Beginselen van XML

Leereenheid 2

Beginselen van XML

INTRODUCTIE

In leereenheid 1 maakte u kennis met algemene kenmerken en toepassingsmogelijkheden van XML. Deze tweede leereenheid is meer gericht op de syntaxis van XML: we gaan na hoe een XML-document is opgebouwd, en hoe de onderdelen die erin kunnen voorkomen zich tot elkaar verhouden. Hierbij beperken we ons voornamelijk tot algemene welgevormdheidskenmerken. Specifieke XML-grammatica's (DTD's en schema's) worden in de leereenheden 3 en 4 behandeld.

LEERDOELEN

Na het bestuderen van deze leereenheid wordt verwacht dat u

- inzicht hebt in structuur en syntaxis van XML-documenten
- inzicht hebt in gebruik en functie van de markuptypen element, entiteitverwijzing, commentaar, verwerkingsinstructie, gemarkeerde sectie en documenttypedeclaratie
- inzicht hebt in het verschil tussen twee typen gegevensinhoud: elementgegevens en attribuutgegevens
- inzicht hebt in welgevormdheid en het gebruik van parsers ter controle daarvan
- inzicht hebt in de noodzaak van namespaces en het gebruik ervan
- inzicht hebt in de ondersteuning van internationale tekensets
- XML-documenten kunt schrijven met een XML-editor
- korte XML-documenten of -fragmenten handmatig kunt controleren op welgevormdheid
- XML-documenten kunt controleren op welgevormdheid met een parser.

Studeeraanwijzingen
De benodigde voorbeelden of bouwstenen van deze leereenheid zijn gebundeld in de oXygen-projecten xml02_beginselen en xml02_reisboeking.

Studielast
De studielast van deze leereenheid bedraagt 6 uur.

Bronnen
– De W3C-aanbeveling voor XML 1.0: http://www.w3.org/TR/REC-xml/
– De W3C-naslag voor XML-namespaces: http://www.w3.org/TR/REC-xml-names/

LEERKERN

1 Voorbeeld: een boeking voor een reis

In deze paragraaf geven we een voorbeeld van een wat groter XML-document. Eigenlijk is het nog steeds klein, want het gemiddelde XML-document is groot! Eerst schetsen we het kader waarin het document begrepen moet worden.

1.1 REISORGANISATIE 'WEGISWEG'

Een reisorganisatie genaamd WegIsWeg omvat een flink aantal kleine, lokale reisbureaus. WegIsWeg heeft een centrale relationele database. Voor de communicatie wordt gebruikgemaakt van XML. Dit betekent dat aanvragen en boekingen in de vorm van XML-documenten worden aangeleverd aan de database, die ze vervolgens (na bewerking) relationeel opslaat. Omgekeerd wordt informatie uit de database, zoals een bevestiging van een voorlopige boeking, in XML-vorm geëxporteerd. De aangesloten reisbureaus zijn vrij in het gebruik van platforms en software, zolang ze zich maar aan de voorgeschreven XML-uitwisselingsformaten houden.

1.2 EEN BOEKINGSBEVESTIGING

Hier volgt een voorbeeld van een XML-boekingsbevestiging. Het document is in de projectmap xml02_reisboeking te vinden onder de naam reisboeking1.xml.
Om het niet al te ingewikkeld te maken hebben we bij geldbedragen geen btw-informatie opgenomen en evenmin valuta-informatie. Alle bedragen zijn in euro's.

Reisboeking1.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!-- van: WegIsWeg; aan: ErOpUit; versie: 20100115T14:15:22; status:
    goedgekeurd -->
3   <?xml-stylesheet type="text/xsl" href="reisboeking_xhtml.xsl"?>
4   <reisboeking nr="A192837" totaal="&amp;#8364; 3425.00">
5       <klant klantnr="12345">
6           <naam>van Amstel</naam>
7           <voorletters>J</voorletters>
8           <geboortedatum>1956-07-17+01:00</geboortedatum>
9           <identification country="NL" idType="passport" id="N1234567"/>
10          <email>j.vanamstel@ou.nl</email>
11          <vasteKlant/>
12      </klant>
13      <reiscomponenten>
14          <vervoer subtotaal="380">
15              <vlucht>
16                  <maatschappij>O&amp;amp;U Airlines</maatschappij>
17                  <vluchtnr>K60</vluchtnr>
18                  <vertrek>
19                      <locatie>Maastricht/Aken Airport</locatie>
20                      <datumTijd>2010-07-01T11:33+01:00</datumTijd>
21                  </vertrek>
22                  <aankomst>
23                      <locatie>Dublin Airport</locatie>
24                      <datumTijd>2010-07-01T11:52+00:00</datumTijd>
25                  </aankomst>
26              </vlucht>
```

```
27              <bedrag>190.00</bedrag>
28          </vervoer>
29          <verblijf subtotaal="270">
30              <hotel>
31                  <naam>Dubliners</naam>
32                  <locatie>Dublin</locatie>
33                  <kamertype>2p en suite</kamertype>
34              </hotel>
35              <periode>
36                  <datumVanaf>2010-07-01+00:00</datumVanaf>
37                  <datumTot>2010-07-04+00:00</datumTot>
38              </periode>
39              <bedrag>270.00</bedrag>
40          </verblijf>
41          <vervoer subtotaal="80">
42              <trein>
43                  <vertrek>
44                      <locatie>Dublin Central Station</locatie>
45                      <datumTijd>2010-07-05T09:43+00:00</datumTijd>
46                  </vertrek>
47                  <aankomst>
48                      <locatie>Banagher</locatie>
49                      <datumTijd>2010-07-05T11:33+00:00</datumTijd>
50                  </aankomst>
51              </trein>
52              <bedrag>40.00</bedrag>
53          </vervoer>
54          <verblijf subtotaal="1450">
55              <jacht>
56                  <jachtwerf>XYachting</jachtwerf>
57                  <jachttype>BaMaSuper</jachttype>
58                  <lengte eenheid="m">9</lengte>
59                  <maxPersonen>3</maxPersonen>
60              </jacht>
61              <periode>
62                  <datumVanaf>05-jul-2010+00:00</datumVanaf>
63                  <datumTot>12-jul-2010+00:00</datumTot>
64              </periode>
65              <bedrag>1450.00</bedrag>
66          </verblijf>
67          <vervoer subtotaal="50">
68              <bus>
69                  <maatschappij>Connemara Tours</maatschappij>
70                  <vertrek>
71                      <locatie>Banagher</locatie>
72                      <datumTijd>2010-07-12T13:28+00:00</datumTijd>
73                  </vertrek>
74                  <aankomst>
75                      <locatie>Galway</locatie>
76                      <datumTijd>2010-07-12T15:25+00:00</datumTijd>
77                  </aankomst>
78              </bus>
79              <bedrag>25.00</bedrag>
80          </vervoer>
81          <vervoer>
82              <vlucht subtotaal="470">
83                  <maatschappij>XFlights</maatschappij>
84                  <vluchtnr>B106</vluchtnr>
85                  <vertrek>
86                      <locatie>Banagher</locatie>
87                      <datumTijd>2010-07-12T18:01+00:00</datumTijd>
```

```
88                  </vertrek>
89                  <aankomst>
90                      <locatie>Schiphol</locatie>
91                      <datumTijd>2010-07-12T21:15+00:00</datumTijd>
92                  </aankomst>
93              </vlucht>
94              <bedrag>235.00</bedrag>
95          </vervoer>
96      </reiscomponenten>
97      <reizigers aantalPersonen="2">
98          <persoon>
99              <naam>van Amstel</naam>
100             <voorletters>J</voorletters>
101             <identification country="NL" idType="passport" id="N1234567"/>
102         </persoon>
103         <persoon>
104             <naam>Maas</naam>
105             <voorletters>P</voorletters>
106             <identification country="NL" type="passport" id="N7654321"/>
107         </persoon>
108     </reizigers>
109 </reisboeking>
```

Toelichting
– Na de bekende XML-declaratie op regel 1 volgt een commentaar tussen de tags <!-- en --> (ook bekend van HTML). De inhoud hiervan wordt genegeerd door de XML-parser, maar de verwerkende applicatie kan er eventueel wel wat mee doen.
– Op regel 3 volgt een *processing instruction*. Dit is een instructie aan de verwerkende applicatie. In dit geval is het een verwijzing naar een *stylesheet*, met instructies voor de manier waarop het document visueel moet worden weergegeven door een browser. (De stylesheet maakt geen deel uit van het project.)
– Zoals in elk XML-document is er één rootelement. Dit is het element reisboeking, met een openingstag op regel 4 en een sluitttag op regel 109.
– Het element reisboeking omvat een element klant, een element reiscomponenten en een element reizigers.
– Het reiscomponenten-element bevat een opeenvolging van vervoer- en verblijfcomponenten. Een vervoercomponent is een vlucht-, een trein- of een bus-element. Een verblijfcomponent is een hotel- of een jacht-element.
– Attributen geven aanvullende kenmerken, bijvoorbeeld bij boeking een uniek nummer, subtotalen en een totaalbedrag.
– Diverse regels (waaronder de regels 8 en 20) bevatten een datum- of een datum/tijd-aanduiding. In paragraaf 1.3 gaan we hier dieper op in.
– Door dieper in de XML-boom te kijken ontdekt u verdere subelementen met hun gegevensinhoud.

De XML-taal ReisboekingML
Het document reisboeking1.xml is er één van vele soortgelijke. Deze zijn geschreven in een gemeenschappelijke taal, de 'XML-taal voor WegIsWeg-reisboekingen'. We zullen deze taal een naam geven: ReisboekingML, de 'reisboeking markup language'. In de leereenheden 3 en 4 zullen we grammatica's ontwerpen voor de taal ReisboekingML.

1.3 WERELDWIJDE DATUM- EN TIJDREKENING

XML heeft als wereldwijd uitwisselingsformaat niet alleen te maken met internationale tekensets, maar ook met internationale tijdrekening. Hiervoor zijn verschillende gestandaardiseerde formaten ontwikkeld. Eén daarvan is de 'dateTime-notatie', geassocieerd met het XML Schema dateTime-datatype, dat in leereenheid 4 wordt behandeld. Een voorbeeld uit reisBoeking1.xml:

```
2010-07-01T11:33+01:00
```

Uitleg

2010-07-01T11:33 is de universele UTC-tijd (datum yyyy-mm-dd gevolgd door tijdsaanduiding Tuu:mm in minuten nauwkeurig). Daarna volgt een 'offset' voor de tijdzone, in dit geval +1 uur, voor de Midden-Europese tijd. Offset (die ook negatief kan zijn) opgeteld bij de UTC geeft de tijd van de tijdzone. De lokale tijd kan nog weer afwijken vanwege seizoenstijd (zomer- of wintertijd). De UTC-tijd is ongeveer gelijk aan GMT (Greenwich Mean Time), maar wijkt hiervan een beetje af, omdat GMT rekening houdt met de (afnemende) rotatiesnelheid van de aarde terwijl UTC puur astronomisch is. De wereldwijde datum- en tijdproblematiek is uiterst complex, maar vanuit XML-standpunt – dat immers een mondiaal standpunt is – erg belangrijk.

Bronnen

Indien u meer hierover wilt weten, raadpleeg dan de volgende W3C-bronnen:
– http://www.w3.org/TR/xmlschema11-2/#dateTime
– http://www.w3.org/TR/timezone/
Ook kunt u zoeken op UTC of ISO 8601.

1.4 UITWISSELINGSFORMAAT VERSUS OPSLAGSTRUCTUUR

De structuur van het XML-document (een boomstructuur) is anders dan een relationele-databasestructuur (een verzameling tabellen). De reden hiervan is dat de XML-structuur een uitwisselingsformaat is en geen opslagformaat. Voor uitwisselingsstructuren zijn – ook bij één achterliggende opslagstructuur – vele alternatieven denkbaar, afhankelijk van het doel van de uitwisseling. Is het document bijvoorbeeld bedoeld om gegevens aan te leveren aan een relationele database? Of is het juist output van een relationele database en input voor een rapportageapplicatie?

Gezien de commentaarregel aan het begin van reisboeking1.xml gaat het hier om een bevestiging (een 'bevestigingsrapport'). We nemen aan dat de gegevens hierin afkomstig zijn uit een relationele database van WegIsWeg. Reisboeking1.xml is als rapport nogal abstract: inhoudelijk compleet, maar nog niet goed leesbaar voor mensen. Om een voor mensen goed leesbaar rapport te genereren is nog een transformatie nodig, afgestemd op het medium (papier, beeldscherm, mobiel device, …).

Merk op dat het XML-document de nodige redundante informatie bevat. Met name zijn alle subtotaal-attribuutwaarden en ook het totaal-attribuut van het root-element afleidbaar uit de bedrag-waarden en het aantal personen. In een relationele opslagstructuur zou dit een probleem zijn. In een rapportage of bericht is dit echter geen bezwaar. We kunnen het

document zien als een rapport dat is 'berekend' uit de database-inhoud van WegIsWeg.

Overigens is de structuur van reisboeking1.xml mede gekozen met het oog op de uitleg van XML-grammatica's in de leereenheden 3 en 4. Daarbij zullen we ook enkele alternatieven geven.

OPGAVE 2.1

Een XML-document kan met elke editor worden aangemaakt en opgeslagen. Maar natuurlijk geniet een specifieke XML-editor de voorkeur, omdat zo'n editor een parser bevat die in staat is te controleren of de tekst XML-welgevormd is en eventueel ook valide met betrekking tot een DTD of schema. Bovendien maakt zo'n editor u het leven gemakkelijk door bijvoorbeeld 'code completion'. Zo'n editor is die van oXygen. U hebt er in leereenheid 1 al mee kennisgemaakt.

a Open in Eclipse/oXygen het XML-document reisboeking1.xml van het project xml02_beginselen.

b Let op de verschillende kleuren voor elementnamen (tevens tagnamen), de gegevensinhoud tussen de elementtags, attribuutnamen en attribuutwaarden.

c Het is nuttig eens een stuk tekst (tenminste een heel element) te verwijderen en opnieuw in te typen. Merk op dat automatische *code completion* plaatsvindt (toevoegen van eindtags en eindaanhalingstekens).

d Maak opzettelijk enige fouten en bekijk de foutmeldingen van Xerces (de onderliggende XML-parser).

Tips Eclipse

1 U kunt een actie ongedaan maken met Ctrl-z.

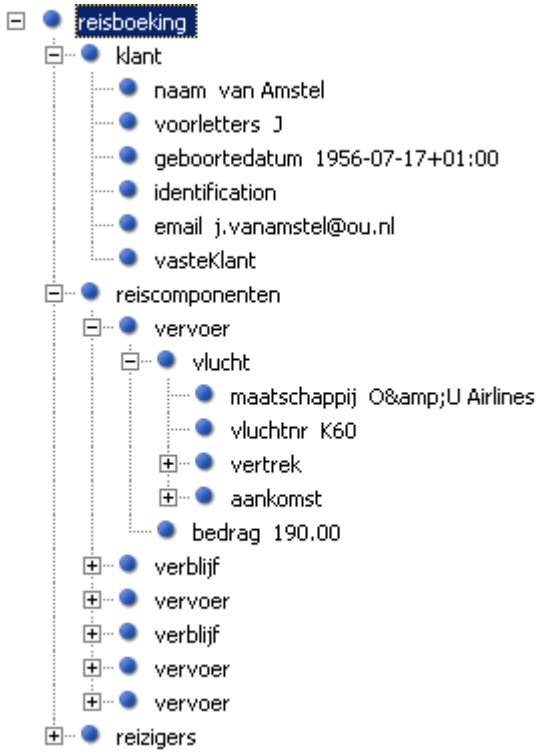
2 Let op met aanhalingstekens wanneer u tekst kopieert vanuit een andere editor of tekstverwerker. Wanneer hierdoor bijvoorbeeld 'gekrulde aanhalingstekens' in de XML-tekst terechtkomen (wat niet goed zichtbaar is), geeft dit een foutmelding.

e Bekijk het welgevormde XML-document in grid-view (figuur 2.1) en in Outline-view (figuur 2.2).

f Experimenteer zoveel als u kunt met de XML-omgeving van oXygen. Bekijk zo nodig nog eens de basishandleiding oXygen en die van Eclipse. Bewaar reisboeking1.xml uiteindelijk in de oorspronkelijke versie.

<?xml	version="1.0" encoding="UTF-8"			
<!--	van: WegIsWeg; aan: ErOpUit; versie: 20100115T14:15:22; status: goedgekeurd			
<?xml-styles...	type="text/xsl" href="reisboeking_xhtml.xsl"			
reisboeking	@xmlns	http://www.wegisweg.com/boekingen		
	@nr	A192837		
	@totaal	€ 3425.00		
	klant	@klantnr	12345	
		naam	van Amstel	
		voorletters	J	
		geboortedatum	1956-07-17+01:00	
		identific...	@country	NL
			@idType	passport
			@id	N1234567
		email	j.vanamstel@ou.nl	
		vasteKlant		
	reiscompo...	vervoer		
		verblijf		
		vervoer		
		verblijf		
		vervoer (2 rows)		
	reizigers			

FIGUUR 2.1 Reisboeking1.xml in grid-view



FIGUUR 2.2 Reisboeking1.xml in Outline-view

2 XML-welgevormdheid

Welgevormdheid

Een XML-document moet aan algemene regels voldoen, die samen de algemene XML-grammatica vormen. Dataobjecten die daaraan voldoen heten XML-*welgevormd*. Meestal zullen we het hebben over *welgevormde* XML-*documenten*, hoewel 'XML-document' al 'welgevormd' impliceert.

Validiteit

Valide documenten
Voor de meeste toepassingen moet een XML-document aan aanvullende, toepassingsspecifieke regels voldoen. XML-welgevormde documenten die daaraan voldoen heten *valide*. Talen waarin zulke aanvullende regels worden vastgelegd zijn onder meer DTD (documenttypedefinitie) en XML Schema. De DTD-taal wordt behandeld in leereenheid 3. XML Schema is het onderwerp van leereenheid 4.

2.1 DE ONDERDELEN VAN EEN WELGEVORMD XML-DOCUMENT

Een welgevormd XML-document is als volgt opgebouwd:

XML-declaratie (optioneel)
nul of meer commentaren
nul of meer processing instructions
} proloog
één root element

Het document begint doorgaans met de XML-declaratie. Daarna mogen commentaren en/of processing instructions volgen, in willekeurige volgorde. Het document eindigt met één (verplicht) *root-element*.
Alles wat aan het root-element voorafgaat heet de *proloog*. In paragraaf 3 gaan we in op de onderdelen van de proloog .

Root-element
Proloog

In paragraaf 4 kijken we naar de structuur van het root-element, met haar subelementen en de attributen van de elementen.

In paragraaf 5 zullen we de character-content, de gegevens die – in de vorm van elementinhoud en attribuutwaarden – met het XML-document aan de ontvangende applicatie worden verstuurd, aan een nader onderzoek onderwerpen.

2.2 XML-PARSERS

Een XML-document is een uitwisselingsdocument. Dit betekent dat er een zenderapplicatie is en een ontvangerapplicatie. De zender moet ervoor zorgen dat het XML-document welgevormd is en de ontvangerapplicatie moet daarop kunnen controleren. Een analoge opmerking kan gemaakt worden over validiteit.

XML-parser

In deze leereenheid zullen we het perspectief van de ontvanger kiezen. Onderdeel van de ontvangerapplicatie is een niet-applicatiespecifieke XML-*parser*. Deze heeft de volgende taken:

1 de fysieke structuur omvormen tot een logische structuur
2 de externe representatie omvormen tot een interne representatie
3 controle op welgevormdheid
4 controle op validiteit.

Toelichting
Ad 1 Een 'logisch' XML-document moet vaak uit verschillende fysieke onderdelen worden geassembleerd. Een voorbeeld is de import van 'karakterentiteiten' voor speciale tekens vanuit bepaalde publiek

toegankelijke DTD's waarnaar vanuit het XML-document wordt verwezen. (Meer hierover in leereenheid 3.)

Ad 2 De voor mensen leesbare vorm van een XML-document ondergaat een bewerking alvorens het aan de applicatie wordt aangeboden. Zo worden alle namespace-aliassen verwijderd en vervangen door de respectievelijke namespace-strings.

Ad 3 Een parser is letterlijk een 'ontleder'. Een parser probeert een aangeboden document te ontleden volgens de in de parser ingebouwde grammaticale XML-regels. Als dat lukt, is het document welgevormd.

Validerende parser

Ad 4 Een *validerende parser* probeert een aangeboden document tevens te ontleden volgens de grammaticale regels van een DTD of schema. Als dat lukt, is het document valide.

XML-processor

Een parser is per definitie gericht op syntaxis en validatie. Een XML-parser maakt vaak deel uit van een meer omvattende XML-*processor*, die alle generieke XML-gerelateerde taken ten behoeve van de applicatie verricht. Een XML-processor kan bijvoorbeeld een applicatie toegang geven tot de juiste Java-XML-API's.

3 De proloog van een XML-document

Proloog

In deze paragraaf gaan we in op de drie optionele onderdelen van de proloog:
– de XML-declaratie
– commentaren
– processing instructions.

3.1 XML-DECLARATIE

XML-declaratie

Een XML-document begint bijna altijd met een *XML-declaratie* <?xml … ?>. De belangrijkste attributen zijn:
– version, voor de XML-versie
– encoding, voor de in het XML-document gebruikte codering van de tekenset (met als default US-ASCII). Een veelgebruikte codering is UTF-8 bij de tekenset Unicode, die we in de meeste van de XML-declaraties van deze cursus zullen aantreffen:

```
<?xml version="1.0" encoding="UTF-8"?>
```

De ontvangende applicatie zal de XML-content (elementinhoud en attribuutwaarden) interpreteren conform de in de XML-declaratie genoemde codering met de daarbij horende tekenset.

De XML-declaratie is niet verplicht. De XML-aanbeveling maakt onderscheid tussen *must*'s and *should*'s. Over de XML-declaratie (zie paragraaf 2.8 Prolog and Documenttypedeclaratie) wordt gezegd: 'XML documents *should* begin with an XML declaration which specifies the version of XML being used'.

Voor de volledigheid geven we de productieregel voor de XML-declaratie, dat is de grammaticale regel die vastlegt hoe een welgevormde XML-declaratie is opgebouwd:

Productieregel, zie http://www.w3.org/TR/REC-xml

```
XMLDecl  ::=  '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
```

Hieruit zien we dat tussen de tags <?xml en ?> verplicht de XML-versie moet worden opgenomen, optioneel encoding-informatie over de gebruikte tekenset en optioneel nog een 'standalone document declaration', die in deze cursus niet van belang is. De S duidt op whitespace; de vraagtekens (buiten de markup <? en ?>) geven optionaliteit aan.

3.2 COMMENTAREN

In een XML-document kan commentaar worden opgenomen. Een voorbeeld zien we in het volgende XML-fragment:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- dit document beschrijft … -->
…
```

Commentaren mogen ook verderop in een XML-document voorkomen, dat wil zeggen na de proloog.

De XML-aanbeveling laat parsers vrij om commentaar al dan niet aan de verwerkende applicatie door te geven. Soms is het essentieel dat het wordt doorgegeven. Dit geldt bijvoorbeeld voor JavaScript-code in XHTML, die wordt genoteerd in een vorm die syntactisch gezien een commentaar is.

3.3 PROCESSING INSTRUCTIONS

Processing instruction

Aan het begin van een XML-document, direct na de XML-declaratie, kunnen één of meer *processing instructions* voorkomen. Dit zijn instructies aan het verwerkende programma die aangeven hoe het XML-document moet worden verwerkt. In reisboeking1.xml hebben we een voorbeeld gezien:

```
<?xml-stylesheet type="text/xsl" href="reisboeking_xhtml.xsl"?>
```

Het gaat hier om een processing instruction aan de ontvangende applicatie (in dit geval een browser) die aangeeft dat het XML-document moet worden weergegeven door middel van een stylesheet, genaamd reisboeking_xhtml.xsl. De stylesheet voert een transformatie uit naar XHTML, wat in de naam tot uitdrukking komt in het suffix _xhtml. De stylesheet (niet aanwezig in de projectmap) is geschreven in de taal XSLT, hetgeen zichtbaar is aan de extensie xsl. XSLT wordt behandeld in leereenheid 7.

Net als commentaren mogen processing instructions ook na de proloog in een XML-document voorkomen.

4 Elementen en attributen

XML-documenten zijn – afgezien van de XML-declaratie, het commentaar en de processing instructions (zie paragraaf 3) – opgebouwd als een hiërarchie van elementen, elk met nul of meer attributen.

4.1 ELEMENTNAMEN EN ATTRIBUUTNAMEN

Belangrijk is te weten dat XML hoofdlettergevoelig is. Dit geldt dus zowel voor elementnamen als voor attribuutnamen. Beide volgen de productieregel voor 'Name' (XML-naam) uit de XML-aanbeveling:

```
Name  ::=  NameStartChar (NameChar)*
```

Dit wil zeggen: een Name bestaat uit een NameStartChar (beginteken) gevolgd door nul of meer NameChar's (naamtekens). Naamtekens omvatten grofweg letters, cijfers, de underscore (_) en nog een aantal andere tekens. De toegestane letters omvatten meer dan alleen A-Z en a-z.

Er is een aantal beperkingen, waaronder:
– het beginteken mag geen cijfer zijn
– een elementnaam mag niet beginnen met 'xml' of een variant hiervan met één of meer hoofdletters.

Hoewel een punt (.) of een streepje (-) niet verboden zijn, kunnen deze tekens beter worden vermeden, daar ze door een applicatie verkeerd kunnen worden opgevat, namelijk als objectoperator respectievelijk rekenkundige bewerking. Ook de dubbele punt is toegestaan, maar gebruik wordt afgeraden omdat de dubbele punt wordt gebruikt om een Name te scheiden van een namespace-alias.

Wanneer u hier meer over wilt weten, kunt u de XML-aanbeveling op dit punt bestuderen. U moet hiervoor wel eerst het onderwerp 'tekenverwijzingen' hebben bestudeerd.

4.2 ELEMENTEN

Root-element

Elementen worden weergegeven door een openings- en een sluittag en kunnen eindeloos in elkaar worden genest. Een welgevormd XML-document bevat altijd één element, het *root-element*, waarbinnen alle andere zijn genest.

Leeg element

Een element kan *leeg* zijn, bijvoorbeeld:

```
<vasteKlant></vasteKlant>
```

Zo'n leeg element kan, zoals in reisboeking1.xml is gebeurd, worden afgekort met één tag:

```
<vasteKlant/>
```

Een leeg element kan wel attributen bevatten. Een voorbeeld uit reisboeking1.xml:

```
<identification country="NL" idType="passport" id="N1234567"/>
```

OPGAVE 2.2
Welke van de volgende namen zijn niet toegestaan als elementnaam? Waarom niet?
a Student

b studieresultaat:tentamencijfer
c Cursus Inschrijving
d 4de_klas_mentor
e XML_cursusdocumenten

OPGAVE 2.3

a Wat is het verschil tussen de tags <abc/> en </abc>?
b Legt het gebruik van één van deze tags condities op aan het XML-document met betrekking tot de welgevormdheid?
c Kunnen beide tags in één XML-document voorkomen?

4.3 ATTRIBUTEN

Attribuut

Een element kan als inhoud (content) character data hebben. Een ander type inhoud kan aan een element worden meegegeven in de vorm van attributen. Een *attribuut* is een tweetal, bestaande uit een attribuutnaam en een attribuutwaarde, en wel als volgt:

<attribuutnaam>="attribuutwaarde"

Attributen: een verzameling
Een element kan nul of meer attributen hebben. Deze vormen een verzameling. Dit betekent dat de volgorde van attributen in een element geen betekenis heeft, in het bijzonder niet voor de verwerkende applicatie. Het betekent ook dat een element geen gelijknamige attributen mag bevatten.

Waarom attributen?
Een belangrijke vraag is: waartoe dienen attributen? Kan hun inhoud niet net zo goed (of zelfs beter) als elementinhoud worden gemodelleerd? Het is niet zo gemakkelijk op deze vraag een antwoord te geven. Het is namelijk altijd mogelijk attributen te vermijden door een XML-documentstructuur te kiezen met alleen elementinhoud (zie opgave 2.4).

Historisch gezien – kijkend naar SGML en HTML – dienden attributen vooral voor metadata, ofwel gegevens óver de gegevens. In (X)HTML zien we dit nog steeds wel, denk bijvoorbeeld aan de trefwoorden voor zoekmachines. In de praktijk ziet men attributen vaak gebruikt worden voor gewone niet-gestructureerde document-content. In deze cursus zullen we geen scherpe criteria geven. De wijze van modelleren (subelement of attribuut) zal vaak willekeurig zijn. Dit wijkt niet af van wat de praktijk te zien geeft. Het is goed hierbij te bedenken dat het bij XML primair om gegevensuitwisseling gaat en dat daarbij niet de strenge regels gelden zoals die voor bijvoorbeeld relationele-datatabaseopslag opgaan.

OPGAVE 2.4

Hierna volgt een stukje uit het XML-document reisboeking1.xml dat betrekking heeft op een element persoon.

```
<persoon>
  <naam>van Amstel</naam>
  <voorletters>J</voorletters>
  <identification land="NL" idType="passport" id="N1234567"/>
</persoon>
```

a Werk het element persoon om, zodanig dat alle attributen worden gehermodelleerd tot subelementen. Probeer te motiveren of dit een verbetering is of juist niet.
b Geef een alternatief waarbij land en idType worden gehandhaafd als attribuut en het id-nummer elementinhoud wordt van identification.

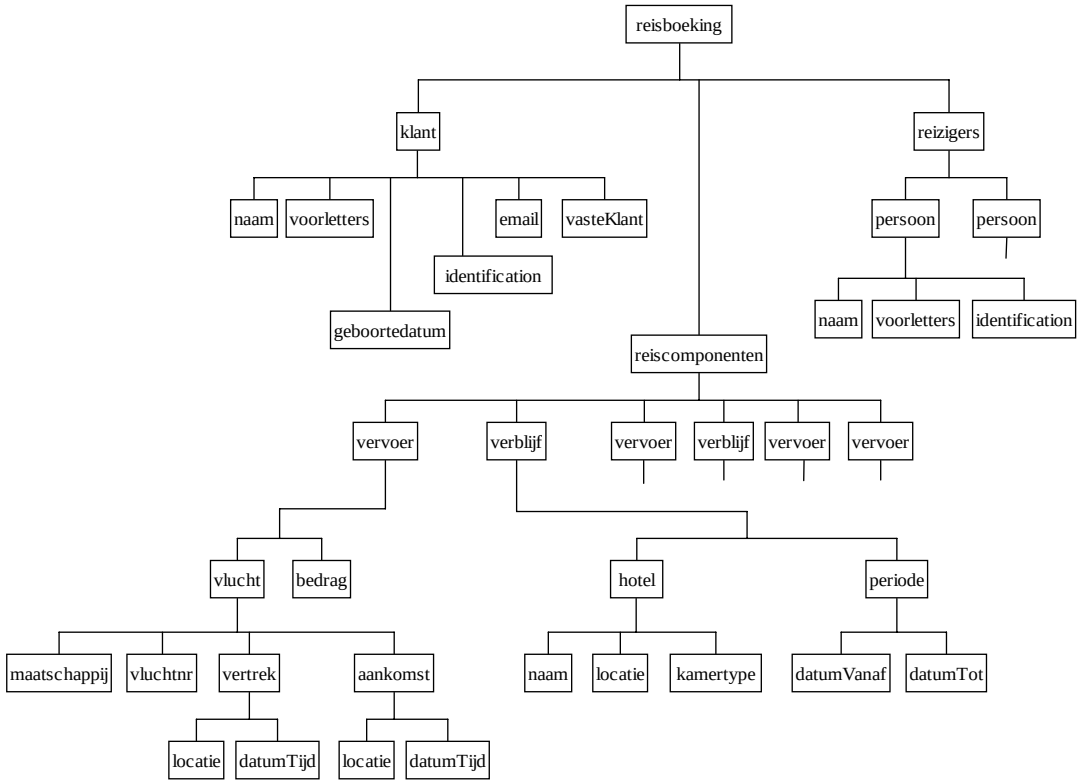
4.4 DE HIËRARCHISCHE STRUCTUUR VAN EEN XML-DOCUMENT

We hebben al herhaaldelijk gezien dat een XML-document een hiërarchische elementenstructuur (boomstructuur) heeft. Zie bijvoorbeeld de weergave van figuur 2.2 of die in een browser, beide met in- en uitklapbare elementen.

Element nodes

Element nodes
Parent
Child

De knooppunten in de boomstructuur heten *nodes*. Specifiek: *element nodes*. Op deze nodes zijn de hiërarchische begrippen *parent* (ouder) en *child* (kind) van toepassing. Bijvoorbeeld: de vervoer- en verblijf-nodes zijn child van de reiscomponenten-node, en de reiscomponenten-node is parent van de vervoer- en verblijf-nodes. Zie figuur 2.4



FIGUUR 2.4 XML-boom van reisboeking1.xml met element nodes

Text nodes

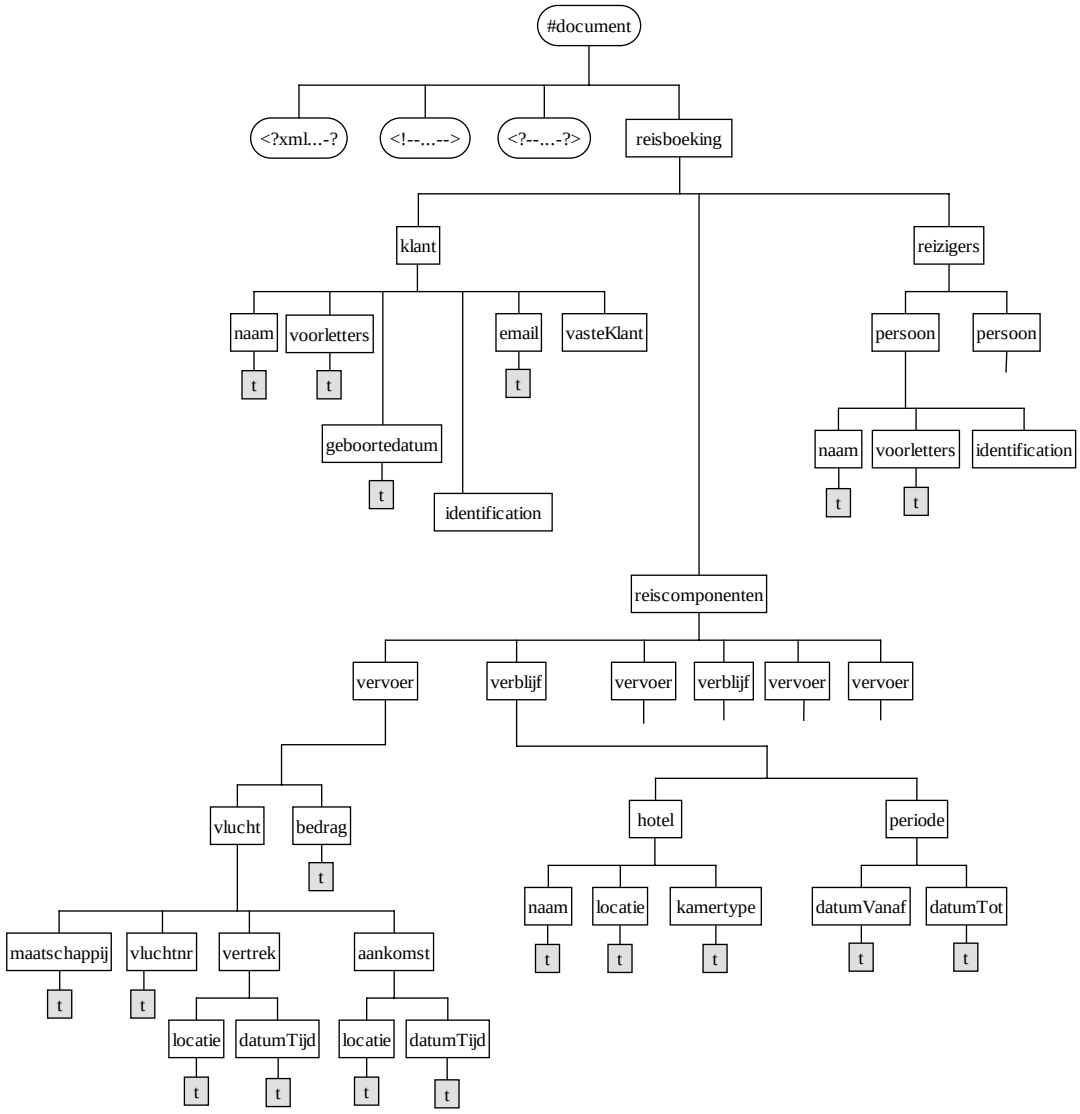
Text nodes

De volledige XML-boomstructuur bevat meer dan alleen elementen: de boomstructuur omvat ook *text nodes*, die childs zijn van hun elementen en die daarvan de character-data-inhoud vormen. In figuur 2.5 zijn *text nodes* weergegeven als grijze t-blokjes. Het t-blokje dat child is van het

email-element staat bijvoorbeeld voor 'j.vanamstel@ou.nl'. Merk op dat lege elementen geen text-childs hebben.
Ook de attributen voegen zich als *attribute nodes* in de hiërarchische structuur. Deze zijn in figuur 2.5 niet gevisualiseerd.

Parent- en child-relaties voor attributen
Formeel ligt de relatie tussen attributen en elementen, en tussen attributen en attribuutwaarden nogal ingewikkeld. Hier lopen we onder andere tegenaan bij de talen XPath en XQuery, waarbij woorden als child en parent worden gebruikt om door een XML-boom te navigeren. Volgens de formele specificatie geldt:
1 Een element is een parent van elk van zijn attribuut nodes.
2 Een attribuut is geen child van zijn ouderelement.
3 Welke relatie er bestaat tussen een attribuutnode en de bijbehorende attribuutwaarde is niet gespecificeerd.

We komen hierop in de betreffende leereenheden terug.



FIGUUR 2.5 XML-boom van reisboeking1.xml met element nodes en text nodes

Proloognodes
Ten slotte bevat de proloog van een XML-document nog één of meer nodes: in reisboeking1.xml zijn dat een node voor de XML-declaratie, een commentaarnode en een processor instruction-node. Beide nodes zijn van hetzelfde niveau ofwel 'zusjes' van de root node. Om deze niet 'in de lucht te laten hangen', is de *documentnode* als extra topnode gedefinieerd die wordt aangeduid met #document. Zie figuur 2.5.

Documentnode

De in deze paragraaf behandelde hiërarchie zal later in deze cursus een belangrijke rol spelen, met name in leereenheid 6 over XPath.

4.5 GEMENGDE INHOUD

In opgave 2.4 hebben we twee alternatieven gegeven voor modellering van het element identification. We geven nu nog een variant, en wel een heel opmerkelijke omdat daarin het element identification een *gemengde inhoud* (*mixed content*) heeft: een opeenvolging van een subelement en character data.

Gemengde inhoud
Mixed content

```
<persoon>
   <naam>van Amstel</naam>
   <voorletters>J</voorletters>
   <identification idType="passport">
      <country>NL</country>N1234567
   </identification>
</persoon>
```

We zien: het subelement country wordt direct gevolgd door character data. Zo'n voorbeeld hebben we nog niet eerder gezien.

Gemengde inhoud en whitespace
Wanneer we in een 'gewoon' document zoals reisboeking1.xml de daarin voorkomende whitespace (of een deel ervan) opvatten als significante character data, dan krijgt een deel van de elementen gemengde inhoud! Dit is natuurlijk bij dit voorbeeld in hoge mate onwaarschijnlijk, maar technisch is het mogelijk dat een bijbehorende XML-grammatica (bijvoorbeeld een schema) een dergelijke structuur voorschrijft. De parser zal dan conform de voorgeschreven structuur van de grammatica bepaalde whitespace als significant markeren en als zodanig aan de toepassing doorgeven. Zie verder paragraaf 5.7.

OPGAVE 2.5
Maak een variant van het element identification, met een gemengde inhoud die een opeenvolging is van een country-element, character data voor het id en een idType-element.

5 Character data en markup

In paragraaf 3 'De proloog van een XML-document' en paragraaf 4 'Elementen en attributen' hebben we XML-documenten bekeken vanuit hun opbouw van begin naar einde: een proloog gevolgd door een root-element, met eventuele subelementen en/of attributen. In deze paragraaf maken we een ander onderscheid in het XML-document, namelijk dat tussen twee soorten XML-tekst: character data en markup.

5.1 CHARACTER DATA

Character data
Content

De *character data* vormen de *content* (inhoud) van het XML-document. Alle character data hebben syntactisch de vorm van een string, die op twee plaatsen kan voorkomen:
– als content van een element, tussen starttag en eindtag
– binnen een attribuut als attribuutwaarde.

Character data dienen te worden geïnterpreteerd conform de in de XML-declaratie genoemde tekenset.

5.2 MARKUP

Markup

In de *markup* wordt de betekenis van de content weergegeven of wordt extra informatie opgenomen ten behoeve van de menselijke lezer of de verwerkende applicatie.

Markup omvat de volgende XML-tekst:
– de XML-declaratie
– commentaren
– processing instructions
– element tags (start tags, end tags en leeg-element tags)
– tekenverwijzingen
– entiteitverwijzingen
– delimiters van gemarkeerde CDATA-secties
– documenttypedeclaraties
– bepaalde whitespace.

De XML-declaratie, commentaren en processing instructions zijn al behandeld in paragraaf 3. Element tags zijn al in paragraaf 4 aan bod gekomen. De overige soorten markup zullen in de nu volgende paragrafen in het kort worden langsgelopen. Indien van toepassing (bij CDATA-secties en whitespace) worden de onderwerpen ook behandeld in relatie met content.

5.3 TEKENVERWIJZINGEN

XML is ontworpen voor wereldwijde communicatie. Om die reden moeten de tekensets van zo'n beetje 'alle' talen worden ondersteund. Daarnaast moet er ondersteuning zijn voor wetenschappelijke symbolen, valutatekens, enzovoort. In het algemeen moet het mogelijk zijn symbolen in een XML-document op te nemen die niet via een toetsenbord of ander invoerapparaat beschikbaar zijn.

Tekenverwijzing
Character reference

Zie paragraaf 7 voor meer informatie over tekensets

In een XML-document kunnen tekens uit een breed scala van tekens worden opgenomen via *tekenverwijzingen* (*character references*). Dit zijn verwijzingen naar tekens uit de Unicode-tekenset.
Een tekenverwijzing heeft één van de volgende vormen:

&#*decimaal volgnummer*;
&#x*hexadecimaal volgnummer*;

Enkele voorbeelden ziet u in tabel 2.1. Ga na dat het voorbeelddocument reisboeking1.xml op regel 3 een decimaal gecodeerde tekenverwijzing voor het euroteken bevat.

TABEL 2.1 Tekenverwijzingen in decimale en hexadecimale notatie (voorbeelden)

teken	*naam*	*tekenverwijzing (decimaal)*	*tekenverwijzing (hexadecimaal)*
©	copy	&#169;	&#x00A9;
¾	frac34	&#190;	&#x00BE;
ñ	ntilde	&#241;	&#x00F1;
π	pi	&#960;	&#x03C0;
€	euro	&#8364;	&#x20AC;
♥	hearts	&#9829;	&#x2665;

De tekennamen in tabel 2.1 zijn niet bruikbaar om naar de tekens te verwijzen, tenzij we op de één of andere manier verwijzen naar een *entiteitenbestand* waarin de tekennamen worden gekoppeld aan een volgnummer. Meer hierover in paragraaf 5.4 en in leereenheid 3.

OPGAVE 2.6
Zoek de tekenverwijzing voor het euroteken in reisboeking1.xml, op regel 3. Verwijder de processing instruction (omdat de stylesheet niet bestaat), sla het docment op en open het in een browser. Constateer dat de tekenverwijzing € als € wordt weergegeven.

5.4 ENTITEITEN EN ENTITEITVERWIJZINGEN

Entiteit
Entiteitverwijzing

Vaak is het handig om dataobjecten, zoals een speciaal karakter, te bewaren onder een herkenbare naam. Een XML-dataobject, met een naam en een inhoud, heet een *entiteit* (entity). Een *entiteitverwijzing* (entity reference) verwijst naar de inhoud van een entiteit.
Een entiteitverwijzing heeft de volgende vorm:

&*entiteitnaam*;

De W3C-aanbeveling kent vijf voorgedefinieerde entiteiten: quot, amp, apos, lt en gt. Zie tabel 2.2.

TABEL 2.2 Voorgedefinieerde entiteiten in XML 1.0

teken	*entiteit-verwijzing*	*Unicode*	*omschrijving*
“	&quot;	0022 (34)	dubbel aanhalingsteken (quote)
@	&amp;	0026 (38)	ampersand
‘	&apos;	0027 (39)	enkel aanhalingsteken (apostrophe)
<	&lt;	003C (60)	kleiner dan (less than)
>	&gt;	003E (61)	groter dan (greater than)

De tekens bij deze voorgedefinieerde entiteiten zijn precies de tekens die een bijzondere rol vervullen in XML-markup en daarom niet zomaar in content mogen worden opgenomen. De XML-parser verwacht bij het teken < bijvoorbeeld het begin van een tag. Daarom mag dat teken niet in de content voorkomen. Het wordt gecodeerd als een entiteit en de applicatie weet hoe deze moet worden weergegeven.

Andere entiteiten kunnen worden gedefinieerd in een DTD. Zo'n definitie koppelt een entiteitnaam aan een teken (via de decimale of hexidecimale code daarvan) of aan langere tekst (een rij van tekens). Er zijn publiek beschikbare DTD's met entiteitdefinities waarnaar vanuit een DTD verwezen kan worden. Zo is er voor (X)HTML een DTD waarin een hele reeks bijzondere tekens als entiteit wordt gedefinieerd. Bijvoorbeeld: @hearts; voor het teken ♥ (zie ook tabel 2.1). We komen daar in leereenheid 3 op terug.

OPGAVE 2.7

a Ga na dat het volgende XML-document een foutmelding oplevert en geef de verbeterde code.

```
<?xml version="1.0" encoding="UTF-8"?>
<ongelijkheid>
   3 < 5
</ongelijkheid>
```

b Waarom is voor de ampersand een entiteit nodig?

c Zoek de voorgedefinieerde entiteit voor een ampersand in reisboeking1.xml. Open het document in een browser en constateer dat O&U als O&U wordt weergegeven.

5.5 GEMARKEERDE CDATA-SECTIES

CDATA-sectie

Voor de volledigheid vermelden we hier dat het kan voorkomen dat een deel van de gegevensinhoud van een XML-document genegeerd moet worden door de XML-parser en rechtstreeks moet worden doorgegeven aan de applicatie. Dergelijke inhoud kan in een CDATA-sectie worden geplaatst. Een CDATA-sectie heeft als algemene vorm:

<![CDATA[… *inhoud* …]]>

Een CDATA-sectie zou bijvoorbeeld de binaire code van een foto of ander media-object kunnen bevatten. Al is dat niet de meest gebruikte manier om media-objecten met een XML-document mee te sturen (in leereenheid 3 zien we andere manieren), het is er wel één van.

CDATA-secties spelen geen rol in deze cursus.

5.6 DOCUMENTTYPEDECLARATIES

Zie paragraaf 1.4 van leereenheid 1

In leereenheid 1 hebt u kennisgemaakt met DTD's (documenttypedefinitie). Door middel van een DTD kan een XML-document aan specifieke grammaticale beperkingen worden gebonden. Ter controle kan een XML-parser het document valideren tegen de DTD. Bij het voorbeeld van leereenheid 1 was de DTD een extern bestand bestelling.dtd, ter validatie van het XML-document bestelling.xml. In opgave 1.6 hebt u gezien hoe vanuit bestelling.xml naar de DTD wordt verwezen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bestelling SYSTEM "bestelling.dtd">
<bestelling>
   ...
```

```
</bestelling>
```

Documenttype-declaratie

De regel <!DOCTYPE …> heet een *documenttypedeclaratie*.

Opmerking

Merk het onderscheid op tussen een documenttypedeclaratie (wordt nooit afgekort) en een documenttypedefinitie (wordt meestal afgekort tot DTD).

In leereenheid 3, waarin we uitgebreid ingaan op documenttypedeclaraties en DTD's, zullen we zien dat een DTD ook geheel of gedeeltelijk intern, binnen de documenttypedeclaratie, kan worden opgenomen.

5.7 WHITESPACE

Whitespace

Spaties, tab-tekens, regeleinden en carriage returns worden tezamen *whitespace* genoemd. In XML-documenten, computerprogramma's en veel andere teksten dienen whitespace-karakters vaak voor de opmaak, zodat de leesbaarheid van de tekst voor de menselijke lezer wordt vergroot.

Opmerking

Whitespace hoort tot de markup voor zover deze buiten het root-element voorkomt en geen deel uitmaakt van andere markup. Overige whitespace behoort (per definitie) tot de character data.

We zullen nu de rol van whitespace in XML-documenten nader bekijken.

OPGAVE 2.8

Productieregel

Het aanbevelingsdocument van W3C voor XML 1.0 bevat een groot aantal *productieregels.* Dat zijn grammaticale regels van een bepaalde vorm (herschrijfregel) die aangeven hoe een grammaticaal correct onderdeel van een XML-document hoort te zijn samengesteld. De productieregel voor whitespace heeft de volgende vorm:

```
S  ::=  expressie voor geldige whitespace-karakterstring
```

a Zoek de productieregel op in de XML-aanbeveling.
b Hoe zou u de regel formuleren in gewoon Nederlands?

Einde-regelcodering in Windows en Linux
Zoals wellicht bekend, wordt het einde van een regel in Windows anders gecodeerd dan in Linux, OS X en dergelijke systemen. In Windows is dat als carriage return + line feed, in Linux-like systemen door alleen een line feed. Een XML-parser kan – vanuit het oogpunt van platformonafhankelijkheid – hierin uiteraard maar één koers varen. De XML-aanbeveling vermeldt dat de parser het volgende doet: het teken #xD (carriage return) wordt verwijderd of vervangen door #xA (line feed).

Niet-significante whitespace

Niet-significante whitespace

Whitespace die in XML-documenten alleen voor opmaak dient, wordt *niet-significante whitespace* genoemd. Deze kan worden verwijderd zonder dat de betekenis van het XML-document verandert. Het woord 'betekenis' gebruiken we hier in operationele zin: de verwerking van het XML-document door de verwerkende applicatie.
Voor de menselijke lezer kan niet-significante whitespace heel belangrijk zijn, voor machinale verwerking maakt het niets uit.

In het voorbeelddocument reisboeking1.xml zijn alle inspringspaties en alle regeleinden niet-significant. Indien we alle inspringspaties weghalen krijgen we een XML-document dat slecht leesbaar is, maar met dezelfde betekenis:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- van: WegIsWeg; aan: ErOpUit; versie: 2010-06-18T14:15:22; status:
goedgekeurd -->
<?xml-stylesheet type="text/xsl" href="reisboeking_xhtml.xsl"?>
<klant klantnr="12345">
<naam>van Amstel</naam>
<voorletters>J</voorletters>
<geboortedatum>1956-07-17+01:00</geboortedatum>
<identification country="NL" idType="passport" id="N1234567"/>
<email>j.vanamstel@ou.nl</email>
</klant>
<reiscomponenten>
<vervoer subtotaal="380">
…
</reisboeking>
```

Laten we ook de regeleinden weg, dan blijft het XML-document grammaticaal correct en verandert er niets aan de betekenis:

```
<?xml version="1.0" encoding="UTF-8"?><!-- van: WegIsWeg; aan: ... </reizigers></boeking>
```

Het document bestaat nu uit één heel lange regel.

Het voorbeelddocument bevat geen tab-tekens. Vaak worden die wel gebruikt om in te springen. Parsers vervangen tabs soms automatisch door een vast aantal spaties, afhankelijk van hoe ze zijn ingesteld.

Significante whitespace

Significante whitespace
Whitespace die niet kan worden weggehaald zonder het XML-document inhoudelijk te veranderen of te verminken, heet *significante whitespace*. Voorbeelden zijn de spaties die attributen van elkaar scheiden. Ook spaties in tekenrijen (character strings) die voorkomen als attribuutwaarde of elementinhoud, zijn in het algemeen significant. Een voorbeeld is de spatie in de klantnaam ‘van Amstel’.

Mixed content

Significante whitespace in mixed content
In uitzonderlijke gevallen kan whitespace die ‘normaal gesproken’ niet-significant is (dat wil zeggen volgt op een eindtag) significant zijn. Dat is het geval wanneer een element als inhoud een afwisseling mag hebben van subelementen en character data. We spreken dan van *mixed content*. Die character data kunnen in principe alle karakters zijn, dus ook whitespace.
De vraag rijst nu: hoe kan een XML-parser weten of de betreffende whitespace dient voor opmaak of tot de elementinhoud behoort? Het antwoord is: dit moet blijken uit een bij het document geleverde grammatica (DTD of schema).

6 XML-namespaces

In leereenheid 1 hebben we al kennisgemaakt met namespaces. We gaan er nu wat dieper op in.

6.1 WAAROM NAMESPACES?

Het belangrijkste voordeel van XML is dat er door vele verschillende personen, in verschillende situaties en met verschillende bedoelingen informatie mee is te modelleren. Dat kan echter pas tot zijn recht komen, als de betekenis van de informatie éénduidig vastligt. Als in één document informatie moet worden bijeengebracht vanuit verschillende bronnen en waarvan de betekenis dus op verschillende plaatsen is vastgelegd, dan kunnen naamsconflicten optreden. Om die te vermijden worden namespaces ofwel 'naamruimten' gebruikt. Alleen namen die zijn gekoppeld aan een namespace bezitten mondiale uniciteit, in de zin dat ze een wereldwijd, uniek karakter hebben.

6.2 NAMESPACE-SYNTAXIS

URI en URL

Een namespace is syntactisch gezien 'gewoon' een string. Om aan zijn doel te beantwoorden, moet dat een string zijn die in de mondiale (wereldwijde) context van het internet is te herleiden tot één eigenaar. Als namespaces worden daarom URI's (*universal resource identifiers*) gebruikt. Een goedgekozen URI is te herleiden tot één 'eigenaar'. De meest gebruikte URI's zijn URL's (*universal resource locators*), dat zijn unieke namen voor webresources. De bekendste URL's zijn www-adressen.

Het systeem werkt zolang iedereen die XML-code schrijft, het eigenaarschap van anderen respecteert, dat wil zeggen: géén URI's van andere eigenaars misbruikt in eigen code. Dit betekent niet dat de namen in een XML-document maar tot één namespace mogen behoren. We zullen nog vele voorbeelden tegenkomen van XML-documenten met namen die tot verschillende namespaces behoren.

Gekwalificeerde naam

Elke naam die 'tot een bepaalde namespace behoort', moet daar impliciet of expliciet een koppeling mee hebben. Een naam met zo'n koppeling heet een *gekwalificeerde naam* (Engels: *qualified name*). In de volgende paragrafen zullen we twee soorten kwalificatie behandelen: impliciete en expliciete kwalificatie.

6.3 IMPLICIETE KWALIFICATIE

Impliciete kwalificatie

Een naam kan *impliciet gekwalificeerd* zijn. Dit is het geval wanneer de naam is gekoppeld aan een namespace door middel van het attribuut xmlns. Voorbeelden zagen we in de paragrafen 3.3 (XHTML), 3.4 (SVG) en 4.1 (WSDL) van leereenheid 1. De namespace heet in dit geval een *default namespace*.

Default namespace

Een ander voorbeeld vinden we in het document reisboeking2.xml, waarvan we hier het begin afdrukken:

reisboeking2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- van: WegIsWeg; aan: ErOpUit; versie: 20100115T14:15:22; status:
goedgekeurd -->
<reisboeking xmlns="http://www.wegisweg.com/boekingen"
             nr="A192837" totaal=" &#8364; 3425.00">
    <klant klantnr="12345">
        <naam>van Amstel</naam>
        <voorletters>J</voorletters>
```

```
        <geboortedatum>1956-07-17+01:00</geboortedatum>
        <identification country="NL" idType="passport" id="N1234567"/>
        <email>j.vanamstel@ou.nl</email>
        <vasteKlant/>
    </klant>
    <reiscomponenten>
        <vervoer subtotaal="380">
            <vlucht>
…
</reisboeking>
```

Reisboeking2.xml verschilt op één punt van reisboeking1.xml: het element reisboeking heeft een extra attribuut met de attribuutnaam xmlns en met attribuutwaarde de string 'http://www.wegisweg.com/boekingen'. Alle elementnamen binnen het reisboeking-element (inclusief de elementnaam reisboeking zelf) worden op deze wijze gekoppeld aan de namespace 'http://www.wegisweg.com/boekingen'. De naam klant bijvoorbeeld wordt gekwalificeerd door de naam van de namespace, hij 'behoort tot' deze namespace. De parser zal de elementnamen vervangen door de gekwalificeerde namen:

```
http://www.wegisweg.com/boekingen:reisboeking
http://www.wegisweg.com/boekingen:klant
http://www.wegisweg.com/boekingen:reiscomponenten
…
```

Default namespacedeclaraties kunnen in elk element opgenomen zijn. De naam van dit element en de namen van alle eventuele child-elementen zijn dan gekoppeld aan de betreffende namespace.

Opmerking
Attribuutnamen vallen buiten de werking van een default namespace. Alle attributnamen in reisboeking2.xml zijn dus ongekwalificeerde namen.

6.4 EXPLICIETE KWALIFICATIE

Expliciete kwalificatie

Een naam kan ook *expliciet gekwalificeerd* zijn. Dit is het geval wanneer de naam via een alias is gekoppeld aan een namespace. Een voorbeeld zagen we in paragraaf 3.5 van leereenheid 1 (Open Office). Een ander voorbeeld vinden we in het document reisboeking3.xml, waarvan we hier het begin afdrukken:

Reisboeking3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- van: WegIsWeg; aan: ErOpUit; versie: 20100115T14:15:22; status:
goedgekeurd -->
<bk:reisboeking xmlns:bk="http://www.wegisweg.com/boekingen"
                nr="A192837" totaal=" &#8364; 3425.00">
    <bk:klant klantnr="12345">
        <bk:naam>van Amstel</bk:naam>
        <bk:voorletters>J</bk:voorletters>
        <bk:geboortedatum>1956-07-17+01:00</bk:geboortedatum>
        <bk:identification country="NL" idType="passport" id="N1234567"/>
        <bk:email>j.vanamstel@ou.nl</bk:email>
        <bk:vasteKlant/>
    </bk:klant>
    <bk:reiscomponenten>
```

```
        <bk:vervoer subtotaal="380">
            <bk:vlucht>
…
```

Alias

Via de *alias* bk wordt nu elke elementnaam expliciet gekwalificeerd. Het resultaat, na parsen, is identiek aan dat van reisboeking2.xml. De parser vervangt de alias door de eigenlijke namespace. Merk hierbij op dat ook in dit geval de attribuutnamen ongekwalificeerd zijn.

6.5 ALIASSEN EN COLONIZED NAMES

Local part

Colonized name
Non-colonized name

Een expliciet gekwalificeerde naam bestaat uit drie delen: een namespace-alias, een dubbele punt (Engels: colon) en een zogenaamd *local part*. Zo'n samengestelde naam met een dubbele punt heet in het Engels een *colonized name*. Het local part is een *non-colonized name*. Overigens verplicht de introductie van de alias niet tot het expliciet kwalificeren van elke naam. Nog steeds kan een default-namespace-declaratie zorgen voor impliciete kwalificatie van non-colonized names. Ook kunnen namen buiten het bereik van een namespace vallen.

Een alias is maar een hulpmiddel om op een overzichtelijke manier een namespace te koppelen aan een namespace. De keuze van de alias is daarom in principe vrij; wel moet deze alias in het huidige element of één van zijn parent-elementen gedeclareerd zijn met behulp van het attribuut xmlns:*alias*. Wijziging van de alias 'bk' in 'xyz' heeft voor de verwerking geen enkel gevolg. Zelfs kunnen in één document elementen via verschillende aliassen aan dezelfde namespace worden gekoppeld. In leereenheid 4 zullen we hier nog een voorbeeld van zien.

Het is aan te raden voor aliassen korte namen te kiezen, die voor een menselijke lezer snel herkenbaar zijn en dus gemakkelijk te herleiden tot de bijbehorende namespace. We zullen nog allerlei namespaces tegenkomen met een vaste conventie voor de keuze van de alias.

Namespacedeclaraties voor explicite kwalificatie kunnen in elk element opgenomen zijn. De bijbehorende alias kan gebruikt worden in het element met de declaratie en in alle eventueel aanwezige kindelementen.

De W3C-definitie van qualified names

Dé bron voor XML-syntax en -semantiek in relatie met namespaces is de W3C namespaces recommendation http://www.w2.org/TR/REC-xml-names/. In paragraaf 3 'Qualified names' lezen we:

W3C-citaat Qualified Name

```
[6]  QName ::=  (Prefix ':')? LocalPart
[7]  Prefix ::=  NCName
[8]  LocalPart ::=  NCName
```

Toelichting

Deze regels definiëren de syntax van qualified names (QName's) in een XML-document. Regel [6] zegt dat een qualified name bestaat uit een verplicht local part (LocalPart) dat, optioneel, wordt voorafgegaan door een alias (Prefix) met dubbele punt. De regels [7] en [8] zeggen dat zowel de alias als het local part de vorm hebben van een non-colonized name (NCName, zonder dubbele punt).

Geen namespace en lege namespace

Aan paragraaf 5.2 van de W3C-recommendation ontlenen we de volgende zinnen over namen die niet onder het bereik van een namespacedeclaratie vallen en over

namen die onder het bereik vallen van de lege namespace (aangegeven door een lege string, ""):

W3C-citaten

"If the URI reference in a default namespace declaration is empty, then unprefixed elements in the scope of the declaration are not considered to be in any namespace."
"The default namespace can be set to the empty string. This has the same effect, within the scope of the declaration, of there being no default namespace."

Het eerste citaat houdt in dat een naam die onder het bereik van een lege namespacedeclaratie valt, beschouwd kan worden als een naam die niet onder het bereik van enige namespace valt. Het tweede citaat geeft aan dat ook het omgekeerde geldt.
NB: door een lege default-namespacedeclaratie (met het attribuut xmlns="", waarin "" de lege string is), wordt een eventuele namespacedeclaratie op een hoger niveau overruled. Non-colonized names die onder het bereik van die lege namespacedeclaratie vallen, kunnen we dan beschouwen als namen die niet tot een namespace behoren.

Verwarrend taalgebruik

Wanneer we nu zeggen dat een naam ongekwalificeerd is (unqualified), betekent dat hetzelfde als wanneer we zeggen dat de naam gekwalificeerd is door de lege namespace! En wanneer we zeggen dat een naam niet tot een namespace behoort, betekent dat hetzelfde als wanneer we zeggen dat de naam tot de lege namespace behoort.

De enige en juiste weg uit dit taallabyrint lijkt ons: ervan uitgaan dat er helemaal geen lege namespace bestaat! De namespacedeclaratie xmlns="" is in die visie dan slechts een lege namespacedeclaratie, maar niet een declaratie van de lege namespace! De lege declaratie heeft als effect dat de non-colonized names (NCName's) in het bereik ervan niet tot enige namespace behoren (ongekwalificeerd zijn).

In de voorbeelden van deze leereenheid spelen dit soort subtiliteiten gelukkig geen rol. In leereenheid 4 echter, waar we zullen zien hoe in een XML Schema-grammatica wordt vastgelegd welke namen tot een gegeven namespace (de 'target namespace' van het schema) behoren, moeten we deze kennis paraat hebben om alle details goed te begrijpen.

6.6 HET BEREIK VAN NAMESPACES

Wanneer in een element een default namespace is gedefinieerd (met het attribuut xmlns), dan geldt deze voor de elementnaam zelf en voor de namen van alle child elements, behoudens overruling door een andere namespace (of overruling door 'géén namespace') op een lager niveau van nesting. Wanneer een elementnaam echter via een alias aan een namespace is toegewezen (met het attribuut xmlns:*alias*), dan geldt deze toewijzing niet voor de child elements. Namen van child elements die non-colonized zijn, zijn dan dus ongekwalificeerd. Anders gezegd: ze behoren niet tot een namespace.

Begripsverwarring kan optreden rond de begrippen *gekwalificeerd* (qualified) en *colonized*. Het begrip 'gekwalificeerd' houdt in: behorend tot een namespace, hetzij impliciet (via een default namespace), hetzij expliciet (via een alias). Het begrip 'colonized' is een simpel syntactisch

begrip: een colonized name is een samengestelde naam (in het brondocument, vóór parsering) met een dubbele punt.

Hier volgt nog een voorbeeld ter verduidelijking:

```
<elementA>
   <elementB xmlns="http://www.ou.nl/namespace1">
      <elementC xmlns:al="http://www.ou.nl/namespace2">
         <al:elementD>
            <elementE/>
         </al:elementD>
      </elementC>
   </elementB>
<elementA>
```

– **vet**: elementnaam behoort niet tot een namespace (unqualified name); dit geldt ook voor de attribuutnaam xmlns
– onderstreept: elementnamen behoren tot namespace http://www.ou.nl/namespace1
– onderstreept: elementnamen behoren tot namespace http://www.ou.nl/namespace2
– namespaces worden gedeclareerd met behulp van een verzameling van gereserveerde attribuutnamen. Zo'n attribuutnaam is xmlns of xmlns: gevolgd door een NCName (non-colonized name).

Verdere opmerkingen over namespaces:
– De namespace van een parent element geldt alleen voor de child elementen indien het om een default namespace gaat. Een expliciet gekwalificeerde namespace wordt niet geërfd door child elementen.
– De default namespace, gedefinieerd met behulp van het xmlns-attribuut geldt voor alle non-colonized elementnamen in het betreffende element en de child-elementen daarvan.
– Met behulp van xmlns="" (lege string als attribuutwaarde) wordt een *lege namespace* gedefinieerd. Hiermee is de default namespace te overrulen. Overigens raden we het af om met lege namespaces te werken.
– Een naam die via een alias is gekoppeld aan een lege namespace, geldt als niet-gekwalificeerd. Dus: 'niet behorend tot namespace' is synoniem met 'behorend tot de lege namespace'.
Opmerking: oXygen laat dit niet toe: "Prefixed namespace binding may not be empty".
– De eventuele namespace waartoe een naam behoort, wordt nooit bepaald door informatie buiten het document. In het bijzonder heeft een XML-grammatica (zoals een DTD of schema) geen invloed hierop.
– Namen van attributen vallen niet onder de default namespace. Eventuele kwalificatie moet dus altijd expliciet gebeuren.
– Een attribuutnaam kan alleen op expliciete wijze worden gekoppeld aan een namespace. De waarde van een attribuut valt nooit onder het bereik van een namespace.

6.7 GEKWALIFICEERDE ATTRIBUUTNAMEN

Indien ook attribuutnamen gekwalificeerd moeten worden, zal dat expliciet moeten. We hebben immers gezien dat attribuutnamen niet onder de werking van een default namespace vallen. In de volgende variant van het reisboeking-document, reisboeking4.xml, zijn ook de

(gemarkeerde) attribuutnamen gekwalificeerd en wel met dezelfde namespace als het element waartoe ze behoren.

We geven het begin van reisboeking4.xml:

Reisboeking4.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- van: WegIsWeg; aan: ErOpUit; versie: 20100115T14:15:22; status:
goedgekeurd -->
<bk:reisboeking xmlns:bk="http://www.wegisweg.com/boekingen"
                bk:nr="A192837" bk:totaal=" &#8364; 3425.00">
        <bk:klant bk:klantnr="12345">
        <bk:naam>van Amstel</bk:naam>
        <bk:voorletters>J</bk:voorletters>
        <bk:geboortedatum>1956-07-17+01:00</bk:geboortedatum>
        <bk:identification bk:country="NL" bk:idType="passport"
                           bk:id="N1234567"/>
        <bk:email>j.vanamstel@ou.nl</bk:email>
        <bk:vasteKlant/>
    </bk:klant>
    <bk:reiscomponenten>
        <bk:vervoer bk:subtotaal="380">
            <bk:vlucht>
…
```

Attribuutnamen worden meestal niet gekwalificeerd, omdat ze als lokale namen gezien worden, in de context van vaak wel gekwalificeerde elementnamen.

OPGAVE 2.9

Is het mogelijk om een default namespace te gebruiken voor de elementnamen en tevens de attribuutnamen te kwalificeren met dezelfde namespace?

OPGAVE 2.10

Ga uit van het volgende XML-document order.xml uit het project xml02_beginselen:

```
<order>
    <klant>
        <naam>Jan Pietersen</naam>
        <adres>Kerkstraat 21, Kapelle</adres>
    </klant>
    <bestelling>
        <naam>printer abc</naam>
        <aantal>20</aantal>
    </bestelling>
    <leverancier>
        <naam>PC shop</naam>
        <adres>Edisonstaat 2, Rotterdam</adres>
    </leverancier>
    <transporteur>
        <naam>Transportbedrijf Vogel</naam>
        <adres>Havenplein 198, Dordrecht</adres>
    </transporteur>
</order>
```

Merk op dat er vier verschillende 'namen' zijn: een klantnaam, een bestellingnaam, een leveranciernaam en een transporteurnaam. Evenzo zijn er drie verschillende 'adressen'.
Gegeven zijn de volgende namespaces, met daarbij vier verschillende 'eigenaren':

namespace	*'eigenaar'*
http://www.ebusiness.com	bedrijf dat order.xml creëert
http://www.customers.com	klant
http://www.suppliers.com	leverancier
http://www.logistics.com	transporteur

U zou zich kunnen voorstellen dat de stukjes code die met de genoemde 'eigenaren' corresponderen, afkomstig zijn van andere XML-bronnen en zijn geïmporteerd in order.xml. De namespaces dienen dan om potentiële naamsconflicten uit te sluiten.

Na deze inleiding volgt nu de eigenlijke opgave:
Neem referenties naar de vier namespaces op in order.xml, zodanig dat:
– alle klant-gerelateerde elementen tot de namespace http://www.customers.com behoren
– alle leverancier-gerelateerde elementen tot de namespace http://www.suppliers.com behoren
– alle transporteur-gerelateerde elementen tot de namespace http://www.logistics.com behoren
– alle overige elementen tot de namespace http://www.ebusiness.com behoren.

Opmerking:
Er ontstaan zodoende vier verschillende elementnamen 'naam' en drie verschillende elementnamen 'adres'.

Werk deze opgave uit op drie verschillende manieren:
a met uitsluitend default namespace declaraties
b met uitsluitend expliciete namespace declaraties
c met expliciete namespacedeclaraties voor de elementen klant, leverancier en transporteur (en hun child-elementen) en een default voor de overige.

OPGAVE 2.11
Op de cursussite vindt u een tutorial over namespaces. Hierin worden, in een mooi opgebouwde reeks voorbeelden, namespaces instructief gevisualiseerd met verschillende kleuren. Indien u een aantal zaken rondom namespaces nog eens de revue wilt laten passeren, is deze tutorial heel geschikt.

6.8 EEN VOORBEELD MET TWEE NAMESPACES

We eindigen deze paragraaf met een voorbeeld van een document met meer dan één namespace. Het XML-document reisboeking5.xml is zo'n document. Het is grotendeels gebaseerd op reisboeking2.xml, met de default namespace http://www.wegisweg.com/boekingen. Voor één elementnaam is echter een aparte namespace gedefinieerd: http://interIdService.com, via een namespacedeclaratie in het element reisboeking. Alleen het element identification wordt aan deze namespace

gekoppeld, via de alias id. De achtergrond hiervan is dat WegIsWeg de identificatiegegevens deelt met een internationale dienst, voor controle op het bestaan ervan. Het gaat hier dus om gegevensuitwisseling op een niveau dat het bedrijf overstijgt.

We geven het begin en het einde van reisboeking5.xml:

Reisboeking5.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- van: WegIsWeg; aan: ErOpUit; versie: 20100115T14:15:22; status:
goedgekeurd -->
<reisboeking xmlns="http://www.wegisweg.com/boekingen"
             xmlns:id="http://interIdService.com"
             nr="A192837" totaal="&#8364; 3425.00">
   <klant klantnr="12345">
      <naam>van Amstel</naam>
      <voorletters>J</voorletters>
      <geboortedatum>1956-07-17+01:00</geboortedatum>
      <id:identification country="NL" idType="passport" id="N1234567"/>
      <email>j.vanamstel@ou.nl</email>
      <vasteKlant/>
   </klant>
   <reiscomponenten>
…
   </reiscomponenten>
   <reizigers aantalPersonen="2">
      <persoon>
         <naam>van Amstel</naam>
         <voorletters>J</voorletters>
         <id:identification country="NL" idType="passport" id="N1234567"/>
      </persoon>
      <persoon>
         <naam>Maas</naam>
         <voorletters>P</voorletters>
         <id:identification country="NL" type="passport" id="N7654321"/>
      </persoon>
   </reizigers>
</reisboeking>
```

Toelichting
1 Alle elementnamen worden gekwalificeerd door de default namespace, behalve identification.
2 De elementnamen, na parsen, zijn:

```
http://www.wegisweg.com/boekingen:reisboeking
http://www.wegisweg.com/boekingen:klant
```

… enzovoort, maar:

```
http://interIdService.com:identification
```

7 Internationale taalondersteuning in XML

Voor wereldwijde communicatie is het van evident belang dat de tekensets van alle belangrijke talen op een uniforme, gestandaardiseeerde wijze worden ondersteund. Voor dat doel zijn verschillende tekensets ontworpen. Belangrijke tekensets zijn UCS, ISO 10646 enUnicode. Er verschillende manieren om deze tekensets – al dan niet efficiënt – binair te coderen. De officiële tekenset voor XML is ISO 10646, die vrijwel

equivalent is met Unicode. Belangrijke (want efficiënte) binaire coderingen daarvan zijn UTF-8 en UTF-16.

7.1 UCS EN ISO 10646

UCS (universal tekenset) is in zijn standaardvorm een 4-byte code voor de lettertekens van vrijwel alle denkbare talen, uiteenlopend van het Latijnse, het Griekse en het cyrillische alfabet tot de duizenden Chinese karakters. Ook oude talen, zoals Sanskriet, hebben hun eigen code-interval. Verder zijn er nog bibliotheken vol bijzondere symbolen in UCS gecodeerd. UCS is gestandaardiseerd door de ISO als ISO 10646.

7.2 UNICODE

Unicode stamt van een 'concurrerend' standaardisatieproject, opgezet door een consortium van voornamelijk Amerikaanse producenten van meertalige software. Net als UCS is het een 4-byte codering. Gelukkig hebben beide groepen elkaar tijdig gevonden. Hoewel ze nog steeds hun eigen (voortdurend uitbreidende) codetabellen ontwikkelen, is afgesproken deze door goed overleg compatibel te houden.

7.3 UTF-8

Uit het voorgaande blijkt dat ISO 10646 en Unicode voor de praktijk inwisselbaar zijn. Ze vormen de basis van alle XML-gerelateerde communicatie. Er is echter een probleem: ASCII-tekens (gecodeerd in één byte, beginnend met een 0-bit) maken het leeuwendeel uit van alle uitgewisselde boodschappen (binaire gegevens buiten beschouwing gelaten). Dan is codering in Unicode, met zijn 4 bytes per karakter, wel erg inefficiënt. Om die reden zijn er verschillende manieren bedacht om Unicode te hercoderen, zodanig dat ASCII-tekens niet meer dan één byte innemen en ook andere veelgebruikte karakters niet de volle omvang van een Unicode-karakter hebben. De belangrijkste van deze hercoderingen is UTF-8 (UTF = UCS transformation format). In UTF-8 nemen ASCII-tekens één byte (8 bits) in beslag, vandaar de 8. Er is een beperkt aantal gereserveerde 'escape'-codes om aan te geven dat het betreffende teken meer dan één byte in beslag neemt. Zo kunnen toch alle Unicode-karakters in UTF-8 worden gecodeerd. Dat gebeurt in maximaal 6 bytes, maar zoals gezegd: verreweg de meeste tekens zijn ASCII, en kosten dus maar één byte.

Het volgende tabelletje geeft aan hoe een 4-byte Unicode-karakter wordt gecodeerd als een minimaal 1- tot maximaal 6-byte UTF-8-karakter. De Unicode-intervallen zijn hexadecimaal weergegeven, de UTF-8-weergave is in bits, het minst significante bit rechts. Het interval U-00000000 t/m U-0000007F correspondeert met de 127 ASCII-tekens.

Unicode-interval	*codering in UTF-8*
U-00000000 - U-0000007F	0xxxxxxx
U-00000080 - U-000007FF	110xxxxx 10xxxxxx
U-00000800 - U-0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
U-00010000 - U-001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U-00200000 - U-03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Merk op dat in UTF-8 het aantal begin-1's in de eerste byte gelijk is aan het totaal aantal bytes, uitgezonderd het ASCII-interval. De eerste byte bevat dus de informatie tot hoever de byte-rij (waar het teken deel van uitmaakt) moet worden gelezen voor dit ene teken.

Voorbeelden

Het Unicode-karakter U-000000A9 = 1010 1001 (copyrightteken: ©) wordt in UTF-8 gecodeerd als 11000010 10101001 (binair) = C2 A9 (hexadecimaal).
Het Unicode-karakter U-00002260 = 0010 0010 0110 0000 (ongelijkheidsteken: ≠) wordt in UTF-8 gecodeerd als 11100010 10001001 10100000 (binair) = E2 89 A0 (hexadecimaal).

Het spreekt vanzelf dat voor XML-documenten vol Chinese karakters, om maar een voorbeeld te noemen, UTF-8 niet optimaal is. Unicode zelf is daarvoor een betere keus.

Unicode en UTF-8 zult u in deze cursus nog regelmatig tegenkomen. Voor meer informatie: zie http://www.unicode.org/ respectievelijk http://www.cl.cam.ac.uk/~mgk25/unicode.html#utf-8.
Raadpleeg http://www.eki.ee/letter/ om een indruk te krijgen van de overweldigende rijkdom aan tekens in de talen die u maar wilt, met gedetailleerde informatie per teken.

We sluiten deze paragraaf af met een stukje over de ontwikkeling van Unicode en UTF, speciaal met betrekking tot Java. Dit mede ter verklaring van het fenomeen dat in Java-cursussen en Java-documentatie steeds wordt gesproken over 16-bits (in plaats van 32-bits) Unicode-karakters.

Unicode, UTF en Java
Unicode is begonnen als een 2-byte (16-bit) codering, zonder voorzieningen voor karakters buiten UCS-2. Dit is overgenomen door de ontwikkelaars van Java. In Java beslaat het primitieve type char dan ook 2 byte, goed voor de eerste 64k karakters uit de Unicode-tabel. Ten behoeve van export naar andere omgevingen is voor implementatie van de JVM vereist dat conversie van String van en naar ByteArray mogelijk is volgens minimaal Latin-1 (de eerste 256 characters gecodeerd in 1 byte), UTF-16 en UTF-8 coderingen.
Bron: http://www.cl.cam.ac.uk/~mgk25/unicode.html.

7.4 EÉN TEKEN, VELE CODERINGEN

Ter illustratie van de vele mogelijke coderingen laten we nu één teken, dat voor de Japanse yen:

zien in een hele rij coderingen. Zie tabel 2.3.

TABEL 2.3 Yen-teken in vele coderingen

codering	*code*
HTML Entity (decimal)	&#165;
HTML Entity (hex)	&#xa5;
HTML Entity (named)	&yen;
UTF-8 (hex)	0xC2 0xA5 (c2a5)
UTF-8 (binary)	11000010:10100101
UTF-16 (hex)	0x00A5 (00a5)
UTF-16 (decimal)	165
UTF-32 (hex)	0x000000A5 (00a5)
UTF-32 (decimal)	165
C/C++/Java source code	"\u00A5"
Python source code	u"\u00A5"

ZELFTOETS

1 a Welke onderdelen moeten altijd in een XML-document voorkomen? Geef een zo klein mogelijk XML-document.
b Geef minstens vijf onderdelen die in een XML-document voor kúnnen komen, maar die niet verplicht zijn.
c Welke onderdelen moeten precies één keer voorkomen, welke nul of één keer, en welke onderdelen mogen 0, 1 of meer keer voorkomen?

2 Het volgende document voldoet niet aan de vereisten van welgevormdheid:

```
<?Xml version="1.0"?>
<Fiets type="ATB">
   <model>grasshopper</model>
   <kleur>bush groen</kleur>
   <remmen>shimano 3ZX</remmen>
   <versnelling>shimano XYZ
   <aantal versnellingen>21
   </versnelling></aantal versnellingen>
   <prijs eenheid=euro>549,00</prijs>
   <op voorraad/>
</fiets>
<fabrikant>
<fabrikantnaam>fietsenfabriek 'De tweewieler'</fabrikantnaam>
<fabrikantadres>Kettingplein 56, Wielerdorp</fabrikantadres>
</fabrikant>
```

a Om welke redenen is dit document niet welgevormd?
b Bewerk het document zodat het wél welgevormd is. Maak desgewenst gebruik van oXygen. U vindt het bestand in de projectmap onder de naam fiets.xml.

3 Gegeven is een element prijs:

```
<prijs valuta="USD">14.99</prijs>
```

Werk het element prijs om, zodanig dat het attribuut valuta is gehermodelleerd tot een subelement. Geef twee verschillende oplossingen. Welke van de drie varianten vindt u het beste?

4 Een onderwijsorganisatie (eigenaar van onder meer de portal http://www.edu.org) ontwikkelt een e-learningapplicatie waarmee cursussen worden beheerd via XML-documenten. Een cursus bestaat uit onderwijsmaterialen en studentgegevens.

Neem aan dat specificaties voor de onderwijsmaterialen zijn te vinden op http://www.edu.org/onderwijsmateriaal. In de onderwijsmaterialen komen toetsen voor met vragen in een bepaald genre. De vragen zijn van het type 'meerkeuze' of 'kort antwoord'. Iedere toets kan een bepaald niveau hebben. Een meerkeuzevraag is opgebouwd uit een vraagtekst en antwoordalternatieven. De antwoordalternatieven hebben een volgnummer en uiteraard is aangegeven welk antwoord juist is. Een kort-antwoordvraag bestaat uit een vraagtekst en het correcte antwoord.

Studentgegevens zijn gespecificeerd op http://www.uni.org/student (de portal http://www.uni.org wordt beheerd door een overkoepelende organisatie van universiteiten). Van studenten worden naam, studentnummer en studieresultaat weergegeven. Het studieresultaat is het aantal goed beantwoorde vragen.

Stel een XML-document op voor een cursus met twee meerkeuzevragen en een kort-antwoordvraag en met de studentgegevens. Gebruik uw eigen creativiteit voor de inhoud van de vragen en de antwoorden.

5 Zijn de volgende beweringen waar of onwaar? Motiveer uw antwoorden.
a Is elke qualified name tevens een colonized name? En omgekeerd?
b Ongekwalificeerde namen behoren niet tot een namespace.
c Namen die niet tot een namespace behoren, zijn ongekwalificeerd.
d Namen met alias en dubbele punt behoren tot een namespace.
e Namen die tot een namespace behoren, zijn herkenbaar aan een alias met dubbele punt.

TERUGKOPPELING

1 Uitwerking van de opgaven

2.1 Geen uitwerking.

2.2 De volgende namen zijn niet toegestaan:
b studieresultaat:tentamencijfer – in deze naam is een dubbele punt opgenomen
c Cursus Inschrijving – in deze naam is een spatie opgenomen
d 4de_klas_mentor – deze naam begint met een cijfer
e XML_cursusdocumenten – deze naam begint met XML.

2.3 a Als het laatste symbool in een tag een forward slash is, dan is het element een 'leeg element' (empty element). Er is dan geen tagpaar (openings- en sluitingstag) met daartussen de content, maar slechts één enkele tag. Deze kan overigens wel een of meer attributen bevatten.
De tag <abc /> is dus de enige tag van het lege element abc.
Als het eerste symbool in een tag een forward slash is, dan is die tag de sluittag van een element. De tag </abc > is dus de sluittag van het element abc.

b De tag <abc/> mag overal in een XML-document voorkomen en legt verder geen eisen op aan het document.
Vóór de tag </abc> moet ergens in het XML-document de tag <abc> voorkomen, wil het document welgevormd zijn. Bovendien moeten alle elementen die geopend zijn na <abc>, worden gesloten vóór de corresponderende </abc>.

c De tags <abc/> als <abc> mogen, voor wat betreft de welgevormdheid, tegelijkertijd in een XML-document voorkomen. Verifieer dit desgewenst met oXygen.

2.4 a Het element identification bevat in attribuutvorm informatie over een identificatiedocument: het land dat het document heeft afgegeven, het type en een uniek nummer. Als we geen attributen willen gebruiken, ligt het voor de hand om de attributen te transformeren tot subelementen van identification:

```
<persoon>
   <naam>van Amstel</naam>
   <voorletters>J</voorletters>
   <identification>
      <country>NL</country>
      <idType>passport</idType>
      <id>N1234567</id>
   </identification>
</persoon>
```

Het is moeilijk om aan te geven of dit een verbetering is. De oorspronkelijke vorm is korter en daarom misschien beter. Maar dit is geen sterk argument. De vraag is of de verwerkende applicatie met de ene vorm beter overweg kan dan met de andere. Hier is geen eenduidig antwoord op mogelijk, omdat – zoals we nog zullen zien – zowel voor elementinhoud als voor attribuutwaarden goede verwerkingsmogelijkheden zijn. Houd bij dit soort vragen ook altijd in

het achterhoofd dat XML geen opslagformaat is maar een uitwisselingsformaat. Dit relativeert de vraag naar de beste wijze van modelleren.

b Wanneer we de attributen land en idType handhaven en het id-nummer elementinhoud maken van identification, krijgen we:

```
<persoon>
  <naam>van Amstel</naam>
  <voorletters>J</voorletters>
  <identification country="NL" idType="passport">N1234567</identification>
</persoon>
```

2.5 De gevraagde code:

```
<persoon>
   <naam>van Amstel</naam>
   <voorletters>J</voorletters>
   <identification>
      <country>NL</country>N1234567<idType>passport</idType>
   </identification>
</persoon>
```

2.6 Geen uitwerking.

2.7 a Verbeterde code: vervang 3 < 5 door 3 < 5.
b Voor de ampersand is een entiteit nodig omdat de parser het teken @ voor het begin van een entiteit aanziet.
c Geen uitwerking.

2.8 a Geen uitwerking.
b Een whitespace is een opeenvolging van een willekeurig aantal (*) karakters die elk een spatiekarakter zijn of (|) een tab-karakter of een carriage return-karakter of een line feed-karakter.
Iets minder letterlijk: een whitespace is een willekeurige opeenvolging van spaties, tabs, carriage returns en/of line feeds.

2.9 Ja, dat kan. Ter illustratie geven we het begin van een reisboeking-document waarin de namespace http://www.wegisweg.com/boekingen default namespace is, maar tevens is gekoppeld aan de alias bk. De elementnamen kunnen nu zonder alias (non-colonized) worden gebruikt, maar de attribuutnamen moeten worden gekwalificeerd.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- van: WegIsWeg; aan: ErOpUit; versie: 20100115T14:15:22; status:
goedgekeurd -->
<reisboeking xmlns="http://www.wegisweg.com/boekingen"
             xmlns:bk="http://www.wegisweg.com/boekingen"
             bk:nr="A192837" bk:totaal=" &#8364; 3425.00">
        <klant bk:klantnr="12345">
        <naam>van Amstel</naam>
…
```

2.10 a Oplossing met uitsluitend default namespaces:

```
<order xmlns="http://www.ebusiness.com/order">
    <klant xmlns="http://www.customers.com/klant">
```

```
        <naam>Jan Pietersen</naam>
        <adres>Kerkstraat 21, Kapelle</adres>
    </klant>
    <bestelling>
        <naam>printer abc</naam>
        <aantal>20</aantal>
    </bestelling>
    <leverancier xmlns="http://www.suppliers.com/leverancier">
        <naam>PC shop</naam>
        <adres>Edisonstaat 2, Rotterdam</adres>
    </leverancier>
    <transporteur xmlns="http://www.logistics.com/transporteur">
        <naam>Transportbedrijf Vogel</naam>
        <adres>Havenplein 198, Dordrecht</adres>
    </transporteur>
</order>
```

In deze oplossing wordt de default namespacedeclaratie in regel 1 overruled op een lager niveau. Zodoende behoren de elementnamen order, bestelling, (bestelling)naam en (bestelling)aantal tot namespace http://www.ebusiness.com en bijvoorbeeld klant, (klant)naam en (klant)adres tot namespace http://www.customers.com.

b Indien alleen expliciete namespacedeclaraties worden gebruikt, worden bij het root-element de referenties naar de diverse verschillende definities opgenomen en aan een alias gekoppeld. Vervolgens wordt in iedere tag verwezen naar één van de definities. De uitwerking wordt dan:

```
<or:order
    xmlns:or="http://www.ebusiness.com/order"
    xmlns:kl="http://www.customers.com/klant"
    xmlns:le="http://www.suppliers.com/leverancier"
    xmlns:tr="http://www.logistics.com/transporteur">
    <kl:klant>
        <kl:naam>Jan Pietersen</kl:naam>
        <kl:adres>Kerkstraat 21, Kapelle</kl:adres>
    </kl:klant>
    <or:bestelling>
        <or:naam>printer abc</or:naam>
        <or:aantal>20</or:aantal>
    </or:bestelling>
    <le:leverancier>
        <le:naam>PC shop</le:naam>
        <le:adres>Edisonstaat 2, Rotterdam</le:adres>
    </le:leverancier>
    <tr:transporteur>
        <tr:naam>Transportbedrijf Vogel</tr:naam>
        <tr:adres>Havenplein 198, Dordrecht</tr:adres>
    </tr:transporteur>
</or:order>
```

Opmerking: het is niet strikt noodzakelijk alle namespaces in de root-tag te definiëren. Het kan bijvoorbeeld ook als volgt:

```
<or:order xmlns:or="http://www.ebusiness.com/order"
    <kl:klant xmlns:kl="http://www.customers.com/klant">
        <kl:naam>Jan Pietersen</kl:naam>
            ...
```

Alles in de root is echter wel het meest overzichtelijk.

c De gemengde methode leidt tot de volgende oplossing:

```
<order
    xmlns="http://www.ebusiness.com/order"
    xmlns:kl="http://www.customers.com/klant"
    xmlns:le="http://www.suppliers.com/leverancier"
    xmlns:tr="http://www.logistics.com/transporteur">
    <kl:klant>
        <kl:naam>Jan Pietersen</kl:naam>
        <kl:adres>Kerkstraat 21, Kapelle</kl:adres>
    </kl:klant>
    <bestelling>
        <naam>printer abc</naam>
        <aantal>20</aantal>
    </bestelling>
    <le:leverancier>
        <le:naam>PC shop</le:naam>
        <le:adres>Edisonstaat 2, Rotterdam</le:adres>
    </le:leverancier>
    <tr:transporteur>
        <tr:naam>Transportbedrijf Vogel</tr:naam>
        <tr:adres>Havenplein 198, Dordrecht</tr:adres>
    </tr:transporteur>
</order>
```

2.11 Geen uitwerking.

2 Uitwerking van de zelftoets

1 a De XML-declaratie `<?xml version="1.0"?>` is géén verplicht onderdeel van een welgevormd XML-document. Het enige dat verplicht is, is een root-element. Een kleinst mogelijk XML-document bevat dus alleen een root-element, zonder content. Bijvoorbeeld:

<a/>

b Niet-verplichte onderdelen van een XML-document:
- een XML-declaratie
- een documenttypedeclaratie
- subelementen van het root-element, daar ook weer subelementen van, enzovoort
- attributen van elementen
- gegevens bij elementen
- commentaar
- processing instructions
- verwijzingen naar namespaces bij elementen.

c Het root-element moet precies één keer voorkomen. Een XML-declaratie moet nul of één keer voorkomen. De overige onderdelen mogen alle nul of meer keren voorkomen.

2 a Overtredingen tegen de welgevormdheidsregels:

- Direct al in de XML-declaratie is een eerste fout gemaakt. Omdat XML hoofdlettergevoelig is, moet xml in de declaratie met kleine letters worden geschreven, dus: `<?xml version="1.0"?>`.
- Nog een fout tegen de hoofdlettergevoeligheid: in de openingstag van het root-element staat 'Fiets', in de sluittag staat 'fiets'.
- In het element fiets gaat het met de tags versnelling en aantal versnellingen mis. Die zijn niet goed genest.
- Het element prijs heeft een attribuut eenheid. De waarde die daaraan wordt meegegeven, moet tussen aanhalingstekens worden gezet.
- De namen van de elementen 'aantal versnellingen' en 'op voorraad' bevatten een spatie, dat is niet toegestaan.
- De openings- en sluittag van het element fiets bevatten niet op dezelfde wijze hoofdletters.
- Na het element fiets, dat het root-element lijkt te zijn, komt er nog een element fabrikant. Dat is niet toegestaan. Het element kan eventueel worden ondergebracht in het element fiets.

b Worden genoemde fouten gecorrigeerd, dan leidt dat tot het volgende welgevormde XML-document.

```
<?xml version="1.0"?>
<!DOCTYPE  fiets SYSTEM "fiets.dtd">
<fiets type="ATB">
    <model>grasshopper</model>
    <kleur>bush groen</kleur>
    <remmen>shimano 3ZX</remmen>
    <versnelling>shimano XYZ</versnelling>
    <aantal_versnellingen>21</aantal_versnellingen>
    <prijs eenheid="euro">549,00</prijs>
    <op_voorraad/>
    <fabrikant>
        <fabrikantnaam>fietsenfabriek 'De tweewieler'</fabrikantnaam>
        <fabrikantadres>Kettingplein 56, Wierlerdorp</fabrikantadres>
    </fabrikant>
</fiets>
```

3 We zetten de gegeven structuur en twee oplossingen op een rij.

Gegeven:

```
<prijs valuta="USD">14.99</prijs>
```

Deze structuur zegt: 'prijs is 14.99, waarbij als een soort meta-informatie over de prijs wordt toegevoegd dat deze als valuta USD heeft.

Oplossing 1:

```
<prijs>
   <valuta>USD</ valuta>
   <value>14.99</value>
</prijs>
```

Deze structuur zegt: prijs heeft een valuta (USD) en een prijs (14.99). Hier worden valuta en waarde als afzonderlijke, benoemde eigenschappen van prijs gezien.

Oplossing 2:

```
<prijs>
   <valuta>USD</valuta>
   14.99
</prijs>
```

Deze oplossing lijkt op de gegeven structuur, maar gebruikt gemengde inhoud. Het gebruik van een attribuut vinden we hier beter dan gemengde inhoud.

Het is niet mogelijk absolute uitspraken te doen.

4 De opgave bevat een hint voor het gebruik van namespaces, door het noemen van aan onderwijsmaterialen respectievelijk studentgegevens gerelateerde URI's. Door deze URI's als namespace te gebruiken, worden naamsconflicten vermeden, zoals bij 'naam', dat zowel een studentnaam als een cursusnaam kan aanduiden. Bovendien garandeert het gebruik van gekwalificeerde namen dat de gegevens in het document óók na import in een andere omgeving en in combinatie met XML-gegevens uit andere bronnen, hun eenduidige betekenis behouden.
Een XML-document voor een cursus zou er als volgt uit kunnen zien:

```
<?xml version="1.0"?>
<om:cursus
   xmlns:om="http://www.edu.org/onderwijsmateriaal"
   xmlns:sg="http://www.uni.org/student">

   <om:toets niveau="1">
      <om:opgave type="MC" genre="dieren">
         <om:vraagtekst>Een vleermuis is een ...</om:vraagtekst>
         <om:alternatief volgnr="A">zoogdier</om:alternatief>
         <om:alternatief volgnr="B">vogel</om:alternatief>
         <om:alternatief volgnr="C">reptiel</om:alternatief>
         <om:correctAntwoord>A</om:correctAntwoord>
      </om:opgave>
      <om:opgave type="MC" genre="planten">
         <om:vraagtekst>Een meidoorn is een ...</om:vraagtekst>
         <om:alternatief volgnr="A">kruid</om:alternatief>
         <om:alternatief volgnr="B">heester</om:alternatief>
         <om:alternatief volgnr="C">boom</om:alternatief>
         <om:correctAntwoord>B</om:correctAntwoord>
      </om:opgave>
         <om:opgave type="kortAntwoord" genre="dieren">
         <om:vraagtekst>Hoeveel poten heeft een spin?</om:vraagtekst>
         <om:correctAntwoord>8</om:correctAntwoord>
      </om:opgave>
   </om:toets>

   <sg:student studentnummer="123456789">
      <sg:naam>Jan Pietersen</sg:naam>
      <sg:studieresultaat>3</sg:studieresultaat >
   </sg:student>

</om:cursus>
```

Deze oplossing is te vereenvoudigen (althans visueel, want in wezen verandert er niets) door de meest gebruikte namespace als default namespace te nemen. We krijgen dan:

```
<?xml version="1.0"?>
<cursus
   xmlns="http://www.edu.org/onderwijsmateriaal"
   xmlns:sg="http://www.uni.org/student">

   <toets niveau="1">
      <opgave type="MC" genre="dieren">
         <vraagtekst>Een vleermuis is een ...</vraagtekst>
         <alternatief volgnr="A">zoogdier</alternatief>
         <alternatief volgnr="B">vogel</alternatief>
         <alternatief volgnr="C">reptiel</alternatief>
         <correctAntwoord>A</correctAntwoord>
      </opgave>
      <opgave type="MC" genre="planten">
         <vraagtekst>Een meidoorn is een ...</vraagtekst>
         <alternatief volgnr="A">kruid</alternatief>
         <alternatief volgnr="B">heester</alternatief>
         <alternatief volgnr="C">boom</alternatief>
         <correctAntwoord>B</correctAntwoord>
       </opgave>
       <opgave type="kortAntwoord" genre="dieren">
          <vraagtekst>Hoeveel poten heeft een spin?</vraagtekst>
          <correctAntwoord>8</correctAntwoord>
       </opgave>
    </toets>

   <sg:student studentnummer="123456789">
      <sg:naam>Jan Pietersen</sg:naam>
      <sg:studieresultaat>3</sg:studieresultaat >
   </sg:student>

</cursus>
```

5 a Een qualified name kan een naam zijn die zijn kwalificatie ontleent aan een default-namespacedeclaratie. Het is dan een naam zonder dubbele punt, dus géén colonized name. Omgekeerd: een colonized name bevat per definitie een namespace-alias, en wordt dus gekwalificeerd door de bijbehorende namespace. Met andere woorden: het is een qualified name.

b en c Beide beweringen zijn waar: een gekwalificeerde naam is per definitie een naam die tot een namespace behoort.

d Deze bewering is waar: een alias koppelt altijd een namespace aan een naam.

e Deze bewering is onwaar: een naam zonder alias kán tot een namespace behoren, namelijk wanneer hij valt onder de werking van een default-namespacedeclaratie, al dan niet in een omhullend element.