Inleiding tot XML en aanverwante specificaties

Kris Luyten
Universiteit Hasselt (http://www.luc.ac.be)
Expertisecentrum Digitale Media (http://www.edm.luc.ac.be)

kris.luyten@luc.ac.be

Chris Vandervelpen
Prof. dr. Wim Lamotte
Peter Quax

Inleiding tot XML en aanverwante specificaties

door Kris Luyten
Chris Vandervelpen
Prof. dr. Wim Lamotte
Peter Quax

Wijzigingen

Herziening 0.0.1 7 december 2001
Eerste draft versie. Basismateriaal bestaande uit een verzameling slides beschikbaar gesteld door Chris Raymaekers
Herziening 0.0.2 10 januari 2002
Eerste release versie
Herziening 0.0.3 14 oktober 2002
Eerste draft versie volgende cursus jaar. Verscheidene fouten zijn door studenten gemeld.
Herziening 0.0.4 29 oktober 2002
Toevoegen van SVG en SMIL aan deze cursus
Herziening 0.0.5 03 december 2002
* *Sectie over XMLSchema toegevoegd.* * *Verschillende fouten gefixt dank zij Bert Creemers*
Herziening 0.0.6 15 januari 2003
Hoofdstuk over SVG aangevuld (Chris Vandervelpen)
Herziening 0.0.7 26 maart 2003
* *xsl:variabele uitleg bijgevoegd* * *Hoofdstukken toegevoegd om verder uit te werken* * *BibTeXML case geïntroduceerd.*
Herziening 0.0.8 30 november 2003
* *Licentie is nu FDL 1.2* * *BibTeXML oefening toegevoegd* * *Enkele links toegevoegd* * *xsl:result-document toegevoerd* * *Definitie van predefi*
Herziening 0.0.9 4 februari 2004
* *XPath inleiding is gecorrigeerd (Peter Billen)* * *Kleine verduidelijking bij practicum 1*
Herziening 0.0.10 12 oktober 2004
* *Voorgedefinieerde entity references gecorrigeerd (Jan Fabry)* * *Opdrachten verwijderd achteraan tekst*

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1. Dankwoord

De volgende personen hebben bijdragen geleverd of fouten gemeld voor deze tekst, waarvoor mijn dank: Bert Creemers, Chris Raymaekers, Chris Vandervelpen, Jo Segers, Emile Vrijdags, Davy Sannen, Jan Van den Bergh, Peter Billen, Jan Fabry.

Hoofdstuk 2. Introductie

Deze cursus behandelt de eXtensible Markup Language en aanverwante specificaties. Er wordt ondermeer aandacht besteed aan:

- XML
- DTD
- XML Schema
- XPath
- XSLT
- Java API voor XML processing
- Scaled Vector Graphics
- SMIL
- Andere praktische toepassingsgebieden van XML

In de loop van de tekst zullen de afkortingen verklaard en uitvoerig besproken worden. Er worden oefeningen voorzien bij elk onderdeel van dit boek.

Je kan de laatste versie van dit boek online bekijken of downloaden op de website (http://lumumba.luc.ac.be/~kris/courses/xml/) van deze cursus. Je vindt hier ook een verzameling nuttige links. Het boek wordt gepubliceerd onder de vrije documentatie licentie (http://www.gnu.org/copyleft/fdl.html) van de Free Software Foundation (http://www.gnu.org/).

Deze cursus behandelt niet alleen XML en aanverwante W3C[1] specificaties, maar is tevens zelf in een XML vorm geschreven. Hiervoor werd de DocBook[2] specificatie gebruikt, met de bijbehorende OASIS[3] Document Type Definition. Om het XML document om te zetten naar HTML en pdf werd vervolgens jade[4] gebruikt, een document (SGML) processer.

Noten

1. World Wide Web Consortium
2. http://www.oasis-open.org/docbook/
3. http://www.oasis-open.org/
4. http://openjade.sourceforge.net/

Hoofdstuk 3. XML: eXtensible Markup Language

W3C XML specificatie (http://www.w3.org/XML)

3.1. Inleiding

XML is een markup language gebaseerd op SGML die tekst “annoteert” met de betekenis van de tekst. Als je bijvoorbeeld de titel van dit hoofdstuk beschouwt: “XML: eXtensible Markup Language” zou je door middel van zogenaamde tags (zoals men die ook terugvindt in HTML) kunnen aanduiden dat het om een titel gaat:

```
<titel>XML: eXtensible Markup Language</titel>
```

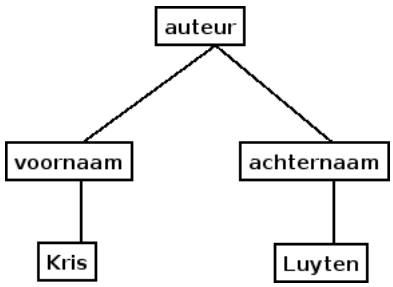
Merk op dat zulke tags uit twee delen bestaan: de start-tag <titel> en de eind-tag </titel>, en tussen deze twee tags komt de data voor waaraan de tag <titel> een betekenis geeft. Er kan ook gewoon niets tussen de start- en eind-tag voorkomen; dan kunnen we de twee tags tot een enkele tag combineren: <titel/>. Op dit moment lijkt dat nog nutteloos, maar als we het over attributen zullen hebben zal dit duidelijker worden en nuttiger blijken. Zulke tags noemt men Element tags, deze tags definiëren de markup. Er zijn geen standaard tags; je kan tags gebruiken die je wil, de betekenis die je aan de tags hangt kan je dan ook zelf definiëren. Daarom spreken we over de *eXtensible* Markup Language. XML beschrijft de structuur en het soort inhoud van een document. Het heeft niets te maken met de opmaak van een document.

Het is ook mogelijk om Element tags te nesten. Bijvoorbeeld:

```
<?xml version="1.0"?>
<auteur>
   <voornaam>Kris</voornaam>
   <achternaam>Luyten</achternaam>
</auteur>.
```

zegt ons dat auteur bestaat uit een voornaam en achternaam. Merk op dat dit als een boom kan uitgedrukt worden, zoals afgebeeld in Figuur 3-1.

Figuur 3-1. De auteur tag uitgedrukt in boom vorm



Elk element in de boom (auteur, naam, voornaam, maar ook de elementen die tekst bevatten zoals “Kris” en “Luyten”) wordt een *node* of knopen genoemd. Er zijn maar een beperkt aantal types nodes: commentaar nodes (blad van de XML boom), tekst nodes (blad), processing-instruction nodes (blad) en de nodes die geen blad zijn van de boom. Deze laatste hebben dan weer een verzameling van kindknopen (*child nodes*). Deze begrippen zullen verder in de tekst toegelicht worden.

> **Opmerking:** Een geldig XML document bevat altijd minimaal <?xml version="1.0"?> op de eerste regel van de tekst.

Voor het verwerken van een XML document gebruikt men een *parser*. Hierop zullen we later nog terugkomen. Het is voldoende dat je nu weet dat een parser een XML document verwerkt en in het geheugen als een boomstructuur kan voorstellen. [1]

3.2. Structuur van een XML document

Een XML document kan bestaan uit de volgende structuren:

- Element tags (met of zonder attributen)
- Entity references
- Commentaar
- Processing instructions
- CDATA secties
- Document type declarations

In de volgende paragrafen zullen we enkele hiervan uitleggen, de rest komt verder in de cursus aan bod.

3.2.1. Attributen

Wat Element tags zijn hebben we reeds in Paragraaf 3.1 uitgelegd. Wat nog niet vermeld is, is dat element tags ook *attributen* kunnen bevatten. Zo krijgen de lege tags ook meer zin. Attributen kunnen we binnen in de tag specifieren. Bijvoorbeeld:

```
<?xml version="1.0"?>
<auteur geslacht="man">
   <voornaam>Kris</voornaam>
   <achternaam>Luyten</achternaam>
</auteur>
```

Hier heeft de auteur element tag een attribuut “geslacht”. De vraag of iets nu als attribuut dan wel tussen de element tags moet staan wordt veel gesteld, maar een formeel correct antwoord is hier niet mogelijk. Vergelijk het met Object-georiënteerd programmeren: ga je iets als object beschouwen of als member variabele van een object? Dit hangt af van de situatie, de granulariteit van de data,... Er is een vuistregel om hierover een redelijke beslissing te nemen: meestal worden attributen gebruikt voor informatie die men aan de element tag zelf wil toevoegen en die niet expliciet in de uitvoer voor de gebruiker getoond moet worden.

3.2.2. Entity References

Entity references kunnen een “macro” definiëren voor veel gebruikte tekst of verwijzen naar andere stukken tekst. Een entity reference begint altijd met een ampersand (&) en eindigt met een puntkomma (;). Bijvoorbeeld &luc; is een entity reference die zal vervangen worden door “Limburgs Universitair Centrum” als &luc; gedefinieerd was als:

```
<!ENTITY luc "Limburgs Universitair Centrum">
```

We zullen later zien dat dit soort definities voorkomt in de zogenaamde Document Type Definition (DTD).

3.2.3. Processing instructions

Een processing instruction wordt niet gebruikt door de parser, maar moet doorgegeven worden aan de applicatie. Deze processing instructions zijn dus “applicatie-afhankelijk”. Bijvoorbeeld: <?python lines=open(’/etc/passwd’).readlines()?> geeft aan de parsende applicatie door dat aan “python” de data **“lines=open(’/etc/passwd’).readlines()”** moet gegeven worden.

3.3. Well-formed XML documents

Na de eerste stappen in de syntax van element tags, kunnen we eens kijken wat nu *juist gevormde* (well-formed) XML documenten zijn. Een well-formed XML document voldoet aan de volgende regels:

- Er is een uniek root element tag
- Alle tags moeten gebalanceerd zijn
- Alle tags moeten goed genest zijn
- Attributen moeten tussen quotes (") staan
- >, < en & moeten juist gebruikt worden
- Er zijn enkel 5 voorgedefinieerde karakter referenties: <, >, &, ' en "

Er zijn verschillende (online) services beschikbaar om je XML document op well-formedness te controleren:

- Expat: ftp://ftp.jclark.com/pub/xml/expat.zip
- Web-based validation services:
  - http://www.xml.com/lpt/a/tools/ruwf/check.html
  - http://www.cogsci.ed.ac.uk/~richard/xml-check.html

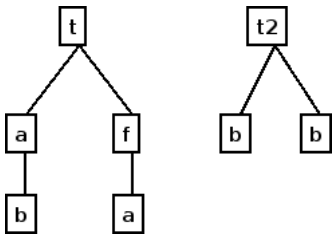
Men zegt ook wel dat een well-formed XML toelaat om een *element grove* te bouwen. Dit betekent dat het mogelijk is om een XML document als een boom document te laten zien.

Een eerste vereiste is dat er een uniek root element is. Beschouw het voorbeeld in listing Voorbeeld 3-1. Dit is *geen* well-formed XML document, omdat het twee root elementen heeft, namelijk <t> en <t2>.

Voorbeeld 3-1. Een verkeerde grove

```
<?xml version="1"?>
<t>
   <a>
      <b>
      </b>
   </a>
   <f>
      <a/>
   </f>
</t>
<t2>
   <b/>
   <b/>
</t2>
```

Figuur 3-2. Een foutieve XML grove

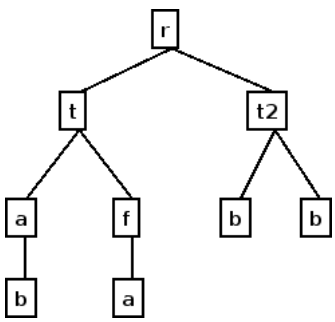


Het voorbeeld in listing Voorbeeld 3-2 is *wel* een goed XML document.

Voorbeeld 3-2. Een juiste grove

```
<?xml version="1.0"?>
<r>
   <t>
      <a>
         <b>
         </b>
      </a>
      <f>
         <a/>
      </f>
   </t>
   <t2>
      <b/>
      <b/>
   </t2>
</r>
```

Figuur 3-3. Een correcte XML grove



Een tweede vereiste is dat alle tags goed gebalanceerd moeten zijn.Voor elke start-tag moet er dus een eind-tag zijn. Denk eraan dat indien er tussen de start-tag en de eind-tag geen verdere data meer voorkomt (behalve de eventuele attributen van de tag) dat je de twee tags kan combineren tot een tag. De derde vereiste is dat de tags *goed genest* zijn. Voorbeelden van *foute* XML constructies zijn

- this text is in bold italic for emphasis
- this text is in bold italic for emphasis
- this text is in bold italic for emphasis

Merk op dat we hiermee niets over de betekenis van de tags zelf gezegd hebben. Een juiste XML constructie zou, voortgaand op de vorige foute voorbeelden, er uit zien als volgt:

```
<p>this is text is in <b>bold <i>italic</i></b> for emphasis</p>
```

of

```
<p>this is text is in <b>bold</b> <i>italic</i> for emphasis</p>
```

Zoals bij de meeste programmeertalen is het ook mogelijk om in je XML document commentaar toe te voegen. Een XML parser zal bij het parsen van het XML document de commentaar overslaan. Commentaar begint met de tekens <!-- en eindigt met -->. Alles wat tussen deze twee tekens staat wordt als commentaar beschouwd en niet door de gebruikte XML parser bekeken. Bijvoorbeeld:

```
<!-- Dit is commentaar, en kan verschillende tekens bevatten: > < ] [ &-->
```

De vierde vereiste is dat alle attributen tussen quotes moeten staan. xml pagina is dus *foutief* en xml pagina is *wel juist*. Let erop dat dit niet alleen voor tekst geldt, maar evenzeer voor getallen. Dus *wel* <item nr="124"/>, maar *niet* <item nr=124/>.

Er zijn enkele “gereserveerde” tekens in XML zoals >,< en &. We moeten ervoor zorgen dat deze tekens niet in de gewone tekst voorkomen, of in de tekst van attributen. Het is ook mogelijk om in XML aan te geven dat een stuk tekst door de parser niet geïnterpreteerd mag worden, maar dat deze “ruw” verwerkt moet worden [2], dit doen we door de zogenaamde *CDATA* sectie. Bijvoorbeeld:

```
<![CDATA[hier kan je </b> inschrijven <i> wat je wil <p>
zonder dat de XML parser daarover struikelt]]>
```

De laatste regel voor een well-formed XML document is dat er maar *vijf* voorgedefinieerde referenties zijn (je kan er zelf natuurlijk ook bij declareren).

- © geeft ©
- ® geeft ®
- &tm; geeft ™
- é geeft é
-   geeft (een spatie dus)

3.4. Lexicale beperkingen

Het eerste karakter van een naam (van een tag bijvoorbeeld) in XML begint met een karakter uit a-zA-Z. De volgende karakters kunnen “0-9a-zA-Z_-.:” zijn. Hierbij heeft ’:’ een speciale betekenis. Dit teken wordt gebruikt om namespaces van elkaar te scheiden. Namespaces zorgen ervoor dat je twee tags die dezelfde naam, maar toch een andere context/betekenis hebben niet conflicteren. Je kan het vergelijken met de Java import: lang.Math.random() en MyMath.random(); twee functies met een identieke naam, random, maar de ene in de “namespace” lang.Math en de andere in de “namespace” MyMath. In een naam mag geen whitespace karakter gebruikt worden. Element namen, attribute namen, attribute waarden en entity references zijn *case sensitive*.

3.5. Well-formedness controleren

- Expat (ftp://ftp.jclark.com/pub/xml/expat.zip)
- RUWF (are you well formed) (http://www.xml.com/lpt/a/tools/ruwf/check.html)
- Richard Tobin XML checker (http://www.cogsci.ed.ac.uk/~richard/xml-check.html)

3.6. Een voorbeeld-toepassing: BibTeXML

Doorheen dit boek zullen we in de oefeningen regelmatig gebruik maken van een BibTeXML (http://bibtexml.sf.net) als illustratief voorbeeld. BibTeXML definieert een op XML gebaseerde syntax om bibliografische data op te slaan, en om te zetten naar verschillende formaten.

In Voorbeeld 3-3 wordt weergegeven hoe we bibiliografische informatie over dit boek in BibTeXML kunnen specifieren.

Voorbeeld 3-3. BibTeXML specificatie voor dit boek

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE bibtex:file PUBLIC
    "-//BibTeXML//DTD XML for BibTeX (extended) v1.0//EN"
    "bibteXML.dtd" >
<bibtex:file xmlns:bibtex="http://bibtexml.sf.net/">
 <bibtex:entry id="luyten">
  <bibtex:book>
   <bibtex:author>Kris Luyten</bibtex:author>
   <bibtex:title>Introductie tot XML en aanverwante specificaties</bibtex:title>
   <bibtex:year>2003</bibtex:year>
<bibtex:publisher>Campus Boekhandel</bibtex:publisher>
   <bibtex:address>Expertisecentrum voor Digitale Media, Wetenschapspark 2</bibtex:address>
  </bibtex:book>
 </bibtex:entry>
</bibtex:file>
```

De <!DOCTYPE ...> regel wordt in het volgende hoofdstuk uit de doeken gedaan. Merk op dat in de regel <bibtex:file xmlns:bibtex="http://bibtexml.sf.net/"> de namespace *bibtex* gedefinieerd wordt. Dit wordt gedaan door dit aan te geven met het attribuut *xmlns* wat staat voor XML namespace. Na de dubbele punt van xmlns komt de naam van de namespace die we in ons XML document kunnen gebruiken. Voor elk element in het XML document getoond in Voorbeeld 3-3 staat de prefix *bibtex* die aanduid dat het element tot de bibtex namespace behoort. Tenslotte moet de namespace nog expliciet aangegeven worden door middel van een URL (een URL is altijd uniek). In dit geval wordt de namespace uniek geïdentificeerd met http://bibtexml.sf.net/.

3.7. Oefeningen

1. Stel een XML bestand op dat je stamboom voorstelt. Gebruik de tag <persoon> die als attribuut “naam” heeft en twee andere persoon elementen kan bevatten. Controleer je bestand op well-formedness.
2. Maak een grafische presentatie (boom) van de volgende XML listing:

Voorbeeld 3-4. Glade User Interface description

```
<?xml version="1.0"?>
<GTK-Interface>
  <project id="1409">
    <name>Project1</name>
    <program_name>project1</program_name>
    <source_directory>src</source_directory>
    <pixmaps_directory>pixmaps</pixmaps_directory>
    <language>C++</language>
    <gnome_support>True</gnome_support>
```

```
  <gettext_support>True</gettext_support>
  <infodoc type="html">docs/html</infodoc>
  <infodoc type="tex"/>
  <infodoc type="docbook">docs/docbook</infodoc>
</project>

<widget>
  <class>GtkWindow</class>
  <name>window1</name>
  <title>window1</title>
  <type>GTK_WINDOW_TOPLEVEL</type>
  <widget>
    <class>GtkVBox</class>
    <name>vbox1</name>
    <widget>
      <class>GtkButton</class>
      <name>Jump</name>
      <label>Jump</label>
    </widget>
    <widget>
      <class>GtkButton</class>
      <name>Bounce</name>
      <can_focus>True</can_focus>
      <label>Bounce</label>
      <relief>GTK_RELIEF_NORMAL</relief>
    </widget>
  </widget>
</widget>
</GTK-Interface>
```

3. Stel een XML bestand op dat voor een catalogus van een CD-shop kan gebruikt worden. Welke tags heb je nodig om de informatie van een CD te beschrijven? Vul zelf de data van enkele CDs in met je XML tags en check het bestand op well-formedness.
4. Breid de vorige oefening uit met tags om boeken en films te beschrijven (het assortiment van de winkel is uitgebreid). Vul het bestand in m.b.v. informatie van bestaande boeken en films. Zorg dat je bestand well-formed is. Hoe kies je of iets een attribuut dan wel een aparte tag moet zijn? Zijn er alternatieven voor je oplossing? Indien ja, waarom is de jouwe beter?

Noten

1. Later zal ook blijken dat de interne boomvoorstelling in het geheugen geen strikte vereiste is (SAX parsers)
2. Een beetje equivalent met de verbatim environment in LaTeX

Hoofdstuk 4. DTD: Document Type Definition

4.1. Een XML document valideren

We hebben het reeds gehad over well-formed XML documenten (Paragraaf 3.3), en aan welke regels zo een well-formed XML document moet voldoen. In dit hoofdstuk gaan we een stap verder: we laten zien hoe we aan de hand van een *schema* een set van regels kunnen opleggen waaraan een XML document moet voldoen. De meest gebruikte vorm van een schema is een *Document Type Definition* (DTD). Een XML document kan dan aan de hand van zijn bijbehorende DTD *gevalideerd* worden. De structuur van het document wordt gedefinieerd aan de hand van de DTD.

De DTD legt restricties op voor:

- element namen
- attribuut namen en waarden
- volgorde van de elementen
- nesting van de elementen

Een XML document dat aan deze restricties voldoet wordt een valid XML document genoemd. Een valid XML document is zowel well-formed als opgesteld volgens de regels die de DTD aangeeft.

Waarom is het nuttig dat een XML document kan gevalideerd worden aan de hand van een Document Type Definition? Op deze manier kan er garantie geboden worden over hetgeen afgeleverd wordt en kan de ontvanger controleren of de data in de verwachte vorm ontvangen werd. Vooral in de B2B[1]applicaties wordt de DTD gezien als een *contract* waaraan een afgeleverd XML document dient te voldoen.

4.2. Een DTD gebruiken

Er zijn twee plaatsen om een DTD te definiëren. De eerste is de DTD vooraan het XML document plaatsen (Voorbeeld 4-1) en de andere is een DTD in een apart bestand te definiëren en er naar te verwijzen vanuit het gewenste XML document (Voorbeeld 4-2). Let op: dit hoeft niet altijd een URL te zijn die naar een website verwijst; het kan evengoed naar een bestand op de harddisk wijzen.

Voorbeeld 4-1. DTD in XML document

```
<?xml version="1.0"?>
<!DOCTYPE Document
[
<!Element Document ..>
  ...
]>
```

```
<Document>
   ...
</Document>
```

Voorbeeld 4-2. DTD gerefereerd vanuit XML document

```
<?xml version="1.0" ?>
<!DOCTYPE Document SYSTEM "http://www.xml.org/doc.dtd" [
   ...
]>
<Document>
   ...
</Document>
```

4.3. Regels op elementen

Een DTD bevat declaraties voor elementen en attributen en wordt uitgedrukt volgens de volgende syntax:

```
<!keyword parameter associated parameter(s)>
```

Voor een element wordt de DTD als volgt gespecifieerd:

```
<!ELEMENT element_name content_model>
```

Hierbij is *element_name* de naam van een element dat kan voorkomen in het XML document en *content_model* de mogelijke vorm van de subtree onder dit element. Dit content model zullen we aan de hand van enkele voorbeelden in meer detail uitleggen.

Het content model bestaat uit:

- textual content
- combined content:
  - textual content
  - namen van de elementen onder dit element
  - namen van de parameter entities
  - combinaties van de voorgaande door middel van logische operatoren

Het simpelste is textual content: dit kan enkel tekst bevatten, verder niets. Het mag bijvoorbeeld geen subelementen bevatten. Textual content wordt aangegeven door de volgende keywords: *#PCDATA*, *EMPTY* of *ANY*. Neem als voorbeeld de definitie uit Voorbeeld 4-3.

Voorbeeld 4-3. DTD regel voor het element naam

```
<!ELEMENT naam (#PCDATA)>
```

Dit geeft weer dat het element naam enkel tekst mag bevatten, zonder subelementen. Als deze regel in de DTD van je XML document voorkomt dan kan je het element naam gebruiken als volgt:

```
<naam>Richard Feynmann</naam>
```

maar het volgende voorbeeld is *fout*:

Voorbeeld 4-4. Naam met subelementen

```
<naam>
   <voornaam>Richard</voornaam>
   <achternaam>Feynmann</achternaam>
</naam>
```

Je mag ook *niet* <naam/> gebruiken. Als je wil aangeven dat je element een leeg element kan zijn gebruik je deze regel in je DTD: <!ELEMENT naam EMPTY>. Je kan dan natuurlijk ook <naam></naam> gebruiken.

4.4. Combined content

Stel dat je wel wil toelaten dat het element naam uit de twee subelementen voornaam en achternaam bestaat, dit op hun beurt enkel tekst bevatten. Dit kunnen we als volgt beschrijven in de DTD:

Voorbeeld 4-5. Combined Content in het naam element

```
<!ELEMENT naam (voornaam, achternaam)>
<!ELEMENT voornaam (#PCDATA)>
<!ELEMENT achternaam  (#PCDATA)>
```

Als er van deze definitie gebruik wordt gemaakt kan men wel de naam beschrijven zoals in Voorbeeld 4-4. Let op de volgorde: deze is strikt gedefinieerd hier, je moet dus eerst de voornaam en dan de achternaam geven. Volgende voorbeelden zijn dus *fout*:

```
<naam>
   <achternaam>Feynmann</achternaam>
   <voornaam>Richard</voornaam>
</naam>
```

```
<naam>
   <voornaam>Richard</voornaam>
   <achternaam></achternaam>
</naam>


<naam>
   <voornaam>Richard</voornaam>
</naam>
```

De vorige alinea laat enkel zien hoe je een opeenvolging van elementen onder een ander element specifieert. Er zijn verschillende mogelijkheden om combined content onder een element uit te drukken:

E

: matcht met element E

E?

: matcht met *hoogstens* 1 element E

E+

: matcht met *minstens* 1 element E

E*

: matcht met *0 of meer* elementen E

E1|E2

: matcht met element E1 *of* element E2

E1,E2

: matcht met element E1 en daarna met element E2; E1 en E2 zijn siblings in die volgorde

De vorige mogelijkheden kunnen gecombineerd worden tot meer uitgebreide mogelijkheden. We krijgen de volgende regels (beschouw e als leaf van de boom-representatie):

```
E->e
E->(E?) | (E+) | (E*) | (e|E) | (e,E)
```

Om deze mogelijkheden te illustreren geven we enkele voorbeelden:

Voorbeeld 4-6. De naam DTD geherdefinieerd

```
<!ELEMENT naam ((voornaam,achternaam)|(achternaam,voornaam))>
<!ELEMENT voornaam (#PCDATA)>
<!ELEMENT achternaam (#PCDATA)>
```

Deze DTD laat toe om een willekeurige volgorde toe te laten voor de voor- en achternaam. Op deze manier is de volgende XML structuur *wel* geldig:

```
<naam>
   <achternaam>Feynmann</achternaam>
   <voornaam>Richard</voornaam>
</naam>
```

Dit geeft ook dadelijk een van de tekortkomingen van DTDs weer (dit werd ook niet opgelost in de opvolger van de DTD; XML Schema): er is geen manier om aan te geven dat de kind elementen van een element in een willekeurige volgorde kunnen voorkomen, zonder al de mogelijkheden op te sommen.

Een uitgebreider voorbeeld halen we uit een bestaande DTD voor Software Version Control[2]. Deze DTD geeft aan hoe je met behulp van XML informatie over een software package kan definiëren.

Voorbeeld 4-7. Deel van de svc.dtd

```
<!ELEMENT softpkg        (name,authors?,abstract?,website?,
                          specification?,buglist?,implementation*)>
<!ELEMENT name           (#PCDATA)>
<!ELEMENT abstract       (#PCDATA)>
<!ELEMENT website        (#PCDATA)>
<!ELEMENT authors        (person+)>
<!ELEMENT person         (name,email?)>
<!ELEMENT email          (#PCDATA)>
<!ELEMENT specification  (rule*)>
<!ELEMENT rule           (desc)>
<!ELEMENT desc           (#PCDATA)>
<!ELEMENT implementation (distro)*>
<!ELEMENT distro         (release-date?,release-info?,buglist?,diffs*)>
<!ELEMENT release-date   (#PCDATA)>
<!ELEMENT release-info   (#PCDATA)>
<!ELEMENT buglist        (bug+)>
<!ELEMENT bug            (#PCDATA)*>
<!ELEMENT diffs          (spec,depend,bug,object,func,var)+>
<!ELEMENT spec           (support,comment)>
<!ELEMENT support        EMPTY>
<!ELEMENT depend         (softpkg)>
<!ELEMENT object         (rel*,desc?,remark*)>
<!ELEMENT rel            (#PCDATA)>
<!ELEMENT func           (returns?,param*,desc?,remark*)>
<!ELEMENT param          (#PCDATA)>
<!ELEMENT returns        (#PCDATA)>
```

```
<!ELEMENT remark         (#PCDATA)>
<!ELEMENT var            (type?,desc?,remark*)>
<!ELEMENT type           (#PCDATA)>
```

Merk op dat hier enkel iets gezegd wordt over de volgorde en structuur van de elementen, maar nog niets over de attributen. In de volgende sectie zullen we laten zien hoe je restricties kan opleggen aan de attributen in een XML document. Een voorbeeld van een geldig XML document (well-formed en valid) vind je in Voorbeeld 4-8.

Voorbeeld 4-8. XML document conform aan de svc.dtd

```
<?xml version="1.0" ?>
<!DOCTYPE softpkg SYSTEM "http://www.openscience.org/~egonw/svc/svc.dtd" [
]>
<softpkg>
   <name>jrtplib</name>
   <authors>
      <person>
         <name>Jori Liesenborgs</name>
         <email>jori@lumumba.luc.ac.be</email>
      </person>
   </authors>
   <abstract>
      JRTPLIB is an object-oriented RTP library written in C++.
   </abstract>
   <website>http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.html</website>
   <implementation>
      <distro>
         <release-date>02 januari 2000</release-date>
      </distro>
      <distro>
         <release-date>02 december 2000</release-date>
         <release-info>
             The library offers support for the Real-time Transport Protocol (RTP),
             defined in RFC 1889
         </release-info>
      </distro>
   </implementation>
</softpkg>
```

Belangrijk: Tenslotte nog een belangrijke opmerking waarmee rekening dient gehouden te worden indien men `#PCDATA` gebruikt binnen een `AND` of `OR` combinatie: `#PCDATA` (plain tekst) kan *niet* gecombineerd worden in een `AND` sequentie met andere elementen. Dit zou tot ambiguïteit leiden. Tekst en elementen kunnen echter wel gecombineerd worden in een `OR` sequentie, op voorwaarde dat de `#PCDATA` eerst komt, zoals aangegeven in Voorbeeld 4-9.

Voorbeeld 4-9. #PCDATA en andere elementen in een OR sequentie

```
<!ELEMENT p (#PCDATA | a | b | i)*>
```

4.5. Regels op attributen

Naast regels op elementen, kunnen we ook regels leggen op attributen met behulp van een Document Type Definition. De syntax voor zulk een definitie is:

```
<!ATTLIST element_naam attribuut_naam waarden default_waarde>
```

Dit definieert de naam van het attribuut, de mogelijke waarden van het attribuut (type), en optioneel de default waarde van dit attribuut. Een voorbeeld van een attribuut wordt gegeven in Voorbeeld 4-10, dit breidt het element "persoon" uit (Voorbeeld 4-7) met attributen.

Voorbeeld 4-10. Persoon uit svc.dtd uitgebreid met attributen

```
<!ELEMENT person EMPTY>
<!ATTLIST person id    CDATA #IMPLIED
                 type  (author|contact|co-author) "author">
```

Het voorbeeld in Voorbeeld 4-10 zegt dat het element person twee attributen kan hebben: *id* en *type*. Het attribuut id is van het type CDATA: dit wil zeggen dat het gewone tekst bevat. Daarachter bevindt zich de attribuut declaratie #IMPLIED: dit geeft aan dat het invullen van dit attribuut verplicht is. Het attribuut type heeft 3 mogelijke waarden: author, contact of co-author, waar de default waarde "author" is. Dit betekent dat het attribuut type niet expliciet moet aangegeven worden: bij het ontbreken wordt het attribuut type automatisch gelijkgesteld aan author. De openingstag van het person element kan er dan als volgt uitzien:

```
<person id="nr021589" type="contact"/>
```

of

```
<person id="abde1165563ed"/>
```

Merk op dat de waarden van attributen *altijd* tussen quotes moeten staan.

Er zijn verschillende types voor attributen, zoals CDATA in het vorige voorbeeld (Voorbeeld 4-10). Mogelijke types zijn:

CDATA

: String zonder < en zonder "

ID

: Unieke ID

IDREF

: Een referentie naar een ID

IDREFS

: Een serie referenties (IDREF), gescheiden door spaties

NMTOKEN

: Een legale nametoken: dit is een geldige XML naam

NMTOKENS

: Een serie nametokens (NMTOKEN), gescheiden door spaties

ENTITY

: Een entity die gedeclareerd is in de DTD

ENTITIES

: Een serie entities (ENTITY), gescheiden door spaties

NOTATION

: Een notatie type die gedeclareerd is in de DTD

Opsomming

: Een opsomming van mogelijke waarden, door de gebruiker gedefinieerd

Naast deze types, zijn er ook 4 soorten attribuut declaraties:

#REQUIRED

: Er moet een waarde aan het attribuut gegeven worden

#IMPLIED

: Geeft aan dat de XML processor expliciet aan de applicatie moet melden dat er geen waarde ingevuld is, indien dit het geval is.

#FIXED

: De waarde die aan het attribuut toegekend wordt *moet* de default waarde zijn.

Default waarde

: Als er een default waarde gegeven is zal deze gebruikt worden indien het attribuut niet gebruikt werd in de desbetreffende tag. Anders wordt de opgegeven waarde gebruikt.

De types CDATA en opsomming kwamen reeds aan bod in Voorbeeld 4-10. Merk op dat onze uitbreiding aan de svc.dtd eigenlijk niet helemaal volgens de regels van het spel gedaan is: we hebben bij het element person "id CDATA" als attribuut bijgestoken. Een beter oplossing zou zijn om ID te gebruiken. We kunnen dan met IDREF naar een ID van een person verwijzen.

Voorbeeld 4-11. ID en IDREF

```
<!ATTLIST person id ID #REQUIRED
                type  (author|contact|co-author) "author">
<!ATTLIST debugger id IDREF #REQUIRED>
<!ATTLIST projectteam ids IDREFS #REQUIRED>
```

We breiden onze DTD uit (Voorbeeld 4-11) met twee nieuwe attribuut declaraties voor twee andere tags. `debugger` is een tag die als attribuut een id heeft. Deze id wijst naar een bestaande andere id in het XML document. Een project team is samengesteld uit een aantal bestaande personen: we gebruiken hier een lijst van IDREFs om dit aan te geven.

Voorbeeld 4-12. ID en IDREF in het XML document

```
<person id="nr123-456-789">
   ...
</person>
<person id="nr321-457-987">
   ...
</person>
<person id="nr987-654-321">
   ...
</person>
<debugger id="nr123-456-789">
   ...
</debugger>
<projectteam ids="nr123-456-789 nr321-457-987 nr987-654-321"/>
   ...
</projectteam>
```

Een voorbeeld van NMTOKEN en NMTOKENS vind je in Voorbeeld 4-13.

Voorbeeld 4-13. NMTOKEN en NMTOKENS

```
<?xml version="1.0"?>
<!DOCTYPE workfloor[
<!ELEMENT Employee ...
<!ATTLIST Employee security_level NMTOKEN #REQUIRED
```

```
                    compartments NMTOKENS #IMPLIED>
...
]>
<workfloor>
   <Employee security_level="trusted" compartments="red green blue">
   ...
```

NOTATION kan gebruikt worden om een formaat aan te geven van data die niet door de XML processor verwerkt dient te worden. Zo kan men bijvoorbeeld voor een png bestand aangeven dat hiervoor de applicatie "the Gimp" moet gebruikt worden. Men moet hiervoor eerst voor deze types een NOTATION declareren. Als dit gebeurd is kan men dit gebruiken verder in de DTD. Een voorbeeld hiervan vind je in Voorbeeld 4-14.

Voorbeeld 4-14. NOTATION

```
<?xml version="1.0"?>
<!DOCTYPE images[
<!NOTATION jpg SYSTEM "eog">
<!NOTATION png SYSTEM "gimp">
...
<!ATTLIST image type NOTATION (png|jpg) "png" >
...
]>
<images>
   <image type="jpg">
   ...
```

Als laatste type tonen we hoe ENTITY en ENTITIES kunnen gebruikt worden. In het inleidende hoofdstuk over XML kwam al ter sprake hoe bijvoorbeeld de entiteit &luc; kon gedefinieerd worden. We kunnen op eenzelfde manier "afkortingen" in de DTD definiëren met behulp van *parameter entities*.

Voorbeeld 4-15. Een parameter entity

```
<!ENTITY % mijnattr  "leeftijd CDATA #IMPLIED
                      geslacht (man|vrouw) #REQUIRED">
```

Let op het teken "%" dat verplicht is voor parameter entities. Dit kan dan in de DTD verder gebruikt worden, zoals getoond in het volgende voorbeeld.

Voorbeeld 4-16. Het gebruik van een parameter entity

```
<!ELEMENT persoon>
<!ATTLIST persoon %mijnattr; naam CDATA #REQUIRED>
```

Dit zal ge-expand worden door de XML parser tot

Voorbeeld 4-17. De parameter entity opgelost

```
<!ELEMENT persoon>
<!ATTLIST persoon leeftijd CDATA #IMPLIED
                  geslacht (man|vrouw) #REQUIRED
                  naam CDATA #REQUIRED>
```

Nu alle mogelijke types en attribuut declaraties gedefinieerd worden geven we als laatste het volledige svc.dtd voorbeeld. In Voorbeeld 4-7 toonden we de software version control DTD zonder mogelijke attributen. In Voorbeeld 4-18 wordt de volledige DTD getoond.

Voorbeeld 4-18. De volledige svc.dtd

```
<!--
 * SVC.dtd version 0.9.1
 *
 * Information can be found
 * at http://openscience.chem.nd.edu/~egonw/svc/
 *
 * Copyright (c) 2000 E.L. Willighagen (egonw@sci.kun.nl)
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place -
 * Suite 330, Boston, MA  02111-1307, USA. -->

<!ELEMENT softpkg        (name,authors?,abstract?,website?,
                          specification?,buglist?,implementation*)>
```

```
<!ATTLIST softpkg        id              CDATA        #REQUIRED
                         version         CDATA        "0.9.1">
<!ELEMENT name           (#PCDATA)>
<!ELEMENT abstract       (#PCDATA)>
<!ELEMENT website        (#PCDATA)>
<!ELEMENT authors        (person+)>
<!ELEMENT person         (name,email?)>
<!ATTLIST person         id              CDATA        #IMPLIED
                         type            (author|contact|co-author)
                                         "author">
<!ELEMENT email          (#PCDATA)>
<!ELEMENT specification  (rule*)>
<!ATTLIST specification  href            CDATA        #IMPLIED>
<!ELEMENT rule           (desc)>
<!ATTLIST rule           id              CDATA        #REQUIRED>
<!ELEMENT desc           (#PCDATA)>
<!ELEMENT implementation (distro)*>
<!ATTLIST implementation lang            CDATA        #REQUIRED
                         os              CDATA        #IMPLIED>
<!ELEMENT distro         (release-date?,release-info?,buglist?,diffs*)>
<!ATTLIST distro         version         CDATA        #REQUIRED>
<!ELEMENT release-date   (#PCDATA)>
<!ATTLIST release-date   href            CDATA        #IMPLIED>
<!ELEMENT release-info   (#PCDATA)>
<!ELEMENT buglist        (bug+)>
<!ELEMENT bug            (#PCDATA)*>
<!ATTLIST bug            id              CDATA        #REQUIRED
                         state           (solved|known)
                                         "known">
<!ELEMENT diffs          (spec,depend,bug,object,func,var)+>
<!ATTLIST diffs          type            (new|changed|removed)
                                         "new">
<!ELEMENT spec           (support,comment)>
<!ATTLIST spec           id              CDATA        #REQUIRED
                         author          CDATA        #IMPLIED
                         type            CDATA        #IMPLIED>
<!ELEMENT support        EMPTY>
<!ATTLIST support        percentage      CDATA        #IMPLIED
                         mark            (none|partly|buggy|full)
                "none">
<!ELEMENT depend         (softpkg)>
<!ATTLIST depend         author          CDATA        #IMPLIED
                         type            CDATA        #IMPLIED>
<!ELEMENT object         (rel*,desc?,remark*)>
<!ATTLIST object         author          CDATA        #IMPLIED
                         name            CDATA        #REQUIRED
                         type            CDATA        #IMPLIED>
<!ELEMENT rel            (#PCDATA)>
<!ATTLIST rel            type            (part-of|subclass|superclass)
                                                      "part-of">
<!ELEMENT func           (returns?,param*,desc?,remark*)>
<!ATTLIST func           author          CDATA        #IMPLIED
                         name          CDATA       #REQUIRED
```

```
                        scope           CDATA       #IMPLIED
                        type            CDATA       #IMPLIED>
<!ELEMENT param         (#PCDATA)>
<!ATTLIST param         order           CDATA       #IMPLIED
                        type            CDATA       #IMPLIED
                        name            CDATA       #REQUIRED>
<!ELEMENT returns       (#PCDATA)>
<!ATTLIST returns       type            CDATA       #REQUIRED>
<!ELEMENT remark        (#PCDATA)>
<!ELEMENT var           (type?,desc?,remark*)>
<!ATTLIST var           author          CDATA       #IMPLIED
                        scope           CDATA       #IMPLIED
                        name            CDATA       #REQUIRED
                        type            CDATA       #IMPLIED>
<!ELEMENT type          (#PCDATA)>
```

4.6. Oefeningen

1. Breid het XML document van Voorbeeld 4-8 uit zodat deze aan de DTD in Voorbeeld 4-18 voldoet.
2. Stel een Document Type Definition op voor het XML document getoond in Voorbeeld 4-19. Wees zo volledig mogelijk! Let erop dat elk item een uniek ID moet hebben.

Voorbeeld 4-19. Product lijst van een winkel

```
<!-- producten lijst -->
<catalog>
   <!-- begin boeken lijst -->
   <productgroep naam="boeken">
      <item nr="nr01">
         <titel>De Muur</titel>
         <auteur>J.P. Sartre</auteur>
         <prijs waarde="1050" munteenheid="BEF"/>
      </item>
      <item nr="nr02">
         <titel>De Nachtvogels</titel>
         <auteur>Jef Geeraerts</auteur>
         <prijs waarde="1470" munteenheid="BEF"/>
      </item>
      <item nr="nr03">
         <titel>De Varkensput</titel>
         <auteur>Willy Spillebeen</auteur>
         <prijs waarde="1225" munteenheid="BEF"/>
      </item>
      <item nr="nr04">
         <titel>De Vreemdeling</titel>
```

```
         <auteur>Albert Camus</auteur>
         <prijs waarde="890" munteenheid="BEF"/>
      </item>
   </productgroep>

   <!-- begin cd lijst -->
   <productgroep naam="cds">
      <item nr="nr05">
         <titel>De Nachten</titel>
<uitvoerder>Sofia</uitvoerder>
         <prijs waarde="650" munteenheid="BEF"/>
      </item>
      <item nr="nr06">
         <titel>The Battle of Los Angeles</titel>
         <uitvoerder>Rage Against the Machine</uitvoerder>
         <prijs waarde="700" munteenheid="BEF"/>
      </item>
      <item nr="nr07">
         <titel>Floodland</titel>
         <uitvoerder>The Sisters of Mercy</uitvoerder>
         <prijs waarde="500" munteenheid="BEF"/>
      </item>
  </productgroep>


   <!-- begin video lijst -->
   <productgroep naam="video">
      <item nr="nr08">
         <titel>Event Horizon</titel>
<categorie>science fiction</categorie>
         <prijs waarde="600" munteenheid="BEF"/>
      </item>
      <item nr="nr09">
         <titel>Three to Tango</titel>
         <categorie>komedie</categorie>
         <prijs waarde="650" munteenheid="BEF"/>
      </item>
      <item nr="nr10">
         <titel>Star Trek: First Contact</titel>
         <categorie>science fiction</categorie>
         <prijs waarde="550" munteenheid="BEF"/>
      </item>
   </productgroep>
</catalog>
```

3. Maak een DTD voor cursus beschrijvingen zoals die in de studiegids voorkomen. Test deze DTD door een XML beschrijving van dit vak op te maken en te valideren aan de hand van de DTD.
4. Bestudeer het BibTeXML (http://bibtexml.sourceforge.net/) schema. BibTeXML is een XML-formaat voor de BibTeX (http://www.ecst.csuchico.edu/~jacobsd/bib/formats/bibtex.html) syntax. Maak een voorbeeld bibliografie entry voor deze cursus in BibTeX en in BibTeXML.

5. Stel een XML document op dat al de referenties opgenomen achteraan in deze cursus (*Bibliografie*) bevat. Het XML document moet voldoen aan het BibTeXML Schema (http://bibtexml.sourceforge.net/).

Noten

1. Business 2 Business
2. http://openscience.chem.nd.edu/~egonw/svc/

Hoofdstuk 5. XML Schema

W3C XML Schema specificatie (http://www.w3.org/XML/Schema): http://www.w3.org/XML/Schema

DTD is niet de enige technologie waarmee we de structuur van een XML document kunnen beschrijven. Een nieuwere technologie voor het valideren van XML documenten is XMLSchema (http://www.w3.org/XML/Schema). XMLSchema wordt vaak gezien als de opvolger van DTD, en de nieuwe standaard schema-taal voor XML documenten. Naast alle functies die aangeboden worden door DTD ondersteunt XMLSchema ook nog de mogelijkheid om complexere datatypes voor ieder element en attribuut aan te duiden. XMLSchema ondersteunt meer dan 40 datatypes (in tegenstelling tot de 10 datatypes die ondersteund worden door DTD). Het is zelfs mogelijk om restricties of uitbreidingen te definiëren voor bestaande datatypes. Men zou bijvoorbeeld kunnen uitdrukken dat de inhoud van een element een integer moet zijn tussen 0 en 12000. Een ander voordeel van XMLSchema is dat het gebruik maakt van de XML syntax zodat een XMLSchema dezelfde voordelen biedt als een gewoon XML document. Bovendien hoeft men dan geen nieuwe taal te leren om een schema te schrijven.

Het volgende voorbeeld laat een XMLSchema zien voor XML documenten die een adresboek voorstellen.

Voorbeeld 5-1. Eenvoudig XMLSchema voor een adresboek

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
 <xsd:element name="adresboek">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="naam" minOccurs="1" maxOccurs="unbound">
     <xsd:complexType>
      <xsd:sequence>
      <xsd:element name="voornaam" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="achternaam" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="straat" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="city" type="xsd:string" minOccurs="1" maxOccurs="1"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Hier zien we dat gebruik gemaakt wordt van een *namespace prefix* (xsd) om te laten zien dat het gaat om elementen die gedefinieerd zijn in de XMLSchema specificatie. In het voorbeeld worden 4 basiselementen van XMLSchema geillustreerd:

xsd:schema

Dit element is de wortel van een XMLSchema en is een verplicht element om een XMLSchema te beginnen

xsd:element

Met dit element wordt een nieuw element gedefinieerd. Dit element kan verschillende attributen hebben:

name

De naam van het element

type

Het type van het element. Dit kan een XMLSchema simple type (zoals xsd:integer), een user-defined type of de naam van een complexType zijn

minOccurs
maxOccurs

Hoeveel keer het element minimaal en maximaal mag voorkomen. De waarden kunnen gaan van 1 tot unbound. Als maxOccurs en minOccus 1 zijn betekent dit dat het element juist 1 keer moet voorkomen en dan mogen deze attributen weggelaten worden

xsd:complexType

Dit element wordt gebruikt om aan te geven dat het parent element andere elementen als kinderen moet hebben. Een mogelijke parameter van xsd:complextype is name. Deze naam kan dan gebruikt worden in het type attribuut van xsd:element om naar een complextype te verwijzen.

xsd:sequence

Dit element geeft aan dat de elementen in een complex type sequentieel moeten voorkomen. Een andere mogelijkheid hier is xsd:choice waarbij er een keuze kan gemaakt worden tussen de kind elementen.

Ondanks de vooruitgang die XMLSchema zeker is ten opzichte van DTD, heeft ook XMLSchema verschillende gebreken. Zo is het niet mogelijk een verzameling elementen in een willekeurige volgorde toe te laten. Bovendien is het gebruik van XML als syntax niet altijd een voordeel: het schema wordt al snel onoverzichtelijk en moeilijk onderhoudbaar.

5.1. Oefeningen

1. Maak een XMLSchema voor een lijst van GSM nummers

2. Bestudeer het XMLSchema van BibTeXML. Hoe worden groepen gedefinieerd? Geef een opsomming van elementen die in de bovenstaande tekst niet uitgelegd worden. Wat is volgens jou (intuïtief) de betekenis van die elementen?
3. Stel zelf een XMLSchema op voor het XML document getoond in Voorbeeld 4-19.

Hoofdstuk 6. XPath

W3C XPath specificatie (http://www.w3.org/TR/xpath)

6.1. Inleiding

XPath is een w3c standaard om nodes, attributen,... te vinden in een XML document. XPath is eigenlijk een soort query taal om delen van een XML document te lokaliseren in de XML boom. Met behulp van XPath kan je vragen stellen als “geef me het attribuut van het vijfde kind-node van al de nodes met de naam “blaai” of “Geef me al de sectie nodes waarbij in de titel het woord ’Babel’ voorkomt”.

XPath wordt uitvoerig gebruikt in de taal XSLT (zie het volgende hoofdstuk), en wordt meestal ook beschouwd als een onderdeel van XSLT. De specificatie van XPath is wel apart van XSLT. Om deze reden, en omdat XPath een uitgebreide en krachtige manier is om delen van een XML document te lokaliseren besteden we een apart hoofdstuk hieraan. Een XPatch expressies kan als output een van de volgende types geven:

1. node-set; een verzameling nodes zonder duplicaten
2. boolean
3. kommagetal
4. string

6.2. XPath Syntax

Een XPath expressie bestaat uit 3 delen:

Axis
: definieert de richting van navigatie doorheen de boom

Node test
: filtert bepaalde element types uit de verzameling elementen die de axis aangeeft

Predicate
: een expressie die bepaalde elementen selecteert uit de verzameling aangereikte elementen. De geselecteerde elementen voldoen aan de expressie die de predikaat expressie oplegt

Samengevoegd ziet de XPath syntax er als volgt uit:

```
Axis::Node test[Predicate]
```

Je kan ook meerdere XPath queries na elkaar hangen en deze “samenstellen”. De verschillende stukken worden dan gescheiden door ’/’:

```
Axis::Node test[Predicate]/Axis::Node test[Predicate]/Axis::...
```

Elke stap werkt dan verder op het resultaat van de vorige stap, dit kan omdat het resultaat altijd terug een node set is. Als je helemaal vooraan “/“ erbij plaatst, dan geef je aan dat je XPath expressie bij de root van het document begint.

Om een duidelijker beeld te geven van een XPath expressie geven we reeds enkele voorbeelden. Deze voorbeelden dienen louter ter illustratie van de syntax, de semantiek ervan zal verduidelijkt worden in de rest van dit hoofdstuk. Met behulp van het volgende XML document zullen de voorbeelden gegeven worden:

Voorbeeld 6-1. Biblio beschrijving in XML

```
<bibliography>
    <title>Bibliografie</title>
      <biblioentry ID="25">
         <abbrev>xslt-pr</abbrev>
         <authorgroup>
            <author>
               <firstname>Michael</firstname>
               <surname>Kay</surname>
            </author>
         </authorgroup>
         <copyright>
            <year>2001</year><holder>Wrox Press</holder>
         </copyright>
         <publisher>
             <publishername>Wrox Press</publishername>
         </publisher>
         <isbn>1-861005-06-7</isbn>
         <title>XSLT, Programmer’s Reference</title>
      </biblioentry>
      <biblioentry ID="26">
         <abbrev>latex-comp</abbrev>
         <authorgroup>
           <author>
              <firstname>Michel</firstname>
              <surname>Goossens</surname>
           </author>
           <author>
              <firstname>Frank</firstname>
              <surname>Mittelbach</surname>
           </author>
           <author>
               <firstname>Alexander</firstname>
               <surname>Samarin</surname>
```

```
            </author>
          </authorgroup>
          <copyright>
             <year>2000</year><holder>Addison-Wesley</holder>
          </copyright>
          <publisher>
             <publishername>Addison-Wesley</publishername>
          </publisher>
          <isbn>0-201-554199-8</isbn>
          <title>The LaTeX Companion</title>
        </biblioentry>
        ...
</bibliography>
```

//author[firstname="Michael"]/surname

Selecteert de achternaam van de auteur(s) met als voornaam “Michael”

/bibliography/biblioentry[@ID > 17]

Geeft alle biblioentry nodes waarvan het ID groter is dan 17

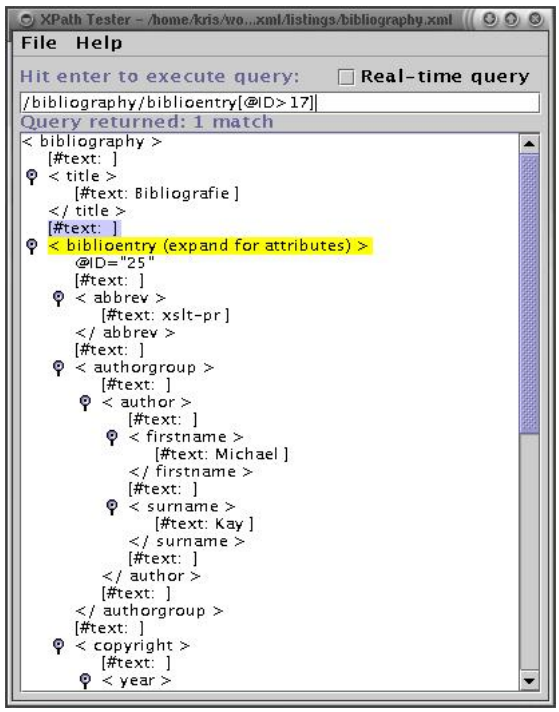
/bibliography/biblioentry[position() = 5]/isbn

Geeft het isbn van de vijfde boek

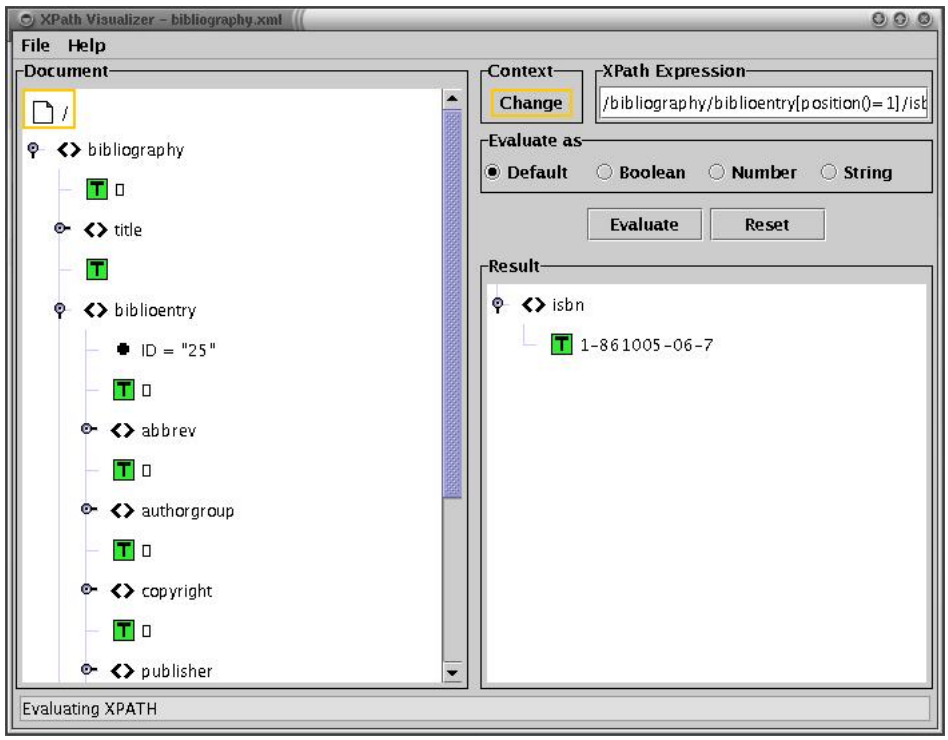
Dit zijn slechts enkele simpele voorbeelden van XPath expressies. We zullen later zien dat er heel wat ingewikkelde en zeer krachtige XPath expressies mogelijk zijn.

Een handige tool om je XPath queries eens uit te testen is XPath Tester (http://www.fivesight.com/downloads/xpathtester.asp) van FiveSight Technologies (http://www.fivesight.com/). Het is gratis (en open source[1]) af te halen van het Internet. Naast deze applicatie zijn er nog een hele resem andere beschikbare tools om XPath queries te testen, zoals XPath Visualiser (http://www.logilab.org/xpathvis/) van Logilab (http://www.logilab.org/). Deze applicatie is eveneens gratis en open source [2]. Een van de redenen waarom het handig is om zo een tool te gebruiken is dat je op voorhand je XPath queries kan testen en dat je tevens een overzicht krijgt van de boomstructuur van je XML document. Sommige applicaties kunnen de XPath expressie evalueren terwijl je deze intypt, wat een beter inzicht in de werking en de correctheid van de query bevordert. Het is aangeraden om eens een grafische applicatie te gebruiken en hiermee XPath querying in te oefenen.

Figuur 6-1. XPath Tester



Figuur 6-2. XPath Visualiser



6.3. Axis

De axis definieert de richting van de navigatie doorheen de boom. Een verzameling keywords worden gedefinieerd om dit te doen (de node van waaruit de XPath expressie wordt uitgevoerd wordt hier “bron-node” genoemd):

self

Selecteert de bron-node zelf

child

Selecteert al de rechtstreekse kinderen van de bron-node

descendant

Selecteert al de afstammelingen van de bron-node

parent

Selecteert de parent node van de bron-node

ancestor

Selecteert al de voorouders van de bron-node

attribute

Selecteert de attribuut nodes van de bron-node

namespace

Selecteert al de namespace nodes die zich in "het gezichtsveld" van de bron-node bevinden

following-sibling

Selecteert al de *volgende* siblings van de bron-node die zich op hetzelfde niveau in de boom bevinden

preceding-sibling

Selecteer al de *vorige* siblings van de bron-node die zich op hetzelfde niveau in de boom bevinden

following

Selecteert al de volgende nodes, maar niet de afstammelingen van de bron-node

preceding

Selecteert al de voorgaande nodes, maar niet de voorouders van de bron-node

descendant-or-self

Selecteert al de nakomelingen van de bron-node *en* de bron-node zelf

ancestor-or-self

Selecteert al de voorouders van de bron-node *en* de bron-node zelf

6.4. Node tests

Terwijl de axis de richting definieert waarin gezocht moet worden, filteren de Node tests de verzameling die de axis aangeeft. De Node test filtert op node type, of op de naam van de node. De mogelijke Node tests zijn:

*

Matcht alle elementen die aangegeven worden in de axis

E

Matcht alle elementen met de naam E

node()

Matcht enkel de nodes

text()

Matcht alle text elements

comment()

Matcht alle commentaar elementen

processing-instruction()

Matcht alle processing instructions

6.5. Predicates

Een predikaat is een expressie die bepaalde elementen selecteert uit de verzameling aangereikte elementen. De geselecteerde elementen voldoen aan de expressie die de predikaat expressie oplegt. De predikaat expressie wordt geëvalueerd ten opzichte van de context node. De context node is de node van waaruit het “predikaat” wordt uitgevoerd. Deze context node kan verschillend zijn afhankelijk van de verschillende selectie-*stappen* in de XPath expressie. De uitvoering van de XPath stap ten opzichte van de context node geldt niet enkel voor een predikaat, maar ook voor de axis en de node-test.

Belangrijk: Het begrip *context* zal nog verschillende keren gebruikt worden. Om dit wat te verduidelijken kan je XPath vergelijken met het browsen door directories via een command line interface. Stel dat je in je home directory `/home/kris` een directory aanmaakt voor je xml oefeningen `/home/kris/xmloef`. Als de huidige directory de root directory is (*/*) en je wil naar je home directory gaan geef je dit aan als volgt: `cd /home/kris`, de “context” waarin je je dan bevindt is `/home/kris` en niet meer `/` zoals voor het uitvoeren van het commando. Als je verder wil gaan naar de directory `xmloef` moet je het `cd` commando dan ook tegen de huidige context uitvoeren, namelijk `cd xmloef`. Als je `cd home/kris/xmloef` typt dan lukt dit niet, maar als je “context” nog altijd `/` zou geweest zijn, dan was dit commando wel gelukt.

Ongeveer hetzelfde principe geldt voor de context waarin een XPath expressie uitgevoerd wordt: dit hang af van de node van de boom waar het programma zich op dat moment bevindt. Vanaf die plaats kan men dan een XPath expressie uitvoeren. Indien nodig kan men wel altijd vanaf de root node een XPath expressie uitvoeren (zoals `/home/kris/xmloef` ook zou gelukt zijn, zelfs als de context `/home/kris` geweest was). Meer gedetailleerde uitleg over de context van de evaluatie kan je in de XPath specificatie (http://www.w3.org/TR/xpath20/#eval_context) vinden.

Het predikaat zelf is ofwel een booleaanse expressie of een numerieke expressie. De expressie komt voor tussen de hoekhaken [en], na de node test. Voor predikaten samen te stellen worden de voorgedefinieerde functies van XPath en XSLT gebruikt. Een beperkte deelverzameling van alle

mogelijke functies worden opgesomd in de volgende sectie. Voor de overige functies moet beschikbare documentatie geraadpleegd worden.

6.6. Voorgedefinieerde functies

In deze sectie geven we een overzicht van de functies die in de *Core Function Library* (http://www.w3.org/TR/xpath#corelib) voorkomen. Er wordt in dit overzicht geen volledigheid nagestreefd, er worden wel enkele handige functies uit de bestaande functies gelicht en besproken. Volledigere en formelere verklaringen vind je terug in de XPath specificatie (http://www.w3.org/TR/xpath).

6.6.1. Node set functies

last():number

Geeft de grootte van de context: dit bedoelt meestal[3] het aantal nodes in de node set momenteel in de context. Een mogelijke aanwending is als van een verzameling nodes, degene die laatste komt anders behandelt moet worden.

position():number

Geeft weer welke node we momenteel aan het verwerken zijn, rekening houdend met de context.

count(node-set):number

Telt het aantal nodes in “node-set”.

6.6.2. String functies

string(object?):string

Geeft een string representatie van het object weer indien er een argument meegegeven werd, anders geeft deze functie een string representatie terug van de huidige context node.

concat(string, string, string*):string

Geeft een concatenatie terug van al de argumenten.

starts-with(string, string):boolean

Geeft *waar* terug indien de eerste string start met de tweede string, anders wordt *vals* teruggegeven.

contains(string, string):boolean

Geeft *waar* terug indien de eerste string de tweede bevat, anders wordt *vals* teruggegeven.

substring-before(string, string):string

Indien string 2 in string 1 voorkomt, geeft deze functie het gedeelte van string 1 *voor* string 2 terug, anders wordt een lege string teruggegeven. Bijvoorbeeld:
`substring-before("docbook","book")==doc`.

substring-after(string, string):string

Indien string 2 in string 1 voorkomt, geeft deze functie het gedeelte van string 1 *achter* string 2 terug, anders wordt een lege string teruggegeven. Bijvoorbeeld:
`substring-after("docbook","doc")==book`

substring(string, number, number?):string

Geeft de substring van string 1 terug vertrekkende vanaf het karakter op plaats number, en indien er een derde argument (2e number) meegegeven wordt, met lengte number 2. Indien er geen derde argument meegegeven wordt zal de substring alle karakters vanaf karakter op plaats number 1 tot aan het einde van de string bevatten.

string-length(string?):number

Geeft de lengte van een string weer: indien er een argument meegegeven wordt zal de lengte van die string teruggegeven worden, anders de lengte van de context node.

normalize-space(string?):string

Verwijdert voorafgaande spaces en spaces achteraan de string.

translate(string, string, string):string

Vervangt karakters in de eerste string, waarbij de te vervangen karakter(s) aangegeven worden door string 2 en de vervangingskarakters door string 3.

6.6.3. Boolean functies

boolean(object):boolean

Probeert een boolean voor het object terug te geven.

not(boolean):boolean

Klassieke not operatie op een boolean:

`not(true)=false`
en

`not(false)=true`

true():boolean

Geeft `true` terug.

false():boolean

Geeft `false` terug

XPath definieert de operatoren `and`, `or` en `not` om met booleaanse waarden te werken. `and` en `or` zijn infix operatoren die op de klassieke manier werken op twee booleaanse waarden. `not` is de logische notatie, die kan toegepast worden als een functie op een booleaanse waarde.

6.6.4. Number functies

number(object?):number

Probeert een getal terug te geven voor object, indien er een argument meegegeven werd. Anders probeert deze functie een getal voor de context node terug te geven

sum(node-set):number

Geeft de som terug van alle nummers (eventueel na conversie) die in node-set zitten

floor(number):number

Geeft het grootste geheel getal kleiner dan het argument terug.

ceiling(number):number

Geeft het kleinste geheel getal groter dan het argument terug.

round(number):number

Geeft een gehele afronding terug van het argument, volgens de klassieke afrondingsregels.

6.6.5. Vergelijkingsoperatoren

Er zijn ook vergelijkingsoperatoren die je kan gebruiken in de XPath expressie: =, !=, >, <, <=, >=. Deze operatoren behouden hun klassieke betekenis uit andere programmeertalen. Ze kunnen gebruikt worden

om nummers, strings, booleans,... mee te vergelijken.

Vergelijkingen met node verzamelingen

Als een van de argumenten een verzameling nodes is, worden de regels iets moeilijker. Raadpleeg de specificatie indien u een zulke vergelijking nodig heeft om het juiste effect hiervan te weten te komen. Meestal komt het erop neer dat de node-set zal geconverteerd worden naar het primitieve type van de andere operand.

6.7. Voorbeelden

De voorbeelden in deze sectie zijn gebaseerd op de XML listing Voorbeeld 6-1. We zullen dadelijk voorbeelden van een redelijke complexiteit geven. De student wordt sterk aangeraden zelf enkele simpele voorbeelden uit te werken.

Waarschuwing

Soms zijn de XPath voorbeelden te lang om op een enkele lijn te zetten in deze cursus. In deze gevallen gaat de XPath expressie op de volgende lijn verder. Bij gebruik moet dit echter een aaneensluitende lijn zijn. Uit de context van het voorbeeld zal blijken of de lijn nog verder gaat of niet.

Voorbeeld 6-2. Alle auteurs van een LaTeX boek

```
descendant-or-self::author[contains(parent::node()/
parent::node()/child::title,'LaTeX')]
```

De eerste listing (Voorbeeld 6-2) selecteert alle auteurs die een boek hebben geschreven waar het woord “LaTeX” in de titel voorkomt. In een stap-voor-stap analyse van deze XPath expressie bekijken we eerst de buitenste axis:*descendant-or-self* , dit geeft aan dat in de richting van alle child nodes van de context node en de context node zelf moet gezocht worden. Vervolgens hebben we de node-test: *author* zegt dat we enkel geïnteresseerd zijn in de author nodes. Welke author nodes juist vinden we in het booleaanse predikaat tussen de hoekhaken: *contains(parent::node()/parent::node()/title,'LaTeX')*. De functie contains checkt hier of de string “LaTeX” voorkomt in de titel geassocieerd met de author (die bij het checken van het predikaat de context node is). Om de titel te gaan ophalen moeten we eerst twee niveaus stijgen (tweemaal na elkaar *parent*) waarbij we aangeven dat we enkel geïnteresseerd zijn in nodes door in de node test de functie *node()* te zetten. Tenslotte wordt er afgedaald naar de node title, en wordt de inhoud daarvan door de contains functie vergeleken met de opgegeven string.

Stel dat we niet de nodes van alle auteurs willen, maar wel het totale aantal, dan kunnen we simpelweg de functie count toevoegen die het aantal nodes in een nodeset telt:

Voorbeeld 6-3. Het totale aantal auteurs die een LaTeX boek geschreven hebben

```
count(descendant-or-self::author[contains(parent::node()/
parent::node()/child::title,'LaTeX'))]
```

Voorbeeld 6-4. Het aantal XML boeken met een copyright in het jaar 2000 *en* in 1996

```
count(descendant-or-self::biblioentry[child::copyright/child::year=
'2000'| child::copyright/child::year='1996'])
```

Een stap-voor-stap analyse van deze XPath expressie (Voorbeeld 6-4) beschouwt eerst de axis: *descendant-or-self*, en we selecteren alle biblioentry nodes uit deze verzameling van nodes die de axis aangeeft. Vervolgens testen we of de nodes *aan 1 van de* condities voldoet in het predikaat. Hiervoor gebruiken we de unie operatie '|': ofwel is het copyright jaar 2000 ofwel is het 1996, geeft ons alle nodes die minstens aan 1 van de twee condities voldoen.

Let op

'|' staat *niet* voor de logische *of* operatie, enkel voor de unie. Als het om booleaanse types gaat kunnen ook de logische operatoren `and`, `or` en `not` gebruikt worden, waarbij `not` werkt als een functie op een argument, terwijl `and` en `or` in de infix notatie kunnen gebruikt worden. Bijvoorbeeld:

Voorbeeld 6-5. Het aantal XML boeken met een copyright in het jaar 2000 *en* in 1996

```
count(descendant-or-self::biblioentry[
child::copyright/child::year='2000' and
child::copyright/child::year='1996'])
```

Dit geeft natuurlijk altijd 0 terug. Indien we of (or) gebruikt hadden, was het resultaat hetzelfde als bij het nemen van de unie.

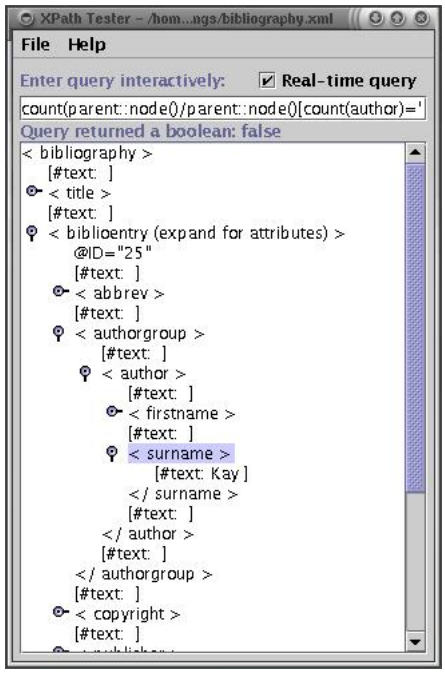
Voorbeeld 6-6. Een XPath expressie met een boolean resultaat

```
parent::node()/parent::node()[count(child::author)='2' and
child::author/child::surname='Kay']
```

Opdat deze XPath expressie true zou teruggeven, is het ten eerste nodig dat de context node van waar deze expressie uitgevoerd wordt, twee niveaus lager is dan de node authorgroup (dit wordt aangegeven

door parent::node()/parent::node()). Met andere woorden, moet de context node surname of firstname zijn.

Figuur 6-3. XPath Tester



Voorbeeld 6-7. Attribuut selectie

```
sum(descandant-or-self::node()/attribute::ID)
```

Als laatste voorbeeld, een XPath expressie (Voorbeeld 6-7) die ook een attribuut van een element gebruikt. Deze expressie geeft, indien de context node de root van de boom is, de som van de waardes van alle attributen met als naam "ID" (van eender welk element die een dergelijk attribuut heeft). Probeer zelf uit te vissen wat de volgende XPath expressie (Voorbeeld 6-8) als resultaat heeft, en onder welke condities.

Voorbeeld 6-8. Attribuut selectie

```
sum(descandant-or-self::
biblioentry[count(child::authorgroup/child::author)>2]/attribute::ID)
```

6.8. Afkortingen

Complexe XPath expressies hebben de neiging zeer lange uitdrukkingen te worden. Daarom worden er enkele mogelijke *afkortingen* gedefinieerd. Deze sectie geeft een opsomming van veel gebruikte afkortingen

> **Tip:** Gebruik pas afkortingen als je voldoende ervaring hebt met XPath expressies. Dadelijk gebruik maken van de mogelijke afkortingen kan (en zal) tot verwarring leiden.

In plaats van naar de rechtstreekse kinderen van een node te verwijzen met *child::*, kan dit gewoon weggelaten worden. De volgende twee XPath expressies zijn dus equivalent met elkaar:

Voorbeeld 6-9. child::

```
count(descendant-or-self::biblioentry[child::copyright/child::year='2000'
and child::copyright/child::year='1996'])
count(descendant-or-self::biblioentry[copyright/year='2000' and
copyright/year='1996'])
```

Een andere veelgebruikte afkorting is *descendant-or-self::node()* te vervangen door *//*. De volgende twee XPath expressies zijn dus equivalent met elkaar.

Voorbeeld 6-10. descendant-or-self::node()

```
count(descendant-or-self::biblioentry[copyright/year='2000' and
copyright/year='1996'])
count(//biblioentry[copyright/year='2000' and copyright/year='1996'])
```

attribute:: kan afgekort worden door @. De volgende twee XPath expressies zijn ook equivalent met elkaar:

Voorbeeld 6-11. attribute::

```
sum(descandant-or-self::biblioentry[
count(child::authorgroup/child::author)>2]/attribute::ID)
sum(//biblioentry[count(authorgroup/author)>2]/@ID)
```

De volgende afkorting is voor *self::node()*, dit kan vervangen worden door ".". De volgende twee XPath expressies zijn dus equivalent:

Voorbeeld 6-12. self::node()

```
self::node()/attribute::ID
./@ID
```

Een soortgelijke afkorting als voor self::node() bestaat er ook voor *parent:node()*, namelijk "..". De volgende twee XPath expressies zijn dus equivalent:

Voorbeeld 6-13. parent::node()

```
parent::node()/parent::node()[count(child::author)='2' and
child::author/surname='Kay']
../..[count(author)='2' and author/surname='Kay']
```

De laatste afkorting die we hier laten zien, is een specifieke afkorting voor *[position()=x]*, wat afgekort kan worden tot *[x]*. De volgende twee expressies zijn dus equivalent:

Voorbeeld 6-14. [position()=x]

```
/descandant-or-self::bibliography/child::biblioentry[position() = 5]/
child::isbn
//bibliography/biblioentry[5]/isbn
```

6.9. Oefeningen

Voor de XPath oefeningen dient er van een van de twee voorgestelde applicaties (Figuur 6-1, Figuur 6-2) gebruik gemaakt te worden. Haal ook de DocBook code van deze tekst af op de website (http://lumumba.luc.ac.be/~kris/courses/xml/), we gebruiken het bestand `xpath.xml`.

1. Localiseer alle voorbeelden (<example></example>) die in dit hoofdstuk voorkomen.
2. Tel al de paragrafen die in dit hoofdstuk voorkomen.
3. Tel hoeveel paragrafen de eerste, de derde, de vijfde en de zevende sectie samen hebben.
4. Localiseer alle mogelijke voorgedefinieerde functies die in dit hoofdstuk voorkomen.
5. Tel alle mogelijke voorgedefinieerde functies die in dit hoofdstuk voorkomen.

Deze oefeningen zijn enkel inleidende oefeningen. In het hoofdstuk over XSLT zal XPath verder gebruikt worden.

Noten

1. Apache Public License
2. GNU General Public License
3. last wordt anders gedefinieerd in de XSLT specificatie, maar heeft het over hetzelfde

Hoofdstuk 7. XSL: eXtensible Stylesheet Language

w3c XSL specificatie (http://www.w3.org/Style/XSL)

Verplichte lectuur: *What kind of language is XSLT?*
(http://www-106.ibm.com/developerworks/xml/library/x-xslt/), Michael Kay

7.1. Inleiding

XSL is een taal die stylesheets in XML uitdrukt. De originele betekenis van een stylesheet is een soort van “theme” op de inhoud plaatsen. Een stylesheet behoudt de inhoud van een document, maar laat het uiterlijk ervan veranderen. Het principe van XSL is hetzelfde, maar gaat een stap verder in de beschikbare functionaliteit. In dit hoofdstuk zullen we niet heel de XSL specificatie bespreken, maar ons beperken tot een deel hiervan: XSLT. Hierbij zal XPath ook een zeer belangrijke rol spelen.

De w3c XSL specificatie bestaat uit drie verschillende delen:

XSL:FO

: XSL Formatting Objects; beschrijft hoe een XML document kan gepresenteerd worden. Hiermee zullen we ons *niet* bezighouden.

XSLT

: XSL Transformations; beschrijft hoe we de element grove van een XML document kunnen aanpassen. XSLT is een transformatietaal voor XML documenten. In dit hoofdstuk zullen we ons in XSLT verdiepen.

XPath

: XPath is een taal om delen van een document te lokaliseren. Zie het vorige hoofdstuk voor meer uitleg hierover.

XSL is een taal die de *structuur* van een XML document beïnvloedt, maar *niet de inhoud*. Een XSL document is zelf een XML document, en kan dus gevalideerd worden met behulp van een DTD of een XML Schema document. De XSL standaard definieert tags die in de XSL namespace voorkomen en die voor een XSL stylesheet kunnen gebruikt worden. Op de meest simpele manier kunnen we een XSLT document voorstellen als een functie die als input een XML document neemt, welk eventueel voldoet aan een bepaalde DTD, en als output een document met dezelfde inhoud maar een andere structuur (dus eventueel voldoet aan een andere DTD). Merk op dat dit geen XML document hoeft te zijn, alhoewel XSLT wel oorspronkelijk voor XML output bedoeld was.

Het principe van de werking van een XSLT document berust nog altijd op het gebruik van stylesheets: *Vertrekkende van een XML document, wordt op dit document een XSLT document met templates*

toegepast, met als resultaat dat het output document eigenlijk het XSLT document is waarbij de templates vervangen werden door elementen van het XML document.

> **Opmerking:** XSLT en XPath zijn zwak getypeerde talen. De typecheck wordt enkel at runtime gedaan.

XSLT zorgt dus voor transformaties:

- Structurele transformaties
- Dynamisch aanmaken van documenten
- Transformatie naar een “rendition language” (zoals HTML)

Bovendien is XSLT helemaal conform met XML: het gebruikt enkel de XML syntax en moet zelf dus ook een well-formed document zijn.

In dit hoofdstuk beperken we ons tot een overzicht voor de belangrijkste begrippen van XSLT. Voor een grondige bespreking van XSLT verwijzen we door naar de W3C specificatie en het boek “XSLT, Programmer’s Reference” van Micheal Kay.

> **XSLT is XML:** Zoals reeds aangehaald is een XSLT document ook een XML document. Hieruit kan je afleiden dat een XSLT document dus ook als template voor een ander XSLT document kan dienen. Een XSLT document kan dan ook als template op zichzelf werken. Omdat voor XSLT documenten ook XML gebruikt wordt krijgen deze templates reflexieve eigenschappen.

7.2. XSLT Elementen: Elementen uit de XSL Namespace

Deze paragraaf geeft een overzicht van de voorgedefinieerde tags die in de xsl namespace voorkomen. Enkel de belangrijkste tags met betrekking tot XSLT worden besproken. De rest kan opgezocht worden in de XSLT specificatie (http://www.w3.org/TR/xslt20/).

<xsl:template>

> Kan een beetje als een functie, zoals we die ook kennen van de imperatieve programmeertalen, beschouwd worden. Het oproepen van een template kan op twee manieren gebeuren: via een expliciete oproep met behulp van “`call-template`” of met behulp van pattern matching, waardoor de template automatisch met een patroon geassocieerd wordt. Dit laatste gebeurt door “`apply-templates`” (zie verder). Indien je een template expliciet oproept met behulp van de naam van de template moet het name attribuut ingevuld zijn, indien je apply-templates gebruikt moet je het match attribuut met een geldige XPath expressie invullen.

Voorbeeld 7-1. xsl:template

```
<xsl:template name="qsort">
   ...
</xsl:template>

<xsl:template match="title">
   <h1><xsl:value-of select="."/></h1>
</xsl:template>

<xsl:template match="title" mode="toc">
<li><xsl:value-of select="."/></li>
</xsl:template>
```

Soms is het ook nodig dat je een element verschillende keren kan verwerken: een bekend voorbeeld is automatisch een table of contents genereren voor al de titels in je document. Dit kan je doen met behulp van het *mode* attribuut.

<xsl:apply-templates>

`apply-templates` zal, afhankelijk van de select statement, al de templates met de overeenkomende match statement selecteren en toepassen op de context node. De context node wordt aangegeven door het select statement.

Voorbeeld 7-2. xsl:apply-templates

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

In Voorbeeld 7-2 worden er bij de `apply-templates` alle title nodes geselecteerd. Dit zou tot gevolg hebben dat voor elke title node de template met het attribuut match gelijk aan “title” uit Voorbeeld 7-1 zou toegepast worden op deze node.

match=“/”: De node <xsl:template match="/"> match met de root van het document. Dit kan dus beschouwd worden als het startpunt vanaf waar het XSLT document toegepast wordt. Merk echter op dat deze template nergens expliciet aangeroepen hoeft te worden. De reden hiervoor is dat er een ingebouwde template rule bestaat die automatisch uitgevoerd wordt, zoals afgebeeld in Voorbeeld 7-3.

Voorbeeld 7-3. Built-in Template Rule

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

xsl:apply-templates zorgt ervoor dat naar alle xsl:template nodes gezocht wordt die matchen met de huidige context node: de root van het document in dit geval. Er zijn nog enkele andere built-in rules, hiervoor verwijzen we door naar de XSLT specificatie.

<xsl:call-template>

De `call-template` werkt zoals een klassieke functie oproep in een imperatieve programmeertaal. Een template wordt expliciet opgeroepen op basis van het ingevulde naam-attribuut van de template. Deze template wordt dan uitgevoerd, waarbij de context node bij het begin van de target template dezelfde is als de context bij het oproepen van die template.

Voorbeeld 7-4. xsl:call-template

```
<xsl:call-template name="makeTOC"/>
```

<xsl:value-of>

Output de waarde die in de bijbehorende XPath expressie geselecteerd wordt. Stel je voor dat er een element titel voorkomt bijv.:<titel>De Nachtvogels</titel>, dan geeft de `value-of` uit Voorbeeld 7-5 als uitvoer “De Nachtvogels”. Op de mogelijke XPath expressie staan geen beperkingen, zolang het maar een juiste expressie is. Als de expressie een NodeSet zou selecteren, dan zal *enkel* het eerste element uit die verzameling in de output verschijnen.

Voorbeeld 7-5. xsl:value-of

```
<xsl:value-of select="titel"/>
```

<xsl:with-param>

Bij het oproepen van een template kan je parameters meegeven. Hiervoor gebruik je de `xsl:with-param` tag als kinderen van de `xsl:apply-templates` of `xsl:call-template` node. Op deze manier kan je traditionele parameter passing ook toepassen bij het oproepen van templates. Aan de parameters wordt een naam gegeven en via een XPath expressie een waarde toegekend.

Voorbeeld 7-6. xsl:with-param

```
  ...
<xsl:call-template name="ggd">
   <xsl:with-param name="x" select="number($x)"/>
   <xsl:with-param name="y" select="number($y)-number($x)"/>
</xsl:call-template>
   ...
<xsl:call-template name="qsort">
   <xsl:with-param name="spil" select="nummer[1]"/>
   <xsl:with-param name="list" select="nummer[position()>1]"/>
</xsl:call-template>
```

<xsl:param>

Specifieert de parameters, gelijkaardig aan de parameters die bij een functie van een imperatieve programmeertaal meegegeven worden. Er hoeft enkel een naam voor de parameters gespecifieerd te worden. Deze parameters kunnen dan in de template body gebruikt worden door er een $-teken voor te plaatsen.

Voorbeeld 7-7. xsl:param

```
<xsl:template name="ggd">
   <xsl:param name="x"/>
   <xsl:param name="y"/>
   ...
   <ggd><xsl:value-of select="$x"/></ggd>
   ...
</xsl:template>
```

<xsl:if>

De traditionele if-test. Merk op dat er geen else aanwezig is; hier zal `xsl:choose` met `xsl:otherwise` voor gebruikt kunnen worden (zie verder). De if-tag gebruikt een XPath expressie als test-voorwaarde, indien nodig wordt het resultaat van deze expressie nog geconverteerd naar een boolean resultaat.

Voorbeeld 7-8. xsl:if

```
<xsl:if test="count($verzElementen)>0">
<teken>positief</teken>
</xsl:if>
```

<xsl:choose>
<xsl:when>
<xsl:otherwise>

Als er meerdere keuzes mogelijk zijn, cfr. switch-statement in Java, dan is er een speciale xsl constructie beschikbaar: `xsl:choose`. Met `xsl:when` wordt in de `xsl:choose` alle mogelijkheden afgegaan. Eventueel (maar niet verplicht!) kan als laatste kind van `xsl:choose` `xsl:otherwise` gebruikt worden om alle andere mogelijkheden op te vangen.

Voorbeeld 7-9. xsl:choose

```
<xsl:choose>
   <xsl:when test="number($x)&gt;number($y)">
      <xsl:call-template name="ggd">
         <xsl:with-param name="x" select="number($x)-number($y)"/>
         <xsl:with-param name="y" select="number($y)"/>
      </xsl:call-template>
```

```
   </xsl:when>
   <xsl:when test="number($x)&lt;number($y)">
      <xsl:call-template name="ggd">
         <xsl:with-param name="x" select="number($x)"/>
         <xsl:with-param name="y" select="number($y)-number($x)"/>
      </xsl:call-template>
   </xsl:when>
   <xsl:otherwise>
     <ggd><xsl:value-of select="$x"/></ggd>
   </xsl:otherwise>
```

<xsl:for-each>

Dit is de tegenhanger van de for-lus in imperatieve programmeertalen. Er is echter een belangrijk verschil: het `xsl:for-each` element itereert *niet* over een bepaald bereik, maar over de elementen in een verzameling (NodeSet)! De boom onder de `xsl:for-each` node wordt dan toegepast op elk element van de geselecteerde NodeSet. Deze NodeSet wordt geselecteerd met behulp van een XPath expressie. Dit wordt dan ook de context waarin de subboom van de `for:each` node toegepast wordt.

Voorbeeld 7-10. xsl:for-each

```
<xsl:for-each select="//example">
   <h2>Example:<xsl:value-of select="title"/></h2>
   <p><xsl:value-of select="programlisting"/></p>
</xsl:for-each>
```

for-each vs apply-templates: `xsl:for-each` geeft een intuïtieve benadering voor de programmeur die een achtergrond heeft in imperatieve programmeertalen. In feite is het een alternatief voor apply-templates, waarbij het gebruik van `xsl:apply-templates` een grotere flexibiliteit toelaat terwijl for-each de logica duidelijker maakt. Het gebruik van apply-templates wordt ook wel *push-processing* genoemd, en for-each *pull-processing*. push-processing omdat de nodes “gedelegeerd” worden, en pull-processing omdat de nodes “binnengehaald” worden.

<xsl:attribute>

Laat toe om attributen bij tags te voegen. Dit is krachtiger dan deze hard-coded erin te zetten, daar de attributen hiermee ook dynamisch ingevuld kunnen worden.

Voorbeeld 7-11. xsl:attribute

```
<A>
 <xsl:attribute name="HREF">
http://www.w3.org/TR/
 </xsl:attribute>
volg deze link
```

```
</A>
```

<xsl:stylesheet>

Dit is het top element van een stylesheet. Het geeft onder meer de namespace aan, meestal is dit xmlns:xsl="http://www.w3.org/1999/XSL/Transform" het stuk ":xsl" geeft aan dat al de elementen, die je gebruikt ivm XSLT in je stylesheet, voorafgegaan worden door "xsl:".

Voorbeeld 7-12. xsl:stylesheet

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

Meestal wordt als eerste kind van de xsl:stylesheet node de xsl:output node gebruikt: deze geeft aan welke type output de stylesheet genereert. De standaard mogelijkheden zijn xml, html en text.

<xsl:variable>

Alhoewel in XSLT het element xsl:variable gedefinieerd wordt, bestaan er eigenlijk geen echte variabelen. Met xsl:variable kan je een variabele creëeren waar je *éénmalig* een waarde aan kan toekennen. Nadien is het niet meer mogelijk deze variabele te veranderen (vandaar dat het geen echte variabele is dus). De belangrijkste reden hiervoor is om geen neveneffecten in XSLT toe te laten. Verschillende XSLT processoren bieden echter een work-around aan zodat dit wel zou kunnen. Het gebruik van deze specifieke work-arounds kan echter beter vermeden worden: men zou imperatief en niet meer functioneel met XSLT gaan werken. Bovendien zorgt de functionele benadering ervoor dat er zich geen neveneffecten kunnen voordoen, waardoor toekomstige XSLT processoren (in theorie) de mogelijkheid hebben om de toepassing van een XSLT document te parallelliseren.

Voorbeeld 7-13. xsl:variable

```
<xsl:variable name="nr" select="number($x)-number($y)"/>
```

Het voorbeeldje toont hoe aan *nr* het resultaat van een XPath expressie toegekend wordt door deze in te vullen in het *select* attribuut.

<xsl:result-document> (http://www.w3.org/TR/xslt20/#element-result-document)

Regelmatig dient een XSLT document de output in verschillende documenten te plaatsen. Voor dit doel is er het element xsl:result-document. Met het attribuut href kan men de naam van het output

document aangeven. Het attribuut format geeft aan welk het type van het output document is (HTML,XML,...).

Het voorbeeldje voor dit element toont hoe er een naam voor het output document uit het element gmr:Name gehaald wordt. In het hoofddocument wordt er dan een link (met behlup van de anchor tag) gelegd naar het document dat met xsl:result-document gecreëerd wordt.

Voorbeeld 7-14. xsl:result-document

```
<xsl:template match="gmr:Sheet" mode="clean">
 <xsl:variable name="filename"
               select="concat(translate(gmr:Name, '\/:-*() ', ''), '.html')"/>
  <li>
   <a>
    <xsl:attribute name="href">
     <xsl:value-of select="$filename"/>
    </xsl:attribute>
    <xsl:value-of select="gmr:Name"/>
   </a>
  </li>
  <xsl:result-document href="{$filename}" format="html">
    <head>
      <xsl:apply-templates select="." mode="sheetheader"/>
    </head>
    <body>
      <h2><xsl:value-of select="gmr:Name"/></h2>
        <table border="1">
          <xsl:apply-templates select="gmr:Cells/gmr:Cell"/>
        </table>
    </body>
  </xsl:result-document>
</xsl:template>
```

De XSLT specificatie definieert nog enkele mogelijke tags die we hier niet uit de doeken doen. Deze worden overgelaten aan de lezer indien deze nodig zouden zijn. De tags die nog in de XSLT specificatie zitten en hier niet besproken worden, zijn: xsl:sort, xsl:result-document, xsl:import, xsl:apply-imports, xsl:include, xsl:fall-back, xsl:attribute-set, xsl:element, xsl:key, xsl:message, xsl:text, xsl:number, xsl:script, xsl:preserve-space, xsl:namespace-alias, xsl:decimal-format, xsl:copy, xsl:copy-of, xsl:comment.

7.3. Voorbeeld: een simpele XML Transformatie

Stel dat je op je website je CV wil publiceren, en je hebt alle data opgeslagen in een XML file. We beperken ons hier tot het verwerken van de informatie over de universitaire studie.

Voorbeeld 7-15. Beschrijving van de universitaire studies in een CV, uitgedrukt mbhv XML

```
<studies>
   <schooljaar jaar="1998-1999">
      <school>Limburgs Universitair Centrum</school>
  <richting jaar="1">Informatica</richting>
  <uitslag>Voldoening</uitslag>
  </schooljaar>
  <schooljaar jaar="1999-2000">
      <school>Limburgs Universitair Centrum</school>
  <richting jaar="2">Informatica</richting>
  <uitslag>Onderscheiding</uitslag>
   </schooljaar>
   <schooljaar jaar="2000-2001">
   <school>University of Glasgow</school>
   <richting jaar="3">Computer Science</richting>
  <uitslag>Uitgesteld</uitslag>
   </schooljaar>
   <schooljaar jaar="2001-2002">
      <school>Limburgs Universitair Centrum</school>
  <richting jaar="3">Informatica</richting>
  <uitslag>Voldoening</uitslag>
   </schooljaar>
   <schooljaar jaar="2002-2003">
      <school>Limburgs Universitair Centrum</school>
  <richting jaar="4">Informatica</richting>
  <uitslag>Grote Onderscheiding</uitslag>
   </schooljaar>
</studies>
```

Het XML document in Voorbeeld 7-15 moet nu omgezet worden naar een gepast HTML document om het te tonen in een webbrowser. We kiezen ervoor om de inhoud in een tabel te gieten. Er worden twee mogelijke voorbeelden gegeven; let op de verschillen tussen de twee oplossingen: Voorbeeld 7-17 en Voorbeeld 7-16. In Voorbeeld 7-17 wordt er de zogenaamde *push* processing methode gebruikt: een regel-gebaseerde aanpak zeg maar. Voorbeeld 7-16 geeft een voorbeeld van *pull* processing: er wordt een for-each verkozen om de elementen te verwerken. Push processing kan best gebruikt worden als de structuur van het document onderhevig is aan wijzigingen of niet op voorhand bekend is, terwijl pull processing best werkt bij statische structuren.

Voorbeeld 7-16. Een mogelijk XSLT document voor de CV (pull processing)

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

<xsl:template match="/">
  <h1>Currivulum Vitae</h1>
  <xsl:apply-templates select="studies"/>
</xsl:template>

<xsl:template match="studies">
  <h2>studies</h2>
  <table border="1" cellpadding="5" cellspacing="0">
    <tr><th>Schooljaar</th>
    <th>richting</th>
    <th>school</th>
    <th>uitslag</th></tr>
    <tbody>
     <xsl:for-each select="schooljaar">
       <tr><xsl:call-template name="verwerkjaar"/></tr>
     </xsl:for-each>
   </tbody>
  </table>
</xsl:template>

<xsl:template name="verwerkjaar">
  <td><xsl:value-of select="@jaar"/></td>
  <td><xsl:value-of select="richting"/></td>
  <td><xsl:value-of select="school"/></td>
  <td><xsl:value-of select="uitslag"/></td>
</xsl:template>

</xsl:stylesheet>
```

Voorbeeld 7-17. Een mogelijk XSLT document voor de CV (push processing)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">
  <h1>Currivulum Vitae</h1>
  <xsl:apply-templates select="studies"/>
</xsl:template>

<xsl:template match="studies">
<h2>studies</h2>
```

```
  <table border="1" cellpadding="5" cellspacing="0">
    <tr><th>Schooljaar</th>
    <th>richting</th>
    <th>school</th>
    <th>uitslag</th></tr>
    <tbody>
      <xsl:apply-templates select="schooljaar"/>
    </tbody>
  </table>
</xsl:template>

<xsl:template match="schooljaar">
 <tr>
   <td><xsl:value-of select="@jaar"/></td>
   <td><xsl:value-of select="richting"/></td>
   <td><xsl:value-of select="school"/></td>
   <td><xsl:value-of select="uitslag"/></td>
  </tr>
</xsl:template>
</xsl:stylesheet>
```

Het tweede voorbeeldje (Voorbeeld 7-19) behandelt een lijst van vakken, en zet deze om in HTML. Het voorbeeldje is louter ter illustratie. Als input kan de listing in Voorbeeld 7-18 gebruikt worden.

Voorbeeld 7-18. XML listing van 2e trimester vakken

```
<?xml version="1.0"?>
<Vakken>
  <Vak>
    <Naam>Technologie van Multimediasystemen en -Software</Naam>
    <Docent>W. Lamotte</Docent>
  </Vak>
  <Vak>
    <Naam>Intelligente Systemen</Naam>
    <Docent>E. Postma</Docent>
  </Vak>
  <Vak>
    <Naam>Optimalisering</Naam>
    <Docent>R. Peeters</Docent>
  </Vak>
</Vakken>
```

Voorbeeld 7-19. XSLT document voor lijst van vakken in HTML te zetten

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">
 <html xmlns="http://www.w3.org/TR/xhtml1/strict">
  <head><title>Vakken</title></head>
  <body>
   <table border="1">
    <xsl:call-template name="TableContents"/>
   </table>
  </body>
 </html>
</xsl:template>

<xsl:template name="TableContents">
<xsl:for-each select="Vakken/Vak">
 <xsl:sort select=".//Naam"/>
  <xsl:choose>
   <xsl:when test="contains( Naam/text(), 'Multimedia' )">
    <xsl:call-template name="TableRow">
     <xsl:with-param name="RowStyle">italic</xsl:with-param>
    </xsl:call-template>
   </xsl:when>
   <xsl:otherwise>
    <xsl:call-template name="TableRow"/>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:for-each>
</xsl:template>

<xsl:template name="TableRow">
 <xsl:param name="RowStyle">normal</xsl:param>
 <tr style="font-style:{$RowStyle}">
  <th align="left">
   <xsl:number value="position()" format="1."/>
   <xsl:value-of select=".//Naam"/>
  </th>
   <td><xsl:value-of select=".//Docent"/></td>
 </tr>
</xsl:template>

</xsl:stylesheet>
```

7.4. Academische Voorbeelden

Bestudeer de volgende voorbeelden grondig en zorg ervoor dat je in staat bent oefeningen van dezelfde complexiteit op te lossen met het handboek langs je. Deze voorbeelden laten zien dat de kracht van XSLT veel verder gaat dan het traditionele toepassen van stylesheets.

7.4.1. Algoritme van Euclides

Het eerste voorbeeld bestaat uit een XSLT document dat als input een lijst van getal paren krijgt, en als output een lijst van de grootste gemene delers van deze getallen geeft. Merk op dat dit een zeer procedurale benadering is van het gebruik van XSL. Dit voorbeeld illustreert samen met het volgende enkel de mogelijkheden van XSLT en heeft tot doel de kracht van de taal kenbaar te maken. De mix van imperatieve en functionele aspecten maken XSLT erg krachtig, doch ook niet altijd even intuïtief.

Als input document wordt het volgende XML document aangereikt:

Voorbeeld 7-20. Input listing van getal paren

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="euclides.xsl"?>
<nummers>
   <paar>
      <nummer>30</nummer>
      <nummer>25</nummer>
   </paar>
   <paar>
      <nummer>40</nummer>
      <nummer>100</nummer>
   </paar>
   <paar>
      <nummer>39</nummer>
      <nummer>1000</nummer>
   </paar>
   <paar>
      <nummer>5550</nummer>
      <nummer>110000</nummer>
   </paar>
</nummers>
```

Hierop wordt het volgende XSLT document toegepast:

Voorbeeld 7-21. Het algoritme van Euclides in XSLT

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

```
   <xsl:template match="/">
      <nummers>
         <xsl:for-each select=".//paar">
            <paar>
               <nummer><xsl:value-of select="nummer[1]"/></nummer>
               <nummer><xsl:value-of select="nummer[2]"/></nummer>
               <xsl:call-template name="ggd">
                   <xsl:with-param name="x" select="nummer[1]"/>
                   <xsl:with-param name="y" select="nummer[2]"/>
               </xsl:call-template>
            </paar>
         </xsl:for-each>
      </nummers>
   </xsl:template>

   <xsl:template name="ggd">
      <xsl:param name="x"/>
      <xsl:param name="y"/>
      <xsl:choose>
         <xsl:when test="number($x)&gt;number($y)">
             <xsl:call-template name="ggd">
                <xsl:with-param name="x" select="number($x)-number($y)"/>
                <xsl:with-param name="y" select="number($y)"/>
             </xsl:call-template>
         </xsl:when>
         <xsl:when test="number($x)&lt;number($y)">
            <xsl:call-template name="ggd">
               <xsl:with-param name="x" select="number($x)"/>
               <xsl:with-param name="y" select="number($y)-number($x)"/>
            </xsl:call-template>
         </xsl:when>
         <xsl:otherwise>
            <ggd><xsl:value-of select="$x"/></ggd>
         </xsl:otherwise>
       </xsl:choose>
    </xsl:template>
</xsl:stylesheet>
```

Als output geeft dit dan:

```
<nummers>
   <paar>
      <nummer>30</nummer>
      <nummer>25</nummer>
      <ggd>5</ggd>
   </paar>
   <paar>
      <nummer>40</nummer>
      <nummer>100</nummer>
      <ggd>20</ggd>
   </paar>
   <paar>
```

```
      <nummer>39</nummer>
      <nummer>1000</nummer>
      <ggd>1</ggd>
   </paar>
   <paar>
      <nummer>5550</nummer>
      <nummer>110000</nummer>
      <ggd>50</ggd>
   </paar>
</nummers>
```

7.4.2. Quicksort in XSLT

De volgende XSLT kan gemapt worden op een XML die rijen getallen bevat. Voor elke rij wordt het quicksort algoritme gebruikt om deze rij te ordenen en zo in de output voor te stellen. Ondanks dat er meer code nodig is dan nodig zou zijn in talen als Lisp, Haskell of Prolog, is dit voorbeeld toch redelijk intuïtief. Merk op dat dit vooral te wijten is aan het expliciet oproepen van de template *qsort*. Vanwege het processing model dat de meeste XSLT processors gebruiken, wordt het moeilijk om enkel met template matching hetzelfde effect te bekomen. De toekenning van bepaalde waarden (zoals een lijst van nodes) aan de parameters van de template laat ons toe dit probleem op een redelijk imperatieve manier op te lossen. Enkel werken met template matching zou een meer functionele manier zijn.

Als input document wordt het volgende document aangereikt:

Voorbeeld 7-22. Input voor de Quicksort stylesheet

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="quicksort.xsl"?>
<nummers>
   <list>
      <nummer>696</nummer>
   </list>
   <list>
      <nummer>30</nummer>
      <nummer>25</nummer>
      <nummer>40</nummer>
      <nummer>100</nummer>
      <nummer>1000</nummer>
      <nummer>27</nummer>
      <nummer>1</nummer>
      <nummer>5550</nummer>
      <nummer>2</nummer>
      <nummer>110000</nummer>
   </list>
   <list>
      <nummer>100</nummer>
```

```
      <nummer>1000</nummer>
      <nummer>27</nummer>
   </list>
   <list>
     <nummer>8994</nummer>
     <nummer>1</nummer>
   </list>
   <list>
      <nummer>1</nummer>
      <nummer>3</nummer>
      <nummer>3</nummer>
      <nummer>-5</nummer>
      <nummer>4</nummer>
      <nummer>3</nummer>
      <nummer>4</nummer>
      <nummer>-6</nummer>
      <nummer>-5</nummer>
   </list>
</nummers>
```

Het volgende XSLT document `quicksort.xsl` wordt dan op dit input document toegepast:

Voorbeeld 7-23. Quicksort in XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html"/>
   <xsl:template match="/">
      <nummers>
         <xsl:apply-templates select=".//list"/>
      </nummers>
   </xsl:template>

   <xsl:template match="list">
      <list>
         <xsl:call-template name="qsort">
            <xsl:with-param name="spil" select="nummer[1]"/>
            <xsl:with-param name="list" select="nummer[position()>1]"/>
         </xsl:call-template>
      </list>
   </xsl:template>

   <xsl:template name="qsort">
      <xsl:param name="spil"/>
      <xsl:param name="list"/>
      <xsl:choose>
         <xsl:when test= "count($list)=0">
            <nummer><xsl:value-of select="$spil"/></nummer>
         </xsl:when>
         <xsl:otherwise>
```

```
            <xsl:variable name="kleiner"
                          select="$list[number() &lt; number($spil)]"/>
            <xsl:variable name="groter-gelijk"
                          select="$list[number() &gt;= number($spil)]"/>
            <xsl:if test="count($kleiner)>0">
               <xsl:call-template name="qsort">
                  <xsl:with-param name="spil" select="$kleiner[1]"/>
                  <xsl:with-param name="list" select="$kleiner[position()>1]"/>
               </xsl:call-template>
            </xsl:if>
            <nummer><xsl:value-of select="$spil"/></nummer>
            <xsl:if test="count($groter-gelijk)>0">
               <xsl:call-template name="qsort">
                  <xsl:with-param name="spil"
                                  select="$groter-gelijk[1]"/>
                  <xsl:with-param name="list"
                                  select="$groter-gelijk[position()>1]"/>
               </xsl:call-template>
            </xsl:if>
         </xsl:otherwise>
      </xsl:choose>
   </xsl:template>
</xsl:stylesheet>
```

Dit voorbeeld laat goed zien dat in XSLT recursie een zeer belangrijke en krachtige programmeertechniek is. De aard van de XSLT taal maakt het zelfs redelijk intuïtief om met recursie te werken. Na het toepassen van de template “`quicksort.xsl`” op de input, zien we de input in de volgende vorm:

Voorbeeld 7-24. Output van de Quicksort XSLT

```
<nummers>
   <list>
      <nummer>696</nummer>
   </list>
   <list>
      <nummer>1</nummer>
      <nummer>2</nummer>
      <nummer>25</nummer>
      <nummer>27</nummer>
      <nummer>30</nummer>
      <nummer>40</nummer>
      <nummer>100</nummer>
      <nummer>1000</nummer>
      <nummer>5550</nummer>
      <nummer>110000</nummer>
   </list>
   <list>
      <nummer>27</nummer>
      <nummer>100</nummer>
```

```
      <nummer>1000</nummer>
   </list>
   <list>
      <nummer>1</nummer>
      <nummer>8994</nummer>
   </list>
   <list>
      <nummer>-6</nummer>
      <nummer>-5</nummer>
      <nummer>-5</nummer>
      <nummer>1</nummer>
      <nummer>3</nummer>
      <nummer>3</nummer>
      <nummer>3</nummer>
      <nummer>4</nummer>
      <nummer>4</nummer>
   </list>
</nummers>
```

xsl:sort: De XSLT specificatie definieert tevens een eigen sorteer-mogelijkheid, namelijk `xsl:sort`.

7.5. Oefeningen

1. Een opwarmer: herschrijf het volgende stuk XSLT code zodanig dat het korter *en* performanter wordt. De stylesheet wordt toegepast op listing Voorbeeld 7-22.

   ```
   <xsl:stylesheet version="1.0"
                   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:template match="/">
       <list>
         <xsl:for-each select="//nummer">
           <xsl:if test="number()>15">
             <nummer>
               <xsl:value-of select="number()-15"/>
             </nummer>
           </xsl:if>
         </xsl:for-each>
       </list>
     </xsl:template>
   </xsl:stylesheet>
   ```

   Wat doet deze stylesheet precies? Leg uit waarom jouw herschreven stuk XSLT performanter is.

2. Pas het XSLT document uit listing Voorbeeld 7-19 toe op het XML document getoond in Voorbeeld 7-18. Observeer het resultaat; wat gebeurt er?
3. Maak een XSLT document dat XML listings, die beantwoorden aan de DTD van Voorbeeld 4-19, omzet naar HTML pagina's. Om de informatie van de producten af te beelden moeten er tabellen in HTML gebruikt worden.
4. Lees in de W3C XSLT de uitleg over de elementen in de xsl namespace voor transformaties die we hier niet behandeld hebben: xsl:sort, xsl:variable, xsl:document, xsl:import, xsl:apply-imports, xsl:include
5. Geef de gelijkenissen aan tussen de nodes xsl:import en xsl:apply-imports en Object Georienteerd programmeren.
6. Maak het voorbeeld van je CV helemaal af door ook de volgende tags en informatie in het XML document te zetten: je naam, adres, leeftijd, geslacht, vorige werkervaringen, interesses, informatica-kennis en je talenkennis. Breid de XSLT uit zodat deze alles kan transformeren in een HTML pagina.
7. Maak een XSLT die als input een getal x parst uit een XML document en als output een XML document geeft waarin de x eerste elementen van de rij van Fibonacci staan.

Hoofdstuk 8. Programmeertalen en XML processing

8.1. Java en XML processing

In dit hoofdstuk wordt er aangeleerd hoe men XML in samenwerking met Java kan gebruiken. Enerzijds laten we zien hoe de boom-gebaseerde DOM API werkt, en anderzijds het event-gestuurde parsen met de SAX API. DOM staat voor *Document Object Model* en SAX voor *Simple API for XML*.

Om gebruik te kunnen maken van DOM of SAX in onze Java applicatie, downloaden we de Xerces (http://xml.apache.org/) library van het Apache project (http://www.apache.org/) [1]. Je kan deze library, samen met documentatie, vinden op de url http://xml.apache.org. Omdat we hier Java gebruiken, is enkel de Xerces Java library nodig, meestal geleverd als een jar-bestand. Zorg ervoor dat deze jar in het classpath staat zodanig dat je de classes in de Xerces library in je code kan gebruiken.

Doorheen dit hoofdstuk zullen we in Java een applicatie schrijven die een XML document ontvangt en dit verwerkt. Het XML document beschrijft een simpele gebruikersinterface: knoppen, tekstvelden en labels. Onze applicatie zal deze beschrijving parsen en deze gebruikersinterface opbouwen en tonen. We maken zo onze eigen XML renderer voor Java gebruikersinterfaces. We gebruiken Java2 met de Swing widget familie voor de implementatie.

8.2. De test case: een DTD voor gebruikersinterfaces

Als test case zullen we een DTD gebruiken die de regels aangeeft om een gebruikersinterface te beschrijven met behulp van XML. Deze DTD is afgebeeld in Voorbeeld 8-1

Voorbeeld 8-1. Een DTD voor simpele gebruikersinterfaces

```
<!ELEMENT ui (title,group*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT group (widget|group)*>
<!ENTITY % pos "x CDATA #IMPLIED y CDATA #IMPLIED">
<!ATTLIST group name CDATA #REQUIRED
               %pos;
               rows CDATA #IMPLIED
               columns CDATA #IMPLIED>
<!ELEMENT widget (button|label|textfield)>
<!ATTLIST widget %pos;>
<!ELEMENT button (text,actie)>
<!ATTLIST button name CDATA #REQUIRED>
<!ELEMENT label (text)>
```

```
<!ATTLIST label name CDATA #REQUIRED>
<!ELEMENT textfield (text)>
<!ATTLIST textfield name CDATA #REQUIRED>
<!ELEMENT text (#PCDATA)>
<!ELEMENT actie (#PCDATA)>
```

We bouwen met de DOM en de SAX interface een applicatie in Java die een XML beschrijving parst, en er vervolgens de gebruikersinterface voor toont. Met andere woorden: we bouwen een “renderer” applicatie die een gebruikersinterface beschrijving neemt en deze op het scherm rendert.

8.3. De DOM API

8.3.1. Het Document Object Model

Het Document Object Model definieert een programmeer interface voor toegang tot XML documenten. Deze interface zorgt ervoor dat je toegang hebt tot het XML document in de vorm van een boomstructuur. Als je dus een DOM parser je XML document laat parsen, zal deze hiervan een boom in het geheugen vormen die je makkelijk kan lezen en veranderen met de DOM functionaliteiten.

> **Waarschuwing**
>
> De aandachtige lezer zal zich waarschijnlijk nu bedenkingen maken in verband met geheugen gebruik. Bij het verwerken van zeer grote XML documenten (wat zeker geen uitzondering is) of bij beperkt beschikbaar geheugen is DOM niet de beste keuze. Alhoewel werken met DOM intuïtiever is en meer mogelijkheden biedt (zoals het schrijven naar het XML document), zal later blijken dat SAX-parsers veel lichter en sneller werken dan DOM-parsers.

Een DOM-parser (een XML parser die zo een DOM boom opbouwt) laat dus niet enkel toe een boom van het XML document op te bouwen, maar biedt ook de mogelijkheid om deze boom te manipuleren en terug te bewaren.

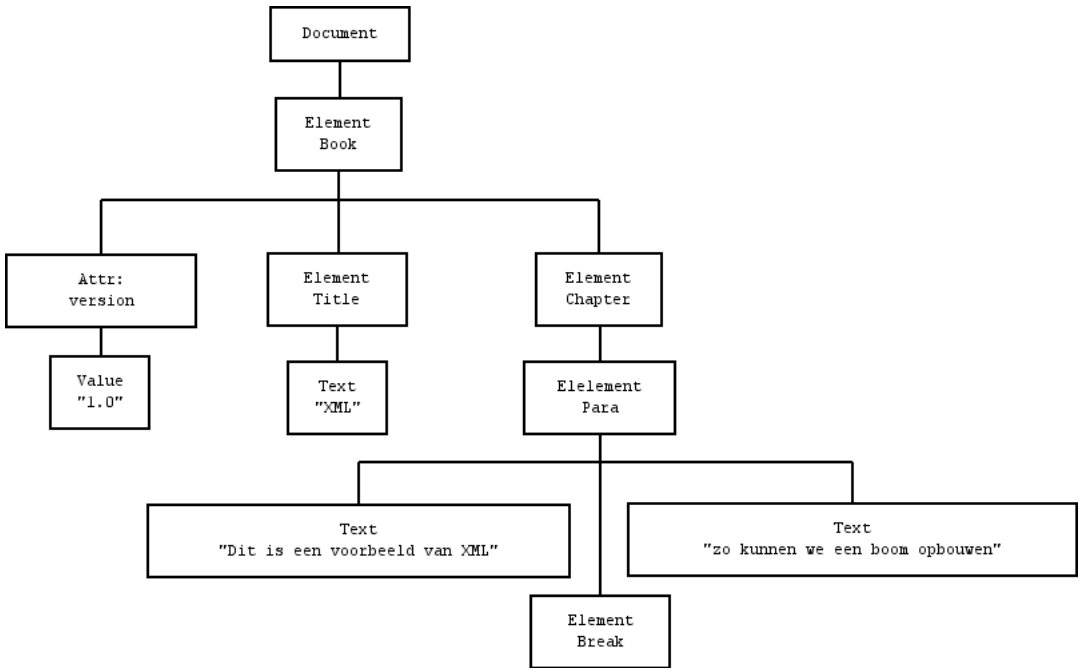
Ter herinnering nog eens een voorbeeld hoe je een XML document als een boom voorstelt, maar nu op de manier waarop het met DOM zou gebeuren. Voorbeeld 8-2 geeft de XML code weer waarvan Figuur 8-1 de grafische afbeelding is.

Voorbeeld 8-2. Mini boek

```
<Book version="1">
  <Title>XML</Title>
  <Chapter>
    <Para>Dit is een voorbeeld van XML.<Break/>
     Zo kunnen we een boom opbouwen.
```

```
    </Para>
  </Chapter>
</Book>
```

Figuur 8-1. DOM boom grafisch voorgesteld



8.3.2. Het package org.w3c.dom

Het package *org.w3c.dom* definieert enkele interfaces om met het DOM te werken. We geven een overzicht van de belangrijkste classes en interfaces:

Interface org.w3c.dom.Node

> Dit is de belangrijkste interface van het DOM. Het stelt een node voor in de Document boom, en biedt de nodige methods aan om te navigeren door de ouders, kinderen en siblings. Er zijn eveneens methods om de boom aan te passen. De interface Node is een superinterface van de meeste interfaces die in het package org.w3c.dom gedefinieerd worden.

Interface org.w3c.dom.Element

Deze interface biedt toegang tot een element in een XML document [2]. Men kan er ondermeer de attributen en de tag naam mee opvragen.

Interface org.w3c.dom.Document

Stelt het hele XML document voor. Een van de handigheidjes van deze interface is dat het directe toegang tot de root node toelaat. Er kan ook mee opgevraagd worden welke DTD er bij het document hoort. Een andere belangrijke functionaliteit is het creëren van allerhande types van nodes (comment, entity, processing instruction, element,...) .

Interface org.w3c.dom.Attr

Stelt de attributen voor van een element. Via de `Element` interface kan je een referentie naar de attributen bekomen. Er is een inconsistentie in de definitie van de `Attr` interface: het is een subinterface van de `Node` interface, alhoewel het door de DOM specificatie niet als een Node aanzien wordt. De method `getParentNode()` zal bijgevolg null teruggeven, evenals `getPreviousSibling()` en `getNextSibling()`.

Interface org.w3c.dom.DocumentFragment

Deze interface wordt gebruikt om een gedeelte van het document voor te stellen: een subtree in de Document tree met andere woorden. Objecten die de `DocumentFragment` interface implementeren voorzien een meer *lightweight* presentatie van een document dan Objecten die de interface `Document` implementeren.

Interface org.w3c.dom.CharacterData

Extends de `Node` klasse om toegang te bieden tot data bestaande uit karakters in de DOM boom. Deze interface wordt voor de gewone DOM processing niet gebruikt: hetr is in feite geen DOM element. Het dient als interface voor subinterfaces zoals `Text` en `CDATASection`.

Interface org.w3c.dom.Comment

Geeft toegang tot de Comment type node van het XML document. Commentaar wordt ook als node van de DOM boom beschouwd wordt. Deze interface erft over van `CharacterData` en bevat de data die tussen de tekens <!-- en --> voorkomt.

Interface org.w3c.dom.Entity

Deze interface geeft toegang tot een parsed of unparsed entity.

Interface org.w3c.dom.Notation

Deze interface geeft toegang tot een notation gedefinieerd in de DTD. Het biedt echter niet veel meer dan de `Node` aanbiedt. Het grote verschil is het invariante kenmerk dat een `Notation` *geen* parent `Node` heeft.

Interface org.w3c.dom.ProcessingInstruction

Deze interface geeft toegang tot een processing instruction. Zoals bij `Notation` ook al het geval was, biedt deze interface weinig extra functionaliteit ten opzichte van de superinterface `Node`. Je kan het eerste token van de processing instruction opvragen en al de data na dat eerste token.

Interface org.w3c.dom.Text

> `Text` geeft toegang tot de inhoud van `Attr` en `Element` indien de inhoud van deze nodes louter tekstueel is. Een `Text` bevat dus gewoon een blok tekst. Een interessant gegeven is dat deze blok, gegeven een offset, in twee kan gesplitst worden en siblings van de twee ontstane nodes maakt.

8.3.2.1. Praktisch gebruik van de DOM Interfaces

In deze sectie wordt getoond hoe er kan gebruik gemaakt worden van de DOM om de implementatie van onze gebruikersinterface renderer te maken. We bespreken de verschillende stukken die nodig zijn apart om tenslotte er één geheel van te maken.

Een eerste stap is de DOM parser aanmaken. We hebben gekozen voor Xerces, dus zullen we de DOM parser hiervan kiezen: `org.apache.xerces.parsers.DOMParser`. We maken een nieuwe instantie van deze klasse aan en geven dan het XML document aan de parser:

Voorbeeld 8-3. Aanmaken en initialiseren van de DOM parser

```
org.apache.xerces.parsers.DOMParser $dparser=
   new org.apache.xerces.parsers.DOMParser();
try{
   File f = new File("onsvoorbeeld.xml");
   $dparser.parse(f.getPath());
}catch(IOException ioe){
   //het inlezen van onsvoorbeeld.xml liep mis
   ...
}catch(SAXException se){
   //onsvoorbeeld.xml was geen geldig xml document
   ...
}
```

Nu we een DOM parser tot onze beschikking hebben, en het document in het geheugen opgebouwd hebben [3] kunnen we wat beginnen te werken met deze DOM boom. In Voorbeeld 8-4 wordt de maximale diepte van de DOM boom berekend. Let op: dit is niet noodzakelijk de maximale diepte van de XML boom, wel van de DOM presentatie van die XML boom.

Voorbeeld 8-4. Maximale diepte van de DOM boom

```
public class Example1{
 public int maxDepth(){
   return maxDepth($dparser.getDocument().getDocumentElement());
 }
```

```
 private int maxDepth(Node n){
   if(n.getChildNodes().getLength()==0)
      return 1;
   else{
      int max=0;
      for(int i=0; i<n.getChildNodes().getLength(); i++){
        if(maxDepth(n.getChildNodes().item(i))>max)
            max = maxDepth(n.getChildNodes().item(i)) + 1;
      }
      return max;
   }
 }

 final public static void main(String args[]){
   try{
      Example1 ex1 = new Example1(args[0]);
      ex1.parse();
      System.out.println("Max tree depth="+ex1.maxDepth());
...
 }
}
```

In Voorbeeld 8-5 doorlopen we de DOM boom en tellen we de verschillende soorten nodes. Recursief wordt de DOM boom tot aan de bladen doorlopen in pre-order, elke Node wordt op type gecheckt en de teller voor dat type node wordt verhoogd. Om de interface van het bepaalde type te gebruiken kan je casten naar die interface. Voorwaarde is natuurlijk dat `getNodeType()` het juiste type geeft. Denk eraan dat Node een superinterface is van alle andere mogelijke types die in de DOM boom voorkomen.

Voorbeeld 8-5. Tel de verschillende types nodes

```
public class Example2{

private int $attrCounter, $cdataCounter,$commentCounter;
private int $docfragCounter, $docnodeCounter;
private int $doctypeCounter, $elemCounter, $entityCounter;
private int $entrefCounter, $notCounter, $procCounter, $textCounter;
private java.util.Vector $elist;

public void listCounts(){
   $attrCounter=0; $cdataCounter=0;
   $commentCounter=0; $docfragCounter=0;
   $docnodeCounter=0; $doctypeCounter=0;
   $elemCounter=0; $entityCounter=0;
   $entrefCounter=0; $notCounter=0;
   $procCounter=0; $textCounter=0;

   $elist = new java.util.Vector();
   listAll($dparser.getDocument().getDocumentElement());
   p("# attr nodes = " + attrCounter);
   p("# cdata nodes = " + cdataCounter);
```

```
   p("# comment nodes = " + commentCounter);
   p("# document fragment nodes = "+docfragCounter);
   p("# document nodes = " + docnodeCounter);
   p("# document type nodes = "+doctypeCounter);
   p("# element nodes = "+elemCounter);
   p("# entity nodes = "+entityCounter);
   p("# entity reference nodes = "+entrefCounter);
   p("# notation nodes = "+notCounter);
   p("# processing instruction nodes = "+procCounter);
   p("# text nodes = "+textCounter);
   p("Element names: ");
   for(int i=0; i<elist.size(); i++){
      System.out.print((String)elist.get(i));
      if(i<elist.size()-1)
        System.out.print(",");
   }
}


private void p(String messg){
   System.out.println(messg);
}

private void listAll(Node n){
   switch(n.getNodeType()){
      case Node.ATTRIBUTE_NODE: // The node is an Attr.
         attrCounter++;
         break;
      case Node.CDATA_SECTION_NODE: // The node is a CDATASection.
         cdataCounter++;
         break;
      case Node.COMMENT_NODE: // The node is a Comment.
         commentCounter++;
         break;
      case Node.DOCUMENT_FRAGMENT_NODE: // The node is a DocumentFragment.
         docfragCounter++;
         break;
      case Node.DOCUMENT_NODE: // The node is a Document.
         docnodeCounter++;
         break;
      case Node.DOCUMENT_TYPE_NODE: // The node is a DocumentType.
         doctypeCounter++;
         break;
      case Node.ELEMENT_NODE: // The node is an Element.
         elemCounter++;
         elist.add(((Element)n).getTagName());
         break;
      case Node.ENTITY_NODE: // The node is an Entity.
         entityCounter++;
         break;
      case Node.ENTITY_REFERENCE_NODE: // The node is an EntityReference.
         entrefCounter++;
         break;
```

```
      case Node.NOTATION_NODE: // The node is a Notation.
         notCounter++;
         break;
      case Node.PROCESSING_INSTRUCTION_NODE: // The node is a ProcessingInstruction.
         procCounter++;
         break;
      case Node.TEXT_NODE: // The node is a Text node.
         textCounter++;
         break;
   }
   for(int i=0; i<n.getChildNodes().getLength();i++){
      listAll(n.getChildNodes().item(i));
   }
}

final public static void main(String args[]){
   try{
      Example2 ex2 = new Example2(args[0]);
      ex2.listCounts();
   }catch(Exception e){
      System.err.println("usage: java Example1 <file.xml>");
      System.exit(0);
   }
}
```

Al de voorbeelden die tot nu toe beschouwd zijn, hebben enkel betrekking tot het lezen van de DOM boom. Men kan ook wijzigingen aanbrengen aan de DOM boom. Hiervoor voorziet de interface `Node` enkele methods:

Node appendChild(Node newChild)

Node insertBefore(Node newChild, Node refChild)

Node removeChild(Node oldChild)

Node replaceChild(Node newChild, Node oldChild)

Meer uitleg over het gebruik van deze methods vindt u in de api docs (http://xml.apache.org/apiDocs/org/w3c/dom/Node.html) van Xerces.

8.3.3. De gebruikersinterface DTD parsen met behulp van DOM

Met de informatie uit de voorgaande secties kunnen we nu een uitgebreider voorbeeld tonen. We maken een DOM parser voor documenten die aan de DTD gedefinieerd in Voorbeeld 8-1 voldoen. De code in het volgende voorbeeld is uitgebreid gedocumenteerd en een werkend voorbeeld vind je in uibuilder.jar. We tonen enkel de code van de Java klasse die het parsen en opbouwen voor de gebruikersinterface voor zijn rekening neemt. Dit voorbeeld kan enkel naar een Swing Java widget set output geven. We bespreken de implementatie van de gebruikersinterface builder hier stuk per stuk.

Voorbeeld 8-6. De constructor en de parse method

```
/**
Creates the DOMUiParser, and initializes the functionality
for the specified output format
@param outputFormat indicates which kind of output should be generated
**/
public DOMUiParser(String outputFormat){
   $dparser = new org.apache.xerces.parsers.DOMParser();
}

/**
Starts the parser and the gebruikersinterface building process
@throws FileNotFoundException if the source document is not found
@throws InvalidXMLException if the source document was badly structured
**/
public void parse() throws FileNotFoundException, InvalidXMLException{
   if(getSourceDocument()==null)
      throw new FileNotFoundException("no file specified");
    try{
       $dparser.parse(getSourceDocument().getPath());
       buildUI($dparser.getDocument());
    }catch(IOException ioe){
          throw new FileNotFoundException("no file found or illegal file [" + ioe.toString(
    }catch(SAXException se){
       throw new InvalidXMLException(se.toString());
    }
}
```

De commentaar bij de code is nog in het Engels geschreven, maar zou toch al het een en ander duidelijk moeten maken. `InvalidXMLException` is een zelf gedefinieerde exceptie. In het voorbeeld wordt de method buildUI opgeroepen, van waaruit het hele processing gebeuren gestart wordt.

Voorbeeld 8-7. buildUI en processNode

```
/**
Makes a dummy frame, sets the title and initiates a recursive UI building process
@param doc the XML document describing the gebruikersinterface
<p><b>requires</b>: doc != null</p>
**/
protected void buildUI(Document doc){
   Element e = doc.getDocumentElement();
   JPanel app = new JPanel();
   processNode(e, app);
   JFrame f = new JFrame();
   f.setTitle(getTitle());
   f.getContentPane().add(app);
   f.pack();
   f.setVisible(true);
}

/**
processes the Nodes, this is the recursive heart of the parsing process.
@param n the node, presenting some information about the gebruikersinterface,
       to be processed
@param container the container where the gebruikersinterface must be embedded
<p><b>requires</b>: n!=null</p>
<p><b>requires</b>: container!=null</p>
**/
private void processNode(Node n, JComponent container){
   if(n.getNodeType()==Node.ELEMENT_NODE)
      if(processElement((Element)n, container)) return;
   NodeList nl = n.getChildNodes();
   if(nl.getLength()>0)
     for(int i=0; i<nl.getLength(); i++)
      processNode(nl.item(i), container);
}
```

Het vorige stuk code maakt duidelijk dat `+processNode` verantwoordelijk is voor de DOM boom te doorlopen en de elementen eruit te pikken. De verwerking van de elementen wordt gedelegeerd naar `processElement`, getoond in het volgende voorbeeld. Indien deze functie true terug geeft duidt dit aan dat de kinderen van de huidige node al behandeld zijn, en dat processNode voor deze node de recursie op de kinderen niet mag uitvoeren. Dit is natuurlijk een implementatie keuze, het is de oplossing die hier gekozen werd maar er zijn nog andere mogelijkheden natuurlijk.

Voorbeeld 8-8. processElement

```
/**
Processes an Element, and adds the appropriate information, which e describes,
to the container
@param e an element describing part of the gebruikersinterface
@param container the container to hold the part of this interface
```

```
@return true if the descendants of e are processed by this method,
false otherwise
<p><b>requires</b>: e!=null</p>
<p><b>requires</b>: container!=null</p>
**/
private boolean processElement(Element e, JComponent container){
   if(e.getTagName().equals("group"))
      return processGroupElement(e,container);
   else if(e.getTagName().equals("widget"))
      return processWidgetElement(e,container);
   else if(e.getTagName().equals("title"))
      setTitle(e.getFirstChild().getNodeValue());
   return false;
}
```

De method `processElement` kijkt om welke tag het gaat, en beslist aan de hand daarvan welke method het element mag verwerken. Er zijn 3 mogelijkheden: het gaat om de titel (<title>, dit kan maar één keer gebeuren!), het gaat om een group element of het gaat over een widget. De respectievelijke functies worden getoond in de voorbeeld code. Als het element *niet* aan één van deze tags beantwoordt geven we false terug, en geven zo aan dat `processNode` zelf de kinderen mag processen. Als we een validerende processor gebruiken (wat hier het geval is) kan dit eigenlijk nooit gebeuren met een valid XML document.

Voorbeeld 8-9. processGroup en processWidget

```
/**
Processes a widget element, and adds it to the container
@param e an element describing a widget
@param container the container to which the widget should be added
@return true if the descendants of e are processed by this method,
false otherwise
<p><b>requires</b>: e!=null</p>
<p><b>requires</b>: e.getChildNodes()!=null</p>
<p><b>requires</b>: e.getChildNodes().getLength()&gt;=1</p>
<p><b>requires</b>: container!=null</p>
**/
private boolean processWidgetElement(Element e, JComponent container){
   GridBagConstraints gbc = new GridBagConstraints();
   try{
      gbc.gridx = Integer.parseInt(e.getAttribute("x"))-1;
      gbc.gridy = Integer.parseInt(e.getAttribute("y"))-1;
   }catch(NumberFormatException nfe){}
   NodeList nl = e.getChildNodes();
   try{
      int i=0;
      //avoid the comment Nodes
      while(nl.item(i).getNodeType() != Node.ELEMENT_NODE)
         i++;
      Element wchild = (Element)nl.item(i);
      String label = getLabel(wchild);
```

```
      JComponent c;
      if(wchild.getTagName().equals("button"))
          c = new JButton(label);
       else if(wchild.getTagName().equals("textfield"))
          c = new JTextField(label);
       else if(wchild.getTagName().equals("label"))
          c = new JLabel(label);
       else
          return false;
        container.add(c,gbc);
   }catch(NullPointerException npe){
      return false;
   }
   return true;
}

/**
Processes the grouping element and adds the grouped childs
of it to container
@param e an element describing a bunch of logically grouped UI elements
@param container the container to which this group should be added
@return true if the descendants of e are processed by this method,
false otherwise
<p><b>requires</b>: e!=null</p>
<p><b>requires</b>:container=!null</p>
**/
private boolean processGroupElement(Element e, JComponent container){
   String name = e.getAttribute("name");
   int rows,cols;
   try{
      cols = Integer.parseInt(e.getAttribute("columns"));
      rows = Integer.parseInt(e.getAttribute("rows"));
   }catch(NumberFormatException nfe){
      return false;
   }
   JPanel current = new JPanel(new GridBagLayout());

   if((e.getAttribute("x").length()>0)&&(e.getAttribute("y").length()>0)){
      try{
         GridBagConstraints gbc = new GridBagConstraints();
         gbc.gridx = Integer.parseInt(e.getAttribute("x"))-1;
         gbc.gridy = Integer.parseInt(e.getAttribute("y"))-1;
         container.add(current, gbc);
      }catch(NumberFormatException nfe){
         container.add(current);
      }
   }else
      container.add(current);
   NodeList nl = e.getChildNodes();
   for(int i=0; i<nl.getLength(); i++)
     processNode(nl.item(i), current);
   return true;
}
```

Dit was het hele voorbeeld, als je de kleine stukjes voorbeeld code nu aan elkaar hangt krijg je een werkende XML processor die van ons XML document een gebruikersinterface op het scherm tovert. Veel magie komt er eigenlijk niet bij kijken: de stukken voorbeeld code zijn niet echt complex.

Wat voor een resultaat krijgen we nu te zien? Ter illustratie is er in Voorbeeld 8-10 een beschrijving van een login-scherm gegeven.

Voorbeeld 8-10. Beschrijving van een login scherm

```
<?xml version="1.0" ?>
<!DOCTYPE ui SYSTEM "http://lumumba.luc.ac.be/kris/courses/xml/simpleui.dtd" [
]>
<ui>
   <title>Login</title>
   <group name="loginpanel" rows="2" columns="1">
      <group name="info" x="1" y="1" rows="2" columns="2">
         <widget x="1" y="1">
            <label name="userlabel">
               <text>User name:</text>
            </label>
         </widget>
         <widget x="1" y="2">
             <label name="passwordlabel">
                <text>Password:</text>
             </label>
         </widget>
         <widget x="2" y="1">
             <textfield name="username">
                 <text>             </text>
             </textfield>
         </widget>
         <widget x="2" y="2">
            <textfield name="password">
                <text>              </text>
            </textfield>
         </widget>
      </group>
      <group name="buttons" x="1" y="2" rows="1" columns="2">
         <widget x="1" y="1">
            <button name="shutdown">
               <text>Shutdown</text>
               <actie>exit</actie>
            </button>
         </widget>
         <widget x="2" y="1">
            <button name="login">
               <text>Login</text>
               <actie>login</actie>
            </button>
```

```
        </widget>
      </group>
    </group>
</ui>
```

Deze beschrijving wordt verwerkt door onze DOM parser, en geeft ons een Java Swing gebruikersinterface. Deze is afgebeeld in Figuur 8-2.

Figuur 8-2. Het Login panel in Java Swing

Merk op dat de group elementen eigenlijk de dimensie die ze kunnen bevatten (rows en columns) niet hoeven te specifieren hier: de layoutmanager `GridBagLayout` is flexibel genoeg om zonder deze waardes nog een juiste layout te creëren. Andere widget sets zouden hier echter wel gebruik van kunnen maken.

Met dit uitgebreide voorbeeld wordt deze sectie afgesloten, en ook de uitleg over DOM XML processing. Om zelf (met Xerces) aan de slag te gaan, heb je vooral de javadoc documentatie van de Xerces Java API nodig. Je kan deze vinden op xml.apache.org (http://xml.apache.org).

8.4. De SAX API

8.4.1. De Simpele Api voor XML

Verplichte lectuur: “SAX, the power API”, Benoit Marchal, *DeveloperWorks: SAX, the power api* (http://www-106.ibm.com/developerworks/xml/library/x-saxapi/index.html): http://www-106.ibm.com/developerworks/xml/library/x-saxapi/index.html

In tegenstelling tot DOM-parsers bouwen SAX gebaseerde parsers *geen* boom structuur op in het geheugen. In plaats hiervan zal het sequentieel het document afhandelen en voor elke openings- en sluitingstag die het tegenkomt een *event* afvuren. Door te “luisteren” naar deze events weten we welke tags aan de beurt zijn. Dit betekent dat er minder geheugen verbruikt wordt, maar ook dat de programmeur zelf meer werk moet doen. Zo zal er zelf uitgekeken moeten worden in welke context de events zich bevinden, zoals onder welke parent ze staan.

8.4.2. Het package org.xml.sax

De structuur van het package org.xml.sax is helemaal anders gestructureerd als het org.w3c.dom package. Dit is niet verwonderlijk, daar de architectuur ook helemaal anders is. Deze event-gebaseerde aanpak gebruikt geen boom om het XML-document voor te stellen, maar vuurt events af voor elke open en sluit tag die het tegenkomt. Als men SAX gebruikt en een interne boom-structuur van het XML-document nodig heeft, zal men daar dan ook zelf voor moeten zorgen.

SAX werkt als volgt: de applicatie levert een *document handler* aan de XML parser. Vervolgens zal de applicatie de parser de opdracht geven het XML document te parsen. De parser kent de document handler (omdat deze afgeleverd werd door de applicatie), en zal voor elke open- en sluit-tag events afvuren naar deze document handler.

Interfaces org.xml.sax.DocumentHandler en org.xml.sax.ContentHandler

Deze interfaces zijn “de spil van het spel”, waarbij de interface `DocumentHandler` de oude interface is (gebruikt in SAX1) en vervangen werd door de interface `ContentHandler` in SAX2. De twee belangrijke methods in deze interfaces zijn `startElement` en `endElement`. `startElement` wordt “afgevuurd” wanneer er een open tag ontmoet wordt en `endElement` wanneer er een sluit tag ontmoet wordt. Idem kan men gebruik maken van `startDocument` en `endDocument`.

Om gebruik te maken hiervan, is het nodig de interface `ContentHandler` (voor SAX2) of `DocumentHandler` (voor SAX1) over te erven. Om het de programmeur makkelijker te maken wordt er de klasse `org.xml.sax.helpers.DefaultHandler` aangeboden. Het wordt dan ook aangeraden deze te gebruiken (zie ook Voorbeeld 8-11). De volgorde van de events die afgevuurd worden zijn dezelfde als de volgorde van de tags in het document. Als je het XML document als een boom bekijkt, betekent dit dat de events in pre-order zullen afgevuurd worden.

De method `startElement` krijgt als argumenten onder meer de namespace, de tag naam en een lijst van attributen van de tag mee. Hieruit kan men dan afleiden voor welke tag het event afgevuurd was.

Interface org.xml.sax.XMLReader

Om de events met behulp van `ContentHandler` of `DocumentHandler` te verwerken, is er iets nodig dat het XML document daadwerkelijk doorleest en zorgt dat de events afgevuurd worden. `XMLReader` biedt deze functionaliteit aan: het kan een XML Document inlezen en parsen (method parse()). Vooraf dienen dan de ContentHandlers aangegeven te worden (we beschouwen hier enkel SAX2) door middel van de method `setContentHandler`.

Class org.xml.sax.SAXException

Dit is de belangrijkste exceptie als men met SAX werkt. Alle andere excepties in verband met SAX worden van deze exceptie afgeleid.

8.4.3. Een voorbeeld met de SAX API

Als voorbeeld van het gebruik van de SAX API tonen we een simpele zelf gemaakte XML formatter die ook uitschrijft wat de maximale diepte (pad van root node naar blad) is van het XML document. De formatter schrijft de xml code naar de standaard output stream, en zorgt dat de indentatie mooi staat. De code vindt u in Voorbeeld 8-11. De interface `org.xml.sax.XMLReader` (http://xml.apache.org/xerces-j/apiDocs/org/xml/sax/XMLReader.html) zorgt ervoor dat het XML document kan ingelezen worden door gebruik te maken van callbacks. Dit is de interface die door de SAX parser zelf moet geïmplementeerd worden. Een instantie van een SAX parser wordt door een factory aangemaakt, deze factory krijgt de naam van de gewenste parser mee (in ons geval `org.apache.xerces.parsers.SAXParser`), en geeft een referentie naar deze parser terug (`XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser")`). Het geeft ook de mogelijkheid om de verschillende features van de parser op te vragen en er bepaalde waarden aan toe te kennen. Bekijk de API docs voor meer informatie hierover.

Om via de SAX API een document te parsen, moet men gebruik maken van een ContentHandler waar de events naar kunnen afgevuurd worden. Er is in Voorbeeld 8-11 voor gekozen dit in dezelfde klasse te doen[4]: dit zie je aan extends DefaultHandler bij de klassedefinitie.

Om de Character contents op te kunnen vragen, is het mogelijk om de method `characters` te herdefiniëren. Wegens efficiëntie redenen zal de data in chunks afgeleverd worden: de parser beslist zelf hoe deze de character data opsplitst. Let er dus op dat je de verschillende chunks nog achter elkaar moet hangen, vandaar het gebruik van de `CharArrayWriter` (http://java.sun.com/products/jdk/1.2/docs/api/java/io/CharArrayWriter.html). Dit had natuurlijk ook gedaan kunnen worden met behulp van `StringBuffer` (http://java.sun.com/j2se/1.3/docs/api/java/lang/StringBuffer.html) of iets dergelijks.

Voorbeeld 8-11. Simpele XML formatter

```
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.InputSource;
import java.io.FileReader;
import java.io.IOException;
import java.io.CharArrayWriter;

public class SAXExample extends DefaultHandler{
    private static String pname="org.apache.xerces.parsers.SAXParser";
```

```
 private XMLReader $xr;
 private int $maxDepth=0;
 private int $currentDepth;
 private CharArrayWriter $charWriter;


 public SAXExample(String filename){
     $charWriter=new CharArrayWriter();
     try{
        $xr = XMLReaderFactory.createXMLReader(pname);
     }catch(SAXException saxe){
        System.err.println("Could not load XMLParser:" +
                           saxe.toString());
     }
     try{
         InputSource is= new InputSource(new FileReader(filename));
         $xr.setContentHandler(this);
         $xr.parse(is);
     }catch(SAXException saxe){
         System.err.println();
     }catch(IOException ioe){
         System.err.println("error reading the XML document: " +
                            ioe.toString() );
     }catch(Exception e){
         System.err.println("Unexpected exception: " + e.toString());
     }
 }



 private String buildString(String ln){
     StringBuffer sb = new StringBuffer(ln);
     for(int i=0; i<$currentDepth; i++)
         sb.insert(0, "  ");
     return sb.toString();
 }

 /**
   Wordt opgeroepen als de parser een open tag leest
  **/
public void startElement(String nameSpaceURL, String localName,
                         String qName, Attributes attr)
                  throws SAXException{
     $currentDepth++;
     if($maxDepth<$currentDepth)
         $maxDepth = $currentDepth;
     System.out.println(buildString("<" + localName + ">"));
 }

 /**
   Wordt opgeroepen als de parser een sluit tag leest
  **/
 public void endElement(String nameSpaceURL, String localName,
```

```
                          String qName) throws SAXException{
      System.out.println(buildString("  "+$charWriter.toString().trim()));
      System.out.println(buildString("</" + localName + ">"));
      $currentDepth--;
      $charWriter.reset();
    }

    public void endDocument() throws SAXException{
      System.out.println("<!-- Maximum tree depth is " +
                         $maxDepth + " -->");
    }

    /**
      Vangt de character contents op in chunks van karakter data
     **/
     public void characters(char[] ch, int start, int len)
        throws SAXException{
        $charWriter.write(ch,start,len);
    }

    /**
      Lege functie zorgt ervoor dat  overbodige
      whitespace, tabs etc. weggegooid worden.
      Opzettelijk lege method
     **/
    public void ignorableWhitespace(char[] ch, int start, int len){
    }


    public static void main(String[] args){
        if(args.length != 1){
            System.out.println("Usage: java SAXExample <filename>");
            return;
        }
        new SAXExample(args[0]);
    }
}
```

8.5. Oefeningen

1. Maak een Java programma dat het stuk XML uit Voorbeeld 7-15 parst en deze in een `javax.swing.JTable` toont en toelaat deze gegevens te veranderen en terug te bewaren in de XML boom.

2. Vul de code uit Voorbeeld 8-11 aan zodat de formatter ook de tag attributen met de bijbehorende waarde uitschrijft naar de standaard output.
3. Schrijf een equivalent programma voor Voorbeeld 8-5 met behulp van de SAX API.
4. Herschrijf het programma (DOMUiParser.java) om de gebruikersinterface DTD te parsen en op te bouwen met behulp van de SAX API in plaats van met DOM.

Noten

1. Xerces was origineel XML4J van IBM. IBM heeft echter besloten de code vrij te geven voor gebruik in de Open Source gemeenschap. Het gevolg hiervan is de kwalitatief hoogstaande XML library voor Java en C++: Xerces.
2. Dit geldt eveneens voor HTML documenten.
3. Door de method `parse` op te roepen op het object $dparser van het type `org.apache.xerces.parsers.DOMParser`.
4. Let erop dat dit niet de beste manier is: het is beter een OO programma op te delen in verschillende objecten, elk met zijn eigen verantwoordelijkheid. Hier werd dat niet gedaan om het voorbeeld beperkt te houden

Hoofdstuk 9. SVG: Scalable Vector Graphics

W3C SVG Specificatie (http://www.w3.org/TR/SVG11)

9.1. Inleiding

Scalable Vector Graphics (SVG) is een formaat om twee-dimensionele grafische objecten te beschrijven in XML. De mogelijkheden bestaan ondermeer uit:

Vector graphic shapes

Images

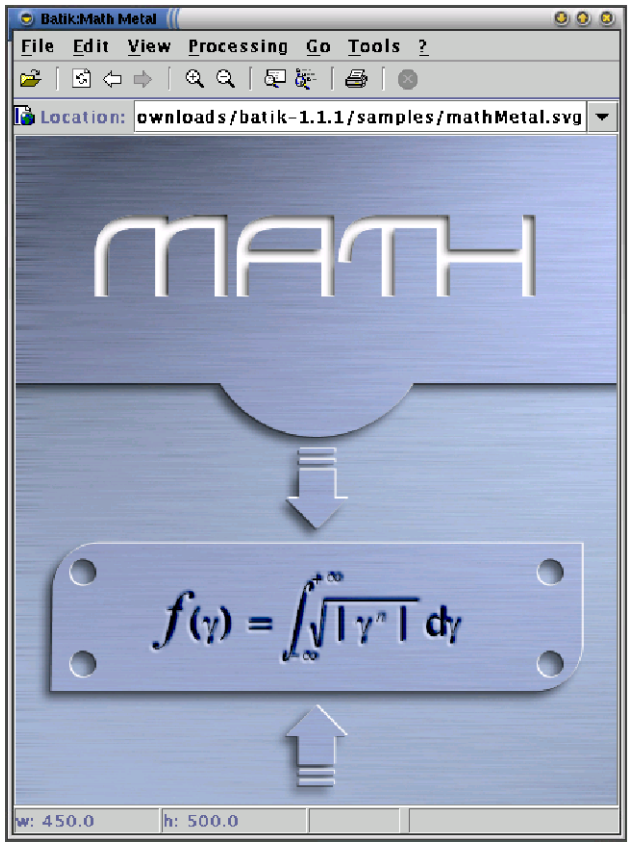
Text

Geometrische objecten (lijnen en curves)

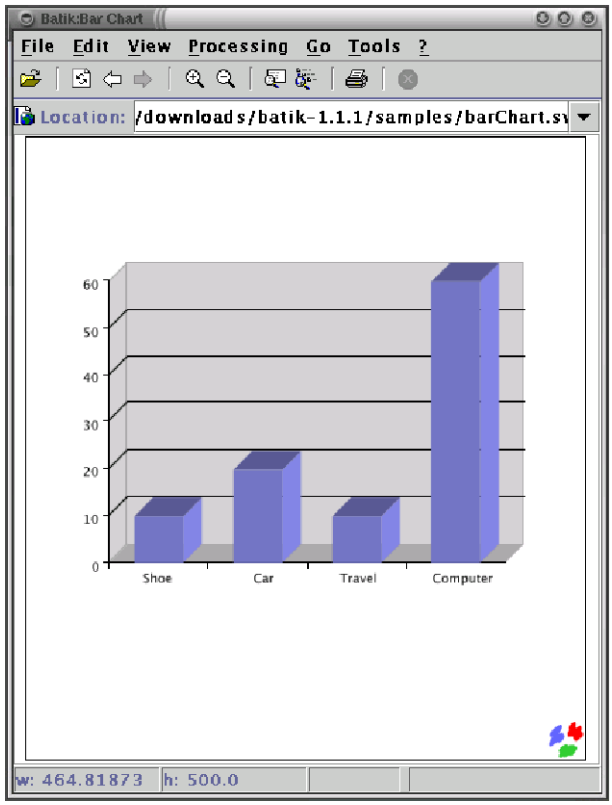
Een SVG document kan zelfs dynamisch en interactief zijn, met andere woorden: het is mogelijk om interactieve animaties te bouwen met SVG.

In dit hoofdstuk bekijken we hoe we zelf simpele SVG figuren kunnen maken. In veel gevallen zal de XML die een SVG figuur beschrijft niet met de hand gemaakt worden, maar met behulp van een tekenpakket of door middel van automatische generatie van de SVG code. In Figuur 9-1 en Figuur 9-2 vind je twee voorbeelden van figuren die beschreven zijn in SVG.

Figuur 9-1. SVG voorbeeld: formule



Figuur 9-2. SVG voorbeeld: barchart



Er zijn verschillende vrij beschikbare tekenpaketten die reeds kunnen exporteren naar SVG formaat: CSIRO SVG Toolkit (http://sis.cmis.csiro.au/svg/), Squiggle (http://xml.apache.org/batik/svgviewer.html) als onderdeel van Batik (http://xml.apache.org/batik/), Sodipodi (http://sodipodi.sourceforge.net/), Amaya (http://www.w3.org/Amaya/Amaya.htm) en X-Smiles (http://www.xsmiles.org/). Daarnaast zijn er ook commerciële implementaties beschikbaar. Verschillende browsers ondersteunen reeds SVG figures zoals de reeds vermelde browsers X-Smiles en Amaya, maar ook de allerlaatste versie van de browser Mozilla (http://www.mozilla.org/projects/svg/) en de browser Microsoft Internet Explorer.

De specificatie van SVG is, zoals vele specificaties van W3C (http://www.w3.org), zeer groot. Als men de specificatie als een doorlopend document bekijkt, dan heeft men meer dan 600 bladzijden! Dit betekent dat we enkel ruimte hebben voor de basis van SVG uit de doeken te doen. De geïnteresseerde lezer wordt doorverwezen naar de specificatie om de details van SVG te ontdekken.

Niet alle browsers ondersteunen SVG even volledig, dus maken we gebruik van een vrij beschikbare applicatie om SVG documenten mee te bekijken: Batik (http://xml.apache.org/batik). Deze applicatie is in Java geschreven en vereist enkel een recente Java Virtual Machine. Natuurlijk kan men ook andere

applicaties gebruiken om SVG documenten mee te bekijken. We beschouwen Batik als referentie applicatie voor dit hoofdstuk.

9.2. Structuur van een SVG document

Een SVG document beantwoordt aan een bepaalde Document Type Definition. Dit dient ook steeds aangegeven te worden bij een stand-alone SVG document. Tevens is het root element ook steeds een tag met als elementnaam *svg*. Een voorbeeld van de structuur van een svg document is gegeven in Voorbeeld 9-1.

Voorbeeld 9-1. SVG document structuur

```
 <?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="10cm" viewBox="100 100 1000 1000"
     xmlns="http://www.w3.org/2000/svg">
  <title>Hier komt de titel</title>
  <desc>Hier komt de beschrijving van de figuur</desc>
  <!-- Hier komt de svg code -->
</svg>
```

9.3. Het path element

Het path element is het basiselement uit de SVG specificatie. Het laat toe om *paden* te definiëren; allerhande complexe vormen kunnen door middel van paden voorgesteld worden. Een gelijkaardige manier om lijntekeningen te maken wordt gebruikt in postscript (http://partners.adobe.com/asn/developer/technotes/postscript.html). Men kan een pad vergelijken met het tekenen van een figuur met behulp van pen op papier. Een pad bestaat uit een opeenvolging van *data* en *instructies*. De mogelijke instructies die voorkomen in pad data zijn:

moveto

Aangeduid door de letters M of m. Definieert een nieuw punt in het pad.

lineto

Tekent een lijn van punt naar punt. Aangeduid door de letters H of h voor horizontale lijnen, V of v voor verticale lijnen en L of l voor lijnen die tussen twee punten getekend worden.

curve

Er zijn verschillende soorten curves mogelijk: cubische Bezier curves (aangeduid door de letters C, c, S of s) en quadratische Bezier curves (aangeduid door Q, q, T of t).

arc

De elleptische boog, aangeduid door de letters A of a.

closepath

Sluit het huidige sub-pad af. Wordt aangeduid door de letters Z of z.

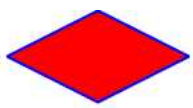
Het verschil tussen kleine letter en hoofdletter om de operatie aan te geven heeft te maken met het soort coördinaten. Een kleine letter betekent dat de coördinaten die erop volgen relatieve coördinaten zijn, een hoofdletter duidt op absolute coördinaten.

We maken deze instructies wat duidelijker aan de hand van enkele praktische voorbeelden. We beginnen met het tekenen van een ruit: hiervan weten we dat we vier hoekpunten nodig hebben. We nemen de volgende hoekpunten: (50,50), (100,100), (100,0), (200,50). Voorbeeld 9-2 toont hoe de ruit wordt getekend met behulp van het path element. Hiervoor wordt het *d* attribuut geparst: *M 0 50 L 100 100 L 200 50 L 100 0 z*. Men verplaatst de huidige positie naar de coördinaten (0,50): *M 0 50*. Vanaf dat punt trekt men een lijn naar coördinaten (100,100): *L 100 100*. Vervolgens wordt de lijn verder getrokken naar coördinaten (200,50): *L 200 50*. De voorlaatste stap trekt de lijn dan naar coördinaten (100,0): *L 100 0*. Merk op dat het path nu nog gesloten dient te worden: om een volledige ruit te bekomen ontbreekt nog een lijn van (100,0) naar (0,50). Met *z* wordt het path dan afgesloten door vanaf het huidige punt naar het intiële punt een lijn te trekken. Tenslotte zijn er nog enkele andere attributen die de kleur e.d. bepalen van het path. Indien men dit SVG document bekijkt met een SVG-viewer, krijgt men hetzelfde te zien als in figuur Figuur 9-3.

Voorbeeld 9-2. Een ruit met het path element

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <title>Een ruit</title>
  <desc>Een pat tekent een ruit</desc>
  <path d="M 0 50 L 100 100 L 200 50 L 100 0  z"
        fill="red" stroke="blue" stroke-width="3" />
</svg>
```

Figuur 9-3. SVG voorbeeld: ruit



SVG laat ook toe om op een simpele manier Bézier curves te tekenen. Cubische Bézier curves worden aangeduid met de letters c of C. Dit zijn curves met *twee* controle punten. Men kan ook opteren om S of s te gebruiken, welke een soort shorthand is voor C of c. Voor het gebruik van S of s verwijzen we naar de SVG specificatie. Voorbeeld 9-3 laat een voorbeeld zien van het gebruik van C. Merk op dat de coördinaten meegegeven bij de operaties naast spaties ook door komma's gescheiden kunnen worden. Het resultaat van Voorbeeld 9-3 is afgebeeld in Figuur 9-4.

Voorbeeld 9-3. Een cubische Bézier curve

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <path d="M0,200 C0,100 300,100 300,250"
        fill="none" stroke="blue" stroke-width="4" />
</svg>
```

Figuur 9-4. SVG voorbeeld: cubische Bézier curve

Quadratische Bézier curves worden eveneens ondersteund. Dit zijn curves met *één* controlepunt.

9.4. De basisvormen

SVG definieert *6* basisvormen:

- Rechthoeken
- Cirkels
- Ellipsen
- Lijnen
- Polylijnen

- Polygonen

Deze basisvormen steunen allemaal op het *path* element.

9.4.1. Rechthoeken

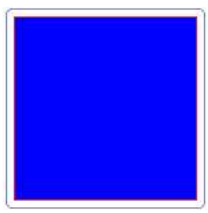
Om rechthoeken weer te geven in SVG wordt het rect element gebruikt. Voorbeeld 9-4 laat een voorbeeld zien van een rechthoek die gedefinieerd is door middel van de standaard attributen. Rond de binnenste rechthoek wordt een rechthoek getekend met afgeronde hoeken. Deze afgeronde hoeken kan men verkrijgen door de rx en ry attributen (optioneel) te gebruiken in de definitie van de rechthoek. Het resultaat van dit voorbeeld is te zien in Figuur 9-5.

Voorbeeld 9-4. Een rechthoek

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

  <svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
     <desc>Rectangle with sharp corners in round rectangle</desc>
     <rect x="20" y="20" width="360" height="360"
        fill="blue" stroke="red" stroke-width="2"/>
<rect x="5" y="5" width="390" height="390" rx="10" ry="10"
             fill="none" stroke="navy" stroke-width="1"  />
</svg>
```

Figuur 9-5. SVG voorbeeld: rechthoek met scherpe hoeken met daarrond een rechthoek met afgeronde hoeken



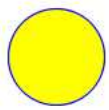
9.4.2. Cirkels

Het circle element wordt gebruikt voor de creatie van cirkels in SVG. Een cirkel wordt gedefinieerd door de coördinaten van het middelpunt van de cirkel (cx en cy) en zijn straal (r). Een voorbeeld dat een cirkel specificeert is te zien in Voorbeeld 9-5. Het resultaat van deze cirkel definitie is te zien in Figuur 9-6.

Voorbeeld 9-5. Een cirkel

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example of a circle</desc>
  <circle cx="200" cy="200" r="95"
        fill="yellow" stroke="blue" stroke-width="3"  />
</svg>
```

Figuur 9-6. Een SVG cirkel



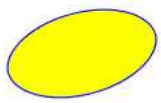
9.4.3. Ellipsen

Een andere basisvorm is de ellips. De creatie van een ellips in SVG gebeurt via het ellipse element. De attributen die kunnen meegegeven worden aan het element zijn de coördinaten van het middelpunt van de ellips (cx en cy), de straal in de x-richting (rx) en de straal in de y-richting (ry). Merk in dit voorbeeld ook het transform attribuut op, waarmee het huidige coördinatensysteem getransformeerd kan worden. Zonder deze transformatie zou het niet mogelijk zijn om bijvoorbeeld een schuine ellips te tekenen. Meer informatie over transformaties vind je in de SVG specificatie. Voorbeeld 9-6 laat een voorbeeld van een ellips zien. Het voorbeeld wordt gevisualiseerd door Figuur 9-7.

Voorbeeld 9-6. Een ellips

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example of a circle</desc>
  <ellipse cx="200" cy="200" rx="150" ry="80" transform="translate(-50 20) rotate(-15)"
        fill="yellow" stroke="blue" stroke-width="3"  />
</svg>
```

Figuur 9-7. Een SVG ellips



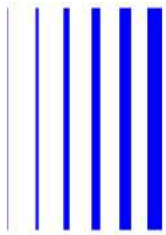
9.4.4. Lijnen

Het line element definieert een lijn van een startpunt (x1, y1) naar een eindpunt (x2, y2). Voorbeeld 9-7 is een voorbeeld waarin een aantal lijnen worden getekend met verschillende dikte. Het resultaat van deze code is te zien in Figuur 9-8.

Voorbeeld 9-7. Lijnen van verschillende dikte

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example of the line element</desc>
      <line x1="50" y1="10" x2="50" y2="390" stroke-width="1"/>
      <line x1="100" y1="10" x2="100" y2="390" stroke="blue" stroke-width="1"/>
      <line x1="150" y1="10" x2="150" y2="390" stroke="blue" stroke-width="5"/>
      <line x1="200" y1="10" x2="200" y2="390" stroke="blue" stroke-width="10"/>
      <line x1="250" y1="10" x2="250" y2="390" stroke="blue" stroke-width="15"/>
      <line x1="300" y1="10" x2="300" y2="390" stroke="blue" stroke-width="20"/>
      <line x1="350" y1="10" x2="350" y2="390" stroke="blue" stroke-width="25"/>
</svg>
```

Figuur 9-8. Lijnen van verschillende dikte



9.4.5. Polylijnen

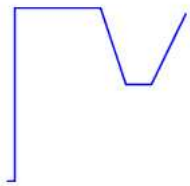
Met het polyline element kunnen we een opeenvolging van lijnen definiëren waarbij de volgende lijn begint waar de laatste eindigde. Het points attribuut wordt hierbij gebruikt om de opeenvolging van punten te geven die moeten worden verbonden door lijnsegmenten. Deze opsomming bestaat uit x,y coördinaatparen: x1,y1; x2,y2; x3,y3 ... Voorbeeld 9-8 laat een voorbeeld zien van een polylijn. Het resultaat is te zien in Figuur 9-9.

Voorbeeld 9-8. Polylijnen

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example of the line element</desc>
  <polyline fill="none" stroke="blue" stroke-width="4"
            points="15,390 30,390 30,50 200,50 250,200 300,200 370,60"/>

</svg>
```

Figuur 9-9. Een polylijn voorbeeld



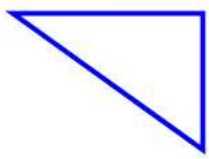
9.4.6. Polygonen

Net als een polylijn wordt een polygoon gedefinieerd door een lijst van punten. Bij een polygoon wordt echter het laatste punt verbonden met het eerste punt in de lijst om op deze manier een gesloten figuur te vormen. Voorbeeld 9-9 laat een voorbeeld zien van de definitie van een polygoon. Het resultaat van deze code is te zien in Figuur 9-10,

Voorbeeld 9-9. Een polygoon

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example of the line element</desc>
  <polygon fill="none" stroke="blue" stroke-width="10"
  points="15,30 390,30 390,300"/>
</svg>
```

Figuur 9-10. SVG polygoon



Hoofdstuk 10. SMIL: Synchronized Multimedia Integration Language

w3c SMIL 2.0 Specificatie (http://www.w3.org/TR/smil20)

10.1. Inleiding

De Synchronized Multimedia Integration Language (SMIL) biedt de mogelijkheid om verschillende onafhankelijke multimedia-objecten te integreren in een gesynchroniseerde multimedia-presentatie.

In dit hoofdstuk wordt verwezen naar een aantal artikels die SMIL uit de doeken doen. Deze artikels moeten als verplichte literatuur gezien worden.

10.2. Verplichte artikels

- *Multimedia Standards: Building Blocks of the Web* (http://http://www.cwi.nl/~media/publications/mediaImpact.pdf), Lloyd Rutledge
- SMIL 2.0: XML For Web Multimedia, Lloyd Rutledge

10.3. SMIL basis

In deze sectie wordt kort de SMIL basis elementen besproken. De voorbeelden kunnen in de X-Smiles (http://www.x-smiles.org/) browser worden uitgeprobeerd. We bespreken achtereenvolgens:

Bijlage A. GNU Free Documentation License

Version 1.2, November 2002

A.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject.

(Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this

License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

A.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

A.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.12. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliografie

Boeken

[xslt-pr] Michael Kay, 2001, Wrox Press, 1-861005-06-7, *XSLT, Programmer's Reference*.

[xml-pro] Richard Anderson, Mark Birbeck, Michael Kay, Steven Livingstone, Brian Loesgen, Didier Martin, Stephen Mohr, Nikola Ozu, Bruce Peat, Jonathan Pinnock, Peter Stark, en Kevin Williams, 2000, Wrox Press, 1-861003-11-0, *Professional XML*.

[xml-21days] Simom North en Paul Hermans, 1999, 1-57521-396-6, *Teach Yourself XML in 21 Days*, Sams.

[docbook-1.0.3] Norman Walsh en Leonard Muellner, 1999, 1-56592-580-7, *DocBook: The Definitive Guide (http://www.oasis-open.org/docbook/documentation/reference/html/docbook.html)*.

[DesignPatterns] Eric Gamma, Richard Helm, Ralph Johnson, en John Vlissides, 1994, 0-201-63361-2, *Design Patterns: Elements of Reusable Object-Oriented Software*.

Artikels

[sax-power] Benoit Marchal, 2001, IBM, *SAX, the power API (http://www-106.ibm.com/developerworks/xml/library/x-saxapi/)*.

[drday-00] Don R. Day, 2000, IBM, *XSL for fun and diversion (http://www-106.ibm.com/developerworks/library/hands-on-xsl/)*.

[xml-faq] Peter Flynt, 2002, *XML Frequently Asked Questions (http://www.ucc.ie/xml/)*.

[xsl-faq] Dave Pawson, 2001, *XSL Frequently Asked Questions (http://www.dpawson.co.uk/xsl/xslfaq.html)*.

[xsl-what] Michael Kay, 2000, IBM, *What kind of language is XSLT? (http://www-106.ibm.com/developerworks/xml/library/x-xslt/)*.

[XML-Java-99] Doug Tidwell, 1999, IBM, *Tutorial: XML Programming in Java (http://www-106.ibm.com/developerworks/education/xmljava/)*.

[SVG-nl-tut] 2002, DeDS, *SVG tutorial (http://home.deds.nl/~svg/)*<.

Nieuwsgroepen - Mailinglists

comp.text.xml.

XSLT Mailing List (http://www.mulberrytech.com/xsl/xsl-list/).

DocBook Mailing List (http://www.oasis-open.org/docbook/mailinglist/).

DocBook Applications Mailing List (http://www.oasis-open.org/docbook/mailinglist/).