verzuimpakket ("package.xml" op het hoogste niveau) en de andere pakketten ("package.xml" op een lager niveau).

Door in de sitemap van Cocoon 2 onze generator te specificeren [Sectie 9.2.3.2.3.2.] en een aantal ingangen in de sitemap bij te voegen, krijgen we een live-koppeling tussen Cocoon 2 en jnome. Zo een ingang in de sitemap ziet er als volgt uit:

Voorbeeld 13.1.

```
...
 <map:match pattern="**/*.html">
    <map:generate type="mygenerator" src="{1}/{2}.xml">
       <map:parameter name="host" value="supermachien.kotnet.org"/>
       <map:parameter name="port" value="1099"/>
       <map:parameter name="bindname" value="JnomeServer"/>
    </map:generate>
    <map:transform src="stylesheets/class.xsl"/>
    <map:serialize/>
 </map:match>
...
```

Voor uitleg over de betekenis hiervan verwijzen we naar [Sectie 9.2.3.2.3.] . Het **map:parameter** element wordt uitgelegd in [Appendix sectie E.3.3.4.] .

13.2.4. De toekomst

Er zijn nog een aantal zaken waar verder aan gewerkt kan worden, bijvoorbeeld bij het koppelen van jnome aan Cocoon 2. We zullen de mogelijkheden hier kort opsommen:

- Nu gebeurt de communicatie tussen jnome en Cocoon 2 nog via RMI. Men zou er aan kunnen denken de communicatie via SOAP [TRSOAP] te laten verlopen.
- Op dit moment worden de Java bestanden éénmaal ingelezen bij het starten van jnome. Er wordt geen rekening gehouden met bestanden die achteraf nog worden gewijzigd. Het zou mooi zijn moesten de wijzigingen onmiddellijk weerspiegeld worden in het objectmodel en de gegenereerde uitvoer.
- Er zou gewerkt kunnen worden aan het ontwikkelen van stylesheets die het geheel tot een zeer gebruiksvriendelijk geheel kunnen maken. Dit is wat de eindgebruikers zullen zien en de eerste indruk is ook hier heel belangrijk.

14. Vergelijking tussen Java en XML/XSLT

Op het eerste gezicht lijkt het misschien vreemd om een vergelijking te maken tussen een object geörienteerde programmeertaal en een "systeem" om de structuur van de inhoud van een document te beschrijven. In dit hoofdstuk zullen we bespreken op welke manier wij een parallel hebben gezien. Deze beschrijving is niet noodzakelijk volledig, maar dient eerder als een illustratie van het feit dat de gehanteerde concepten vrij gelijkaardig zijn.

14.1. De parallellen

We gaan eerst enkele algemene parallellen vermelden zonder er dieper op in te gaan. De concepten overerving, polymorfisme en dynamische binding zullen we wel uitgebreid bespreken. We trachten deze begrippen in verband te brengen met enkele mechanismen in XML Schema en XSLT. Tot slot zeggen we ook nog iets over variabelen in XSLT.

14.1.1. Algemene overeenkomsten

Wanneer we een Java klasse definiëren, beschrijven we eigenlijk heel algemeen de structuur van een object van die klasse. Dit is ook wat XML Schema doet voor XML documenten. Een Java object is een instantie van een Java klasse definitie, net zoals een XML document beschouwd kan worden als een instantie van een XML Schema.

Ook zou je kunnen zeggen dat XSL-FO de vergelijking kan doorstaan met Swing. Beide zijn bedoeld om een interface naar de buitenwereld te definiëren, alhoewel de werkwijze natuurlijk totaal verschillend is. Daarnaast zou je de templates in XSLT ook nog op een heel ruwe manier kunnen vergelijken met methodes in Java.

14.1.2. XPath en Java

XPath biedt een syntax aan om, op een ëënduidige manier, delen van een document aan te spreken. Het document kunnen we beschouwen als een ruimte waarin XPath op zoek kan gaan naar elementen die voldoen aan de XPath expressie. Indien we in Java gebruik maken van, bijvoorbeeld het **import** statement, dan gaat Java in het **classpath** op zoek naar de gespecificeerde klasse of interface. Het classpath beschrijft de ruimte waarin Java op zoek kan gaan. We hebben in beide gevallen een ruimte waar een mechanisme met een bepaalde syntax op losgelaten wordt om iets te localiseren. XPath definieert ook wel een aantal functies waarvoor we bij Java niet onmiddellijk een tegenhanger vinden en vice versa.

14.1.3. Overerving

14.1.3.1. Overerving en XML Schema

Net zoals het mogelijk is om een overervingsrelatie te hebben tussen verschillende Java klassen, is het ook mogelijk om een hiërarchie van XML Schema documenten te definiëren. Een voorbeeld van zo een schemahiërarchie kan je terugvinden in [Sectie 3.4.] .

In Java heb je eigenlijk twee overervingsmechanismen, met name **extends** en **implements**. Daarnaast kan je ook nog gebruik maken van het **import** mechanisme om gebruik te maken van klassen van buitenaf. We gaan nu uitleggen hoe je deze concepten kan terugvinden in XML Schema.

In XML Schema heb je voor overerving de beschikking over **xsd:include** en **xsd:redefine**. Daarnaast is er ook nog het **xsd:import** element. Een bespreking van deze elementen en een uitleg van de terminologie **ingevoegde** en **invoegende** is te vinden in [Sectie 3.5.2.] . We gaan deze elementen één voor één bespreken en uitleggen met welke concepten in Java ze kunnen vergeleken worden. Ook gaan we het heel eventjes over de **abstract** qualifier hebben.

14.1.3.1.1. xsd:include en xsd:redefine

Wanneer je gebruik maakt van het **xsd:include** element, dan zijn de definities van het ingevoegde schema beschikbaar in het invoegende schema. Dit is juist alsof het **xsd:include** element vervangen wordt door de definities van het ingevoegde document. Er kan in het invoegende document zonder meer gebruik gemaakt worden van de definities in het ingevoegde document. Dit is te vergelijken met het **extends** mechanisme in Java. Door dit te gebruiken beschik je ook over de definities (methodes, variabelen, ...) waarover de superklasse beschikt.

Maar er zijn toch wel enkele verschilpunten. Indien je gebruik maakt van het **xsd:include** element, dan beschik je wel over de definities van het ingevoegde document (de superklasse), maar kan je de definities niet overschrijven. In Java zou dit betekenen dat alle definities waarover de superklasse beschikt als **final** gedeclareerd zijn.

In Java kan je in een subklasse nooit de definities gebruiken die als **private** gedeclareerd zijn in de superklasse (of hoger in de hiërarchie). Afhankelijk van de pakketten waarin deze klassen zitten kan er al dan niet in de subklasse gebruikt gemaakt worden van de **package accessible** definities. Een dergelijk mechanisme hebben we niet teruggevonden in de W3C XML Schema Language. Bij XML Schema is het dus alles of niets.

Een laatste verschilpunt vinden we op het gebied van namespaces. Namespaces in XML zijn te vergelijken met de pakketstructuur in Java. Om een geldige relatie tussen twee schema's te definiëren met behulp van **xsd:include** moet er aan één van de volgende regels voldaan zijn:

- het ingevoegde document mag geen targetNamespace definiëren
- de targetNamespace van het invoegende en het ingevoegde document moet gelijk zijn

Vertaald naar Java zou dit betekenen dat de superklasse tot hetzelfde pakket of het verzuimpakket zou moeten behoren. Gelukkig bestaat deze beperking in Java niet.

Tot slot vermelden we ook het **xsd:redefine** element. Er is eigenlijk maar één verschilpunt tussen dit element en het **xsd:include** element. Indien er gebruik gemaakt wordt van het **xsd:redefine** element is het wel mogelijk om de ingevoegde definities te herdefiniëren. Dit maakt dat er, in vergelijking met **xsd:include** één verschilpunt minder is ten opzichte van Java.

14.1.3.1.2. xsd:import

Het **xsd:import** element is vergelijkbaar met het **import** mechanisme in Java. **xsd:import** zorgt ervoor dat je gebruik kan maken van de definities die geïmporteerd worden. Je kan wel beschikken over deze definities, maar, in tegenstelling tot **xsd:include**, is het niet alsof de definities in het importerende document zijn opgenomen. Met **xsd:import** geef je de namespace aan waarvan je de definities tot je beschikking wilt hebben. Deze namespace **moet** je associëren met een voorvoegsel. Indien je een definitie uit die namespace wil gebruiken, moet je dit aangeven door de naam van de definitie te laten voorafgaan door **<voorvoegsel>:**.

Het aanspreken van de definities kan vergeleken worden met het aanspreken van statische methodes of (niet-private) variabelen in de geïmporteerde klasse. Je kan de methodes uit de geïmporteerde klasse ook niet herdefiniëren, natuurlijk op voorwaarde dat er geen overervingsrelatie bestaat met de geïmporteerde klasse klasse.

14.1.3.1.3. implements

We hebben het niet over het **implements** mechanisme van Java gehad omdat we geen concept gevonden hebben in XML Schema dat daar goed mee overeenstemt.

14.1.3.1.4. abstract

In XML Schema is het ook mogelijk om bepaalde elementen en types **abstract** te declareren [TRXS0-47]. Indien een element of type abstract gedeclareerd is, wil dit zeggen dat in een instantie van een XML document, dat gebruik maakt van dit schema, dit element of type niet gebruikt mag worden. Wel mag er een ander element of type gebruikt worden dat als basiselement of basistype het abstracte type gebruikt. In Java mag je in een programma ook geen gebruik maken van abstracte klassen. Op dit gebied kan dit vergeleken worden.

Indien je een schema invoegt (door bijvoorbeeld gebruik te maken van **xsd:include**) dat alleen maar abstracte elementen bevat, dan is er geen verplichting om elementen te voorzien die deze elementen als basistype gebruiken. Indien je in Java met een niet-abstracte klasse overerft van een abstracte klasse, ben je wel verplicht om de abstracte methodes te implementeren.

14.1.3.2. Overerving en XSLT

Net zoals er in XML Schema een mechanisme bestaat voor overerving, bestaat er een gelijkaardig mechanisme in XLST. In XSLT beschik je over de elementen **xsl:import** en **xsl:include**. Deze elementen hebben een lichtjes andere semantiek dan dat bij XML Schema het geval is en maken het mogelijk meerdere stylesheets samen te voegen.

14.1.3.2.1. xsl:include

Indien je het **xsl:include** gebruikt om te verwijzen naar een andere stylesheet, dan zal een XSLT processor de hoofdstylesheet beschouwen als één grote stylesheet die zowel de definities van de hoofdstylesheet bevat als de definities van de ingevoegde stylesheet. Het zou best kunnen dat er op die manier meer dan één template is die in aanmerking komt voor de verwerking van een element. Indien er na het overlopen van een in de standaard gedefinieerde procedure [TRXSLTCR] meer dan één template overblijft, dan is dit eigenlijk een fout. De meeste XSLT processoren verkiezen echter om deze fout te negeren en de laatst gedefinieerde template te kiezen, wat ook toegestaan is in de XSLT standaard.

Dit gedrag is te vergelijken met het overervingsmechanisme (**extends**) in Java. De definities (methodes, variabelen, ...) van de superklasse zijn beschikbaar in de subklasse, maar de methodes kunnen overschreven worden. Je zou in dit geval kunnen zeggen dat de laatste definitie van de methode gebruikt wordt. Dit is dus te vergelijken met het gedrag van de meeste XSLT processoren, die, in geval van conflict, werken met de laatste definitie van de template. Hierdoor is het wel niet mogelijk een eerder gedefinieerde template te gebruiken. In Java is het wel mogelijk een "eerder gedefinieerde" methode (een methode in de superklasse) aan te spreken door gebruik te maken van **super**.

Net zoals bij XML Schema zijn er in XSLT geen equivalenten aanwezig voor de Java sleutelwoorden **private**, **protected** en **public**. Ook hier geldt de stelregel dat ofwel alle definities worden overgenomen ofwel geen enkele (er wordt niet verwezen naar die stylesheet).

14.1.3.2.2. xsl:import

Het **xsl:import** element heeft eigenlijk hetzelfde gedrag als het **xsl:include** element. Het **verschilpunt** is echter dat de templates in de importerende stylesheet voorrang hebben op de templates in de geïmporteerde stylesheet. Er kunnen natuurlijk meerdere **xsl:import** elementen voorkomen. Hoe de voorrang bepaald wordt tussen deze verschillende templates wordt uitgelegd in [Sectie 5.3.1.14.3.] .

Net zoals hierboven, bij **xsl:include**, kan de vergelijking gemaakt worden met het overervingsmechanisme in Java, met de daar reeds aangehaalde nuances. Omdat echter de templates van de importerende stylesheet voorrang hebben op de templates van de geïmporteerde stylesheet zal het niet voorkomen dat er meerdere templates voor één element in de importerende stylesheet overblijven na de conflictresolutieprocedure [TRXSLTCR] (tenzij je in de importerende stylesheet natuurlijk meerdere templates definieert voor dat element). Er zal dus altijd gewerkt worden met de templates in de hoofdstylesheet (de importerende stylesheet), tenzij daar een template voor een element niet gedefinieerd is. Dan zal er in de geïmporteerde stylesheets gezocht worden in een bepaalde volgorde, zoals we juist hebben aangegeven.

Het is echter wel mogelijk om een beroep te doen op de templates in de geïmporteerde stylesheets, zelfs indien er in de importerende stylesheet een geschikte template voorhanden is. Door gebruik te maken van **xsl:apply-imports** maak je aan de XSLT processor duidelijk dat er enkel in de geïmporteerde stylesheets gezocht mag worden naar geschikte templates. Het zoeken gebeurt in de volgorde zoals uitgelegd in [Sectie 5.3.1.14.3.] . Dit is dus te vergelijken met het gebruik van **super** in Java.

14.1.4. Polymorfisme en dynamische binding

14.1.4.1. De concepten in de context van XML Schema

We gaan de concepten polymorfisme en dynamische binding hier niet uitleggen. We gaan er van uit dat iedereen deze begrippen beheerst.

In de bespreking van XML Schema hebben we eigenlijk al polymorfisme en dynamische binding behandeld [Sectie 3.5.8.] . In die sectie hebben we gezien dat het mogelijk is om voor de inhoud van een element een keuze te maken uit een aantal types, waar de inhoud van dat element dan aan moet voldoen. Een element wordt eigenlijk met een bepaald type geassocieerd, in dit geval een **union** type. Het is dit **union** type dat de keuzes voor de inhoud beschrijft. We kunnen hier de vergelijking maken met polymorfisme, maar met enige nuances. Het **union** type is hier de superklasse en de types waar het **union** type naar verwijst kunnen beschouwd worden als de subklassen. We kunnen dus een element (variabele) aanmaken waarvan we alleen weten dat het geassocieerd is met een **union** type (abstracte superklasse), maar het eigenlijke type van de inhoud van het element, is één van de types (subklasse) die beschreven is in het **union** type.

Zoals in Java de dynamische binding gebeurt tijdens de uitvoering van het programma, zal dit bij een XML document gebeuren wanneer het gevalideerd wordt aan de hand van een XML

Schema [Sectie 3.3.2.] . Zoals in [Sectie 3.5.8.] besproken, zal een validator het element (en de inhoud) proberen te valideren aan de hand van de mogelijke types die zijn opgegeven. Maar er is hier wel een **groot verschil** met Java. In Java wordt de definitie gebruikt van de meest specifieke klasse (**best match**) waaraan de variabele voldoet, als er een methode op wordt uitgevoerd. Bij het valideren echter worden de opgegeven types gewoon in volgorde afgegaan en het element wordt gevalideerd met het eerste overeenkomende type (**first match**). De evaluatievolgorde kan bij het valideren wel overschreven worden door gebruik te maken van het **xsd:type** attribuut, zoals in [Sectie 3.5.8.] uitgelegd wordt. Er is dus wel dynamische binding aanwezig bij het valideren, maar er is wel enig verschil, zoals wij hier aangehaald hebben.

14.1.4.2. De concepten in de context van XSL(T)

Het is mogelijk dat er tussen de gedefinieerde templates een aantal templates zitten die kandidaat zijn om een bepaald element te matchen. Je kan dit bekijken alsof je een abstracte superklasse hebt die de algemene kenmerken voor deze templates definieert. Daarnaast heb je de gedefinieerde templates die in aanmerking komen. Deze templates kunnen beschouwd worden als concrete klassen die overerven van de abstracte klasse. Het is tussen de concrete klassen (templates) dat een keuze gemaakt moet worden indien we een element tegenkomen waarvoor deze templates in aanmerking komen. Deze uitleg is dus in feite niet meer dan een beschrijving van polymorfisme in het kader van XSL(T).

Ook dynamische binding is hier aanwezig. Zoals in uitgelegd wordt, wordt steeds geopteerd voor de meest specifieke template (**best match**), indien er meerdere templates mogelijk zijn voor dat element. Dit is juist zoals in Java, maar er kan ook hier weer een klein beetje van afgeweken worden. Het is namelijk mogelijk om een stylesheet op te delen in meerdere bestanden. In de hoofdstylesheet kan dan met behulp van **xsl:import** en **xsl:include** [Sectie 5.3.1.14.] verwezen worden naar de andere stylesheets. In de hoofdstylesheet kan je templates herdefiniëren en het is de laatst gedefinieerde template die wordt uitgevoerd, indien er twee kandidaten zijn voor de **best match**. Indien je echter het **xsl:import** element gebruikt, kan je gebruik maken van het **xsl:apply-imports** element om aan te geven dat er alleen gekeken moet worden naar de templates gedefinieerd in de bestanden waarnaar verwezen wordt door de verschillende **xsl:import** elementen. Op die manier kan er toch lichtjes afgeweken worden van het **best match** principe.

14.1.5. XSLT variabelen en Java

Het Java equivalent dat het **xsl:variable** element, gedefinieerd binnen een XSLT template, het best benadert, is een finale lokale variabeledeclaratie die geïnitialiseerd wordt. De reden hiervoor is dat, eens een variabele gedeclareerd in een XSLT template, we de waarde van de variabele niet meer kunnen veranderen. We kunnen wel een variabele met dezelfde naam definiëren op een bepaalde plaats in de XSLT template. Binnen de scope van deze tweede variabele [Sectie 5.3.1.11.1.] zal de waarde van deze tweede variabele gebruikt worden en binnen de scope van de eerste variabele, maar buiten de scope van de tweede variabele, zal met de waarde van de eerste variable gewerkt worden. Dit valt te vergelijken met de scope van variabelen in Java.

We geven nu een voorbeeldje van de verschillende variabeledeclaraties:

Voorbeeld 14.1. Variabele declaratie binnen een template

```
<xsl:variable name="x"> waarde </xsl:variable>
```

Deze variabeledeclaratie heeft een gelijkaardige semantiek als onderstaand voorbeeld:

Voorbeeldcode 14.1 Java declaratie

```
final OBJECT x = "value";
```

XSLT heeft geen equivalent voor de Java toekennings operator, zoals die hier onder gegeven wordt.

Voorbeeldcode 14.2 Java toekenningsoperator

```
x = "value";
```

14.2. Besluit

Alhoewel het op het eerste zicht lijkt alsof er totaal geen overeenkomst is tussen Java en XML/XSLT/XML Schema, hopen we dat we deze scepsis hebben kunnen weerleggen in dit hoofdstuk. Het is frappant hoeveel gelijkenis er bestaat tussen deze twee, maar, zoals we hebben beschreven, zijn enkele nuances bij de gelijkenissen toch op hun plaats.

15. Toekomst

In deze thesis hebben we een uitgebreide studie gemaakt van de mogelijkheden van XML en XSLT. We hebben als onderdeel van deze studie ook een aantal praktisch gerichte zaken gerealiseerd, zoals het onwikkelen van een XML formaat voor onze thesistekst en het schrijven van de bijhorende stylesheets en het aanbrengen van enkele wijzigingen en uitbreidingen aan jnomedoc.

Het doel van dit hoofdstuk is het beschrijven van een aantal mogelijke onderzoeksonderwerpen voor een vervolgthesis. We zullen kort bespreken wat wij verstaan onder deze verschillende voorstellen.

15.1. jnome en jnomedoc

jnome is een project in volle ontwikkeling. Er zijn nog veel uitbreidingen mogelijk aan jnome en jnomedoc. Het metamodel en de XML Schema documenten kunnen onder de loep genomen worden en eventueel verfijnd worden.

Voor jnomedoc zou de koppeling met Cocoon 2 herbekeken kunnen worden en misschien met SOAP [TRSOAP] kunnen gebeuren. Ook zou er een cache-mechanisme geïmplementeerd kunnen worden dat nagaat wanneer de Java bronbestanden gewijzigd zijn en overeenkomstig de objectstructuur in het geheugen aanpast.

Ook kunnen de XSL stylesheets die voor jnomedoc geschrijven zijn, herwerkt worden om een betere lay-out en bruikbaarheid van de gegenereerde documentatie te bekomen. De stylesheets kunnen aangepast worden aan het type gebruiker: de programmeur van dienst, een programmeur die gebruik maakt van de klassen door de eerste programmeur gemaakt, een geïnteresseerde persoon, ...

Het is ook mogelijk om verschillende uitvoerformaten te voorzien door te werken met verschillende transformaties. Zo zou er één groot PDF-bestand gegenereerd kunnen worden in tegenstelling tot de verschillende HTML-pagina's.

15.2. XMLdoc

In Java is het mogelijk om commentaarblokken te plaatsen bij de verschillende definities (klassedefinities, variabeledefinities, ...) in een klasse. Op basis van deze commentaarblokken en de signatuur van de definities is het mogelijk om met tools zoals javadoc en jnomedoc documentatie te genereren van de code.

Vermits XML Schema vergeleken kan worden met een klassedefinitie [Hoofdstuk 14.] , kan je de vergelijking volledig doortrekken en ook commentaarblokken voorzien om de definities van types, elementen, enzovoort duidelijk te maken. Er zijn al voorzieningen om commentaar op te nemen in XML Schema documenten [Sectie 3.3.3.1.] . Er is echter nog geen woordenschat voorhanden, die automatisch door een programma verwerkt kan worden, en die dient om het doel van de verschillende definities uit te leggen.

Het zou daarom interessant zijn om hiervoor een woordenschat te ontwikkelen. Op die manier moet het mogelijk zijn dat iedereen het schema kan begrijpen en kan inzien wat er wel en niet toegestaan is in een XML document, zonder de finesses van de W3C XML Schema Language eerst te moeten leren.

15.3. Java en XML

Zoals in [Sectie 4.2.6.] beschreven, hebben we een eigen XML formaat ontworpen om Java te kunnen beschrijven met behulp van XML. Zoals daar vermeld is, zijn niet alle gemaakte keuzes exhaustief of semantisch volledig correct. Het XML formaat voor Java kan op deze plaatsen gecorrigeerd worden en aangevuld, zodat we uiteindelijk een XML beschrijving hebben van Java.

We denken hierbij aan tools, zoals jnomedoc, die een aantal klassen inlezen en deze omzetten naar hun XML beschrijving. Dit kan dan resulteren in een **pretty printer** voor Java. Zoals we met jnome bewezen hebben [Sectie 13.2.3.] , is het mogelijk om dit te realiseren zonder XML bestanden op schijf te bewaren. Indien met behulp van de tool gedetecteerd kan worden wanneer een Java bronbestand veranderd is, dan kan de XML beschrijving van dat document aangepast worden, waardoor de wijzigingen quasi onmiddellijk zichtbaar zijn in de XML beschrijving van dat bestand. Iedereen kan dan ook de broncode bekijken zoals hij of zij dat wil. Aan de XML beschrijving moet niks veranderen, iedereen die een andere uitvoer wil, moet gewoon zijn eigen stylesheets schrijven.

Omdat dit toch een grondige studie van Java impliceert, zou er misschien ook gewerkt kunnen worden aan het verder onderzoeken van de overeenkomsten tussen Java en XML/XSLT [Hoofdstuk 14.] .

15.4. Koppeling XSLT en XML Schema

Zoals uit de bespreking van XSLT [Hoofdstuk 5.] blijkt, wordt er voor het uitvoeren van templates naar de namen van de elementen gekeken. Het is niet ondenkbaar dat je een XML Schema hebt voor je document en dat je wil dat het matchen gebeurt op basis van de types die deze elementen in het XML Schema hebben in plaats van gewoon op basis van de namen.

Dit kan bijvoorbeeld het geval zijn indien je veel elementen hebt met verschillende namen, maar die toch van hetzelfde XML Schema type zijn. In dit geval wens je alle elementen van hetzelfde type op dezelfde manier te behandelen. Ofwel moet je veel templates schrijven die allemaal hetzelfde doen ofwel krijg je een heel groot match patroon. Je stylesheet zou echter duidelijker worden indien je kon matchen op type. Dit vereist natuurlijk dat er met het XML document een XML Schema geassocieerd is.

We vinden een verwante gedachtengang bij de ontwikkelaars van Xalan, een XSLT processor. Zoals in [Sectie 12.2.5.] vermeld, wordt ook bij het gebruik van SAX nog een boomstructuur opgebouwd om de transformaties te kunnen toepassen. In het design [XALANDES] van Xalan onthullen de ontwikkelaars hun plannen om ook beroep te doen op de met de XML documenten geassocieerde schema's (indien aanwezig). Op die manier wil men de boom die in het geheugen opgebouwd wordt, zo klein mogelijk houden. Het lijkt ons dus een interessant idee om deze koppeling te bestuderen en te implementeren.

15.5. XSL-FO

Uit het hoofdstuk over XSL-FO [Hoofdstuk 7.] blijkt dat deze taal nogal moeilijk onder de knie te krijgen is. Als voornaamste redenen kunnen we de uitgebreide woordenschat en de, soms verwarrende, terminologie aanhalen.

Om deze drempel te verlagen zou het mooi zijn moesten er grafische editors ontwikkeld worden waarin je kan aangeven welke vormgeving je aan je pagina wilt geven, zonder dat je

een specialist moet zijn op het gebied van XSL-FO. Uitgaande van een XML Schema zou het in zo een applicatie mogelijk moeten zijn om aan te geven hoe bepaalde elementen op het blad moeten verschijnen (in een kader, in het vet, enzovoort). Deze applicatie is dan verantwoordelijk voor het genereren van de XSLT stylesheets die zorgen voor de omzetting van een XML document (een instantie van het schema) naar het overeenkomstige FO document. Zo'n applicatie zou voor XML/XSLT/XSL-FO kunnen zijn wat Dreamweaver en Frontpage zijn voor HTML.

15.6. Besluit

Zoals duidelijk blijkt uit de bespreking van de verschillende toekomstmogelijkheden kan je met XML/XSLT alle kanten uit. Een ware uitdaging voor de menselijke fantasie dus.

16. Besluit

16.1. Bestudeerde onderwerpen

Het doel van deze thesis was het uitvoeren van een verkennende studie over XML en XSL en daarop enkele toepassingen maken. Dit onderzoeksgebied is relatief jong en bestaat uit een zeer brede waaier aan onderwerpen. Bijna wekelijks duiken er nieuwe toepassingen op. We hebben hier dus te maken met een zich snel uitbreidend onderzoeks- en toepassingsgebied. Dat de ontwikkelingen zeer snel gaan, valt ook af te lezen uit onze bibliografie: er staan maar een paar boeken tussen de lange lijst met bibliografie-items, boeken die dan ook nog eens online ter beschikking zijn. Deze grote verscheidenheid aan onderwerpen heeft ons echter ook gedwongen keuzes te maken, waardoor sommige onderwerpen uit de boot vielen. Er zijn verschillende onderwerpen, zoals XLink en XPointer, die we ook nog wilden bestuderen, maar wegens tijdsgebrek is dit niet gebeurd.

Wij zijn in onze verkennende studie heel breed gegaan. Zo breed als binnen ons beperkt tijdsbestek mogelijk was. Het positieve hiervan is dat we de kans hebben gehad om vele onderwerpen te bestuderen en van alles een beetje te proeven. Maar de keerzijde van de medaille is dat we daardoor sommige onderwerpen niet diep genoeg hebben kunnen uitspitten. Een bijkomend nadeel van deze benadering is dat dit ons een heel dikke thesis opgeleverd heeft. Wij hebben ons best gedaan om de moeilijke begrippen zo goed mogelijk uit te leggen, opdat de lezer na het lezen van onze thesis zelf met XML en XSL zou kunnen werken. Dit bleek vaak niet evident want de technische aanbevelingen waarop wij ons baseerden zijn vaak nogal vaag en bronnen die wij op het net raadpleegden waren soms niet meer nauwkeurig genoeg of spraken elkaar tegen. Het is meer dan eens gebeurd dat we empirische vaststellingen hebben moeten doen omdat de bewoordingen in de technische aanbeveling volgens ons niet éénduidig te interpreteren vielen.

Wij hebben als onderdeel van onze studie ook een bijdrage geleverd aan het Apache Cocoon open source project. Wij hebben een handleiding voor de Cocoon 2 sitemap geschreven, omdat zo'n handleiding op het moment dat wij met Cocoon 2 begonnen te experimenteren totaal onbestaande was. Dit betekende natuurlijk een extra hinderpaal: we hebben het gebruik van de sitemap moeten leren met vallen en opstaan. Verder hebben we ook een howto geschreven die handelt over het schrijven van een generator, één van de componenten waar Cocoon 2 gebruik van maakt. Deze howto, zo hebben we ondertussen vernomen, zal deel gaan uitmaken van de documentatie van het Cocoon 2 project. We kunnen op dit moment nog geen precieze locatie geven waar dit document zal terug te vinden zijn, omdat de ontwikkelaars zich nog eerst over dit document moeten buigen. Hiermee zijn we ook de eerste auteurs van het Cocoon 2 documentatie project.

Juist voor het ter perse gaan bereikte ons het bericht dat de howto die wij hadden geschreven opgenomen is in de documentatie van het Cocoon 2 project. Deze documentatie is voortaan ook terug te vinden op *http://cvs.apache.org/viewcvs.cgi/xml-cocoon2/src/documentation/xdocs/tutorial/*.

Verder hebben we aanpassingen aangebracht in jnomedoc, de applicatie die een onderveel vormt van het jnome project, een open source project aan de Faculteit Toegepaste Wetenschapen, departement Computerwetenschappen, van de Katholieke Universiteit Leuven.

Ons grootste project was onze eigen thesis. De thesistekst die je gelezen hebt, is volledig in XML geschreven (met uitzondering van het titelblad). Met behulp van XSLT, XSL-FO,

Cocoon 2 en FOP is het mogelijk om door de HTML versie van onze thesis te browsen als door een gewone website of om er een PDF-bestand van te genereren. Aanpassingen in onze tekst worden onmiddellijk weerspiegeld in beide formaten zonder dat we er bijzondere inspanningen voor moeten doen. Nog één van de voordelen van het gebruik van XML.

16.1.1. XML en XML Schema

XML Schema levert een manier om een XML document te beschrijven met behulp van een XML syntax. De syntax op zich is niet al te moeilijk, maar de uitgebreidheid kan soms nogal overweldigend zijn. XML Schema heeft nog enkele beperkingen, waaronder de onmogelijkheid om entiteiten te definiëren. Dit verplicht je om nog altijd DTD's te gebruiken. Om XML Schema overal te verspreiden, zullen er echter programma's moeten komen die de hele standaard ondersteunen en die kunnen instaan voor de validatie van de documenten. Deze programma's geraken, zij het met mondjesmaat, langzaamaan beschikbaar. Om de omschakeling van DTD's naar XML Schema te kunnen maken is het ook belangrijk dat er programma's zijn die deze omzetting automatisch kunnen doen. Deze programma's zijn er, en alhoewel ze nog geen perfecte XML Schema's afleveren, leveren ze toch een goede basis om mee te beginnen. XML Schema is een veelbelovende standaard, maar er moet wel een uitgebreide ondersteuning voor beschikbaar zijn vooraleer vele bedrijven de overstap zullen wagen van DTD's naar XML Schema.

16.1.2. XML en XSLT

XSLT beschikt over een zeer groot arsenaal aan verschillende syntaxconstructies. Op het eerste zicht kan dit wel een beetje overdonderend lijken, maar uit onze eigen ervaring blijkt dat het niet zoveel tijd kost om een rudimentaire stylesheet te leren schrijven. Als je XML document goed gestructureerd is, kom je al heel ver met een aantal elementaire templates en een aantal eenvoudige XPath expressies om wat eenvoudige uitvoer te genereren. Het verfijnen van de templates kost echter meer werk, vooral als je kruisverwijzingen in je documenten wilt verwerken. In dat geval moet je met vrij ingewikkelde XPath expressies werken.

Wat de toekomst van XML en XSLT betreft, denken wij dat er voor standaardtoepassingen gedetailleerde stylesheets geschreven zullen worden. Als je deze standaardstylesheets wilt aanpassen, door bijvoorbeeld een paar templates te overschrijven, kan je dit doen door jouw specifieke stylesheet samen te voegen met de standaardstylesheet door gebruik te maken van **xsl:import** en **xsl:include**. Deze standaardstylesheets zullen natuurlijk gedocumenteerd moeten worden. Op dat vlak is nog niet zoveel onderzoek verricht, maar wij vermoeden dat dit in de toekomst niet kan uitblijven.

16.1.3. XML en XSL-FO

XSL-FO heeft ontzettend veel mogelijkheden op het gebied van vormgeving. Deze lange waslijst van mogelijkheden, zorgt echter, veel meer nog dan dit voor XSLT en XML Schema het geval is, voor een vrij hoge drempel. Je moet heel veel objecten en eigenschappen kennen om de lay-out van je tekst exact te krijgen zoals jij het wilt. XSL-FO is, zoals we zelf ondervonden hebben, ook niet altijd makkelijk in het gebruik. De tools die de omzetting van XSL-FO document naar een ander medium doen, bieden ofwel een zeer beperkte ondersteuning met de nodige bugs ofwel een vrij volledige ondersteuning, maar daar betaal je dan ook voor. Nog een nadeel is de compatibiliteit met CSS die wordt nagestreeft.

16.1.4. Cocoon 2

Cocoon 2 is een geweldig framework dat een aantal mogelijkheden biedt waarnaar wij op zoek waren toen we op de beperkingen van Cocoon 1 stuitten. De sitemap is zo een krachtig concept voor het beheren van websites dat we ons afvragen waarom deze manier van beheren nog geen algemene ingang heeft gevonden. Cocoon 2 is een prachtig hulpmiddel geweest in onze thesis en we zijn blij dat we ons steentje hebben kunnen bijdragen aan dit project. Toch zijn er nog enkele kleine puntjes van kritiek, waar gelukkig door de ontwikkelaars aandacht aan geschonken wordt. Bij de eerste versies van Cocoon 2 was het zo dat, indien je een element vergat af te sluiten, je gewoon de melding kreeg dat een element met die naam niet was afgesloten. Je moest dan zelf uitvissen waar dit was. In de meest recente versies krijg je ook een lijnnummer waar de fout vermoedelijk zit. Het lijnnummer is niet altijd volledig correct, maar het zit er nooit ver naast. Dit is al een hele hulp in het opsporen van fouten.

Wegens de weinige documentatie die er in de beginmaanden van onze thesis voorhanden was, was het niet evident om met Cocoon 2 te beginnen. Door bijdragen van de gebruikers (waaronder enkele bijdragen van onszelf) komt er gelukkig geleidelijk meer documentatie ter beschikking. Deze documentatie zorgt voor een lagere instapdrempel. Om het gebrek aan documentatie op te lossen is men op dit moment volop bezig met het opstarten van een Cocoon 2 documentatie project. Dit zou als resultaat moeten hebben dat alle losse bijdragen gegroepeerd worden en elk aspect van Cocoon 2 uiteindelijk gedocumenteerd wordt.

16.1.5. jnome en jnomedoc

Wat het jnome project betreft, hebben we de gemaakte DTD's geconverteerd naar XML Schema documenten. De gemeenschappelijke delen van deze documenten hebben we in aparte bestanden geplaatst en zo hebben we een overervingshiërarchie gedefinieerd waardoor één definitie ook maar in één bestand voorkwam.

We hebben ook geleerd hoe jnomedoc werkte. jnomedoc is een goede tool die we op het gebied van de uitvoer nog wat hebben kunnen verbeteren. Door de uitvoer te genereren met behulp van JDOM in plaats van met een PrintWriter, moet er alleen nog gecontroleerd worden of de uitvoer op de juiste plaats wordt toegevoegd en niet meer of bijvoorbeel een element wel correct wordt afgesloten of niet. Het correct opbouwen van XML documenten is iets waar JDOM voor zorgt en waar de programmeur zich geen zorgen over hoeft te maken.

Daarnaast hebben we ook nog een koppeling gerealiseerd tussen jnomedoc en Cocoon 2. Het heeft heel wat doorzettingsvermogen gevraagd om Cocoon 2 te doorgronden. Het voordeel dat deze koppeling biedt is het feit dat de gegenereerde XML bestanden niet meer op schijf moeten opgeslagen worden. Hierdoor wordt de deur geopend voor een breder scala aan toepassingen. De jnomedoc server kan in principe ook door andere programma's aangesproken worden, dit hoeft niet noodzakelijk door Cocoon 2 te gebeuren.

16.1.6. XML en Java

We hebben in onze thesis drie manieren besproken om Java en XML te combineren. Er bestaan Java API's om XML documenten op te bouwen en te manipuleren. We hebben een beschrijving van Java code in XML formaat ontwikkeld. Tenslotte we hebben ook een vergelijking gemaakt tussen XML en Java structuren.

Van de Java API's die we gezien hebben onthouden we vooral JDOM en SAX. JDOM levert, in tegenstelling tot DOM, een eenvoudige syntax om XML documenten aan te maken, in te lezen, te manipuleren en uit te schrijven. Het volstaat om een eenvoudig artikel over JDOM te lezen om met JDOM te kunnen werken, wat niet gezegd kan worden van DOM. We hebben

jnomedoc aangepast door de gegenereerde uitvoer via JDOM te laten verlopen. Daarnaast is SAX belangrijk omdat SAX geldt als een de facto standaard en het lijkt te halen van DOM. Wij hebben SAX voornamelijk bestudeerd in het kader van Cocoon 2.

Om fragmenten Java code voor te stellen hebben we ook een eigen XML formaat ontwikkeld waarin Java code kan beschreven worden. Dit is vrij aardig gelukt, enkele semantische foutjes niet te na gesproken. Deze modellering is verre van volledig maar volstaat in het kader van onze thesis.

Tenslotte hebben we ook nog onderzocht of er overeenkomsten bestonden tussen Java enerzijds en XML, XSLT, XML Schema en XSL-FO anderzijds. Vooral op het gebied van overerving, polymorfisme en dynamische binding hebben we deze gelijkenissen uitgediept.

Tot slot willen we vermelden dat we genoten hebben van dit zeer boeiende thesisonderwerp. We hebben op één jaar tijd bijzonder veel nieuwe toepassingen en mogelijkheden van XML, XSLT en aanverwante onderwerpen ontdekt. In het onderzochte gebied zijn we vrij breed gegaan en op sommige aspecten van onze thesis zijn we wat dieper ingegaan dan op andere. Het resultaat is een serieus uit de kluiten gewassen thesis. We hebben geprobeerd om duidelijk, maar toch beknopt te zijn in onze teksten. Dat de tekst minder beknopt is dan we wel zouden willen, is voor een deel te wijten aan de vaagheid van de technische aanbevelingen. Hierdoor waren we gedwongen alles uitgebreid te omschrijven om zo éénduidig mogelijk te zijn. Wij hopen dat de lezer na het lezen van onze thesis een goed inzicht in deze materie heeft gekregen en zelf in staat is te werken met XML en XSLT.

Appendix A. Documentformaten

Het doel van deze appendix is het beschrijven en vergelijken van een aantal verschillende documentformaten. Wij hebben deze formaten onderzocht voordat we met het schrijven van onze thesistekst zijn begonnen, om een beter zicht te krijgen op de verschillende mogelijkheden. We bespreken achtereenvolgens, DocBook, Tex en Latex en ons eigen XML formaat, zetten de afwegingen die wij gemaakt hebben op een rijtje en leggen uit waarom we uiteindelijk geopteerd hebben voor een eigen XML formaat.

A.1. Een bespreking van DocBook

Voor deze bespreking baseren wij ons op [NWLMDB] en [JBDW].

A.1.1. Wat is DocBook?

DocBook is een markup taal (cfr. HTML) die gedefinieerd wordt door een SGML [RCSGML] of XML DTD (Document Type Definition) [RDW3S]. Oorspronkelijk bestond enkel een SGML versie van DocBook, maar sinds februari 2001 is er ook een officiële XML versie beschikbaar. DocBook bestaat uit een verzameling tags die de structuur van een document beschrijven. Op dit vlak is DocBook vergelijkbaar met HTML, maar DocBook gaat nog een stap verder dan HTML, het is mogelijk vertrekkende vanuit een DocBook document verschillende formaten van datzelfde document te genereren. Een document in het DocBook formaat kan omgezet worden naar andere formaten zoals HTML, PostScript, PDF, RTF, DVI of gewone ASCII tekst.

DocBook is op dit moment (2002) ongeveer tien jaar oud. Het werd oorspronkelijk gecreëerd om gemakkelijker UNIX documentatie uit te wisselen. Ondertussen heeft DocBook vier grote versies doorlopen. Momenteel wordt het beheerd door OASIS [OASIS]. Voor uitgebreide documentatie i.v.m. DocBook verwijzen we graag naar [DB].

A.1.2. Hoe DocBook gebruiken?

DocBook en alle tools die gebruikt worden om met DocBook te werken zijn gratis beschikbaar onder Open Source licenties [OPENSOURCE]. DocBook documenten zijn in plain text formaat en kunnen dus met eender welke text editor gemaakt worden. Let er dan wel op het document als plain text te bewaren, anders zal het document niet op de juiste wijze geparst worden.

De eerste lijn van een DocBook document is de document declaratie. Zonder de document declaratie is het document niet geldig ("valid"). Deze document declaratie geeft aan welke versie van DocBook er gebruikt wordt en om welk type document het gaat.

Een article in DocBook versie 4.1 (SGML) krijgt bijvoorbeeld als eerste lijn:

Voorbeeld A.1.

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V4.1//EN">
```

Voor een article in DocBook versie 4.1.2 (XML) ziet de eerste lijn er als volgt uit:

Voorbeeld A.2.

```
<?xml version="1.0"?> <!DOCTYPE article PUBLIC "-//OASIS//DTD XML
V4.1.2//EN">
```

Opgelet: De XML declaratie niet vergeten!

Hieronder volgt een kort voorbeeld dat een goed idee geeft hoe een DocBook document er kan uitzien. Bemerk hoe elk semantisch onderdeel van het artikel door tags ingesloten wordt.

Voorbeeld A.3.

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V4.1//EN">
   <article>
      <articleinfo>
         <title> Voorbeeld van een DocBook Template </title>
         <author>
            <firstname> de voornaam van de auteur van het artikel
            </firstname>
            <surname> de familienaam van de auteur </surname>
         </author>
      </articleinfo>
      <sect1>
         <title> de title van de eerste sectie 1 van het artikel </title>
         <para> Een paragraaf </para>
         <sect2 label="1">
            <title> Titel van de eerste sectie 2 </title>
            <para> Nog een paragraaf! </para>
         </sect2>
         <sect2 label="2">
            <title> Titel van een tweede sectie 2 </title>
            <para> Nog een
               <emphasis> paragraaf </emphasis>
            ! </para>
         </sect2>
      </sect1>
   </article>
```

Het **label** attribuut bij de <sect> tag geeft een nummering aan de verschillende secties. Bemerk dat in het DocBook document enkel de structuur van het artikel wordt weergegeven. Hoe de layout hiervan er precies zal uitzien, wordt later beslist. Zo kan de <emphasis> tag bijvoorbeeld in bold of in italic worden weergegeven. De verzameling tags in DocBook is bijzonder groot. Voor een gedetailleerde lijst verwijzen we naar [DB].

Er bestaan drie verschillende doctypes, met name **book**, **chapter** en **article**. Wat book en chapter wil zeggen, spreekt voor zichzelf. Article wordt gebruikt als het gaat om een kleinere hoeveelheid tekst.

A.1.3. Ervaringen met DocBook

We zullen nu wat dieper ingaan op de eigenschappen van DocBook waar wij mee in aanraking zijn gekomen. Dit overzicht is verre van volledig. we hebben ons beperkt tot het maken van twee artikels, één in SGML en één in XML. De verschillen tussen deze beide versies beperken zich tot de eerste regel, met name, de document declaratie.

Een eerste eigenaardigheid is volgens ons dat er nog steeds <para> tags tussen <listitem> tags genest moeten worden. Dit is waarschijnlijk een gevolg van het feit dat SGML al wat ouder is

en in die tijd er nog niet met overerving gewerkt werd. Anders zou men bijvoorbeeld <listitem> kunnen laten erven van <para> om zo de eigenschappen van een paragraaf door te geven aan een listitem. Bij nader inzien is deze keuze te verklaren vanuit het standpunt dat een <listitem> misschien wel meerdere paragrafen kan bevatten. Wij vinden echter dat dit indruist tegen wat je intuïtief zou verwachten van een listitem.

Als er speciale tekens zoals < of > in een tekst gebruikt moeten worden, kan dit alleen maar door < en > te gebruiken. Dit is vervelend, maar niet te vermijden in SGML en XML gebaseerde dialecten, omdat deze tekens in SGML en XML zo'n speciale betekenis hebben.

De tools die wij uitgetest hebben om ons DocBook document om te zetten naar HTML, PostScript en PDF zijn de volgende: **docbook2html** , **docbook2ps** en **docbook2pdf** . De meeste linuxdistributies hebben deze programma's standaard beschikbaar. Deze tools zijn zeer eenvoudig in het gebruik: gewoon de naam van de tool en het om te zetten DocBook document intypen in de commandline en het juiste formaat voor het document wordt gegenereerd. Helaas zijn de foutmeldingen soms niet al te duidelijk, een minpunt dat wel meer programma's ten toon spreiden. Het is wel heel belangrijk dat je de juiste versie van de DTD's en de tools hebt. Dit heeft ons verscheidene keren voor problemen gesteld. Het feit dat er tegelijkertijd zoveel verschillende versies in gebruik zijn, pleit ook niet meteen in het voordeel van DocBook. Het is soms echt zoeken. Het probleem is ook dat ze op *http://www.docbook.org* de verschillende versies van DocBook een beetje door elkaar gebruiken in hun voorbeelden, waardoor het geheel soms nogal onoverzichtelijk wordt.

Naar ons inzicht is de scheiding tussen inhoud en vorm ook niet zo absoluut als wij wel zouden willen. In tabellen (op zich eigenlijk ook al layout) bijvoorbeeld is het mogelijk om aan te geven hoe je de inhoud wilt gealigneerd zien. Dit zijn echte lay-out instructies, maar het is onze bedoeling om enkel op de structuur van het document te focussen.

Uiteindelijk zijn we er toch in geslaagd zowel een werkende versie van een .SGML als een .XML DocBook document ineen te knutselen. Op *http://www.docbook.org* wordt als document declaratie voor XML de volgende regel gebruikt:

Voorbeeld A.4.

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
"http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd">
```

Dit bleek bij ons echter niet te werken. Pas nadat alle docbook-bibliotheken geupdated waren en we de url hadden weggelaten, werkten de tools zonder foutmeldingen.

Omdat onze ervaringen met DocBook niet eenduidig positief waren, hebben we vervolgens TeX en LateX bestudeerd. Wij bespreken deze laatste twee documentformaten in de volgende sectie.

A.2. TeX en LaTeX

A.2.1. Oorsprong

TeX is 'uitgevonden' door Donald E. Knuth, toen hij op zoek was naar een manier om de opmaak van zijn serie boeken 'The Art Of Computer Programming' te verbeteren. TeX stamt al uit het einde van de jaren '70 en is een zeer krachtig pakket voor desktop publishing (DTP). Ondanks zijn leeftijd wordt TeX nog volop gebruikt (onder andere dankzij verschillende

macropakketten zoals LaTeX van Leslie Lamport). LaTeX is (volgens ons) zo populair geworden dat er meer over LaTex wordt gesproken dan over TeX. Er zijn nog tal van andere bibliotheken ontwikkeld die bovenop TeX gebruikt kunnen worden, allemaal met hun specifieke doeleinden.

A.2.2. Hoe TeX gebruiken?

TeX maakt gebruik van opmaakinstructies die allemaal beginnen met \. Het geraamte van een simpel TeX-document ziet er als volgt uit:

```
\documentclass {letter}
\begin {document}
\begin {letter}
Het is vandaag
\today\
en het
\emph {regent}
.
\end {letter}
\end {document}
```

Laten we bovenstaand voorbeeld eens verder in detail bekijken:

```
\documentclass {letter}
```

Het meegeven van de documentklasse is verplicht. Het argument tussen de accolades specificeert welk type documentklasse gebruikt wordt. In dit geval gaat het om een brief (letter) in Amerikaanse stijl. Voorbeelden van andere mogelijkheden zijn:

```
\documentclass {dinbrief}
\documentclass (12pt,a4paper,twoside) {book}
\documentclass (a4paper,12pt) {report}
```

Hieruit blijkt tegelijkertijd de kracht en de (door ons niet gewenste) mogelijkheden van (La)TeX. Er zijn zeer veel mogelijkheden op het gebied van documentklassen, maar het is belangrijk op te merken dat er effectieve lay-out instructies in het document opgenomen worden. Dit is wat wij eigenlijk willen vermijden, want de structuur wordt zo onnodig belast met opmaakinstructies. En dit past niet in ons streven naar maximale scheiding van inhoud en voorstellingswijze. Vanuit de oorsprong van TeX als een Desktop Publishing-oplossing is deze manier van werken echter logisch te noemen.

```
\begin {document}
```

Met dit commando geef je aan dat de eigenlijke inhoud van het document begint. Het document moet afgesloten worden met

```
\end {document}
```

Tussen `\documentclass {<>}` en `\begin {document}` kunnen nog allerhande declaraties staan, zoals het includen van extra pakketten, het herdefiniëren van commando's, het instantiëren van bepaalde 'variabelen', enzovoort. In plaats van `\begin {document}` of samen met deze tag kunnen er verschillende bestanden ge-include worden. Dit helpt om de individuele bestanden klein te houden en de bestanden volgens een bepaalde logica in te delen, bijvoorbeel een apart bestand voor elk hoofdstuk van een boek.

```
\today\
```

`\today\` is een macro die de huidige datum in het document invoegt, volgens een algemene default-waarde of volgens de waarde voor die documentklasse of misschien ook nog volgens een specifieke opmaak op die plaats. Het is belangrijk om op te merken dat na een macro-oproep een spatie moet volgen die met een escape-teken is beschermd.

```
\emph {<tekst>}
```

Zoals de naam van dit element al aangeeft, dient dit om de nadruk (emphasis) te leggen op de tekst tussen de accolades. Het programma dat het (La)TeX-bestand omzet naar een ander formaat beslist hoe de nadruk op de tekst gelegd wordt, bijvoorbeeld door de tekst vet of schuin weer te geven.

Tot hier een kort inleidend overzicht op (La)TeX. Dit voorbeeld diende alleen om het algemene functioneren van TeX duidelijk te maken. Een overzicht geven van alle mogelijkheden zou ons te ver leiden (geïnteresseerden verwijzen we naar 'The TeX Book' [DEKTB]).

A.2.3. Ervaringen met TeX en LaTeX

Alhoewel we de bruikbaarheid en het nut van TeX/LaTeX niet in twijfel willen trekken, was dit niet het formaat wat we zochten. We zochten naar een volledige scheiding tussen tekst en lay-out, die hier duidelijk niet aanwezig was.

We vonden ook dat sommige tools een rare manier van werken hanteren. Je moet er zelf voor zorgen dat je document minstens twee maal verwerkt wordt om de referenties juist te krijgen. Een rondvraag bij mensen die TeX/LaTeX al lang gebruiken leerde ons dat dit gedrag echt afhankelijk is van de TeX/LaTeX processor die je gebruikt. Op zich is het een elegante oplossing om de referenties te resolven, maar dat zou dan op zijn minst toch door bijna alle tools automatisch moeten gebeuren.

A.3. Onze afwegingen

In deze secties overlopen we onze motivatie om noch voor Docbook, noch voor (La)TeX te kiezen, maar een eigen documentformaat te ontwikkelen.

DocBook is zonder twijfel heel geschikt voor grootschalige technische documenten en projecten in boekvorm. Het is zeer flexibel en heeft veel (naar onze mening, een beetje te veel) features. Er zijn genoeg bestaande tools en packages beschikbaar die DocBook XML/SGML naar een aantal verschillende formaten kunnen omzetten. Helaas is de scheiding tussen inhoud en voorstellingswijze niet consequent doorgevoerd.

Ook (La)TeX is een heel krachtige tool, omdat er veel aandacht wordt besteed aan de structuur van de inhoud. Er bestaan veel tools om LaTeX bestanden om te zetten naar andere bestanden (PDF, HTML, ...), maar toch zien wij ook hier een probleem. Wil je namelijk je document omzetten naar een ander formaat, waar nog geen tool voor bestaat, dan zul je zelf een tool moeten schrijven. Of je kan proberen dit via omwegen te realiseren (latex2html gevolgd door html2formaat), maar per tussenstap gaat er veel structuurinformatie verloren. Ook bij (La)TeX is er sprake van het verwerken van layout-opdrachten in het document zelf.

Wij denken dat voor het schrijven en opmaken van onze thesistekst zowel DocBook als LateX hun doel voorbij schieten. Volgens ons zijn we meer gebaat met een zelfgedefinieerde XML syntax bestaande uit een beperkt aantal tags die voor de semantiek van onze tekst geschikt

zijn. Dit maakt het een stuk gemakkelijker om het overzicht te bewaren. Eens we over die zelfgedefinieerde syntax beschikken, is het niet moeilijk meer om daarvoor een aantal XSL transformaties te schrijven. Het schrijven van een XSL stylesheet is ook eenvoudiger dan het schrijven van een volledig programma om je (La)TeX document om te zetten naar een ander gewenst formaat. Bij XSL transformaties kan je altijd van je oorspronkelijke bron vertrekken, waardoor geen structurele informatie verloren gaat.

Verder willen wij ook nog vermelden dat de conversie van TeX naar een ander formaat statisch is. Met statisch bedoelen wij dat de omzetting gebeurt zodra het document af is en dat vervolgens de omgezette vorm op een informatiedrager opgeslagen wordt. Als je achteraf aanpassingen wilt maken aan het oorspronkelijke TeX document, moet je deze conversie opnieuw uitvoeren. Bij XML/XSLT hoeft dit niet het geval te zijn. Als je bijvoorbeeld gebruik maakt van Cocoon, gebeurt deze omzetting on the fly, eventuele aanpassingen in het brondocument krijg je dus ook dadelijk te zien in het uitvoerdocument. Het grote voordeel van deze dynamische conversie is dus dat de consistentie tussen alle verschillende documentformaten gegarandeerd blijft.

Tenslotte menen wij ook dat een verzameling van zelfgedefinieerde tags de consistentie in onze documenten alleen maar ten goede kan komen. Als we merken dat we behoefte hebben aan extra tags, dan definiëren we die toch gewoon, op die manier zijn wij het die de touwtjes in handen hebben. Dit is net het grote pluspunt van XML, het laat je toe documenten te structureren op een manier die jij het handigst vindt, dus waarom zouden we daar geen gebruik van maken?

A.4. Ons zelfgedefinieerd documentformaat

Ter afsluiting van deze appendix geven we het volledige XML Schema van ons eigen ontwikkeld formaat. Voor uitleg over dit schema verwijzen we naar [Hoofdstuk 4.] .

Voorbeeld A.1. Volledig schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:attribute name="accessibility">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="public"/>
            <xs:enumeration value="private"/>
            <xs:enumeration value="protected"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="occurs">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="many"/>
            <xs:enumeration value="more"/>
            <xs:enumeration value="optional"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="id">
      <xs:simpleType>
         <xs:restriction base="xs:string"/>
      </xs:simpleType>
   </xs:attribute>
   <xs:group name="TitleGroup">
      <xs:choice>
         <xs:element name="INDEX" type="xs:string"/>
```

```
      <xs:element name="EMPH" type="xs:string"/>
    </xs:choice>
  </xs:group>
  <xs:complexType name="EmptyType"/>
  <xsd:simpleType name="identifierType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-zA-Z_$][a-zA-Z0-9_$]*"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xs:simpleType name="composedIdentifierType">
    <xs:restriction base="xs:string">
      <xs:pattern
      value="[a-zA-Z_$][a-zA-Z0-9_$]*(\.[a-zA-Z_$][a-zA-Z0-9_$])*"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="THESIS" type="ThesisType"/>
  <xs:complexType name="ThesisType">
    <xs:sequence>
      <xs:element name="TITLE" minOccurs="1" maxOccurs="2"
      type="TitleLangType"/>
      <xs:element name="AUTHOR" minOccurs="1" maxOccurs="unbounded"
      type="xs:string"/>
      <xs:element name="PROMOTOR" minOccurs="1" maxOccurs="unbounded"
      type="xs:string"/>
      <xs:element name="COACH" minOccurs="1" maxOccurs="unbounded"
      type="xs:string"/>
      <xs:element name="READER" minOccurs="1" maxOccurs="unbounded"
      type="xs:string"/>
      <xs:element name="ABSTRACT" minOccurs="1" maxOccurs="1"
      type="SummaryType"/>
      <xs:element name="ACKNOWLEDGEMENT" minOccurs="1" maxOccurs="1"
      type="SummaryType"/>
      <xs:element name="CHAPTER" minOccurs="1" maxOccurs="unbounded"
      type="ChapterType"/>
      <xs:element name="APPENDIX" minOccurs="0" maxOccurs="unbounded"
      type="ChapterType"/>
      <xs:element name="READINGS" minOccurs="0" type="ReadingsType"/>
      <xs:element name="BIBLIOGRAPHY" type="BibliographyType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TitleLangType" mixed="true">
    <xs:sequence>
      <xs:group ref="TitleGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="language" type="xs:language" use="optional"/>
  </xs:complexType>
  <xs:complexType name="ChapterType">
    <xs:sequence>
      <xs:element name="TITLE" minOccurs="1" maxOccurs="1"
      type="TitleType"/>
      <xs:element name="SUMMARY" minOccurs="0" type="SummaryType"/>
      <xs:element name="SECTION" maxOccurs="unbounded"
      type="SectionType"/>
    </xs:sequence>
    <xs:attribute ref="id" use="optional"/>
  </xs:complexType>
  <xs:complexType name="TitleType" mixed="true">
    <xs:sequence>
      <xs:group ref="TitleGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SummaryType">
    <xs:sequence>
      <xs:element name="PARA" minOccurs="1" maxOccurs="unbounded"
      type="ParaType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SectionType">
    <xs:sequence>
```

```
        <xs:element name="TITLE" minOccurs="1" maxOccurs="1"
        type="TitleType"/>
        <xs:element name="SUBTITLE" minOccurs="0" type="TitleType"/>
        <xs:choice minOccurs="1" maxOccurs="unbounded">
          <xs:element name="PARA" type="ParaType"/>
          <xs:element name="SECTION" type="SectionType"/>
        </xs:choice>
      </xs:sequence>
      <xs:attribute ref="id" use="optional"/>
    </xs:complexType>
    <xs:complexType name="ParaType" mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <!--Elementen voor index/referenties-->
        <xs:element name="INDEX" type="xs:string"/>
        <xs:element name="BIBREFERENCE" type="xs:string"/>
        <xs:element name="REFERENCE" type="xs:string"/>
        <!--URI gerelateerde elementen-->
        <xs:element name="SITE" type="SiteType"/>
        <xs:element name="URL" type="xs:anyURI"/>
        <xs:element name="DIRECTORY" type="xs:anyURI"/>
        <!--Lijst gerelateerde elementen-->
        <xs:element name="LIST" type="ListType"/>
        <xs:element name="EXPLANATIONS" type="ExplanationsType"/>
        <!--XML-gerelateerde elementen-->
        <xs:element name="TAG" type="TagType"/>
        <xs:element name="XML" type="StructureType"/>
        <xs:element name="DOCDECLARATION" type="DocDeclarationType"/>
        <xs:element name="PI" type="PiType"/>
        <xs:element name="ELEMENT" type="ElementType"/>
        <xs:element name="ATTRIBUTE" type="AttributeType"/>
        <xs:element name="COMMENT" type="xs:string"/>
        <xs:element name="SGML" type="StructureType"/>
        <xs:element name="HTML" type="StructureType"/>
        <!--DTD gerelateerde elementen-->
        <xs:element name="DTD" type="DTDType"/>
        <xs:element name="PCDATA" type="EmptyType"/>
        <!--Java gerelateerde elementen-->
        <xs:element name="JAVA" type="JavaType"/>
        <xs:element name="METHOD" type="MethodType"/>
        <xs:element name="VARIABLE" type="VariableType"/>
        <xs:element name="ARGUMENT" type="MethodArgumentType"/>
        <!--Image en Quote-->
        <xs:element name="IMAGE" type="ImageType"/>
        <xs:element name="QUOTE" type="QuoteType"/>
        <!--TeX/LaTeX gerelateerde elementen-->
        <xs:element name="TEXDOCUMENT" type="TexDocumentType"/>
        <xs:element name="TEXMARKUP" type="TexMarkupType"/>
        <!--aantal andere elementen-->
        <xs:element name="EMPH" type="xs:string"/>
        <xs:element name="FORMATTED" type="xs:string"/>
        <xs:element name="CONFIGURATION" type="xs:string"/>
        <xs:element name="CDATA" type="xs:string"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="SiteType">
      <xs:sequence>
        <xs:element name="TITLE" minOccurs="1" maxOccurs="1"
        type="TitleType"/>
        <xs:element name="URL" minOccurs="1" maxOccurs="unbounded"
        type="xs:anyURI"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ListType" mixed="true">
      <xs:sequence>
        <xs:element name="LISTITEM" minOccurs="1" maxOccurs="unbounded"
        type="ParaType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ExplanationsType">
```

```
      <xs:sequence>
        <xs:element name="TITLE" type="TitleType"/>
        <xs:element name="HEADING" type="HeadingType"/>
        <xs:element name="EXPITEM" maxOccurs="unbounded"
        type="ExpItemType"/>
      </xs:sequence>
      <xs:attribute ref="id" use="optional"/>
    </xs:complexType>
    <xs:complexType name="HeadingType">
      <xs:sequence>
        <xs:element name="NAME" type="xs:string"/>
        <xs:element name="EXPLAINS" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ExpItemType">
      <xs:sequence>
        <xs:element name="NAME" type="xs:string"/>
        <xs:element name="EXPLANATION" type="ExplanationType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ExplanationType" mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="REFERENCE" type="xs:string"/>
        <xs:element name="EMPH" type="xs:string"/>
        <xs:element name="BIBREFERENCE" type="xs:string"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="ImageType">
      <xs:sequence>
        <xs:element name="LOCATION" type="xs:anyURI"/>
        <xs:element name="COMMENT" minOccurs="0" type="xs:string"/>
        <xs:element name="COPYRIGHT" minOccurs="0" type="xs:string"/>
        <xs:element name="WIDTH" minOccurs="0"
        type="ImgMeasurementType"/>
        <xs:element name="HEIGHT" minOccurs="0"
        type="ImgMeasurementType"/>
      </xs:sequence>
      <xs:attribute ref="id" use="optional"/>
    </xs:complexType>
    <xs:simpleType name="ImgMeasurementType">
      <xs:restriction base="xs:string">
        <xs:pattern value="[1-9][0-9]*px"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="QuoteType" mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="PARA" type="ParaType"/>
        <xs:element name="EMPH" type="xs:string"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="TexDocumentType" mixed="true">
      <xs:sequence>
        <xs:element name="TEXMARKUP" type="TexMarkupType"
        maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TexMarkupType">
      <xs:choice>
        <xs:element name="MACRO" type="xs:string"/>
        <xs:sequence>
          <xs:element name="NAME" minOccurs="0" type="xs:string"/>
          <xs:element name="ARGUMENTS" minOccurs="0"
          type="TexArgumentsType"/>
          <xs:element name="TEXCONTENT" type="xs:string"/>
        </xs:sequence>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="TexArgumentsType">
      <xs:sequence>
```

```
      <xs:element name="ARGUMENT" maxOccurs="unbounded"
      type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="TagType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="StructureType">
   <xs:sequence>
      <xs:element name="NAME" minOccurs="0" maxOccurs="1"
      type="xs:string"/>
      <xs:element name="PI" minOccurs="0" maxOccurs="1"
      type="PiType"/>
      <xs:element name="DOCDECLARATION" minOccurs="0" maxOccurs="1"
      type="DocDeclarationType"/>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
         <xs:element name="DOTS" minOccurs="0" type="EmptyType"/>
         <xs:element name="ELEMENT" minOccurs="0" type="ElementType"/>
         <xs:element name="PI" minOccurs="0" type="PiType"/>
         <xs:element name="COMMENT" minOccurs="0" type="xs:string"/>
      </xs:choice>
   </xs:sequence>
   <xs:attribute ref="id" use="optional"/>
</xs:complexType>
<xs:complexType name="PiType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:element name="ATTRIBUTE" minOccurs="0" maxOccurs="unbounded"
      type="AttributeType"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="AttributeType">
   <xs:sequence>
      <xs:element name="NAME" minOccurs="1" maxOccurs="1"
      type="xs:string"/>
      <xs:element name="CONTENT" minOccurs="1" maxOccurs="1"
      type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="DocDeclarationType">
   <xs:sequence>
      <xs:element name="ROOTELEMENT" type="xs:string"/>
      <xs:element name="TYPE" type="xs:string"/>
      <xs:element name="URI" type="xs:anyURI" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="ElementType">
   <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="ATTRIBUTE" type="AttributeType"/>
         <xs:element name="CONTENT" type="xs:string"/>
         <xs:element name="COMMENT" type="xs:string"/>
         <xs:element name="ELEMENT" type="ElementType"/>
         <xs:element name="PI" type="PiType"/>
         <xs:element name="DOTS" type="EmptyType"/>
      </xs:choice>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="DTDType">
   <xs:sequence>
      <xs:element name="NAME" minOccurs="0" type="xs:string"/>
      <xs:choice maxOccurs="unbounded">
         <xs:element name="ETD" minOccurs="0" maxOccurs="unbounded"
         type="ETDType"/>
         <xs:element name="ADL" minOccurs="0" maxOccurs="unbounded"
         type="ADLType"/>
```

```
      </xs:choice>
    </xs:sequence>
    <xs:attribute ref="id" use="optional"/>
  </xs:complexType>
  <xs:complexType name="ETDType">
    <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:choice>
        <xs:element name="EMPTY" type="EmptyType"/>
        <xs:element name="CONTENTMODEL" type="ContentmodelType"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ContentmodelType">
    <xs:choice>
      <xs:element name="PCDATA" type="EmptyType"/>
      <xs:sequence>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="CONTENT" minOccurs="0"
          type="ContentType"/>
          <xs:element name="CHOICE" minOccurs="0">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="CONTENT" maxOccurs="unbounded"
                type="ContentType"/>
              </xs:sequence>
              <xs:attribute ref="occurs" use="optional"/>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="ContentType" mixed="true">
    <xs:attribute ref="occurs" use="optional"/>
  </xs:complexType>
  <xs:complexType name="ADLType">
    <xs:sequence>
      <xs:element name="NAME" type="xs:string"/>
      <xs:element name="ATTRIBUTE">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="NAME" type="xs:string"/>
            <xs:element name="CONTENTMODEL">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="PCDATA" type="EmptyType"/>
                  <xs:element name="CONTENT" type="xs:string"/>
                  <xs:element name="CHOICE">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="CONTENT"
                        maxOccurs="unbounded" type="xs:string"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:choice>
              </xs:complexType>
            </xs:element>
            <xs:element name="REQUIRED" minOccurs="0"
            type="EmptyType"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="MethodArgumentType">
    <xs:sequence>
      <xs:element name="TYPE" type="identifierType"/>
```

```
        <xs:element name="NAME" type="identifierType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="VariableType">
      <xs:sequence>
        <xs:element name="FINAL" minOccurs="0" maxOccurs="1"
        type="EmptyType"/>
        <xs:element name="STATIC" minOccurs="0" maxOccurs="1"
        type="EmptyType"/>
        <xs:element name="TYPE" type="identifierType"/>
        <xs:element name="NAME" type="identifierType"/>
        <xs:element name="INITIAL" minOccurs="0" type="xs:string"/>
      </xs:sequence>
      <xs:attribute ref="accessibility" use="optional"/>
    </xs:complexType>
    <xs:complexType name="MethodType">
      <xs:sequence>
        <xs:element name="STATIC" minOccurs="0" type="EmptyType"/>
        <xs:element name="RETURNTYPE" minOccurs="0"
        type="identifierType"/>
        <xs:element name="NAME" type="identifierType"/>
        <xs:element name="ARGUMENTS" type="MethodArgumentsType"/>
        <xs:element name="EXCEPTIONS" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="NAME" maxOccurs="unbounded"
              type="identifierType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="BODY" minOccurs="0" type="BodyType"/>
      </xs:sequence>
      <xs:attribute ref="accessibility" use="optional"/>
    </xs:complexType>
    <xs:complexType name="MethodArgumentsType">
      <xs:sequence>
        <xs:element name="ARGUMENT" minOccurs="0" maxOccurs="unbounded"
        type="MethodArgumentType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="BodyType">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="ASSIGNMENT" type="AssignmentType"/>
        <xs:element name="COMMENT" type="CommentType"/>
        <xs:element name="VARIABLE" type="VariableType"/>
        <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
        <xs:element name="CONSTRUCTION" type="ConstructionType"/>
        <xs:element name="TRYBLOCK" type="TryBlockType"/>
        <xs:element name="IFBLOCK" type="IfBlockType"/>
        <xs:element name="FORLOOP" type="ForLoopType"/>
        <xs:element name="RETURNSTATEMENT" type="RetStmtType"/>
        <xs:element name="THROW" type="ThrowType"/>
        <xs:element name="DOTS" type="EmptyType"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="AssignmentType">
      <xs:sequence>
        <xs:element name="TO">
          <TYPE minOccurs="0" type="identifierType"/>
          <NAME type="identifierType"/>
        </xs:element>
        <xs:choice>
          <xs:element name="FROM" type="xs:string"/>
          <xs:element name="CONSTRUCTION" type="ConstructionType"/>
          <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CommentType" mixed="true">
```

```
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="PARAM" type="CommentParamType"/>
        <xs:element name="RETURN" maxOccurs="1" type="xs:string"/>
      </xs:choice>
      <xs:attribute name="type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="impl"/>
            <xs:enumeration value="doc"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
    <xs:complexType name="CommentParamType">
      <xs:sequence>
        <xs:element name="NAME" type="identifierType"/>
        <xs:element name="DESCRIPTION" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ApplyMethodType">
      <xs:sequence>
        <xs:element name="METHODNAME" minOccurs="1" maxOccurs="1"
        type="identifierType"/>
        <xs:element name="APPLYTO" minOccurs="0" maxOccurs="1"
        type="ArgumentType"/>
        <xs:element name="ARGUMENTS" minOccurs="1" maxOccurs="1"
        type="ArgumentsType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ArgumentsType">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="ARGUMENT" type="ArgumentType"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="ArgumentType" mixed="true">
      <xs:choice minOccurs="0">
        <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
        <xs:element name="CONSTRUCTION" type="ConstructionType"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="ConstructionType">
      <xs:sequence>
        <xs:element name="CONSTRUCTNAME" minOccurs="0" maxOccurs="1"
        type="identifierType"/>
        <xs:element name="CONSTRUCTCLASS" minOccurs="0" maxOccurs="1"
        type="composedIdentifierType"/>
        <xs:element name="CLASS" minOccurs="1" maxOccurs="1"
        type="composedIdentifierType"/>
        <xs:element name="ARGUMENTS" minOccurs="1" maxOccurs="1"
        type="ArgumentsType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TryBlockType">
      <xs:sequence>
        <xs:element name="BODY" type="BodyType"/>
        <xs:element name="CATCHBLOCK" maxOccurs="unbounded"
        type="CatchBlockType"/>
        <xs:element name="FINALLY" minOccurs="0" type="FinallyType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CatchBlockType">
      <xs:sequence>
        <xs:element name="EXCEPTION">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="NAME" type="identifierType"/>
              <xs:element name="SHORTNAME" type="identifierType"/>
            </xs:sequence>
          </xs:complexType>
```

```
      </xs:element>
      <xs:element name="BODY" type="BodyType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="FinallyType">
    <xs:sequence>
      <xs:element name="BODY" type="BodyType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="IfBlockType">
    <xs:sequence>
      <xs:element name="TEST" type="TestType"/>
      <xs:element name="BODY" type="BodyType"/>
      <xs:element name="ELSEIF" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="TEST" type="TestType"/>
            <xs:element name="BODY" type="BodyType"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="ELSE" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="BODY" type="BodyType"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="TestType">
    <xs:sequence>
      <xs:choice>
        <xs:sequence>
          <xs:element name="ARGUMENT" minOccurs="2" maxOccurs="2"
          type="ArgumentType"/>
        </xs:sequence>
        <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="type" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="equals"/>
          <xs:enumeration value="notequals"/>
          <xs:enumeration value="equalsmethod"/>
          <xs:enumeration value="applymethod"/>
          <xs:enumeration value="notapplymethod"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="ForLoopType">
    <xs:sequence>
      <xs:element name="INITIALIZATION" type="xs:string"/>
      <xs:element name="TERMINATION" type="xs:string"/>
      <xs:element name="ITERATION" type="xs:string"/>
      <xs:element name="BODY" type="BodyType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="RetStmtType">
    <xs:sequence>
      <xs:element name="ARGUMENT" type="ArgumentType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ThrowType">
    <xs:sequence>
      <xs:element name="ARGUMENT" type="ArgumentType"/>
    </xs:sequence>
```

```
    </xs:complexType>
    <xs:complexType name="InterfaceType">
      <xs:sequence>
        <xs:element name="NAME" type="identifierType"/>
        <xs:element name="EXTEND" minOccurs="0" maxOccurs="unbounded"
        type="identifierType"/>
        <xs:element name="BODY" type="InterfaceBodyType"/>
      </xs:sequence>
      <xs:attribute ref="accessibility" use="optional"/>
    </xs:complexType>
    <xs:complexType name="InterfaceBodyType">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="VARIABLE" type="VariableType"/>
        <xs:element name="METHOD" type="InterfaceMethodType"/>
        <xs:element name="COMMENT" type="CommentType"/>
        <xs:element name="DOTS" type="EmptyType"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="InterfaceMethodType">
      <xs:sequence>
        <xs:element name="STATIC" minOccurs="0" type="EmptyType"/>
        <xs:element name="RETURNTYPE" minOccurs="1"
        type="identifierType"/>
        <xs:element name="NAME" type="identifierType"/>
        <xs:element name="ARGUMENTS" type="MethodArgumentsType"/>
        <xs:element name="EXCEPTIONS" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="NAME" type="identifierType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute ref="accessibility" use="optional"/>
    </xs:complexType>
    <xs:complexType name="ClassType">
      <xs:sequence>
        <xs:element name="NAME" type="identifierType"/>
        <xs:element name="EXTEND" minOccurs="0" type="identifierType"/>
        <xs:element name="IMPLEMENT" minOccurs="0" maxOccurs="unbounded"
        type="identifierTYpe"/>
        <xs:element name="BODY" type="ClassBodyType"/>
      </xs:sequence>
      <xs:attribute ref="accessibility" use="optional"/>
    </xs:complexType>
    <xs:complexType name="ClassBodyType">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="VARIABLE" type="VariableType"/>
        <xs:element name="METHOD" type="MethodType"/>
        <xs:element name="COMMENT" type="CommentType"/>
        <xs:element name="DOTS" type="EmptyType"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="JavaType">
      <xs:sequence>
        <xs:element name="NAME" minOccurs="0" maxOccurs="1"
        type="xs:string"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="PACKAGE" type="composedIdentifierType"/>
          <xs:element name="COMMENT" type="CommentType"/>
          <xs:element name="IMPORT" type="composedIdentifierType"/>
          <xs:element name="DOTS" type="EmptyType"/>
          <xs:element name="CLASS" type="ClassType"/>
          <xs:element name="INTERFACE" type="InterfaceType"/>
          <xs:element name="VARIABLE" type="VariableType"/>
          <xs:element name="METHOD" type="MethodType"/>
          <xs:element name="ASSIGNMENT" type="AssignmentType"/>
          <xs:element name="TRYBLOCK" type="TryBlockType"/>
          <xs:element name="IFBLOCK" type="IfBlockType"/>
```

```
            <xs:element name="FORLOOP" type="ForLoopType"/>
            <xs:element name="CONSTRUCTION" type="ConstructionType"/>
            <xs:element name="APPLYMETHOD" type="ApplyMethodType"/>
         </xs:choice>
      </xs:sequence>
      <xs:attribute ref="id" use="optional"/>
   </xs:complexType>
   <xs:complexType name="BibliographyType">
      <xs:sequence>
         <xs:element name="BIBITEM" maxOccurs="unbounded"
         type="BibItemType"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="BibItemType">
      <xs:sequence>
         <xs:element name="REFERENCE" type="xs:string"/>
         <xs:element name="DESCRIPTION" type="xs:string"/>
         <xs:element name="URL" minOccurs="0" type="xs:anyURI"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ReadingsType">
      <xs:sequence>
         <xs:element name="READER" maxOccurs="unbounded"
         type="ReaderType"/>
         <xs:choice minOccurs="1" maxOccurs="unbounded">
            <xs:element name="ARTICLE" type="ArticleType"/>
            <xs:element name="BOOK" type="BookType"/>
         </xs:choice>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ReaderType">
      <xs:sequence>
         <xs:element name="NAME" type="xs:string"/>
         <xs:element name="EMAIL" type="xs:string"/>
         <xs:element name="ID" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ArticleType">
      <xs:sequence>
         <xs:element name="TITLE" type="xs:string"/>
         <xs:element name="URI" minOccurs="0" type="xs:anyURI"/>
         <xs:element name="CONTENTS" type="xs:string"/>
         <xs:element name="AUTHOR" maxOccurs="unbounded"
         type="xs:string"/>
         <xs:element name="REVIEWER" minOccurs="0" maxOccurs="unbounded"
         type="ReviewerType"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="BookType">
      <xs:sequence>
         <xs:element name="TITLE" type="xs:string"/>
         <xs:element name="AUTHOR" maxOccurs="unbounded"
         type="xs:string"/>
         <xs:element name="ISBN" minOccurs="0" type="xs:string"/>
         <xs:element name="CONTENTS" type="xs:string"/>
         <xs:element name="REVIEWER" minOccurs="0" maxOccurs="unbounded"
         type="ReviewerType"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ReviewerType">
      <xs:sequence>
         <xs:element name="ID" type="xs:string"/>
         <xs:element name="REVIEW" type="xs:string"/>
         <xs:element name="RATING">
            <xs:restriction base="xs:string">
               <xs:minInclusive value="0"/>
               <xs:maxInclusive value="10"/>
            </xs:restriction>
         </xs:element>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
```

Appendix B. XSLT stylesheets en browsers

B.1. XSLT stylesheets en Internet Explorer

In de eerste template regel van elke XSLT stylesheet moet het rootelement van het te transformeren XML document gematcht worden. Die eerste template moet er volgens de W3C standaarden dus als volgt uitzien:

Voorbeeld B.1. Template volgens de W3C standaard

```
<xsl:template match="root_element">
   <!--Hier wordt beschreven hoe de gematchte knoop verwerkt zal worden
   in het uitvoerdocument-->
</xsl:template>
```

Als je Internet Explorer 5.0 of 5.5 [IE] gebruikt voor XSLT dan gelden bovenstaande regels over de vorm van XSLT stylesheets niet. Alhoewel MSXML 3.0 en MSXML 4.0 (XML parser/XSLT processors voor Internet Explorer) al veel dichter komen bij het ondersteunen van de standaard W3C XSLT specificatie, worden beiden niet standaard meegeleverd bij Internet Explorer 5.0 en 5.5. Je bent dus verplicht om MSXML 3.0 [MSXML] of XML Core Services (MSXML) 4.0 [MSXMLCS] zelf af te halen en te installeren. Voor meer details daarover verwijzen wij naar de Microsoft webpagina [MS]. Opgelet echter, ook deze versies volgen niet volledig de W3C standaarden. MSXML 3.0 verwacht bijvoorbeeld nog steeds het MIME type **text/xsl** in plaats van **text/xml**.

We overlopen nu de verschillende afwijkingen van de standaarden voor het gebruik van XSLT onder Internet Explorer 5.5. Ten eerste gebruikt Internet Explorer 5.5 een andere namespace voor de XSLT elementen, namelijk *http://www.w3.org/TR/WD-xsl* in plaats van *http://www.w3.org/1999/XSL/Transform*. Ten tweede verwacht Internet Explorer 5.5 ook een niet standaard MIME type **text/xsl** in de xml-stylesheet processing instruction (op voorwaarde dat deze aanwezig is) in plaats van **text/xml**. De processing instruction die toegevoegd wordt aan het XML document ziet er dus als volgt uit:

Voorbeeld B.2. Stylesheet associëren met een XML document

```
<?xml-stylesheet type="text/xsl" href="URI van de stylesheet"?>
```

Ten derde implementeert XSLT voor Internet Explorer de default regels [Sectie 5.3.1.4.] **niet**. Kort samengevat zorgen de default regels er voor dat er toch uitvoer gegenereerd wordt voor elementen waarvoor niet expliciet een template voorzien is. Dit geldt dus niet voor Internet Explorer, waar je voor elk element in de hiërarchie van je document een template nodig hebt. Ten slotte moet in een XSLT stylesheet die je onder Internet Explorer gebruikt altijd eerst het root knooppunt gematcht worden. Rekening houdend met deze opmerkingen krijgen we de volgende stylesheet:

Voorbeeld B.3. Uitzicht van een stylesheet onder Internet Explorer

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
       <!--Hier komt de inhoud van deze template-->
```

```
    </xsl:template>
  </xsl:stylesheet>
```

Appendix C. HOW-TO: Cocoon 1 installeren

We geven hier een beknopte, maar duidelijke omschrijving van hoe wij Cocoon 1 geïnstalleerd hebben. We beginnen eerst met een opsomming van de nodige componenten. Daarna beschrijven we hoe we deze componenten hebben geïnstalleerd. We zijn ook enkele problemen tegengekomen hierbij. We vermelden deze problemen en onze oplossing daarvoor. Een bepaalde kennis van de terminologie die gebruikt in verband met de Apache webserver is vereist.

C.1. Vereiste packages

De meest recente versie van Cocoon 1 [COCOON1] was bij het begin van onze thesis versie **1.8.2**. Cocoon 2 verkeerde op dit moment nog in een alfa-stadium. Nu we stilaan de deadline naderen blijkt dat versie **1.8.2** de laatste release is geweest van Cocoon 1. Van de distributiesite [C1DIST] hebben we **Cocoon-1.8.2.tar.gz** gedownload. We hebben vervolgens de installatie-instructies [C1INST] en de vereisten gelezen.

Als webserver hadden we **apache 1.3.12 met DSO (apxs)** draaien op een Compaq Digital Alpha Personal Workstation onder Redhat Linux 7.1. We hadden ook een JDK nodig. De door ons gebruikte JDK was in dit geval versie 1.3.1-1 van de JDK die Compaq ontwikkelde voor AlphaLinux.

Daarnaast moesten we de volgende programma's afhalen:

- Apache JServ servlet engine [APJSERV]. Deze servlet engine kan geïntegreerd worden met Apache. Van de distributiesite [APJSRVDIST] hebben we **ApacheJServ-1.1.2.tar.gz** gedownload.
    - deze servlet engine vereist JSDK2.0 (Java Servlet Development Kit). Versie 2.1 bleek te recent, we moesten echt versie 2.0 hebben. Onze zoektocht leidde ons langs [JSDK] en [JSDKDL] uiteindelijk toch naar [JSDKARCH] waar we de juiste versie konden vinden.

Na het downloaden van deze pakketten, hebben we ze uitgepakt en zijn we begonnen met het lezen van de installatie-instructies.

C.2. Vereiste pakketten installeren - JServ 1.1.2

We hebben dit pakket uitgepakt en de installatie-instructies gelezen. Er zijn 2 mogelijkheden om te installeren:

- **compile-it-in**: veronderstelt dat je Apache nog zelf wilt compileren (deze methode viel af)
- **DSO**: vereist dat je Apache 1.3.* hebt met DSO-ondersteuning (dat is het geval voor onze Apache)

Er wordt aangeraden om de DSO installatie-instructies te volgen als je niet zeker bent, maar aangezien we deze instructies toch moeten volgens maakt dit voor ons niks uit.

C.2.1. Vereiste pakketten installeren - JSDK

Uit de installatie-instructies van Apache JServer blijkt dat het bestand **jsdk.jar** reeds in het classpath moet staan. We plaatsen dit Java archief in de *lib/* directory. Het bestand **jsdk.jar**

zat in het gedownloade bestand met de naam **jsdk20-solaris2-sparc.tar.Z**. Alhoewel in het bestand **README** duidelijk staat: *"If you are interested in developing servlets with JDK 1.2, there is no need to use this JSDK as the servlet API is bundled with JDK1.2."* Alhoewel wij JDK1.3.1-1 gebruiken, kunnen we dat **jsdk.jar** nergens terugvinden. Het bleek dus toch nodig om dit bestand op de juiste plaats te zetten.

C.3. Vereiste packages installeren - JServ 1.1.2 - vervolg

C.3.1. Configuratie-opties (voor DSO)

We geven het pad van apxs (**AP**ache e**X**ten**S**ion tool) op. Deze optie is in het configuratiescript ingedeeld onder **Apache directory**.

```
Apache directory:
        --with-apxs=/usr/sbin/apxs
```

Het installatiepad is in orde voor ons, dus dat wijzigen we niet.

```
Prefix Path
        default: /usr/local/jserv
```

Het configuratiescript (**configure**) maakt ook gebruik van de systeemvariabelen **JDK_HOME** en **JAVA_HOME**. Aangezien we deze variabelen niet geconfigureerd hebben, maken we gebruik van enkele configuratieopties die ervoor zorgen dat deze systeemvariabelen niet nodig zijn. In ons geval wordt dit:

```
JDK Programs
        --with-jdk-home=/usr/local/JDK122
        --with-java=/usr/local/JDK122/bin/java
        --with-javac=/usr/local/JDK122/bin/javac
        --with-javadoc=/usr/local/JDK122/bin/javadoc
        --with-jar=/usr/local/JDK122/bin/jar
```

Om er zeker van te zijn dat het configuratiescript de JSDK vindt, geven we dit ook mee aan het configuratiescript.

```
JSDK classes
        --with-JSDK=/usr/local/jsdk/lib/jsdk.jar
```

Omdat Apache in onze configuratie gebruik maakt van **mod_ssl** voor beveiligde verbindingen is het nodig een extra optie aan het configuratiescript mee te geven. Hiermee wordt duidelijk gemaakt dat Apache Jserv mogelijk ook gebruikt wordt voor beveiligde verbindingen.

```
Support voor EAPI
        --enable-EAPI
```

Dit zorgt ervoor dat we het configuratiescript op de volgende manier gaan uitvoeren:

```
./configure \
        --with-apxs=/usr/sbin/apxs \
        --with-jdk-home=/usr/local/JDK122 \
        --with-java=/usr/local/JDK122/bin/java \
        --with-javac=/usr/local/JDK122/bin/javac \
        --with-javadoc=/usr/local/JDK122/bin/javadoc \
        --with-jar=/usr/local/JDK122/bin/jar \
        --with-JSDK=/usr/local/jsdk/lib/jsdk.jar \
```

```
                --enable-EAPI
```

Merk op dat het **JSDK** is en niet **jsdk**. Dit verloopt gelukkig zonder foutmeldingen en genereert de volgende uitvoer:

```
Apache JServ configuration complete...

Apache Installed Directory: /usr

Module Type: dynamic (DSO will be used to link mod_jserv into server dynamically)

Apache include flags:
  -I/usr/include/apache

Default configuration files will be installed in:
  /etc/httpd/conf/jserv

Default log files will be created in:
  /usr/logs

Default servlets will be installed in:
  /usr/servlets

ApacheJServ.jar file will be installed in:
  /usr/lib/apache/ApacheJServ.jar

Documentation will be installed in:
  /usr/local/jserv/docs

Build classpath is:
/opt/thesis/ApacheJServ-1.1.2/src/java:/usr/local/JDK122/jre/lib/rt.jar:/usr/local/jsdk/lib/jsdk.

You might consider putting frequently used options into
./configure-options, one per line.

+-STEP 1-----------------------------------------------------+
|Run 'make; make install' to make a .jar file, compile the C  |
|code and copy the appropriate files to the appropriate       |
|locations.                                                   |
+-------------------------------------------------------------+

+-STEP 2-----------------------------------------------------+
|Put this line somewhere in Apache's httpd.conf file:         |
|Include /etc/httpd/conf/jserv/jserv.conf
|                                                             |
|Then start Apache and try visiting the URL:                  |
|http://supermachien:SERVER_PORT/servlets/Hello
|                                                             |
|If that works then you have successfully setup Apache JServ. |
|                                                             |
|If that does not work then you should read the               |
|troubleshooting notes referenced below.                      |
+-------------------------------------------------------------+

+-Troubleshooting--------------------------------------------+
|Html documentation is available in the docs directory.       |
|                                                             |
|Common Errors:                                               |
|    Make sure that the log files can be written to by the    |
|    user your httpd is running as (ie: nobody). If there are |
|    errors in your configuration, they will be logged there. |
|                                                             |
|Frequently asked questions are answered in the FAQ-O-Matic:  |
|                                                             |
|           http://java.apache.org/faq/                       |
+-------------------------------------------------------------+
```

C.3.2. Installeren van Apache JServ 1.1.2

We volgen de instructies die we hier op het scherm zien:

```
[root@supermachien ApacheJServ-1.1.2]# make
[root@supermachien ApacheJServ-1.1.2]# make install
```

We brengen nu de nodige wijzigingen aan in **httpd.conf** en herstarten Apache:

```
[root@supermachien conf]# /etc/rc.d/init.d/httpd restart
Syntax error on line 132 of /etc/httpd/conf/jserv/jserv.conf:
Invalid command 'SetHandler', perhaps mis-spelled or defined by a module not
included in the server configuration
```

We hadden de lijn die we moesten invoegen **voor** de loadmodule en addmodule directieven geplaatst (we hadden wel verwacht dat dit zou mislukken). We plaatsen regel nu **na** deze directieven en herstarten Apache nogmaal.

```
[root@supermachien conf]# /etc/rc.d/init.d/httpd restart
[root@supermachien conf]#
```

Vervolgens testen we met lynx, een browser voor de console, of Apache JServer werkt.

```
[root@supermachien conf]# lynx localhost/servlets/Hello
```

Dit levert de volgende uitvoer op:

```
                                                        Example Apache JServ
Servlet

                           Example Apache JServ Servlet

Congratulations, ApacheJServ 1.1.2 is working!
```

Dat ziet er dus goed uit. We moeten nu alleen nog Cocoon 1.8.2 installeren.

C.4. Cocoon 1.8.2 installeren

Omdat ook de configuratie- en installatiescripts van Cocoon 1 gebruik maken van de systeemvariabelen **JAVA_HOME** en **JDK_HOME**, hebben we deze variabelen gedefinieerd zodat iedereen op het systeem er gebruik van kan maken. Na het uitpakken van van het gedownloade bestand hebben we eerst de installatie-instructies gelezen. Deze installatie-instructies zijn terug te vinden door het bestand *index.html* te openen en vervolgens te kiezen voor **Documentation** en daar voor **Install**. Het bestand *index.html* bevindt zich in de hoofddirectory van het uitgepakte bestand.

Naast alle componenten die we reeds geïnstalleerd hebben, vereist Cocoon 1 nog een aantal andere componenten om te kunnen werken. In tegenstelling tot eerdere distributies maken deze componenten wel deel uit van deze distributie. Daar moeten we dus niet meer naar zoeken. Daarnaast wordt ook geëist dat het bestand *$JDK_HOME/lib/tools.jar* deel uitmaakt van het classpath en dat de meegeleverde *xerces_1_2.jar* als eerste XML parser verschijnt in het classpath.

Bij het lezen van de installatie-instructies merken wij dat ze veronderstellen dat je alles al in een definitieve directory hebt geplaatst. We hebben daarom alle bestanden verplaatst naar */usr/local/cocoon/*. Grappig vinden we wel de volgende opmerking bij de installatie-instructies: *" This is an EXAMPLE only and may not be up to date. If you get errors, first check that all the required jar files (see top of page) from cocoon/lib are on your CLASSPATH, and spelled correctly. "* Omwille van deze opmerking besluiten we eerst alle jar

bestanden (die in */usr/local/cocoon/lib/* staan) toe te voegen aan het classpath. Dat zijn er veertien, maar omwille van een andere opmerking moeten we maar dertien jar bestanden in het classpath plaatsen. Deze opmerking luidde: *" JServ is a Servlet 2.0 compliant servlet engine and will not work if you place the Servlet_2.2.jar in its classpath. So ignore the servlet_2.2.jar package that is shipped with Cocoon if you use Jserv. "*

We kwamen dan uiteindelijk tot het volgende classpath:

```
CLASSPATH=/usr/local/JDK122/lib/tools.jar:\
/usr/local/cocoon/bin/cocoon.jar:\
/usr/local/cocoon/lib/xerces_1_2.jar:\
/usr/local/cocoon/lib/ant_1_1.jar:\
/usr/local/cocoon/lib/bsf.jar:\
/usr/local/cocoon/lib/bsfengines.jar:\
/usr/local/cocoon/lib/fesi.jar:\
/usr/local/cocoon/lib/fop_0_15_0.jar:\
/usr/local/cocoon/lib/sax-bugfix.jar:\
/usr/local/cocoon/lib/stylebook-1.0-b2.jar:\
/usr/local/cocoon/lib/turbine-pool.jar:\
/usr/local/cocoon/lib/w3c.jar:\
/usr/local/cocoon/lib/xalan_1_2_D02.jar:\
/usr/local/cocoon/lib/xml.jar:\
/usr/local/cocoon/lib/xt.jar
```

Vervolgens moet het bestand *jserv.properties* aangepast worden. De locatie van dit bestand is */etc/httpd/conf/jserv/*, waar ook het bestand **jserv.conf** zich bevindt, zoals de uitvoer van het configuratieproces van Apache Jserv al aangaf. Voor elk van de jar bestanden die we ook in het classpath hebben geplaatst, moeten we de volgende regel toevoegen aan het *jserv.properties*:

```
wrapper.classpath=[path-to-jar]/[jar-name].jar
```

Dit levert ons de volgende ingangen op in het *jserv.properties* bestand.

```
# CLASSPATH environment value passed to the JVM
# Syntax: wrapper.classpath=[path] (String)
# ...)

wrapper.classpath=/usr/local/JDK122/lib/tools.jar
wrapper.classpath=/usr/local/cocoon/bin/cocoon.jar
wrapper.classpath=/usr/local/cocoon/lib/xerces_1_2.jar
wrapper.classpath=/usr/local/cocoon/lib/ant_1_1.jar
wrapper.classpath=/usr/local/cocoon/lib/bsf.jar
wrapper.classpath=/usr/local/cocoon/lib/bsfengines.jar
wrapper.classpath=/usr/local/cocoon/lib/fesi.jar
wrapper.classpath=/usr/local/cocoon/lib/fop_0_15_0.jar
wrapper.classpath=/usr/local/cocoon/lib/sax-bugfix.jar
wrapper.classpath=/usr/local/cocoon/lib/stylebook-1.0-b2.jar
wrapper.classpath=/usr/local/cocoon/lib/turbine-pool.jar
wrapper.classpath=/usr/local/cocoon/lib/w3c.jar
wrapper.classpath=/usr/local/cocoon/lib/xalan_1_2_D02.jar
wrapper.classpath=/usr/local/cocoon/lib/xml.jar
wrapper.classpath=/usr/local/cocoon/lib/xt.jar
```

Als volgende stap staat vermeld: *" At this point, you must set the Cocoon configuration. To do this, you must choose the servlet zone(s) where you want Cocoon to reside. If you don't know what a servlet zone is, open the zone.properties file. "* Er wordt ons hier ook bijster veel hulp verschaft. Na wat zoekwerk vinden we dat dit bestand in de directory */etc/httpd/conf/jserv/* staat. We hebben dit bestand eens geopend, maar zonder verdere instructies wilden we hier niks aan veranderen.

Als volgende stap stond vermeld dat, om Cocoon te configureren, we de locatie van het bestand *cocoon.properties* aan de servlet engine moesten doorgeven door de volgende line

aan de servlet zone (**zone.properties**) toe te voegen.

```
servlet.org.apache.cocoon.Cocoon.initArgs=properties=[path-to-cocoon]/bin/cocoon.properties
```

Hierbij is **[path-to-cocoon]** een absoluut pad. Het probleem is echter dat *cocoon.properties* zich in de directory *[path-to-cocoon]/conf/* staat. Gelukkig hebben we dit weten te vinden.

Nu zou deze servlet correct geconfigureerd moeten zijn. We moeten er wel nog voor zorgen dat Apache de oproepen voor XML bestanden (of andere bestanden die door Cocoon behandeld moeten worden) doorgeeft aan de Cocoon servlet. Om dit te doen moeten we de volgende regels toevoegen aan *jserv.conf*:

```
Action cocoon /servlet/org.apache.cocoon.Cocoon
AddHandler cocoon xml"
```

Dit vereist wel dat Apache over de Action module beschikt.

Na al deze configuratiestappen kunnen we eindelijk Apache herstarten en proberen we de statuspagina (*http://localhost/Cocoon.xml* te openen. We horen en zien dat er iets gebeurt, maar we krijgen een foutmelding:

```
Cocoon 1.8.2

      java.lang.RuntimeException: Can't create store repository: ./repository.
                Make sure it's there or you have writing permissions.<br/>
                In case this path is relative we highly suggest you to change this
                to an absolute path so you can control its location directly and
                provide valid access rights. See also the FAQ.
               at
               at
               at
               at
               at
               at
               at
               at
               at
               at java.lang.Thread.run(Thread.java:475)

 Warning: this page has been dynamically generated.

                                Copyright (c) 1999-2001 The Apache XML Project.
                                           All rights reserved.
```

In de FAQ [C1FAQNR] vinden we het volgende antwoord op ons probleem: *" Do what the error message tells you! Create a directory which the servlet engine has read and write permissions for. (This directory, the repository, is used to store compiled XSP pages.) Then change the following configuration in cocoon.properties to match the absolute path (where /absolute/path/to/repository should be replaced by the actual path of the repository directory on your system): "*

```
processor.xsp.repository = /absolute/path/to/repository
```

" Finally restart your servlet engine (which you always need to do after changing cocoon.properties). Warning: Since this directory may contain security-sensitive information, make sure you deny access (even read-only) to untrusted users. "

We maken een directory */usr/local/cocoon/repository* aan en zorgen ervoor dat deze directory van de gebruiker **apache** en de groep **apache** is. We zorgen er ook voor dat alleen de gebruiker **apache** toegang heeft tot deze directory. We maken ook de nodige aanpassing in

het bestand cocoon.properties. We herstarten nu nogmaal onze Apache webserver. We krijgen nu wel een mooie statuspagina te zien indien we surfen naar *http://localhost/Cocoon.xml*.

Het is ons uiteindelijk gelukt om Cocoon 1 te kunnen installeren en te laten samenwerken met de Apache webserver. Het heeft ons wel wat moeite gekost om alle componenten en vooral de juiste versie van de componenten te kunnen vinden. Ook hebben we zelf nog enkele fouten in de uitleg ontdekt. Het installeren op zich is niet zo moeilijk. Er is wel veel uitleg over de installatie, maar er wordt met geen woord gerept over eventuele problemen.

Appendix D. Installatie van Cocoon 2

We zullen hier in het kort onze installatie van Cocoon 2 beschrijven. We behandelen hier versie **rc1a** van Cocoon 2. Dit is de versie waarvoor dit document geschreven is, maar op dit moment draagt de stabiele release het versienummer **2.1.0-dev**.

D.1. Machinegegevens

De machine waarop we Cocoon 2 hebben geïnstalleerd heeft de volgende specificaties:

- Digital Personal WorkStation 500au Miata EV56
- Redhat Rawhide Linux
- 256MB RAM
- Compaq JDK 1.3.1-1 voor Alpha

D.2. Vereiste downloads

- De laatste distributie van Cocoon 2 (op dat moment dus **rc1a**) hebben we gedownload van de distributiesite van Cocoon 2 [C2DIST]. Bestandsnaam: **Cocoon-2.0rc1a.tar.gz**.
- Een Servlet 2.2 engine was ook vereist. Op de site met installatie-instructies voor Cocoon 2 wordt daarvoor verwezen naar Jakarta Tomcat [JAKTOM]. Van deze site downloaden we versie **3.2.3**, op dat moment dat laatste stabiele relaese. Bestandsnaam: **jakarta-tomcat-3.2.3.tar.gz**.

Ook een JDK1.2 of hoger was vereist, maar die hadden we al.

D.3. Installeren van Tomcat

We willen Tomcat installeren in de directory /usr/local/. We pakken daarom het bestand **jakarta-tomcat-3.2.3.tar.gz** uit naar die directory. Er is nu een subdirectory jakarta-tomcat-3.2.3/ bijgekomen in /usr/local/. In die nieuwe directory staat ook een directory bin/ waarin alle opstartscripts staan.

Om een of andere reden zijn de opstartscripts niet uitvoerbaar. We hebben dit daarom zelf opgelost door de shellscripts uitvoerbaar te maken. Verder verwachten de opstartscripts ook nog dat de omgevingsvariabelen **JAVA_HOME** en **TOMCAT_HOME** gedefinieerd zijn. Op onze configuratie krijgen deze omgevingsvariabelen de volgende waarden:

```
JAVA_HOME=/usr/java/jdk1.3.1
TOMCAT_HOME=/usr/local/jakarta-tomcat-3.2.3
```

Daarna voeren we het script **startup.sh** uit in de directory /usr/local/jakarta-tomcat-3.2.3/bin/. Dit doen we met het commando **./startup.sh** uit te voeren in deze directory. We krijgen dan de volgende meldingen op het scherm te zien:

```
[root@supermachien bin]# ./jakarta.sh
2001-11-09 14:11:38 - ContextManager: Adding context Ctx( /examples )
2001-11-09 14:11:39 - ContextManager: Adding context Ctx( /admin )
Starting tomcat. Check logs/tomcat.log for error messages
2001-11-09 14:11:39 - ContextManager: Adding context Ctx(  )
2001-11-09 14:11:39 - ContextManager: Adding context Ctx( /test )
2001-11-09 14:12:03 - PoolTcpConnector: Starting HttpConnectionHandler on 8080
Server 1.6 is running
Press [Ctrl]+[C] to abort
```

Dit vertelt ons dat Tomcat correct werkt. We surfen naar *http://localhost:8080/* en we krijgen de startpagina van Tomcat te zien waardoor we weten datTomcat inderdaad correct werkt.

D.4. Installeren van Cocoon 2

We hebben het archief van de distributie naar een willekeurige directory uitgepakt. Vermits het nog om alfasoftware gaat was er alleen een source distributie. Dit betekende dat we Cocoon 2 volledig zelf moesten compileren. Gelukkig zitten er duidelijke installatie-instructies bij de distributie waarin de volgende stappen beschreven staan:

- Zorg dat je een Servlet 2.2 of 2.3 compatibele engine geïnstalleerd hebt.
- Zet de **JAVA_HOME** omgevingsvariabele.
- Voor een automatische installatie voor je het commando **./build.sh -Dinclude.webapp.libs=yes -Dinstall.war=$TOMCAT_HOME/webapps install** uit.
- Herstart de servlet engine.
- Probeer met je favoriete browser te surfen naar *http://localhost/cocoon/*, *http://localhost:8080/cocoon/* of *http://localhost:port/cocoon/*. Welke URL je moet proberen is afhankelijk van je configuratie.
- Wacht enkele ogenblikken (Cocoon moet nog een aantal onderdelen compileren).
- Maak kennis met Cocoon 2.

De eerste twee stappen hadden we reeds uitgevoerd. We proberen dus het commando **./build.sh -Dinclude.webapp.libs=yes -Dinstall.war=$TOMCAT_HOME/webapps install** uit te voeren. Dit lukte echter niet van de eerste keer. Het build-script riep andere scripts aan die niet als uitvoerbaar gemarkeerd waren. Dit hebben we dan opgelost om dan te merken dat Linux kwam klagen deze bestanden niet uitgevoerd konden worden. De fout, die reeds bekend was, bleek te liggen in het formaat van de andere scripts. De andere scripts waren "gemerkt" als DOS-bestanden. Na ook dit probleem opgelost te hebben verliep het compileren van Cocoon 2 vlekkeloos.

D.5. Jakarta Tomcat afstemmen op Cocoon 2

Op de installatiepagina van Cocoon 2 [C2INST] staan wel nog enkele zaken vermeld die we eerst moeten uitvoeren. Dit omdat we met Tomcat 3.2.3 werken. In de directory $TOMCAT_HOME/lib/ staat een bestand **jaxp.jar** dat verwijderd moet worden. Een ander bestand in diezelfde directory, met name **parser.jar** moet hernoemd worden naar **zparser.jar**. Vervolgens moet de met Cocoon 2 geleverde versie van Xerces, in dit geval **xerces_1_4_3.jar** naar die directory gekopieerd worden. Dit is ook de reden waarom **parser.jar** hernoemd moest worden naar **zparser.jar**. Bij het opstarten voegt Tomcat de bestanden in de $TOMCAT_HOME/lib/ directory toe aan het classpath, blijkbaar in alfabetische volgorde. Op die manier wordt er voor gezorgd dat Cocoon 2 de juiste parser kan gebruiken.

Daarna hebben we vol spanning Tomcat herstart. We kregen weer dezelfde opstartmeldingen, echter met één belangrijke regel extra:

```
2002-11-09 14:11:39 - ContextManager: Adding context Ctx( /cocoon )
```

De installatie van Cocoon 2 is dus blijkbaar gelukt en gedetecteerd door Tomcat. Vervolgens hebben we een willekeurige browser gestart en gesurfd naar *http://localhost:8080/cocoon/*. We hebben het installatieproces op twee bijna identieke machines uitgevoerd. Op de ene machine werkte Cocoon 2 zonder problemen maar op de tweede machine gaf Cocoon 2 niets als problemen.

Om deze problemen op te lossen hebben we in de logs ($TOMCAT_HOME/webapps/cocoon/WEB-INF/logs/) gekeken en vervolgens op het internet gezocht naar de oplossing. De oorzaak van de problemen bleek een **java.lang.UnsatisfiedLinkError** te zijn die kwam melden dat in een bepaalde library het symbool **XpGetDocumentData** niet gedefinieerd was. Na eerst op internet gezocht te hebben naar mogelijke oplossingen (die geen resultaat hadden) hebben we de volledige configuraties van de beide systemen vergeleken. Na een beetje zoeken en rondvragen wisten we dat het probleem waarschijnlijk bij aan "te oude" versie van openmotif kon liggen. Op het systeem met de werkende Cocoon 2 bleek **openmotif-2.1.30-8** geïnstalleerd te zijn. Op het andere systeem was **openmotif-2.1.30-5** geïnstalleerd. Onder het motto **baat het niet, dan schaadt het niet**, hebben we een kleine upgrade uitgevoerd met als resultaat een werkende versie van Cocoon 2 op de beide systemen.

D.6. Besluit

We vinden dat al bij al de installatie van Cocoon 2 nog is meegevallen. De installatieprocedure verliep vrij vlot en was redelijk gedocumenteerd. Het enige echte probleem bleek voort te komen uit een juist niet voldoende recente versie van de openmotif-bibliotheek. We willen er tenslotte nog aan toe voegen dat dit probleem bij slechts één component van Cocoon 2 ligt. Op mogelijke problemen met deze component wordt nu wel voldoende gewezen op de verschillende pagina's van Cocoon 2.

Appendix E. How to write your own Generator for Cocoon 2

E.1. Preface

This document is written in context of the thesis of Carolien Coenen and Erwin Hermans at the department of Computer Science at the Katholieke Universiteit Leuven [CSKUL].

At some point in our thesis the need for extending the functionality of Cocoon 2 became imminent. At that moment, we worked with an application that generated XML documents from Java source files (a sort of JavaDoc tool). We wrote some XSL stylesheets for transforming these documents into HTML, so they could easily be served by Cocoon 2. But **every time** the documentation comments changed in the Java source files, these XML documents **required (re)generation**. The ultimate goal of this project was to be able to do all the intermediate steps of the document generation in memory. This way the HTML documents would change whenever the Java source files were modified, **without the need to regenerate the XML documents**.

To reach this goal, we made some modifications in our program. Our output documents were built in memory using the JDOM API [JDOM]. At the time of writing this API supported 3 output methods, which are **DOM** [TRDOM], **SAX** [SAX] and **XML** [TRXML]. The DOM output method outputs a DOM tree from an existing JDOM tree. The XML output method outputs an XML file to an OutputStream or a Writer. But the most interesting of these three outputmethods is SAX. The generators that come with Cocoon 2 don't supply a method (or at least we haven't found any) to take the SAX output of an arbitrary Java program and feed it into the transformers (in our case). That's why we wanted to write a generator that would be able to supply SAX events that were outputted by our (or an arbitrary) Java program and to feed this SAX events into the transformer pipeline. This generator would be responsible for starting the Java program and delegating the received SAX events to the Cocoon 2 transformation pipeline in some way.

To accomplish this task we decided to write our own generator and this documentation in parallel, as there was no documentation available, except for the following phrase on Apache's Cocoon 2 website [COCOON2]: *"A Generator generates XML content as SAX events and initializes the pipeline processing."* Apache's Cocoon 2 website also enumerates the available Generators, which package contains them, which interfaces and helper classes there are in that package and which of those should be implemented/extended. But at that point, the documentation stopped. So the only way to actually be able to write a generator ourselves, was by studying the Cocoon 2 code (more specific the Generators) and figure out for ourselves how to write a generator. Because we want to make a contribution to the Cocoon 2 community, we have written this document. We think our experiences may come in handy for developers who are also thinking about extending Cocoon 2 with their own generators, .

The writing of this generator and all the testing of our code were performed with the following configuration:

- Compaq/Digital Personal Workstation 500a (Alpha 500MHz processor)
- Redhat Linux 7.1
- Compaq JDK 1.3.1-1 (Classic VM (build 1.3.1-1, native threads, jit))
- Jakarta Tomcat 3.2.3

- Cocoon 2.0.2-dev

E.2. Planning

Here you'll find a list of consequent steps that we expect will be necessary to write our own Generator. It is of course possible that in this first draft of our planning we have forgotten a few steps or that some steps actually form one step.

- Find out which classes should be extended and which interfaces implemented.
- Examine these superclasses and interfaces and find which methods should be actually implemented and what is excepted from these methods.
- Write a first Generator as a **proof of concept** to see if our conlusions in relation to the methods are correct, and let this Generator generate some SAX events (hard coded in the Generator) to 'feed' to Cocoon2.
- Find out how to feed the SAXOutput of a JDOM document (hardcoded in the Generator) to Cocoon2.
- Modify our program so it can generate SAXOutput when this is requested.
- Feed the SAXOutput from our program to the Generator (we think it shall require running our program from the Generator). This SAXOutput shall then be passed to Cocoon 2. Again, every call to our program and the parameters will be hardcoded in our Generator.
- Find out how we can use the sitemap to pass parameter values to our Generator (= examing how the sitemap is transformed into Java code and how we can read the parameter values from this code). This will no longer be hardcoded then.
- Examine how we can define in the most general way the syntax from the sitemap, so that it is possible to define which class should be called to generate SAXOutput and with which values and methods this class should be called. This also implies that we must study if this is possible in Java.
- This will be tested with our program and our Generator (hopefully we'll get this far) will become heavily parameterized.
- Modify our program once again, so that it satisfies our final needs.
- Submit our generator, and this document to the Cocoon 2 project.

E.3. Our planning, step by step

E.3.1. Classes and interfaces to be extended/implemented

In this section, we'll discuss which classes should/can be extended to implement our own Generator. Also, we'll take a closer look at these classes to see which functionality they provide and (try to) discuss their functionality. Let it be clear that it is **not required** to extend one of the existing (abstract) generator classes. But by doing so, it'll make your work a great deal easier.

The second part of this section will discuss the interface(s) that have to be implemented to write our own generator. We'll look at what the methods that are defined in the interface should do. There is one interface that has to be implemented if you write your own generator: the **org.apache.cocoon.generation.Generator** interface.

E.3.1.1. Classes

According to the Cocoon 2 website at the time of writing (21st november 2001) there are four helper classes in the org.apache.cocoon.generation package that can be extended. These four are (they will be discussed later):

- AbstractGenerator
- AbstractServerPage
- ComposerGenerator
- ServletGenerator

Java only supports single inheritance, so you'll have to choose one of these for your Generator. We want to use the AbstractGenerator, but to help the reader of this document in making a well motivated choice, we'll discuss each of these options briefly as to what specific functionality they provide.

There is a hierarchy between these classes, namely:

- AbstractGenerator
- ComposerGenerator extends AbstractGenerator
- ServletGenerator extends ComposerGenerator
- AbstractServerPage extends ServletGenerator

So the choice of which class to extend will depends mostly on which is the level of abstraction required by your generator.

E.3.1.1.1. AbstractGenerator

Extend this one for easier building of your own Generator

This **abstract class** extends the class **org.apache.cocoon.xml.AbstractXMLProducer** and implements the interface **org.apache.cocoon.generation.Generator**.

The **Generator** interface extends the interfaces **org.apache.cocoon.xml.XMLProducer** and **org.apache.cocoon.sitemap.SitemapModelComponent**.

The interface **XMLProducer** is a top-level interface.

The interface **SitemapModelComponent** extends the interface **org.apache.avalon.framework.component.Component**, which in turn is a top-level interface.

The abstract class **AbstractXMLProducer** extends the abstract class **org.apache.avalon.framework.logger.AbstractLoggable** and implements the interfaces **org.apache.cocoon.xml.XMLProducer** and **org.apache.avalon.excalibur.pool.Recyclable**.

AbstractLoggable is a top-level abstract class to provide logging capabilities. This is deprecated and **AbstractLogEnabled** should be used instead. AbstractLoggable implements the interface **org.apache.avalon.framework.logger.Loggable**, but as mentioned in the API docs **org.apache.avalon.framework.logger.LogEnabled** should be used instead.

The interface **Recyclable** extends the interface **org.apache.avalon.excalibur.pool.Poolable**, which is a top-level interface.

The following methods are defined for **AbstractGenerator**, some of which already have an implementation:

- From **org.apache.avalon.excalibur.pool.Poolable**:

- None

This interface is implemented bij components if it is reasonable to Pool the object. It marks the component Poolable.

- From **org.apache.avalon.excalibur.pool.Recyclable**:
  - `public void` **`recycle`** `() ;` : this method should be implemented to remove all costly resources in the object. These resources can be object references, database connections, threads, etc. What is categorised as "costly" resources is determined on a case by case analysis.

- From **org.apache.avalon.framework.logger.Loggable** (Deprecated):
  - `public void` **`setLogger`** `(org.apache.log.Logger` **`logger`**`) ;` : provide component with a logger.

Interface can be implemented by Components that need to log.

- From **org.apache.avalon.framework.logger.AbstractLoggable** (Deprecated):
  - `protected org.apache.log.Logger` **`getLogger`** `() ;` : Helper method to allow sub-classes to acquire logger (implemented).
  - `protected void` **`setupLogger`** `(java.lang.Object` **`component`**`) ;` : Helper method to setup other components with the same logger (implemented).
  - `protected void` **`setupLogger`** `(java.lang.Object` **`component`**`, org.apache.log.Logger` **`logger`**`) ;` : Helper method to setup other components with logger (implemented).
  - `protected void` **`setupLogger`** `(java.lang.Object` **`component`**`, java.lang.String` **`subCategory`**`) ;` : Helper method to setup other components with logger (implemented).

This is a utility class to allow the construction of easy components that will perform logging.

- From **org.apache.cocoon.xml.XMLProducer**
  - `void` **`setConsumer`** `(XMLConsumer` **`xmlconsumer`**`) ;` : set the XMLConsumer that will receive XML data. The XMLConsumer interface extends **org.xml.sax.ContentHandler** and **org.xml.sax.ext.LexicalHandler**.

This interface identifies classes that produce XML data, sending SAX events to the configured XMLConsumer.

- From **org.apache.cocoon.xml.AbstractXMLProducer**:
  - `void` **`setConsumer`** `(XMLConsumer` **`xmlconsumer`**`) ;` : set the XMLConsumer that will receive XML data (implemented).
  - `public void` **`setContentHandler`** `(ContentHandler` **`consumer`**`) ;` : Set the ContentHandler that will receive XML data (implemented).
  - `public void` **`setLexicalHandler`** `(LexicalHandler` **`handler`**`) ;` : Set the LexicalHandler that will receive XML data (implemented).
  - `public void` **`recycle`** `() ;` : Recycle the producer by removing references (implemented).

- From **org.apache.cocoon.generation.Generator**:
  - `void` **`generate`** `() throws` **`IOException`**`,` **`SAXException`**`,` **`ProcessingException`**`;` : generate the SAX events to initialize a pipeline.

- From **org.apache.cocoon.sitemap.SitemapModelComponent**:
    - `void` **`setup`** `(SourceResolver` **`resolver`**`, Map` **`objectmodel`**`, String` **`src`**`, Parameters` **`par`**`) ;` : set the SourceResolver, objectmodel Map, the source and sitemap Parameters used to process the request.

- From **org.apache.avalon.framework.component.Component**:
    - None

  When a class implements this interface, it allows itself to be managed by a ComponentManager and by an outside element called a Composable. The Composable must know what type of Component it is accessing, so it will re-cast the Component into the type it needs.
- From **AbstractGenerator** itself:
    - `void` **`setup`** `(SourceResolver` **`resolver`**`, Map` **`objectmodel`**`, String` **`src`**`, Parameters` **`par`**`) ;` : set the SourceResolver, objectmodel Map, the source and sitemap Parameters used to process the request (implemented).
    - `public void` **`recycle`** `() ;` : Recycle the generator by removing references (override implementation).

If we carefully analyse this list, we see that the only method left unimplemented is the **generate()** method. So if we extend the **AbstractGenerator** class to make our own generator, the only method we'll have to implement is the **generate()** method.

The following variables are defined in the different interfaces and classes:
- From **org.apache.avalon.excalibur.pool.Poolable**:
    - None

- From **org.apache.avalon.excalibur.pool.Recyclable**:
    - None

- From **org.apache.avalon.framework.logger.AbstractLoggable** (Deprecated):
    - None

- From **org.apache.avalon.framework.logger.AbstractLoggable** (Deprecated):
    - `private` **`org.apache.log.Logger`** `m_logger;`: the base logger instance. Provides component with a logger. The **getLogger()** method should be used to aquire the logger.

- From **org.apache.cocoon.xml.XMLProducer**
    - None

- From **org.apache.cocoon.xml.AbstractXMLProducer**:
    - `protected` **`XMLConsumer`** `xmlConsumer;`: The XMLConsumer receiving SAX events. Can be accessed via **super.xmlConsumer**.
    - `protected` **`ContentHandler`** `contentHandler;`: The ContentHandler receiving SAX events. Can be accessed via **super.ContentHandler**.
    - `protected` **`LexicalHandler`** `lexicalHandler;`: The LexicalHandler receiving SAX events. Can be accessed via **super.LexicalHandler**.

We here access these variables via the **super.** qualifier, this is only done for clarity. They can be equally well accessed via using the **this.** qualifier (or omitting this). For reasons of clarity, if we access such a variable in our code, we will use the **super.** qualifier, although when summing up all the variables we will use **this.**.

- From **org.apache.cocoon.generation.Generator**:
  - `String ROLE = "org.apache.cocoon.generation.Generator";`

- From **org.apache.cocoon.sitemap.SitemapModelComponent**:
  - None

- From **org.apache.avalon.framework.component.Component**:
  - None

- From **AbstractGenerator** itself:
  - `protected SourceResolver resolver = null;`: The current SourceResolver. Can be accessed via **this.resolver**.
  - `protected Map objectModel = null;`: The current Map objectModel. Can be accessed via **this.objectModel**.
  - `protected Parameters parameters = null;`: The current Parameters. Can be accessed via **this.parameters**.
  - `protected String source = null;`: The source URI associated with the request or **null**. Can be accessed via **this.source**.

This gives us a list of variables that we can use throughout our own generator.

E.3.1.1.2. ComposerGenerator

Can be used as base class if you want your Generator to be an Avalon Composer

This **abstract class** extends **org.apache.cocoon.generation.AbstractGenerator** and extends the interfaces **org.apache.avalon.framework.component.Composable** and **org.apache.avalon.framework.activity.Disposable**.

In addition to all the methods introduced in the **AbstractGenerator** class, these two interfaces introduce som new methods:

- From **org.apache.avalon.framework.component.Composable**:
  - `public void compose (ComponentManager componentManager) throws ComponentException;` : Pass the ComponentManager to the composer. The Composable implementation should use the specified ComponentManager to acquire the components it needs for execution.

- From **org.apache.avalon.framework.activity.Disposable**:
  - `public void dispose () ;` : The dispose operation is called at the end of a components lifecycle. Components use this method to release and destroy any resources that the Component owns.

  The Disposable interface is used when components need to deallocate and dispose resources prior to their destruction.
- From **ComposerGenerator** itself:
  - `public void compose (ComponentManager componentManager) throws ComponentException;` : Pass the ComponentManager to the composer. The Composable implementation should use the specified ComponentManager to acquire the components it needs for execution.

(implemented)

- `public void` **`dispose`** `() ;` : The dispose operation is called at the end of a components lifecycle. Components use this method to release and destroy any resources that the Component owns. (implemented - implementation sets the ComponentManager to null)

We see that this class provides a default implementation of the methods introduced by the two new interfaces. The only method that needs to be implemented remains the **generate()** method, if we are satisfied with the default implementations.

Besides these methods, also a new variable is introduced:

- From **ComposerGenerator** itself:
  - `protected` **`ComponentManager`** `manager = null;`: the component manager instance, can be accessed via **this.manager**.

E.3.1.1.3. ServletGenerator

If you want to generate servlets. This is the base class for the ServerPagesGenerator

extends ComposerGenerator

E.3.1.1.4. AbstractServerPage

[FIXME: This seems to be intended as basis for the ServerPagesGenerator, but it seems to be obsolete now?]

extends ServletGenerator

E.3.1.2. Interface(s)

Following the somewhat little pointers on develop-part of the Cocoon-website [DVAPXML], we find that the only interface that should be implemented is the Generator interface in the **org.apache.cocoon.generation** package, so we will try to give an overview as complete as possible for now.

E.3.1.2.1. Generator

As mentioned earlier this public interface is situated in the **org.apache.cocoon.generation** package. This interface in its turn extends the **org.apache.cocoon.xml.XMLProducer** and the **org.apache.cocoon.sitemap.SitemapModelComponent** interfaces.

The interface **XMLProducer** is a top-level interface.

The interface **SitemapModelComponent** extends the interface **org.apache.avalon.framework.component.Component**, which in turn is a top-level interface.

Analyzing these interfaces tells us that the following methods should be implemented when implementing the **Generator** interface:

- From **org.apache.cocoon.xml.XMLProducer**
    - `void` **`setConsumer`** `(XMLConsumer` **`xmlconsumer`**`) ;` : set the XMLConsumer that will receive XML data. The XMLConsumer interface extends **org.xml.sax.ContentHandler** and **org.xml.sax.ext.LexicalHandler**.

  This interface identifies classes that produce XML data, sending SAX events to the configured XMLConsumer.
- From **org.apache.cocoon.sitemap.SitemapModelComponent**:
    - `void` **`setup`** `(SourceResolver` **`resolver`**`, Map` **`objectmodel`**`, String` **`src`**`, Parameters` **`par`**`) ;` : set the SourceResolver, objectmodel Map, the source and sitemap Parameters used to process the request.

- From **org.apache.avalon.framework.component.Component**:
    - None

  When a class implements this interface, it allows itself to be managed by a ComponentManager and by an outside element called a Composable. The Composable must know what type of Component it is accessing, so it will re-cast the Component into the type it needs.
- From **org.apache.cocoon.generation.Generator** itself:
    - `void` **`generate`** `() throws` **`IOException`**`,` **`SAXException`**`,` **`ProcessingException`**`;` : generate the SAX events to initialize a pipeline.

We decided that the easiest way of writing a custom generator was to extend the **AbstractGenerator** class. The only method required to implement was the **generate** method, for now, we would settle with the provided default implementations of the other methods.

E.3.2. Writing a test generator

After making these decisions, and looking at the implementations of the classes, we could begin the implementation keeping in mind the following:

- We have to provide SAX events to the XMLConsumer, that is set via the **setConsumer** method.
- We can access the XMLConsumer via **super.xmlConsumer** (analysis of code of **FileGenerator** and definition of the **xmlConsumer** variable as **protected** in the **AbstractXMLProducer** class). The **super.** modifier is only used for clarity, since it can also be accessed via **this.xmlConsumer.**
- We will extend the **org.apache.cocoon.generation.AbstractGenerator** class.
- We have to implement the **generate** method which purpose is to produce SAX events and feed them to the XMLConsumer.

E.3.2.1. The code of our first generator

As a first test we decided to parse a string containing the following XML content and feed the SAX events to the XMLConsumer:

Voorbeeld E.1.

```
<doc> My first Cocoon 2 generator! </doc>
```

First, we will give our code and then we will explain what it does and why we made these choices.

Voorbeeldcode E.1. test/MyGenerator.java

```
package test;

import java.io.IOException;
import java.io.StringReader;
import org.xml.sax.XMLReader;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReaderFactory;
import org.apache.cocoon.ProcessingException;
import org.apache.cocoon.generation.AbstractGenerator;

public class MyGenerator extends AbstractGenerator {
  public void generate () throws IOException, SAXException, ProcessingException {
      String message = "<doc>My first Cocoon 2 generator!</doc>";

      XMLReader xmlreader = XMLReaderFactory.createXMLReader();
      xmlreader.setContentHandler(super.xmlConsumer);
      InputSource source = new InputSource(new StringReader(message));
      xmlreader.parse(source);
  }

}
```

First of all, in our working directory (may be any directory) we made a directory *test* and in that directory we created the Java source file **MyGenerator.java**. We also decided to put this class in a package and named that package **test**. This can be easily changed afterwards.

The obvious **import** statements are those to import the **AbstractGenerator** class and those to import the exceptions thrown by the **generate** method. The other import statements serve to parsing our string and generating SAX events.

The code itself is pretty straightforward. We have our class definition containing one method definition. First of all, in the **generate** method, we define the variable **message** containing the XML content we want to generate SAX events for.

Voorbeeldcode E.2.

```
XMLReader xmlreader = XMLReaderFactory.createXMLReader();
```

Here we make a new **XMLReader** via the **XMLReaderFactory**. Since XMLReader is an interface, the XMLReaderFactory has to provide us with a class that implements the XMLReader interface, commonly known as a **SAXParser**. Therefore the XMLReaderFactory uses the system variable **org.xml.sax.driver** to determine which class to instantiate to provide us with an XMLReader. An example of how this is done is provided after we have discussed the rest of the code.

Voorbeeldcode E.3.

```
xmlreader.setContentHandler(super.xmlConsumer);
```

With this line of code, we tell the XMLReader which object will receive the SAX events that will be generated when parsing. You can see that we use **super.xmlConsumer** to receive the SAX events.

Voorbeeldcode E.4.

```
InputSource source = new InputSource(new StringReader(message));
xmlreader.parse(source);
```

With the second line we tell the XMLReader to start parsing the source, provided as argument of the **parse** method. This parse method can only be supplied with an **org.xml.sax.InputSource** argument or a **String** that represents a system identifier or URI. To parse our string we must encapsulate it in an InputSource object. Since the InputSource class can not be passed an XML document that is contained in a string, we first must encapsulate our string into another object, which we then pass to an InputSource object. In this example we have chosen for a **StringReader**. A StringReader can be given as argument when constructing an InputSource object and a StringReader can be given a String object as argument for its construction. This way we succeed in parsing our string.

The next step is compiling our newly written class. We give here an overview of the our work environment and show how we compiled this Java-file. All the commands from this example were carried out on a PC running Linux, but with respect to a few minor modifications, these commands will also work on a PC running Windows. The commands were carried out in the directory */home/erwin/cocoon2/generator/*. This directory has three subdirectories:

- *test/*: directory containing the source files
    - **MyGenerator.java**: source file for our generator

- *jar/*: directory containing the necessary jar (Java Archive) files
    - **xerces.jar**: Xerces has an implementation for the XMLReader interface which we use
    - **cocoon.jar**: contains the classes from Cocoon 2.0.2-dev, needed to extend AbstractGenerator. This is in fact a symbolic link to $TOMCAT_HOME/webapps/cocoon/WEB-INF/lib/cocoon-2.0.2.jar. Under Windows you will have to copy this file or point directly to this file.

- *compiled/*: outputdirectory for javac. The compiled files will end up in this directory
    - **test/MyGenerator.class**: after compiling, we will have this file here

```
[erwin generator]$ javac -classpath .:jar/cocoon.jar:jar/xerces.jar -d compiled
test/MyGenerator.java
[erwin generator]$
```

Now we have our compiled class, we can make the big step of putting it to work. To make sure there were no errors in our code, we tested our code by using another class as the ContentHandler of our generator. After these tests were completed (without errors), we tried to deploy our generator from within Cocoon 2.

E.3.2.2. Deploying MyGenerator

The next step is deploying our custom written generator. First of all we stopped the Tomcat engine (and thus Cocoon 2). We also emptied the **work** directory, located at *$TOMCAT_HOME/work/*. Experience learned that this is something you have to do every time you want to try something like this with Cocoon 2.

For the next step, we changed the main sitemap to be able to use or generator in the following way:

- Under the **map:generators** element, we added the following:

  Voorbeeld E.2.

```
<map:generator name="mygenerator" src="test.MyGenerator"/>
```

- Under the **map:pipelines** element, we added the following:

  Voorbeeld E.3.

```
<map:pipeline>
   <map:match pattern="mygenerator.xml">
      <map:generate type="mygenerator"/>
      <map:serialize type="xml"/>
   </map:match>
   <map:handle-errors>
      <map:transform src="stylesheets/system/error2html.xsl"/>
      <map:serialize status-code="500"/>
   </map:handle-errors>
</map:pipeline>
```

  If the page **mygenerator.xml** is requested, we tell Cocoon 2 to use our generator, which we have named **mygenerator**. We do not define the **src** attribuut, since we do not use it in our generator. Once we get the content, we serialize it as xml, so we can check if the input matches the output. In the event that an error occurs, we use one of the stylesheets of Cocoon 2 or another pipeline to signal the error to the user.

After the changes to the sitemap, we added the directory */home/erwin/cocoon2/generator/* to the classpath. After these changes, we restarted Tomcat and tried to access the page *http://localhost:8080/cocoon/mygenerator.xml*. After waiting a while, we received a fatal error. Inspection of the log-files (which is something you should always do when receiving an error that is not so clear) showed that the following exception was the cause of that fatal error:

```
ERROR   (2002-03-27) 23:21.40:190   [sitemap] (/cocoon/)
Thread-23/Handler: Error compiling sitemap
java.lang.NoClassDefFoundError:
org/apache/cocoon/generation/AbstractGenerator
        at java.lang.ClassLoader.defineClass0(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java, Compiled Code)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java,
Compiled Code)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java, Compiled Code)
        at java.net.URLClassLoader.access$100(URLClassLoader.java, Compiled Code)
        at java.net.URLClassLoader$1.run(URLClassLoader.java, Compiled Code)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java, Compiled Code)
        at java.lang.ClassLoader.loadClass(ClassLoader.java, Compiled Code)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java, Compiled Code)
        at java.lang.ClassLoader.loadClass(ClassLoader.java, Compiled Code)
        at
```

```
org.apache.tomcat.loader.AdaptiveClassLoader.loadClass(AdaptiveClassLoader.java,
Compiled Code)
        at java.lang.ClassLoader.loadClass(ClassLoader.java, Compiled Code)
        at java.lang.ClassLoader.loadClass(ClassLoader.java, Compiled Code)
        at org.apache.cocoon.util.ClassUtils.loadClass(ClassUtils.java, Compiled
Code)
        at
org.apache.cocoon.sitemap.AbstractSitemap.load_component(AbstractSitemap.java,
Compiled Code)
        at
org.apache.cocoon.www.sitemap_xmap$Configurer.configGenerators(sitemap_xmap.java,
Compiled Code)
        at org.apache.cocoon.www.sitemap_xmap.configure(sitemap_xmap.java, Compiled
Code)
        at
org.apache.avalon.excalibur.component.DefaultComponentFactory.newInstance(DefaultComponentFactory
Compiled Code)
        at
org.apache.avalon.excalibur.component.ThreadSafeComponentHandler.initialize(ThreadSafeComponentHa
Compiled Code)
        at
org.apache.cocoon.components.language.generator.GeneratorSelector.addGenerator(GeneratorSelector.
Compiled Code)
        at
org.apache.cocoon.components.language.generator.ProgramGeneratorImpl.addCompiledComponent(Program
Compiled Code)
        at
org.apache.cocoon.components.language.generator.ProgramGeneratorImpl.generateResource(ProgramGene
Compiled Code)
        at
org.apache.cocoon.components.language.generator.ProgramGeneratorImpl.createResource(ProgramGenera
Compiled Code)
        at
org.apache.cocoon.components.language.generator.ProgramGeneratorImpl.load(ProgramGeneratorImpl.ja
Compiled Code)
        at org.apache.cocoon.sitemap.Handler.run(Handler.java, Compiled Code)
        at java.lang.Thread.run(Thread.java:484)
```

Puzzled by this error, we mailed to the cocoon-users mailinglist (cocoon-users@xml.apache.org) and explained our situation. The answer we received was to put our generator in the *$TOMCAT_HOME/webapps/cocoon/WEB-INF/classes/*. We stopped Tomcat, emptied the work-directory, removed the directory */home/erwin/cocoon2/generator/* from the classpath and made a directory *test/* under the *$TOMCAT_HOME/webapps/cocoon/WEB-INF/classes/* and placed **MyGenerator.class** in that directory. We then restarted Tomcat and once again tried to access *http://localhost:8080/cocoon/mygenerator.xml*. But after making that request in our browser, we got a message from the browser saying that the server could not be reached. Looking at the xterm from which we started Tomcat, we saw the following error:

```
IGSEGV   11*  segmentation violation
si_signo [11]: SIGSEGV   11*  segmentation violation
si_errno [0]: Success
si_code [128]: unknown siginfo
sc_pc: 0x20010164f08, r26: 0x200001e19e0

thread pid: 26157
stackpointer=0x3fffc5f69b8

Full thread dump Classic VM (1.3.1-1, native threads):
...
...
```

Removing our class (and commenting out our changes in the sitemap for safety) would resolve the problem, but then we can't use our generator.

Somewhere on the Web we had read a mail from someone who was also having **NoClassDefFoundError**s that he was able to solve by unzipping all the jar-files (a jar is

basically a zip file containing the compiled classes) from *$TOMCAT_HOME/webapps/cocoon/WEB-INF/lib/* into the *$TOMCAT_HOME/webapps/cocoon/WEB-INF/classes/* directory. We stopped Tomcat, emptied the work-directory and started Tomcat again.

After restarting Tomcat we had our hopes up that this time it would work. We also started our browser and tried to access *http://localhost:8080/cocoon/mygenerator.xml*, again. After waiting a while (Cocoon 2 had to recompile its sitemap and some other components) we got the see our XML file. Cocoon 2 produced the following XML document:

Voorbeeld E.4.

```
<?xml version="1.0" encoding="UTF-8"?>
 <doc> My first Cocoon 2 generator! </doc>
```

So, after a bit of struggling, we finally succeeded in deploying our own generator.

E.3.2.3. Considerations afterwards

After seeing our example and having some experience with Cocoon 2 one might ask why we reinvented the wheel by instantiating a parser and not using the one provided by Cocoon 2. It is evident that a start of a pipeline is a generator that fires SAX events, there must be a SAXParser available throughout Cocoon 2 that can be easily accessed. This is in fact the case. There are a number of reasons why we had not chosen that approach the first time around:

- Limited knowledge of the whole underlying architecture, not really enhanced by the documentation.
- We wanted to keep the time-to-test as short as possible, so we didn't spend time finding this information in the source in the first phase.
- We didn't see any other possibility of testing our code before we tried to integrate it with the Cocoon 2 project.

We would still like to point the reader to an alternative solution, i.e. the solution that is used throughout Cocoon 2. We will give the code fragments here and we will then explain what it does.

Voorbeeldcode E.5.

```
...

import org.apache.avalon.excalibur.xml.Parser;

...

Parser parser = null;

try {
    parser = (Parser)this.manager.lookup(Parser.ROLE);
    parser.parse(this.getInputSource(),handler);
} catch (SAXException e) {
   // Preserve original exception
   throw e;
} catch (Exception e) {
   throw new ProcessingException("Exception during processing of " + this.getSystemId(),e);
} finally {
  if (parser != null) {
       this.manager.release(parser);
  }
}
```

```
...
```

An extra **import** statement is added. The **Parser** interface of the Avalon/Excalibur project [AVEXJA] defines the following method:

- `void` **`parse`** `(InputSource` **`in`**`, ContentHandler` **`consumer`**`) throws` **`SAXException`**`,` **`IOException`**`;` : the implementation of this method should parse the **InputSource** and send the SAX events to the **consumer**. The consumer can be an **XMLConsumer** or an object that implements **LexicalHandler** as well.

This interface defines a variable **ROLE** of the type String that is given the value **org.apache.avalon.excalibur.xml.Parser**. This variable is used to ask the **ComponentManager**, which is accessed by **this.manager**, to **lookup** a **Component** that has that role. The returned Component is then casted to a **Parser** type. We can then apply the parse method to any **org.xml.sax.InputSource** object and to an object that implements the **ContentHandler** interface. Finally, we have to tell the ComponentManager that we are finished using the parser. This allows the ComponentManager to handle the End-Of-Life Lifecycle events associated with this Component.

NOTE: if you want to use this method to obtain a parser, it would be better to extend the **ComposerGenerator** class, instead of the **AbstractGenerator** class. The ComposerGenerator is defined to make use of a **ComponentManager**, while this is not the case for the **AbstractGenerator** class. You should envisage the given code as part of a class that extends the **ComposerGenerator** class or one of its children.

E.3.3. Going the distance

We have succeeded in implementing a first test to find out how everything works, but a generator that only sends a fixed string to Cocoon 2 is not that interesting. Since we have written an application that can serve XML documents contained in a String object (using JDOM [JDOM]), we want to be able to retrieve these documents through our browser, which sends this request to Cocoon 2. Cocoon 2 then fires up our generator to retrieve the requested XML document and can start the pipeline for processing that document.

Since we had experimented with Java RMI in one of our courses, we decided to try a setup where our generator was a client for the document server and the communication would happen via RMI. For this section, we will first look at setting up the server, next we will look at accessing the server from within MyGenerator and finally we will put it all together. If we get this to work, we then can ponder about looking up parameters defined in the sitemap to use in MyGenerator. We used [RMIGS] as a basis for getting started with RMI. If you have never used RMI, we recommend that you read this document to develop a basic understanding of working with RMI.

E.3.3.1. Setting up a RMI server

After reading the document [RMIGS] and having deployed the example, we started writing our own interface, called **Serverfunctions** that defines the methods that should be implemented by a program that wishes to serve as a server for **MyGenerator**. This interface looks like this:

Voorbeeldcode E.6. Interface ServerFunctions.java

```
package test;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ServerFunctions extends Remote {
  /**
    This method returns a String, containing a well-formed XML fragment/document. This
    String contains information about the application implementing this interface. Choosing
    what information is put into this String is left to the application designer.
  */
   String sayHello () throws RemoteException { }

  /**
    This method returns a String, containing a well-formed XML fragment/document. To
    determine the information that should be returned, a systemId is passed to this method.
  */
   String getResource (String systemId) throws RemoteException { }

}
```

This interface defines two methods that should be implemented. Since these methods can be invoked via RMI we must declare that these methods can throw a RemoteException. These methods should return well-formed XML, as specified.

With interfaces alone we cannot build an application. We also must have a class that implements this interface. The following example demonstrates how this can be implemented. We used JDOM [JDOM] for reading in a XML document and converting it to a String.

Voorbeeldcode E.7. Class Server.java

```
package test;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;
import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import test.ServerFunctions;

public class Server extends UnicastRemoteObject implements ServerFunctions {
  Document doc = null;

  public Server () throws RemoteException {
      super();
  }

  public String sayHello () throws RemoteException {
      return "<doc>My First RMI Server!</doc>";
  }

  public String getResource (String systemId) throws RemoteException {
      try {
          SAXBuilder sb = new SAXBuilder();
          Document newdoc = sb.build(systemId);
          return ( new XMLOutputter() ) .outputString(newdoc);
      } catch (JDOMException jde) {
          System.out.println("JDOM error: " + jde.getMessage());
          jde.printStackTrace();
          // Rethrow the exception so the other
          // side knows something is wrong
          throw new RemoteException("JDOMException while processing " + systemId,jde);
      }
  }

  public void main (String args[]) {
```

```
        // Create and install a security manager
        // For testing purposes only, set this to null
        System.setSecurityManager(null);
        try {
            Server obj = new Server();
            // Bind this object instance to the name "MyServer"
            Naming.rebind("MyServer",obj);
            System.out.println("MyServer bound in registry");
        } catch (Exception e) {
            System.out.println("Server error: " + e.getMessage());
            e.printStackTrace();
        }
    }

}
```

We first have the necessary import-statements. This class implements the **ServerFunctions** interface we defined before. We also extend the **UnicastRemoteObject**. The Java API docs ([JAVA13APID]) tell us the following about UnicastRemoteObject: *"The UnicastRemoteObject class defines a non-replicated remote object whose references are valid only while the server process is alive. Objects that require remote behavior should extend RemoteObject, typically via UnicastRemoteObject."* This allows us, by calling the constructor of this superclass, to use the behavior of the UnicastRemoteObject for our RMIServer. This is typically done by calling the **super()** constructor in the constructor of our class.

Next, we have the implementation of the two methods defined in our interface. The **sayHello** method just returns a string representing the following XML fragment:

Voorbeeld E.5.

```
<doc> My First RMI Server! </doc>
```

We then also implement the **getResource** method. In the body of the try-block we first build a JDOM Document using the given systemId. This means that an XML file, at the given location, is read and a JDOM Document object is created. Next, we use the method **outputString(Document doc)** of the **XMLOutputter** class to convert the JDOM Document to a string. It is this string that is returned to the client. In the event that there may be an error building the document, a **JDOMException** is thrown. If this is the case, we print the info to stdout and rethrow the exception, encapsulated in a RemoteException.

We then only need a **main** method to have a Java application at hand. The first thing we do is disabling the **SecurityManager**. For security reasons, this should only be done only for testing purposes on an isolated system and in production environments. We did this so we could bind this server in the rmiregistry without rewriting any Java policy files. Next, we make a new **Server** object and bind this in the rmiregistry, where it is associated with the name **MyServer**. We end with printing out a line that we have bound this object in the rmiregistry.

E.3.3.2. Setting up a RMI client

The next step in the process is to implement a Java application that can connect to our RMI server and invoke its methods. Once again, we will first give our code and then explain what it does.

Voorbeeldcode E.8. Class Client.java

```
package test;

import java.rmi.Naming;
import java.rmi.RemoteException;
import test.ServerFunctions;

public class Client {
  public void main (String args[]) {
      String message = "blank";

      try {
          // "obj" is the identifier that we'll use to refer
          // to the remote object that implements the
          // "ServerFunctions" interface
          ServerFunctions obj = (ServerFunctions)Naming.lookup("//myhost.com/MyServer");
          message = obj.sayHello();
          System.out.println(message);
          message = obj.getResource("index.xml");
          System.out.println(message);
      } catch (Exception e) {
          System.out.println("Server exception: " + e.getMessage());
          e.printStackTrace();
      }
  }

}
```

Our client only defines a **main** method. We first initialize the variable, to which we will assign the return value of the **sayHello** method. Next, we try to **lookup** an object that is bound to *//myhost.com/MyServer* (note that myhost.com is a random chosen example). The lookup method returns an object, that is casted to the **ServerFunctions** type. We then invoke the sayHello method on the object and we print this message out. We also invoke the **getResource** method and print the result out. If this succeeds, we know everything works correctly. If an exception occurs, we print out the message from this exception plus its stack trace.

E.3.3.3. Testing the RMI components

We will first test if the RMI communication works. If it doesn't work there is no point in trying to integrate RMI communication in MyGenerator. Located in the directory */home/erwin/cocoon2/generator/*, which has the subdirectory *test/* containing our files, we execute the following commands:

```
[erwin generator]$ javac -classpath .:jar/jdom.jar:jar/xerces.jar -d compiled/
test/*.java
[erwin generator]$ rmic -classpath .:jar/jdom.jar:jar/xerces.jar -d compiled/
test.Server
[erwin generator]$ rmiregistry &
[erwin generator]$ java -classpath compiled/:jar/jdom.jar:jar/xerces.jar
-Djava.rmi.server.codebase=http://myhost.com/~erwin/cocoon2/generator/compiled/
test.Server
MyServer bound in registry
```

If you forget to define the **java.rmi.server.codebase** system property or give it a wrong value, you are most likely to get the following exception:

```
HelloImpl err: RemoteException occurred in server thread; nested exception is:
        java.rmi.UnmarshalException: error unmarshalling arguments; nested
exception is:
        java.lang.ClassNotFoundException: test.Server_Stub
java.rmi.ServerException: RemoteException occurred in server thread; nested
```

```
exception is:
        java.rmi.UnmarshalException: error unmarshalling arguments; nested
exception is:
        java.lang.ClassNotFoundException: test.Server_Stub
java.rmi.UnmarshalException: error unmarshalling arguments; nested exception is:
        java.lang.ClassNotFoundException: test.Server_Stub
java.lang.ClassNotFoundException: test.Server_Stub
        at
sun.rmi.transport.StreamRemoteCall.exceptionReceivedFromServer(StreamRemoteCall.java,
Compiled Code)
        at sun.rmi.transport.StreamRemoteCall.executeCall(StreamRemoteCall.java,
Compiled Code)
        at sun.rmi.server.UnicastRef.invoke(UnicastRef.java, Compiled Code)
        at sun.rmi.registry.RegistryImpl_Stub.rebind(Unknown Source)
        at java.rmi.Naming.rebind(Naming.java, Compiled Code)
        at test.Server.main(Server.java, Compiled Code)
```

We now can start the client to test if everything works. Notice that the resource requested in the code is in fact a relative URI. It is relative to the path from where we started the server application. The file index.xml contains the following information:

Voorbeeld E.6.

```
<?xml version="1.0"?>
  <document>
     <title> This is a document </title>
     <para> This is the first paragraph. </para>
  </document>
```

The client is started with the following command:

```
[erwin generator]$ java -classpath compiled/ test.Client
```

This resulted in the following output:

```
<doc>My First RMI Server!</doc>
<?xml version="1.0" encoding="UTF-8"?>
<document>
        <title>This is a document</title>

        <para>This is the first paragraph.</para>
</document>
```

This is exactly the output we expected, except for the encoding attribute. But this is something that is added by JDOM.

NOTE: we would like to conclude this section with a final note about the RMI server application. If you wish to deploy an RMI server application in the real world, you may wish to delete the code that disables the SecurityManager. If no other settings are changed, you may get the following error when starting your server application (depending on the configuration in your **java.policy** file):

```
HelloImpl err: access denied (java.net.SocketPermission 127.0.0.1:1099
connect,resolve)
java.security.AccessControlException: access denied (java.net.SocketPermission
127.0.0.1:1099 connect,resolve)
        at
java.security.AccessControlContext.checkPermission(AccessControlContext.java,
Compiled Code)
        ...
        at test.Server.main(Server.java, Compiled Code)
```

The most likely reason is that the default policy does not permit your server to bind its name in the rmiregistry. You have to change the security policy specified in the *$JAVA_HOME/jre/lib/security/java.policy* file. Since we are no experts in security we cannot

give you any advice in this matter, but a general advice in security related matters is that you are better safe then sorry.

E.3.3.4. Putting the pieces together

We now have been able to setup a generator and use RMI communication, now it is time to integrate these two pieces so we have a fully blown RMIGenerator for Cocoon 2. But before we do that, we will look how we can access the parameters and source that are passed from the sitemap to MyGenerator.

We have seen that the method **setup** is implemented in the **AbstractGenerator** class. One of the arguments of this method is String src ,. The value of the **src** attribute in the sitemap is passed via this argument and the variable **source** will be assigned this value. If for instance the following is a small part of the sitemap:

Voorbeeld E.7.

```
...
 <map:match pattern="mygenerator.xml">
   <map:generate type="mygenerator" src="example.xml"/>
   <map:serialize type="xml"/>
 </map:match>
...
```

If we request *$FULL_URL_PATH/mygenerator.xml*, the value of the **src** attribute will be passed to **MyGenerator** using the setup method. This value, **example.xml** can then be accessed via the **this.source** variable in our code.

As for now, we still have hardcoded in MyGenerator to which RMI server our generator should connect and also which bindname should be looked up. This is not desirable, we wish to have a configurable generator. *"Compile once, run many"* is maybe the way you could describe this. We wish to pass these values as parameters to the generator. Clearly, these values should be specified in the sitemap. Amongst the elements allowed in the sitemap there is a **parameter** element. If we want to use this element to pass parameters to our generator this element has to appear as a child of the **generate** element. Our sitemap fragment will then look like this:

Voorbeeld E.8.

```
...
 <map:match pattern="mygenerator.xml">
   <map:generate type="mygenerator" src="example.xml">
     <map:parameter name="host" value="myhost.com"/>
     <map:parameter name="port" value="1099"/>
     <map:parameter name="bindname" value="MyServer"/>
   </map:generate>
   <map:serialize type="xml"/>
 </map:match>
...
```

We define three parameters:

- **host**: tells the generator at which host the RMI server application is running. **REQUIRED**.

- **port**: tells the generator at which port at the remote host the rmiregistry process is running. If no value is specified Java uses the default port (1099). **OPTIONAL**.
- **bindname**: tells the generator which name should be looked up in the remote registry to obtain access to the RMI server object. **REQUIRED**.

We only need these three parameters to define the remote server object. We do not need to specify which methods should be invoked since we demand that a remote server implements the **ServerFunctions** interface. This is something that may be considered in the future.

We now have defined the host, port and bindname parameters, but how can we access the value of these parameters in our code? The setup method has an argument Parameters par ,. It is via this argument that the parameters defined in the sitemap will be passed to the generator. This argument will be assigned to the **parameters** variable defined in AbstractGenerator. To obtain the value of each parameter we can invoke the following method on the parameters variable. `public java.lang.String` **`getParameter`** `(java.lang.String` **`name`**`) throws` **`ParameterException`**`;` This method returns the value of the specified parameter, or throws an exception if there is no such parameter.

With all this in mind, we can finally build our configurable RMIGenerator. Also, this time we are going to extend the **ComposerGenerator** instead of the **AbstractGenerator** class. This way, we can make use of the **ComponentManager** to obtain a SAXParser.

At this moment we decide that if there is no value given to the src attribute in the sitemap (**source is null**), we will invoke the **sayHello** method and otherwise the getResource with the appropriate parameter. When the value of the src attribute is the **empty string**, the **getResource** method is invoked, so this should be **handled by the RMI server application**. After a little bit of thinking about how to code all this, we eventually wrote the following generator:

Voorbeeldcode E.9. test/MyGenerator.java

```
package test;

// import the necessary classes from the java.io package
import java.io.IOException;
import java.io.StringReader;
// import the necessary classes from the java.rmi package
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
// import the necessary SAX classes
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
// import of the classes used from Cocoon 2
import org.apache.cocoon.ProcessingException;
import org.apache.cocoon.generation.ComposerGenerator;
// import of the classes from the
// Avalon Framework
import org.apache.avalon.framework.parameters.Parameters;
import org.apache.avalon.framework.parameters.ParameterException;
import org.apache.avalon.framework.component.ComponentException;
// needed for obtaining parser in Cocoon
import org.apache.avalon.excalibur.xml.Parser;
import test.ServerFunctions;

public class MyGenerator extends ComposerGenerator {
  public void generate () throws IOException, SAXException, ProcessingException {
      String host;

      // lookup parameter 'host'
      try {
          host = parameters.getParameter("host");
```

```
          // test if host is not the empty string
        if (host == "") {
              throw new ParameterException("The parameter 'host' may not be the empty
              string");
        }
      } catch (ParameterException pe) {
          // rethrow as a ProcessingException
          throw new ProcessingException("Parameter 'host' not specified",pe);
      }
      String bindname;

      // lookup parameter 'bindname'
      try {
          bindname = parameters.getParameter("bindname");
          // test if bindname is not the empty string
        if (bindname == "") {
              throw new ParameterException("The parameter 'bindname' may not be the empty
              string");
        }
      } catch (ParameterException pe) {
          // rethrow as a ProcessingException
          throw new ProcessingException("Parameter 'bindname' not specified",pe);
      }
      String port = "";

      // lookup parameter 'port'
      try {
          port = parameters.getParameter("port");
          port = ":" + port;
      } catch (ParameterException pe) {
          // reset port to the empty string
          // port is not required
          port = "";
      }
      try {
          ServerFunctions obj = (ServerFunctions)Naming.lookup("//" + host + port + "/" +
          bindname);
          String message = "";

          // determine the method to invoke
          // depending on value of source
        if (this.source == null) {
              message = obj.sayHello();
        } else {
              message = obj.getResource(source);
        }
          Parser parser = null;

          parser = (Parser)this.manager.lookup(Parser.ROLE);
          InputSource inputSource = new InputSource(new StringReader(message));
          parser.parse(inputSource,super.xmlConsumer);
      } catch (NotBoundException nbe) {
          throw new ProcessingException("Error looking up the RMI server application",nbe);
      } catch (ComponentException ce) {
          throw new ProcessingException("Error obtaining a SAXParser",ce);
      }
  }

}
```

Since we have already explained every step that happens in this generator, we are confident that everyone will understand the code. We are now ready to deploy this generator.

E.3.3.5. The final step: deployment

We can now compile our classes and put the generator, along with the ServerFunctions interface, in the right place. For compiling, we used the following command:

```
[erwin generator]$ javac -classpath
.:jar/xerces.jar:jar/cocoon.jar:jar/framework.jar:jar/excalibur.jar:jar/exc-scratchpad.jar
-d compiled/ test/ServerFunctions.java test/MyGenerator.java
```

where xerces.jar is a symbolic link to *$TOMCAT_HOME/webapps/cocoon/WEB-INF/lib/xercesImpl-2.0.0.jar*, framework.jar to *$TOMCAT_HOME/webapps/cocoon/WEB-INF/lib/avalon-framework-4.1.2.jar*, excalibur.jar to *$TOMCAT_HOME/webapps/cocoon/WEB-INF/lib/avalon-excalibur-4.1.jar* and exc-scratchpad.jar to *$TOMCAT_HOME/webapps/cocoon/WEB-INF/lib/avalon-excalibur-scratchpad-20020212.jar*. If your platform does not allow the use of symbolic links, you should use the complete path to the corresponding jar-files.

Now that these classes are compiled we can place them in the *$TOMCAT_HOME/webapps/cocoon/WEB-INF/classes/* directory as before. Now all that is left is shutting down Tomcat/Cocoon, emptying the work directory, modifying the sitemap, setting up a RMI server application and starting Tomcat/Cocoon.

E.4. Future plans

The first version of this generator was written as a proof-of-concept. The latest version (as given here, extending the ComposerGenerator) only foresees in the **generate** method. There are a number of plans we still have to extend the functionality and thus usability of this generator:

- allow passing of a (J)DOM document instance as a resource to our generator. JDOM does require an additional entry in the classpath.
- supply a possibility for caching documents
- if the RMI server application can generate SAX events, try to pass the xmlConsumer to the server application as the ContentHandler

These are some of the extensions we have in mind for this generator. Our goal is to complete these steps within a few weeks (it will probably be a bit longer since the deadline of our thesis is only three weeks a way at the time of writing).

Overview of articles read as a study into XML/XSLT

Title:	**A Brief Introduction to XSL**	
Contents:	An introduction to XSL based on the example of the Docbook stylesheets	
Author:	Bob Stayton	
Review:	**Read by:**	Carolien Coenen
	Review:	Interesting article because it contains lot's of examples from a real life application. The stylesheets of DocBook are used for illustration. Particularly interesting because, in the begin stage of our thesis, we studied DocBook.
	Rating:	9

Title:	**A gentle guide to DocBook**	
Contents:	This article explains what DocBook is and how to create a simple document using DocBook.	
Author:	Joe "Zonker" Brockmeier	
Review:	**Read by:**	Carolien Coenen
	Review:	Like mentioned in the title, this article is a very gentle introduction to DocBook. If you don't know anything about DocBook, this is a good starting point. The article explains the most common Docbook tags and how to use SGML-tools Lite to to parse the document and make HTML, PostScript, plain-text, and PDF versions of the document. It also contains some useful links for a deeper study of DocBook.
	Rating:	8

Title:	**An Introduction to XSL**	
Contents:	The article explains briefly what XSL is and makes the link whit DSSSL and CSS	
Author:	Henry S. Thompson	
Review:	**Read by:**	Carolien Coenen
	Review:	Old article (from 1997) that shows some basics about XSL, only interesting because it shows the influences of DSSSL and CSS on XSL.
	Rating:	5

Title:	**An XML Wish List**	
Contents:	The author composes a wishlist with ten features to make XML development a pleasure	
Author:	Russel Jones	
Review:	**Read by:**	Carolien Coenen
	Review:	Interesting to have a look at what a known developer finds important for improving future XML development. Apparently we are not the only people to experience some problems with entities. :-)

	Rating:	6

Title:	**A technical introduction to XML**	
Contents:	A basic overview of XML on a technical level	
Author:	Norman Walsh	
Review:	**Read by:**	Erwin Hermans
	Review:	As mentioned in the title, a technical view onto XML. The goals of XML are examined with a closer look on how this is technically specified, with links to the necessary XML specifications. Although this article is written in the winter of 1997 (with an update about a year late), it is still a good introduction to get to know the technique behind XML.
	Rating:	7
Review:	**Read by:**	Carolien Coenen
	Review:	A cleary written article, that discusses XML in a more technical way (the specifications behind it). It was quite interesting, but the last part of the article about links in XML was sometimes difficult to understand. So if you are looking for information on this subject, you should search for an other article.
	Rating:	7

Title:	**DevGuru XSLT Quick Reference guide**	
Contents:	A reference source for all of the elements and functions that compose the eXtensible Stylesheet Language Transformation (XSLT) language version 1.0.	
Author:	Infinite Software Solutions	
Review:	**Read by:**	Carolien Coenen
	Review:	The Reference guide gives a quick overview of some important XSLT elements and XPath functions. Although being far from complete, this reference guide describes in a clear way all the elements and functions it does contains. The description of the elements is very thourough (possible attributes they might contain, and their attribute values) and the examples are clear. To bad that the examples have no indentation, because of this you need more time to comprehend the structure of the stylesheets. The functions are sometimes rather briefly explained. May come in handy if you want to lookup something fast. It's a pity it doesn't contain all the possible elements en functions.
	Rating:	7

Title:	**DTD Tutorial**
Contents:	Tutorial that covers the basic principles of DTDs
Author:	Refsnes Data

Review:	**Read by:**	Carolien Coenen
	Review:	If you don't know anythins about DTD's, this is a very good tutorial. In a few steps the most important things-to-know-about-DTD are explained. The tutorial ends with a few short examples to give you an idea how an actual DTD might look like.
	Rating:	8

Title:	**Easy Java/XML integration with JDOM, Part 1**	
Contents:	In this article, the first of two parts, Hunter and McLaughlin (the creators of JDOM) explain how to use JDOM to read XML from an existing source.	
Author:	Jason Hunter	
Author:	Brett McLaughlin	
Review:	**Read by:**	Carolien Coenen
	Review:	Article written bij the creators of JDOM. Good stuff. Easy to understand, easy to read. Perfect explanation of JDOM.
	Rating:	9
Review:	**Read by:**	Erwin Hermans
	Review:	Very clear article on this API. It shows how easy-to-use this API is.
	Rating:	9

Title:	**Easy Java/XML integration with JDOM, Part 2**	
Contents:	In this second article the authors focus on how you can use JDOM to create and mutate XML.	
Author:	Jason Hunter	
Author:	Brett McLaughlin	
Review:	**Read by:**	Carolien Coenen
	Review:	The secon part of the article that is written by the creators of JDOM. Same comment as before. Nice article. Very useful if you want to learn JDOM.
	Rating:	9
Review:	**Read by:**	Erwin Hermans
	Review:	After this article, I was ready to begin working with JDOM. JDOM is very easy to use and if you have a basic understanding of Java en XML, you'll love this API.
	Rating:	9

Title:	**Enabling XML documents for globalization**	
Contents:	How to make content in multiple languages.	
Author:	Erich Magee	
Review:	**Read by:**	Erwin Hermans
	Review:	Short explanation of the disadvantages of the translation of one XML-document (with all the tags and attributes) to other languages. Discussion of a way to

		separate the translations of the XML-document. I found it a little disappointing though that it is only mentioned how the do this in Java and not in XSLT. (it is always useful to have an example).
	Rating:	6

Title:	**Formatting Objects**	
Contents:	Nederlandstalige uiteenzetting over Formatting Objects	
Author:	Bjorn Boxstart	
Author:	Ronald Walraven	
Review:	**Read by:**	Carolien Coenen
	Review:	Geeft een heel summier overzicht van de meest gebruikte vormgevingsobjecten. Ikzelf vond de uitleg te summier om echt goed door te hebben hoe XSL-FO juist ineen zit. Op het einde van het artikel wordt een heel kort overzicht gegeven van de FO ontwikkelingen op dit moment. Maar ook dit is heel summier verwoord. Niet echt een aanrader dus, wat mij betreft.
	Rating:	6

Title:	**Microsoft's XML Gamble**	
Contents:	Comment on Microsoft's XML politics	
Author:	Elizabeth Montalbano	
Review:	**Read by:**	Carolien Coenen
	Review:	This article discusses standards and the way in which large companies like Microsoft try to manipulate them. In this case it involves specifically the XML standard. The article is interesting, but nog very useful.
	Rating:	7

Title:	**Namespaces in XML**	
Contents:	W3C Recommendation on XML namespaces	
Author:	World Wide Web Consortium	
Review:	**Read by:**	Carolien Coenen
	Review:	A typical recommendation document. Goes into all the details. It is recommended that you are familiar with regular expressions for a language because they are used frequently in the document. Too much detail for someone who is looking for a quick overview.
	Rating:	7
Review:	**Read by:**	Erwin Hermans
	Review:	A technical recommendations explaining how to use namespaces in XML. You should be familiar with regular expressions and grammars because they are used to describe the namespaces. Good for devlopers
	Rating:	7

Title:	**Oral Presentation Advice**	
Contents:	Explains how to give a decent presentation before an audience	
Author:	Mark D. Hill	
Review:	**Read by:**	Carolien Coenen
	Review:	This article explains in a clear way how to deal with presentations. Very funny is the part on "How to Give a Bad Talk" by David Patterson. Very helpfull.
	Rating:	8

Title:	**Practical Transformation using XSLT en XPath (XSL Transformations and the XML Path Language)**	
Contents:	Transformation of XML documents using XSLT and XPath	
Author:	Crane Softwrighs Ltd.	
Review:	**Read by:**	Carolien Coenen
	Review:	Very schematic discussion of XML/XSL/XSLT. Rather difficult explanation in English that is of quite a high degree of difficulty. It was sometimes hard to keep track of the structure. This document is suited for people who have had their hands on the subject more than once. Does contain a few useful links though.
	Rating:	6

Title:	**Preliminary Design of JML: A Behavioral Interface Specification Language for Java**	
Contents:	This paper discusses the goals of JML, the overall approach, and describes the basic features of the language through examples.	
Author:	Gary T. Leavens	
Author:	Albert L. Baker	
Author:	Clyde Ruby	
Review:	**Read by:**	Carolien Coenen
	Review:	Long and a bit boring paper which discusses JML, a behavioral interface specification language tailored to Java. The paper is very theoretically, although it uses lot's of examples to illustrate it's theory. I found it hard to concentrate on it. The paper assumes you also have some knowlegde about Larch/C++, which the authors developed earlier, Eiffel and refinement calculus. They use references to Larch throughout the whole paper, which adds to the difficulty of it, when you are, like me, not familiar with Larch. I liked the object and example index though, might come in handy in the future.
	Rating:	6
Review:	**Read by:**	Erwin Hermans
	Review:	This paper present JML, a behavioral interface specification language tailored to Java. It is a long paper, that is somewhat difficult to understand, inspite of examples to illustrate it's theory. A lot of preliminary knowledge is required, and this makes this paper only

		suited for specialistst.
	Rating:	6

Title:	**Recurse, not divide, to conquer**	
Contents:	Why not divide an HTML document over multiple XSLT templates? And what is the right thing to do in that case.	
Author:	Benoît Marchal	
Review:	**Read by:**	Carolien Coenen
	Review:	A very specific article that encloses a mistake often made by starting XSLT programmers. It is always good to be warned. Very useful article with a large example.
	Rating:	8
Review:	**Read by:**	Erwin Hermans
	Review:	A good article written to demonstrate how one should use and not abuse XSLT. Important but simple article.
	Rating:	7

Title:	**Schemas**	
Contents:	Schemas, the difference with DTD and why is it better/worse than DTD?	
Author:	Elliotte Rusty Harold	
Review:	**Read by:**	Erwin Hermans
	Review:	This chapter of the book gives a good insight into the XML Schema Language. For me, I knew what a DTD was supposed to do, but I had no insight into DTD or into the XML scheme language. After reading this chapter, I understood really well what a DTD and a XML schema where supposed to do, and why a XML schema was better in many cases then a 'simple' DTD. However. I still don't know when you could better use a DTD then an XML schema (it is mentioned that it sometimes is better), but on the other hand, that wasn't the goal of this chapter.
	Rating:	9

Title:	**Simplify XML programming with JDOM**	
Contents:	Explains the advantages of JDOM and how to use it.	
Author:	Wes Biggs	
Author:	Harry Evans	
Review:	**Read by:**	Carolien Coenen
	Review:	Great article, fun to read. It gives a very clear view on the use of JDOM and explains the reasons for it's design, as well as the advantages of it. This articles contains lot's of examples which will help you learn how to use JDOM. A good starting point if you've never heard of JDOM before in your life.
	Rating:	9

Title:	**The Apache XML Project: Technologies used**	
Contents:	A short overview of the different technologies used by Cocoon.	
Author:	The Apache Software Foundation	
Review:	**Read by:**	Carolien Coenen
	Review:	Very good article that explains clearly the different technologies used by Cocoon: XML, XSLT, XSL:FO and XSP. The author of the article writes in a very humoristic way, which makes this reading experience very pleasant. With a minimum of examples he clearly shows the almost unlimited possibilities of XML/XSL. This article will not learn you how to use Cocoon of XML/XSL, but will make you understand better it's advantages.
	Rating:	8

Title:	**The Design of the DocBook XSL Stylesheets**	
Contents:	This paper explores some of the design issues confronted by the author in designing XSL stylesheets for DocBook	
Author:	Norman Walsh	
Review:	**Read by:**	Carolien Coenen
	Review:	Great article. Well written, good examples. Gives you a clear insight on how XSL works and which design issues are considered while developing stylesheets, in this case, the DocBook stylesheets.
	Rating:	9

Title:	**The Jakarta Project: Element Construction Set**	
Contents:	A short introduction to ECS	
Author:	The Apache Software Foundation	
Review:	**Read by:**	Carolien Coenen
	Review:	The article explains briefly what ECS is and what it does. It gives you a general idea of the possibilities. Good introduction.
	Rating:	7
Review:	**Read by:**	Erwin Hermans
	Review:	The article explains briefly what ECS is and what it does. ECS is very simalar to JDOM, but we haven't used it though. Although it now has matured, when we first looked at this page, everyting was still very much under construction.
	Rating:	7

Title:	**The XML Revolution, Technologies for the future Web**
Contents:	Tutorial that gives an introduction and overview of XML, Namespaces, XInclude, XML Base, XLink, XPointer, XPath, DTD, XML Schema, DSD, XSLT, XQuery, DOM, SAX, and JDOM
Author:	Anders Møller
Author:	Michael I. Schwartzbach

Review:	**Read by:**	Carolien Coenen
	Review:	Great tutorial! Covers almost all the issues that are important to us. I'd recommend this tutorial to anyone who wants to know more about XML and other related subjects
	Rating:	9

Title:	**Tutorial: Introduction to XML**	
Contents:	A basic introduction to XML	
Author:	Doug Tidwell	
Review:	**Read by:**	Erwin Hermans
	Review:	The difference between XML and HTML and why XML is considered to be better is discussed in a limited number of slides. The present possibilities of XML are briefly touched.
	Rating:	7
Review:	**Read by:**	Carolien Coenen
	Review:	General introduction to XML with some marketing for IBM products, it does describe in a very brief way the essence of XML. I would say: for dummies.
	Rating:	6

Title:	**Tutorial: Transforming XML documents**	
Contents:	3 parts: XML to HTML, XML to SVG, XML to PDF	
Author:	Doug Tidwell	
Review:	**Read by:**	Erwin Hermans
	Review:	Part 1: Both XSLT and Java are used (the easiest solution is given), XSLT is in lot's of cases easier. A lot of different techniques are used in the transformations, with in almost every part useful tips. Details that could optimize stylesheets are mentioned, but there is nog mentioning of a source and no examples are given. In part two the transformation of an XML document into a SVG is discussed, with a clear explanation on the used SVG elements. It is shown how to transform a text into an image, in this example it is only a sonnet and the positions are hard coded in the XSLT stylesheet. But the the more complex examples like cake and columncharts are also dealth with. Finally it is discussed how XML documents can be transformed tot PDF whit XSL FO. Including tips on how to offer this automatically by Cocoon on your webserver.
	Rating:	9

Title:	**W3C Tutorial**
Contents:	Tutorial that covers the basic principles of the W3C
Author:	Refsnes Data

Review:	**Read by:**	Carolien Coenen
	Review:	Interesting tutorial because it gives a brief explanation on what the W3C is and what it is working on. The tutorial gives an overview of all the different recommendations and how they were archived. I think it is interesting to have this kind of overview on the development of the standards. Of course I liked the part about XML en XSL the most.
	Rating:	8

Title:	**What are XPath Match Patterns?**	
Contents:	This chapter explains know to create match patterns in XSLT	
Author:	Steven Holzner	
Review:	**Read by:**	Carolien Coenen
	Review:	Gives an overview of the most important match patterns, but doesn't explain them all. Some extra explanation could be useful. This chapter is een extract from a book and you clearly miss the information provided in the earlier chapters.
	Rating:	6

Title:	**XLinks**	
Contents:	XLinks, what are they, how can we use them?	
Author:	Elliotte Rusty Harold	
Review:	**Read by:**	Erwin Hermans
	Review:	The difference between XLinks and HTML links is discussed and also the possible practical uses the author sees present in XLinks. Well structured and with lot's of examples.
	Rating:	9

Title:	**XML Basics**	
Contents:	The basic principles of XML	
Author:	Jan Egil Refsnes	
Review:	**Read by:**	Carolien Coenen
	Review:	Nice article, explains in short steps in a very comprehensible way what it is that XML does, the advantages and the future perspectives. A must for people who don't know anything about XML, especially because the article takes it one step at a time.
	Rating:	9

Title:	**XML for Data: Using XML Schema archetypes**
Contents:	Kevin Williams describes the benefits of using archetypes in XML Schema designs for data and provides some concrete examples. He discusses both simple and complex types, and some advantages of using each. Code samples in XML Schema are provided.

Author:	Kevin Williams	
Review:	**Read by:**	Erwin Hermans
	Review:	Good introductory article into using types in XML Schema. Not a big article, but a great help in learning to work with XML Schema.
	Rating:	8

Title:	**XML in 10 punten**	
Contents:	Deze samenvatting in 10 punten probeert genoeg basis-ideeen te bieden zodat een beginner door de bomen het bos kan zien.	
Author:	Bert Bos	
Review:	**Read by:**	Carolien Coenen
	Review:	Nederlandstalige tekst, die een beetje amateuristisch overkomt, het lijkt alsof de auteur zelf niet al te bekend is met zeker concepten ivm XML. Vond het niet zo interessant om te lezen en zou het ook niet aanraden. Bovendien heeft deze tekst ook last van het niet kunnen weergeven van entiteiten zoals ï en ë. Tsss. ;-)
	Rating:	5

Title:	**XML in a Nutshell: A Desktop Quick Reference: Chapter 9 Xpath**	
Contents:	A gentle introduction to the XPath language	
Author:	Elliotte Rusty Harold	
Author:	W. Scott Means	
Review:	**Read by:**	Erwin Hermans
	Review:	For me, this chapter was not that useful, because almost all of it is also mentioned in Chapter 17 of the XML Bible: Second Edition. Nonetheless, this is a very good written chapter, that gives a very good insight in the XPath Language and learns you how to you use it, even if you are a complete novice in this domain (you have to have a basic understanding of how XML/XSLT works). For others it is a good reference.
	Rating:	9

Title:	**XML Schema Tutorial**	
Contents:	Tutorial that covers the basic principles of XML Schema	
Author:	Refsnes Data	
Review:	**Read by:**	Carolien Coenen
	Review:	Good tutorial, covers the basic principles you need when you want to use XML Schema. Gives also a clear overview of the different xsd elements in the references sector.
	Rating:	8

Title:	**XML turorial**	
Contents:	This XML tutorial intends to provide an easy to unterstand introduction into XML programming.	
Author:	Shermin Voshmgir	
Review:	**Read by:**	Carolien Coenen
	Review:	Good tutorial explain what XML is and adresses some important issues as the constraints that apply to well-formed XML documents and the use of entities in your documents. It also adresses briefly XSL and XLL.
	Rating:	9

Title:	**XML Tutorial**	
Contents:	Tutorial that covers the basic principles of XML	
Author:	Refsnes Data	
Review:	**Read by:**	Carolien Coenen
	Review:	Very good tutorial. Covers all the principles of XML worth knowing and even discusses more advanced items for those who is interested. I'd sure recommend this to someone who is new to XML. Very complete. I also liked the quiz, because I got a perfect score. ;-)
	Rating:	9

Title:	**XPointers**	
Contents:	XPointers, what are they, how can we use them?	
Author:	Elliotte Rusty Harold	
Review:	**Read by:**	Erwin Hermans
	Review:	XPointers can be used in those area's where you want to point for example to a part or a particular element of a document without the possibility of putting an anchor in the source document (you've got for in-stancek only read-access to the source document).
	Rating:	9

Title:	**XSL-FO: state of the union, state of the art**	
Contents:	A brief overview of the essential aspects of the XSL-FO standard	
Author:	Karen Lease	
Review:	**Read by:**	Carolien Coenen
	Review:	Good article, clearly written with lot's of figures to clarify things. It gives you a good overview of the possibilities XSL-FO has to offer. A good introduc-tion if you want to start exploring XSL-FO
	Rating:	8

Title:	**XSLT & XPath Tutorial**
Contents:	Tutorial that covers the basic principles of XSL and XPath
Author:	Tracey Wilson

Review:	**Read by:**	Carolien Coenen
	Review:	Good tutorial, written by a woman. (I sure liked that...) Gives an overview of the most important techniques used by XSLT.
	Rating:	8

Title:	**XSL Transformations**	
Contents:	The transformation part of the eXtensible Stylesheet Language	
Author:	Elliotte Rusty Harold	
Review:	**Read by:**	Erwin Hermans
	Review:	For persons with basic knowledge of XML(or HTML, this is a gooed introduciton to XSL
	Rating:	9
Review:	**Read by:**	Carolien Coenen
	Review:	Very extensive overview of xsl features. Explained in a very clear fashion and documented with lot's of examples.
	Rating:	9

Title:	**XSL Transformations: Formatting Objects**	
Contents:	Formatting objects (FO) in the eXtensible Stylesheet Language	
Author:	Elliotte Rusty Harold	
Review:	**Read by:**	Erwin Hermans
	Review:	With basic knowlegde of XML (or HTML), this is a good introduction to XSL-FO
	Rating:	9
Review:	**Read by:**	Carolien Coenen
	Review:	This chapter of the XML bible gives a good overview of the most common objects en attributes used in XSL-FO. Although I've found quite some mistakes in the text, I found it very useful to get acquainted with the different features of XSL-FO. This article is a good starting point for getting to know XSL-FO if you have basic knowlegde of XML en XSLT. Documented with lot's of examples.
	Rating:	9

Title:	**XSL Tutorial**	
Contents:	Tutorial that covers the basic principles of XSL	
Author:	Refsnes Data	
Review:	**Read by:**	Carolien Coenen
	Review:	I found the tutorial itself a disappointing compared to the other W3schools tutorials I've read, because it covers only a small part of the possibilities that XSLT has to offer. I did like the references section though where all the possible XSLT elements and their attributes are described. Very useful (and also very complete) if you want to look up quickly how to use

		an XSLT element.
	Rating:	7

Title:	**XSL - What's in it for us?**	
Contents:		
Author:	Janus Boye	
Review:	**Read by:**	Carolien Coenen
	Review:	Short introduction to XSL. Very general explanation of the concepts. Perhaps useful if we want to know what XSL is whitout going into much detail.
	Rating:	7

Title:	**Java en XML**	
ISBN:	90 395 1718 5	
Contents:	The basics of XML, the use of SAX and DOM API's,design of document-types with CTC's and XML-Schema, writing programs for generating XML-data, XSLT, something about Apache Cocoon, etc.	
Review:	**Read by:**	Carolien Coenen
	Review:	Very nice book, I haven't read all the chapters yet, but it has proven it's use by helping me to install Co-coon2. It touches a lot of XML/XSL related stuff, I can surely recommend it because it also shows the links with Java.
	Rating:	9

Bibliografie

[AAR] Adobe Acrobat Reader
http://WWW.ADOBE.COM/products/acrobat/readstep.html

[AMMISXML] The XML Revolution, Technologies for the future Web door Anders Møller en Michael I. Schwartzbach
http://www.brics.dk/~amoeller/XML/xml/index.html

[APACHE] The Apache software foundation
http://www.apache.org/

[APFSXSLF] Antennehouse Professional Formatting Solutions ontwikkelt XSL Formatter 2.1 met PDF Output Option
http://www.antennahouse.com/

[APJSERV] The Apache JServ Project
http://java.apache.org/jserv/index.html

[APJSRVDIST] The Apache JServ Project distributions
http://java.apache.org/jserv/dist/

[APXMLP] The Apache XML Project
http://xml.apache.org/

[ASP] Application Server Pages
http://www.microsoft.com/asp/

[AT] Arbortext
http://www.arbortext.com/

[AVEXJA] Avalon/Excalibur van The Jakarta Project
http://jakarta.apache.org/avalon/excalibur/index.html

[BGJMSU] EMERGING TECHNOLOGIES, Accessibility and Web Design, Why Does It Matter? door Bob Godwin-Jones
http://llt.msu.edu/vol5num1/emerging/default.html

[BMAAJAXP] All about JAXP, Sun's Java API for XML Parsing, Brett McLaughlin
http://www-106.ibm.com/developerworks/library/x-jaxp/index.html

[BMCONDOM] Tip: Converting from DOM, When you need SAX or JDOM output from DOM, Brett McLaughlin
http://www-106.ibm.com/developerworks/xml/library/x-tipcdm.html?dwzone=xml

[BMJRJAXP] Sun's Java API for XML Parsing, Version 1.1, JAXP revisited, Brett McLaughlin
http://www-106.ibm.com/developerworks/xml/library/x-jaxp1.html

[C1DIST] Cocoon 1 distributions
http://xml.apache.org/cocoon1/dist/

[C1FAQCP] Frequently Asked Questions, chaining pipelines
http://xml.apache.org/cocoon1/faqs.html#faq-pidesign

[C1FAQNR] Frequently Asked Questions, no repository
http://xml.apache.org/cocoon1/faqs.html#faq-norepository

[C1FAQXO] Frequently Asked Questions, xsl:output
http://xml.apache.org/cocoon1/faqs.html#faq-xsloutput

[C1INST] Installing Cocoon
http://xml.apache.org/cocoon1/install.html

[C2DEVML] Cocoon 2 Developers mailinglist archive
http://marc.theaimsgroup.com/?l=xml-cocoon-dev&r=1&w=2

[C2DIST] Cocoon2 van The Apache XML project, Distribution site
http://xml.apache.org/cocoon/dist/

[C2INST] Cocoon2 van The Apache XML project, Installing Apache Cocoon
http://xml.apache.org/cocoon/installing/index.html

[C2SITEMAP] Cocoon 2, The Sitemap
http://xml.apache.org/cocoon/userdocs/concepts/sitemap.html

[C2USML] Cocoon 2 Users mailinglist archive
http://marc.theaimsgroup.com/?l=xml-cocoon-users&r=1&w=2

[CERN] European Organization for Nuclear Research
http://welcome.cern.ch

[COCOON1] Cocoon1 van The Apache XML project
http://xml.apache.org/cocoon1/

[COCOON2] Cocoon2 van The Apache XML project
http://xml.apache.org/cocoon/index.html

[CSKUL] Department of Computer Science
http://www.cs.kuleuven.ac.be/cwis-cs/frames/index-E.shtml

[CSSCOLOR] CSS Colors, door W3Schools.com
http://www.w3schools.com/css/css_colors.asp

[DB] DocBook.org
http://www.docbook.org/

[DB2HTML] docbook2html
http://linuxcommand.org/man_pages/docbook2html1.html

[DB2PDF] docbook2pdf
http://linuxcommand.org/man_pages/docbook2pdf1.html

[DB2PS] docbook2ps
http://linuxcommand.org/man_pages/docbook2ps1.html

[DEKTB] The TeX Book door Donald E. Knuth, ISBN: 0-201-13448-9
http://www.amazon.com/exec/obidos/ASIN/0201134489/qid%3D1009140180/ref%3Dsr%5F11%5F0%5F1/104-1406607-5017527

[DEVGURU] DevGuru XSLT Quick Reference guide, Copyright 1999-2002 by Infinite Software Solutions, Inc.
http://www.devguru.com/Technologies/xslt/quickref/xslt_intro.html

[DL] Datalogics
http://www.datalogics.com/

[DMSJDMU] XML in Java: Java document model usage, A look at how different XML document models work in Java, Dennis M. Sosnoski
http://www-106.ibm.com/developerworks/xml/library/x-injava2/index.html

[DSSSL] Document Style Semantics and Specification Language (DSSSL), International Organization for Standardization and International Electrotechnical Commission, ISO/IEC 10179:1996
http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=18196

[DTD2XS] dtd2xs, Translate a Document Type Definition (DTD) into a XML Schema (REC-xmlschema-1-20010502)
http://puvogel.informatik.med.uni-giessen.de/dtd2xs/

[DTDW] Building an XML application, step 1: Writing a DTD door Doug Tidwell
http://www-106.ibm.com/developerworks/xml/library/buildappl/writedtd.html

[DTDWFO] Part 3: Transforming XML into PDF door Doug Tidwell
http://www-106.ibm.com/developerworks/education/transforming-xml/xmltopdf/index.html

[DTXML] Tutorial: Introduction to XML door Doug Tidwell
http://www.xml101.com/xml/default.asp

[DVAPXML] Extending Apache Cocoon van The Apache XML Project
http://xml.apache.org/cocoon/developing/extending.html

[ERHSCHEMAS] Chapter 24 of the XML Bible, Second Edition : Schemas, Elliotte Rusty Harold
http://www.ibiblio.org/xml/books/bible2/chapters/ch24.html

[ERHXSL] XSL Transformations door Elliotte Rusty Harold
http://www.ibiblio.org/xml/books/bible2/chapters/ch17.html

[ERHXSLFO] XSL Transformations: Formatting Objects door Elliotte Rusty Harold
http://www.ibiblio.org/xml/books/bible2/chapters/ch18.html

[FOP] Apache's Formatting Objects Processor
http://xml.apache.org/fop/

[FOPUSML] The FOP Users mailinglist
http://marc.theaimsgroup.com/?l=fop-user&r=1&w=2#

[HSTED] An Introduction to XSL door Henry S. Thompson
http://www.ltg.ed.ac.uk/~ht/swindon.html

[IANA] Internet Assigned Numbers Authorithy
http://www.iana.org/

[IBMSQC] XML Schema Quality Checker
http://alphaworks.ibm.com/tech/xmlsqc

[IE] Microsoft© Internet Explorer
http://www.microsoft.com/windows/ie/default.asp

[IEEE754] IEEE 754: Standard for Binary Floating-Point Arithmetic
http://grouper.ieee.org/groups/754/

[INTROC2] Introduction to Cocoon 2
http://www6.software.ibm.com/developerworks/education/x-cocoon/index.html

[IS03166] IS0-3166 extract, Codes for representation of names of countries, Aug 1988
http://palimpsest.stanford.edu/lex/iso3166.html

[ISO693] Code for the representation of names of languages, International Organization for Standardization, ISO 693:1988
http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=4766&ICS1=1&ICS2=140&ICS3=20

[JAKTOM] Cocoon2 van The Apache XML project, Distribution site
http://xml.apache.org/cocoon/dist/

[JAVA] The source for Java™ technologie van Sun Microsystems, Inc
http://java.sun.com/

[JAVA13APID] Java 2 Platform, SE v1.3 API documentation
http://java.sun.com/j2se/1.3/docs/api/index.html

[JAVADOC] JAVADOC TOOL HOME PAGE Copyright Sun Microsystems, Inc.
http://java.sun.com/j2se/javadoc/

[JAXP] Java™ API for XML Processing (JAXP) van Sun Microsystems, Inc
http://java.sun.com/xml/jaxp/index.html

[JAXPFAQ] Java™ API for XML Processing (JAXP), Frequently Asked Questions
http://java.sun.com/xml/jaxp/faq.html

[JAXPSPEC] Java™ API for XML Processing (JAXP), Version 1.1, Specification
ftp://ftp.java.sun.com/pub/jaxp/89h324hruh/jaxp-1_1-spec.ps

[JBDW] A gentle guide to DocBook door Joe Brockmeier
http://www-106.ibm.com/developerworks/library/l-docbk.html

[JCW3] Comparison of SGML and XML, World Wide Web Consortium Note 15-December-1997 door James Clark
http://www.w3.org/TR/NOTE-sgml-xml

[JDOM] JDOM.org
http://www.jdom.org

[JDOMDOC] JDOM API Documentation
http://www.jdom.org/docs/apidocs/

[JERXML] XML Basics door Jan Egil Refsnes
http://www.xml101.com/xml/default.asp

[JHBM2JW] Easy Java/XML integration with JDOM, Part 2 door Jason Hunter and Brett McLaughlin
http://www.javaworld.com/javaworld/jw-07-2000/jw-0728-jdom2.html

[JHBMJW] Easy Java/XML integration with JDOM, Part 1 door Jason Hunter and Brett McLaughlin
http://www.javaworld.com/javaworld/jw-05-2000/jw-0518-jdom.html

[JNOME] jnome door Jan Dockx, Nele Smeets en Kristof Mertens
www.jnome.org

[JSDK] JAVA™ SERVLET TECHNOLOGY, The Power Behind the Server
http://java.sun.com/products/servlet/index.html

[JSDKARCH] JAVA™ SERVLET TECHNOLOGY, IMPLEMENTATIONS & SPECIFICATIONS ARCHIVE
http://java.sun.com/products/servlet/archive.html

[JSDKDL] JAVA™ SERVLET TECHNOLOGY, IMPLEMENTATIONS & SPECIFICATIONS
http://java.sun.com/products/servlet/download.html

[KLFO] XSL-FO: state of the union, state of the art door Karen Lease
http://www.idealliance.org/papers/xml2001/papers/html/03-05-06.html

[MIME] MIME Media Types, The Internet Corporation for Assigned Names and Numbers
http://www.iana.org/assignments/media-types/

[MMW3] XML Query, W3C overzicht door Massimo Marchiori
http://www.w3.org/XML/Query

[MS] Microsoft.com
http://www.microsoft.com

[MSXML] What's New in the Microsoft XML Parser Version 3.0 Release © 2002 Microsoft Corporation
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmsxml/html/xmlparser.asp

[MSXMLCS] Microsoft XML Core Services 4.0 SP1
http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/001/766/msdncompositedoc.xml

[MW754] The IEEE Format door The Mathworks, Inc.
http://www.mathworks.com/access/helpdesk_r11/help/toolbox/fixpoint/c3_bev13.shtml

[NMDB] The Design of the DocBook XSL Stylesheets door Norman Walsh
http://nwalsh.com/docs/articles/dbdesign/

[NOTEDCD] Document Content Description for XML, Submission to the World Wide Web Consortium 31-July-1998
http://www.w3.org/TR/NOTE-dcd

[NOTEDDML] Document Definition Markup Language (DDML) Specification, Version 1.0, W3C Note, 19-Jan-1999
http://www.w3.org/TR/NOTE-ddml

[NOTESOX] Schema for Object-Oriented XML 2.0, W3C Note 30 July 1999
http://www.w3.org/TR/NOTE-SOX

[NOTEXMLD] XML-Data, W3C Note 05 Jan 1998
http://www.w3.org/TR/1998/NOTE-XML-data/

[NSKMMMJ] Een metamodel voor Java, genereren van documentatie voor Java bestanden als toepassing. Eindverhandeling aangeboden tot het behalen van de graad van gediplomeerde in de aanvullende studies in de toegepaste informatica. Faculteit Wetenschappen, Katholieke Universiteit Leuven, 2000-2001. Nele Smeets en Kristof Mertens, promotor: prof. dr. ir. E. Steegmans.

[NSKMMMJ-212] Een metamodel voor Java, 2.1.2 Gebruik. Nele Smeets en Kristof Mertens.

[NSKMMMJ-345] Een metamodel voor Java, Hoofdstuk 3 tot en met hoofdstuk 5. Nele Smeets en Kristof Mertens.

[NSKMMMJ-35] Een metamodel voor Java, 3.5 AST (Abstract Syntax Tree). Nele Smeets en Kristof Mertens.

[NSKMMMJ-4643] Een metamodel voor Java, 4.6.4.3 Homogene ASTs in SoFT. Nele Smeets en Kristof Mertens.

[NSKMMMJ-532] Een metamodel voor Java, 5.3.2 Het parsen van documentatie. Nele Smeets en Kristof Mertens.

[NSKMMMJ-6] Een metamodel voor Java, 6. Objectmodel voor Java. Nele Smeets en Kristof Mertens.

[NSKMMMJ-66] Een metamodel voor Java, 6.6 Niet geresolvede elementen. Nele Smeets en Kristof Mertens.

[NSKMMMJ-913] Een metamodel voor Java, 9.1.3 Het Worker patroon. Nele Smeets en Kristof Mertens.

[NSKMMMJ-92] Een metamodel voor Java, 9.2 Het Worker patroon in SoFT. Nele Smeets en Kristof Mertens.

[NSKMMMJ-93] Een metamodel voor Java, 9.3 Het Factory patroon. Nele Smeets en Kristof Mertens.

[NSKMMMJ-95] Een metamodel voor Java, 9.5 Voorbeeld. Nele Smeets en Kristof Mertens.

[NWLMDB] DocBook: The Definitive Guide door Norman Walsh and Leonard Muellner
http://www.docbook.org/tdg/en/html/docbook.html

[OASIS] Organization for the Advancement of Structured Information Standards
http://www.oasis-open.org

[OPENSOURCE] The Open Source Initiative van opensource.org
http://www.opensource.org/licenses/

[PHP] PHP: Hypertext Preprocessor
http://www.php.net

[PLPDF] Planet PDF
http://www.planetpdf.com/

[RCSGML] The XML Cover Pages, SGML: General Introductions and Overviews door Robin Cover
http://www.oasis-open.org/cover/general.html

[RCWAP] The XML Cover Pages, WAP Wireless Markup Language Specification (WML) door Robin Cover
http://www.oasis-open.org/cover/wap-wml.html

[RCXLL] The XML Cover Pages, XML Linking and Addressing Languages (XPath, XPointer, XLink) door Robin Cover
http://www.oasis-open.org/cover/xll.html

[RDW3S] DTD School van W3Schools.com copyright Refsnes Data
http://www.w3schools.com/dtd/default.asp

[RDXML] XML Tutorial copyright Refsnes Data
http://www.w3schools.com/xml/default.asp

[RDXSL] XSL Tutorial door Refsnes Data
http://www.w3schools.com/xsl/

[RELAX] RELAX (Regular Language description for XML)
http://www.xml.gr.jp/relax/

[RFC1766] Request for Comments 1766, Tags for the Identification of Languages door H. Alvestrand
http://www.ietf.org/rfc/rfc1766.txt

[RMIGS] Getting Started Using RMI
http://java.sun.com/products/jdk/1.2/docs/guide/rmi/getstart.doc.html

[RTF] Rich Text Format (RTF) Version 1.5 Specification
http://www.biblioscape.com/rtf15_spec.htm

[RX] RenderX verkoopt XEP, een commerciële tool om XSL:FO files te transformeren naar PDF formaat
http://www.renderx.com/

[SAX] The official SAX website
http://www.saxproject.org/

[SAXON] SAXON, the XSLT processor door Michael H. Kay
http://saxon.sourceforge.net/

[SCHEMATRON] The Schematron, An XML Structure Validation Language using Patterns in Trees
http://www.ascc.net/xml/resource/schematron/schematron.html

[SCHEME] Scheme, Massachusetts Instute of Technologie
http://www.swiss.ai.mit.edu/projects/scheme/

[SDB99] E. Steegmans, J. Docks, S. De Backer. Objectgericht programmeren met Java. Acco Leuven, 1999

[SMRRWDXSP] eXtensible Server Pages, Working Draft 2000-01-09 by Stefano Mazzocchi and Ricardo Rocha
http://xml.apache.org/cocoon1/wd-xsp.html

[SQHTSCHEMA] XML Schema door C. M. Sperberg-McQueen en Henry Thompson
http://www.w3.org/XML/Schema

[SSWD] History of XML door Selena Sol
http://wdvl.internet.com/Authoring/Languages/XML/Tutorials/Intro/history.html

[SVG] Scalable Vector Graphics (SVG), W3C Document Formats Domain
http://www.w3.org/Graphics/SVG/Overview.htm8

[SVXML] XML turorial door Shermin Voshmgir
http://www.javacommerce.com/tutorial/xmlj

[SWAG] Management of Australian film collection is based on Tamino XML database door Software AG
http://www.softwareag.com/xml/applications/cinemedia.htm

[TRAREA] Gebiedsmodel (area model) van XSL, W3C specificatie
http://www.w3.org/TR/xsl/slice4.html#area_model

[TRCSS] Cascading Style Sheets level 2, CSS2 Specification, W3C Recommendation 12-May-1998
http://www.w3.org/TR/REC-CSS2/

[TRDOM] Document Object Model (DOM)
http://www.w3.org/DOM/

[TRDOMAC] Document Object Model (DOM) Activity Statement
http://www.w3.org/DOM/Activity

[TRDOMFAQ] Document Object Model FAQ
http://www.w3.org/DOM/faq

[TRDOML0] Document Object Model (DOM) Requirements, W3C Working Draft 19 April, 2001
http://www.w3.org/TR/2001/WD-DOM-Requirements-20010419/

[TRDOML1] Document Object Model (DOM) Level 1 Specification, Version 1.0, W3C Recommendation 1 October 1998
http://www.w3.org/TR/REC-DOM-Level-1/

[TRDOML2] Document Object Model (DOM) Level 2 Core Specification, Version 1.0, W3C Recommendation 13 November, 2000
http://www.w3.org/TR/DOM-Level-2-Core/

[TRDOML3] Document Object Model (DOM) Level 3 Core Specification, Version 1.0, W3C Working Draft 09 April 2002
http://www.w3.org/TR/DOM-Level-3-Core/

[TREX] TREX - Tree Regular Expressions for XML
http://www.thaiopensource.com/trex/

[TRHTML] HTML 4.01 Specification W3C Recommendation 24 December 1999
http://www.w3.org/TR/html401/

[TRMathML] Mathematical Markup Language (MathML™) 1.01 Specification, W3C Recommendation, revision of 7 July 1999
http://www.w3.org/TR/REC-MathML/

[TRPRR] Property Refinement / Resolution, W3C specificatie
http://www.w3.org/TR/xsl/slice5.html#refinement

[TRSOAP] Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000
http://www.w3.org/TR/SOAP/

[TRXHTML] XHTML[tm] 1.0: The Extensible HyperText Markup Language, A Reformulation of HTML 4 in XML 1.0, W3C Recommendation 26 January 2000
http://www.w3.org/TR/xhtml1/

[TRXINCL] XML Inclusions (XInclude) Version 1.0, W3C Candidate Recommendation 21 February 2002
http://www.w3.org/TR/xinclude/

[TRXML] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000
http://www.w3.org/TR/REC-xml

[TRXMLAVN] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, Attribute-Value Normalization
http://www.w3.org/TR/REC-xml#AVNormalize

[TRXMLBASE] XML Base, W3C Recommendation 27 June 2001
http://www.w3.org/TR/xmlbase/

[TRXMLBS4] XML Base, W3C Recommendation 27 June 2001, 4 Resolving Relative URIs
http://www.w3.org/TR/xmlbase/#resolution

[TRXMLCSC] Extensible Markup Language (XML) 1.0 (Second Edition), Common Syntactic Constructs, W3C Recommendation 6 October 2000
http://www.w3.org/TR/2000/REC-xml-20001006#sec-common-syn

[TRXMLEOL] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, End-of-Line Handling
http://www.w3.org/TR/REC-xml#sec-line-ends

[TRXMLNA] Namespaces in XML door het World Wide Web Consortium 14-January-1999
http://www.w3.org/TR/REC-xml-names/

[TRXMLNAMES] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, XML names
http://www.w3.org/TR/REC-xml#NT-Name

[TRXMLNAND] Namespaces in XML, Namespace Defaulting door het World Wide Web Consortium 14-January-1999
http://www.w3.org/TR/1999/REC-xml-names-19990114/#defaulting

[TRXMLSCH0] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001
http://www.w3.org/TR/xmlschema-0/

[TRXMLSCH1] XML Schema Part 1: Structures, W3C Recommendation 2 May 2001
http://www.w3.org/TR/xmlschema-1/

[TRXMLSCH2] XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001
http://www.w3.org/TR/xmlschema-2/

[TRXMLSS] Associating Style Sheets with XML documents Version 1.0, W3C Recommendation 29 June 1999
http://www.w3.org/TR/xml-stylesheet/

[TRXPA] XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999
http://www.w3.org/TR/xpath

[TRXPOINTER] XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation 11 September 2001
http://www.w3.org/TR/xptr/

[TRXS0-22] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, 2.2 Complex Type Definitions, Element & Attribute Declarations
http://www.w3.org/TR/xmlschema-0/#DefnDeclars

[TRXS0-223] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, 2.2.3 Naming Conflicts
http://www.w3.org/TR/xmlschema-0/#NamingConflicts

[TRXS0-252] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, 2.5.2 Mixed Content
http://www.w3.org/TR/xmlschema-0/#mixedContent

[TRXS0-26] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, 2.6 Annotations
http://www.w3.org/TR/xmlschema-0#CommVers

[TRXS0-27] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, 2.7 Building Content Models
http://www.w3.org/TR/xmlschema-0/#groups

[TRXS0-28] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, 2.8 Attribute Groups
http://www.w3.org/TR/xmlschema-0/#AttrGroups

[TRXS0-47] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, 4.7 Abstract Elements and Types
http://www.w3.org/TR/xmlschema-0/#abstract

[TRXS0-B] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, B. Simple Types & Their Facets
http://www.w3.org/TR/xmlschema-0/#SimpleTypeFacets

[TRXS0-C] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, C. Using entities
http://www.w3.org/TR/xmlschema-0/#usingEntities

[TRXS1-314] XML Schema Part 1: Structures, W3C Recommendation 2 May 2001, 3.1.4 White Space Normalization during Validation
http://www.w3.org/TR/xmlschema-1/#section-White-Space-Normalization-during-Validation

[TRXS2-2511] XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001, Atomic datatypes, definition
http://www.w3.org/TR/xmlschema-2/#dt-atomic

[TRXS2-3] XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001, 3 Built-in datatypes
http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

[TRXS2-333] XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001, 3.3.3 language
http://www.w3.org/TR/xmlschema-2/#language

[TRXS2-43] XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001, 4.3 Constraining Facets
http://www.w3.org/TR/xmlschema-2/#rf-facets

[TRXS2-436] XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001, 4.3.6 whiteSpace
http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace

[TRXSL] Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation 15 October 2001
http://www.w3.org/TR/xsl/

[TRXSLNA] De XSL Namespace, W3C specificatie
http://www.w3.org/TR/xsl/slice2.html#xsl-namespace

[TRXSLT] XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999
http://www.w3.org/TR/xslt

[TRXSLTCR] XSL Transformations (XSLT) Version 1.0, Conflict Resolution for Template Rules, W3C Recommendation 16 November 1999
http://www.w3c.org/TR/xslt#conflict

[TRXSLTNU] XSL Transformations (XSLT) Version 1.0, Numbering, W3C Recommendation 16 November 1999
http://www.w3c.org/TR/xslt#number

[TUG] TeX Users Group Home Page
http://www.tug.org/

[TWVB] XSLT Tutorial, eXtensible Stylesheet Language Transformation (XSLT) door Tracey Wilson
http://www.vbxml.com/xsl/tutorials/intro/

[UCBA] Unicode Standard Annex #9, The Bidirectional Algorithm, Unicode, Inc.
http://www.unicode.org/unicode/reports/tr9/

[UNICODE] Unicode Home Page
http://www.unicode.org/

[USAX] Understanding SAX
http://www6.software.ibm.com/developerworks/education/x-usax/index.html

[W3C] The World Wide Web Consortium
http://www.w3.org/

[WBHEDW] Simplify XML programming with JDOM door Wes Biggs en Harry Evans
http://www-106.ibm.com/developerworks/java/library/j-jdom/

[XALAN] Xalan-Java version 2.3.1 van de Apache Software Foundation
http://xml.apache.org/xalan-j/index.html

[XALANDES] Xalan-J 2.0 Design
http://xml.apache.org/xalan-j/design/design2_0_0.html

[XERCES2] Xerces2 Java Parser
http://xml.apache.org/xerces2-j/index.html

[XSPP] XSP Processor
http://xml.apache.org/cocoon1/xsp.html

[ZPDF] PDF Zone.com
http://www.pdfzone.com/

[ZVONDTDTUT] DTD Tutorial
http://www.zvon.org/xxl/DTDTutorial/General/contents.html