

Gedistribueerde Toepassingen Industrieel Ingenieur

Veerle Ongenae

Master in de Industriële Wetenschappen: Informatica
Academiejaar 2018–2019





Gedistribueerde toepassingen (E765017)

Cursusomvang (nominale waarden; effectieve waarden kunnen verschillen per opleiding)

Studiepunten 3.0	Studietijd 90 u	Contacturen 30.0 u
------------------	-----------------	--------------------

Aanbodsessies en werkvormen in academiejaar 2018-2019

A (semester 1)	Nederlands	hoorcollege	12.0 u
		werkcollege: PC-	13.5 u
		groepswork	4.5 u

Lesgevers in academiejaar 2018-2019

Ongenaë, Veerle	TW05	Verantwoordelijk lesgever
-----------------	------	---------------------------

Aangeboden in onderstaande opleidingen in 2018-2019

	stptn	aanbodssessie
Master of Science in de industriële wetenschappen: informatica	3	A

Onderwijstalen

Nederlands

Trefwoorden

Webservices, SOAP, Netwerkprogrammatie, Client/Server-programmatie, gedistribueerde applicaties, WCF, Computerwetenschappen (P170), Informatica (P175), Computertechnologie (T120)

Situering

De doelstellingen van dit opleidingsonderdeel zijn:

- een gedistribueerde applicatie ontwikkelen m.b.v. WCF, Webservices en JAX/WS.
- een client-servertoepassing in een TCP/IP-omgeving programmeren met behulp van het socket-paradigma

Inhoud

- XSLT en XPath
- Webservices: SOA, SOAP
- WCF
- Netwerkprogrammatie in Java, JMS, Java NIO

Begincompetenties

- Basiskennis besturingssystemen Linux.
- Op een doorgedreven niveau webapplicaties kunnen ontwikkelen in het .NET- en J2EE-platform.
- Op een doorgedreven niveau kunnen objectgeoriënteerd programmeren en ontwerpen in Java

Eindcompetenties

- 1 Basisprincipes kennen van client-serverprogrammatie en in staat zijn om een client/server-applicatie te ontwerpen en te implementeren in Java.
- 2 Basisprincipes van webservices kennen en in staat zijn om een webservice te ontwerpen en te implementeren in Java en C#.
- 3 Basisprincipes van JMS en Java NIO beheersen en kunnen implementeren.

Creditcontractvoorwaarde

Toelating tot dit opleidingsonderdeel via creditcontract is mogelijk mits gunstige beoordeling van de competenties

Examencontractvoorwaarde

Dit opleidingsonderdeel kan niet via examencontract gevolgd worden

Didactische werkvormen

Groepswerk, hoorcollege, werkcollege: PC-klasoefeningen

Toelichtingen bij de didactische werkvormen

- Hoorcollege (12 uren)
- Werkcollege (13,5 uren): PC-klasoefeningen (zelfstandig werk aan een individuele PC; aanwezigheid verplicht).
- Groepswerk (4,5 uren): uitgebreid computerlabo (aanwezigheid verplicht)

Leermateriaal

Syllabus, slides en voorbeeldprogramma's, tutorials op internet.

Referenties

Vakinhoudelijke studiebegeleiding

De docent is ter beschikking voor extra uitleg tijdens de labo's, voor of na de theorielessen en eventueel op andere ogenblikken na afspraak.

Evaluatiemomenten

periodegebonden en niet-periodegebonden evaluatie

Evaluatievormen bij periodegebonden evaluatie in de eerste examenperiode

Mondeling examen

Evaluatievormen bij periodegebonden evaluatie in de tweede examenperiode

Mondeling examen

Evaluatievormen bij niet-periodegebonden evaluatie

Werkstuk, vaardigheidstest

Tweede examenkans in geval van niet-periodegebonden evaluatie

Examen in de tweede examenperiode is enkel mogelijk in gewijzigde vorm

Toelichtingen bij de evaluatievormen

De score op de NPE is de combinatie van een test over de labo's (75%) en de groepsofdracht (25%).

Wie niet slaagt voor de NPE, legt in de tweede examenperiode één test over alle opdrachten (labo en groepswerk) af. Deze punten vervangen de quotatie voor de NPE.

Eindscoreberekening

Theorie: mondeling examen (50%)

Labo's: permanente evaluatie via testen en afgeven van opdrachten/projecten (50%).

Inhoudsopgave

1	Extensible Stylesheet Language Transformations	1
1.1	XSL, XPath, XSL-FO en XSLT	1
1.2	Cascading Style Sheets (CSS)	2
1.2.1	CSS Syntax	2
1.3	XSLT	4
1.3.1	Sjablonen (<i>templates</i>)	4
1.3.2	Instructie-elementen	6
2	XML Path Language (XPath)	17
2.1	De boomstructuur van een XML-document	17
2.2	Paden	19
2.2.1	Basispad (<i>Root Location Path</i>)	19
2.2.2	Kindelement-stap (<i>Child Element Location Steps</i>)	20
2.2.3	Attribuut-stap (<i>Attribute Location Steps</i>)	20
2.2.4	Stap naar commentaar, tekst of verwerkingsinstructie	21
2.2.5	Andere padelementen	21
2.2.6	Voorwaarden	23
2.2.7	Lange padnamen	23
2.3	Andere XPath-uitdrukkingen	24
2.3.1	Datatypes	26

2.3.2	Functies	27
3	Webservices	29
3.1	Webservices in Java	40
3.2	Webservices in C#	44
3.3	Externe API's	51
4	Netwerkprogrammatie in Java	53
4.1	Inleiding	53
4.2	Basistechnieken	53
4.2.1	Een voorbeeld: het UFP-protocol	53
4.2.2	Een eenvoudige client	54
4.2.3	De klasse <i>InetAddress</i>	57
4.2.4	Een eenvoudige server	58
4.3	Concurrency	61
4.3.1	Concurrency versus Parallelism	62
4.3.2	Processen en threads	62
4.3.3	Threads in Java	63
4.3.4	Communicatie tussen threads	65
4.3.5	Executors en concurrent collections	67
4.4	Netwerkprogrammatie met behulp van draden	73
4.4.1	Structuur server	73
4.4.2	De draad	76
5	Java NIO	79
6	JMS	85

Hoofdstuk 1

Extensible Stylesheet Language Transformations

1.1 XSL, XPath, XSL-FO en XSLT

Zoals reeds opgemerkt in de cursus Gegevenstechnologieën wordt XML niet gebruikt om gegevens op te maken, maar wel om gegevens te structureren. Om de gegevens uit een XML-documenten te presenteren werden er andere technologieën ontwikkeld. [HM01]

XSL is de afkorting van *Extensible Stylesheet Language Family* en is een standaard om XML-documenten te presenteren en te transformeren. XSL bestaat uit drie delen

- XSL Transformations (XSLT)
- XPath
- XSL Formatting Objects (XSL-FO)

XSLT is een XML-taal om XML-documenten om te vormen naar een ander formaat. Dit formaat kan terug een XML-taal zijn, maar dit hoeft niet. Andere formaten zijn bv. HTML, Latex, PDF, ...

XPath is een taal waarmee stukken uit een XML-document geselecteerd worden of waarmee verwezen wordt naar bepaalde delen uit een XML-document.

XSL-FO is het XML-formaat voor drukwerk. In XSL-FO wordt de layout van tekstpagina's beschreven.

1.2 Cascading Style Sheets (CSS)

Een eerste technologie die gebruikt kan worden om XML-documenten op te maken is *Cascading Style Sheets (CSS)*[Casa]. CCS bepaalt de opmaak van de elementen in een XML-document op dezelfde manier als voor de elementen van een HTML-pagina. Er worden geen transformaties uitgevoerd.

CSS voegt stijlkenmerken toe aan de inhoud van het XML-documenten. Zo kan je met CSS bv. de inhoud van bepaalde elementen in vetjes of in een bepaalde kleur tonen. Maar het is bv. niet mogelijk om met CSS de volgorde van de elementen aan te passen.

1.2.1 CSS Syntax

CSS is geen XML-taal. De CSS-syntax is echter zo eenvoudig dat dat geen probleem is. Een CSS-document bestaat uit een lijst van elementen waarvoor bepaalde stijlkenmerken opgegeven worden, zoals kleur, lettertype, positionering, ...

In het volgend voorbeeld wordt een XML-document dat de beschrijving van twee boeken bevat opge maakt door de CSS *boeken.css*.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="boeken.css"?>
<!DOCTYPE books SYSTEM "boeken.dtd">
<books>
  <book>
    <isbn>0-596-00058-8</isbn>
    <title>XML in a Nutshell</title>
    <author>
      <name>Harold</name>
      <firstname>Elliotte</firstname>
      <firstname>Rusty</firstname>
    </author>
    <author>
      <name>Means</name>
      <initial>W.</initial>
      <firstname>Scott</firstname>
    </author>
  </book>
  <book>
    <isbn>0-201-77641-3</isbn>
    <title>XML, Web Services, and the
      Data Revolution</title>
    <author>
      <name>Coyle</name>
      <initial>P.</initial>
      <firstname>Frank</firstname>
    </author>
  </book>
</books>
```

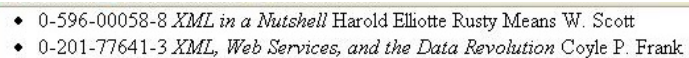

Het bijhorende CSS-document is dan.

```
books {
    margin-left: 0.5cm
}

book {
    display: list-item;
    list-style-position: inside
}

title {
    font-style: italic
}
```

Als een browser het XML-document opent, dan gebruikt die het CSS-bestand om het XML-document te tonen. Het resultaat van het bovenstaande voorbeeld in een browser is als volgt.

- 
- 0-596-00058-8 *XML in a Nutshell* Harold Elliott Rusty Means W. Scott
 - 0-201-77641-3 *XML, Web Services, and the Data Revolution* Coyle P. Frank

Figuur 1.1: Resultaat

Algemeen bestaat een CSS-document uit verschillende elementbeschrijvingen van de volgende vorm.

```
naamTag1, ..., naamTagN {
    eig1: waarde1;
    eig2: waarde2;
    ...
    eigM: waardeM
}
```

Hierbij zijn *naamTag1*, ..., *naamTagN* de namen van elementen met dezelfde opmaak. De opmaak wordt vastgelegd door de kenmerken *eig1*, ..., *eigM* een waarde te geven. Een overzicht van alle eigenschappen en hun mogelijke waarden vind je terug in de CSS-specificatie ([Casb]).

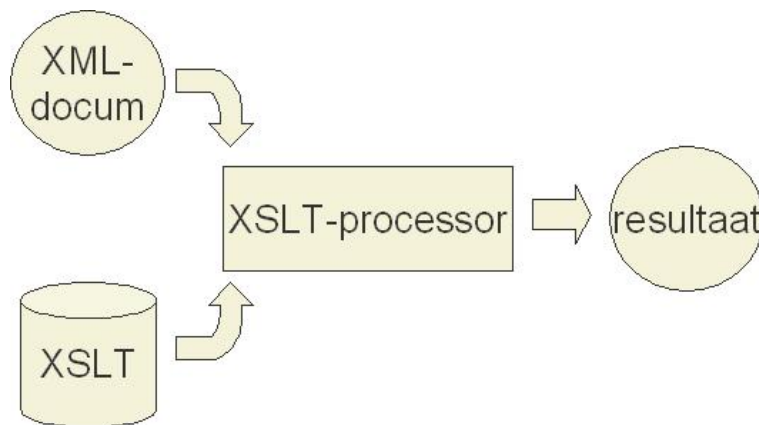
De link tussen het XML-document en het bijhorende CSS-bestand wordt bepaald door een verwerkingsinstructie (*processing instruction*) *xml-stylesheet* in het begin van het XML-document. Het attribuut *type* bepaalt het type van de *style sheet* en het attribuut *href* de locatie.

```
<?xml-stylesheet type="text/css" href="boeken.css"?>
```

1.3 XSLT

XSLT ([XSL]) is een XML-technologie waarmee men regels kan vastleggen om een XML-document te transformeren naar een ander (eventueel XML-) document. Met deze technologie is het o.a. mogelijk om elementen te sorteren of te filteren, om nieuwe informatie toe te voegen, ... Een XSLT-document bestaat uit sjablonen (*templates*) die de transformatie beschrijven. XSLT is een XML-taal en gebruikt dus de XML-syntax.

Een XSLT-processor maakt op basis van een XML-document en een XSLT-bestand een nieuw document aan. Sommige webbrowsers hebben een ingebouwde XSLT-processor, maar ook alleenstaande applicaties worden gebruikt als XSLT-processors.



Figuur 1.2: Principe XSLT-transformatie

1.3.1 Sjablonen (*templates*)

Elk sjabloon bestaat uit een patroon en een beschrijving van de te genereren uitvoer. Een XSLT-processor vergelijkt de elementen en andere knopen in het XML-document met de patronen van de sjablonen. Indien het element of de knoop voldoet aan het patroon, dan wordt op basis van het sjabloon een stuk uitvoer gegenereerd.

XSLT gebruikt XPath om knopen in een XML-documenten te identificeren en patronen te beschrijven.

Voorbeeld XSLT-document

In dit voorbeeld maken we een HTML-document op basis van het XML-document uit §1.2. We gebruiken hiervoor het volgende XSLT-document.

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

```

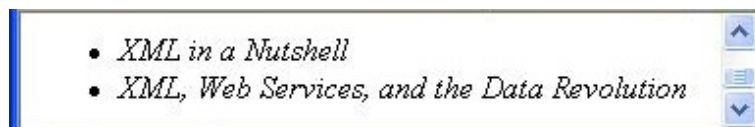
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">
  <html>
    <head>
      <title>Overzicht boeken</title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="books">
  <ul>
    <xsl:apply-templates/>
  </ul>
</xsl:template>

<xsl:template match="book">
  <li>
    <xsl:value-of select="title"/>
  </li>
</xsl:template>
</xsl:stylesheet>

```

Het resultaat van de transformatie van het XML-document uit §1.2 met de bovenstaande sjablonen is een HTML-document. Deze HTML-pagina bevat een lijstje met de titels van de boeken uit het oorspronkelijke XML-document.



Figuur 1.3: Resultaat XSLT-transformatie

Structuur XSLT-document

Een XSLT-document is een XML-document en begint dus met de XML-declaratie (zie [XML]). Het basiselement van een XSLT-document is het *stylesheet*- of *transform*-element. Deze elementen zijn synoniemen.

Alle XSLT-elementen, en dus ook *stylesheet* en *transform* behoren tot de namenruimte (<http://www.w3.org/1999/XSL/Transform>). Voor deze namenruimte wordt standaard de afkorting *xsl* gebruikt. Naast het *xmlns*-attribuut moet het basiselement ook het attribuut *version* met de waarde *1.0*, *2.0* of *3.0* hebben.

Sjablonen bepalen de vorm van de uitvoer van een XSLT-transformatie. Elk sjabloon in het XSLT-document wordt voorgesteld door een *xsl:template*-element. Het *match*-attribuut van dit element bepaalt de elementen of knopen waarop het sjabloon toegepast moet worden. De waarde van het *match*-attribuut is een XPath-uitdrukking. Deze uitdrukking is een patroon dat één of meerdere knopen in het XML-document identificeert. In het voorbeeld zijn de patronen namen van elementen (*books* en *book*) en het begin van het XML-document (/).

De inhoud van het *xsl:template*-element bepaalt de vorm van de uitvoer. Het sjabloon voor een *book*-element in het voorbeeld beschrijft dat voor elk *book*-element de HTML-tag ``, de inhoud van het *titel*-element van het huidige *book*-element en de eind-tag `` naar de uitvoer geschreven moet worden.

Het element *output* bepaalt het formaat van de gegenereerde uitvoer. De *value-of* en *apply-templates* elementen worden beschreven in §1.3.2.

De inhoud van het *template*-element bestaat voor een deel uit instructie-elementen (zie §1.3.2) die behoren tot de namenruimte “<http://www.w3.org/1999/XSL/Transform>” en voor een deel uit XML-stukken die integraal naar de uitvoer geschreven worden.

Aangezien het hele XSLT-document een goedgevormd XML-document moet zijn, moeten de HTML-elementen die deel uitmaken van de sjablonen voldoen aan de XML-syntaxis.

XSLT-processors

Een XSLT-processor voert de transformatie van een XML-document op basis van een XSLT-document uit. Het resultaat kan een XML-document, een HTML-document, ... zijn. Een expliciete link tussen het XML-document en het XSLT-document is niet nodig. Het moet mogelijk zijn om een zelfde XML-document naar verschillende documenten te transformeren gebruik makend van verschillende XSLT-documenten. Bijvoorbeeld een transformatie naar HTML en een transformatie naar XSL-FO, waarmee PDF gegenereerd kan worden.

Enkel indien het XML-document door een browser getransformeerd wordt is er een link met het XSLT-document nodig. Zoals bij CSS (zie §1.2) wordt deze link gerealiseerd met behulp van een verwerkingsinstructie.

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

1.3.2 Instructie-elementen

Instructie-elementen zijn elementen uit de namenruimte “<http://www.w3.org/1999/XSL/Transform>” die voorkomen binnen een *template*-element. Deze elementen hoeven geen directe kinderen van het *template*-element te zijn.

xsl:value-of

Dit element bepaalt de tekst van een XPath-uitdrukking en plaatst die op de uitvoer. In het geval van een element is dat de inhoud van het element. Voor een attribuut is dat de waarde.

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

Het bovenstaande sjabloon schrijft voor elk *book*-element het volgend stukje HTML naar de uitvoer. Hierbij is *titel huidige boek* de inhoud van het *title*-element van het *book*-element waarop het sjabloon toegepast wordt.

```
<li>
  titel huidige boek
</li>
```

xsl:apply-templates

Standaard leest een XSLT-processor een XML-document van voor naar achteren. Voor elk element dat de XSLT-processor tegenkomt voert de processor het sjabloon uit. Is er geen sjabloon beschikbaar dan wordt de inhoud van het element naar de uitvoer geschreven of worden de sjablonen van de kindelementen uitgevoerd. Dit betekent dat de sjablonen uitgevoerd worden volgens de volgorde van de elementen in het XML-document.

Met een sjabloon kan je echter de volgorde van het doorlopen van het XML-document veranderen. Zo werd in §1.3.1 voor een *book*-element enkel de inhoud van het *title* uitgeschreven. De andere kindelementen van een *book*-element werden genegeerd.

Met het element *xsl:apply-templates* kan je expliciet de volgorde van het uitvoeren van de sjablonen bepalen. Het *select*-attribuut bepaalt dan voor welke knopen het sjabloon uitgevoerd moet worden. Is dit attribuut niet aanwezig dan wordt het sjabloon voor elk kindelement uitgevoerd.

Pas het volgende stukje XSLT toe op het XML-bestand met boeken uit §1.2. Het basiselement van het XML-bestand was *books*. Dit element bestond uit een aantal *book*-elementen die een titel, auteurs, ... hadden.

```
<xsl:template match="books">
  <ul>
    <xsl:apply-templates/>
  </ul>
</xsl:template>

<xsl:template match="book">
  <li>
```

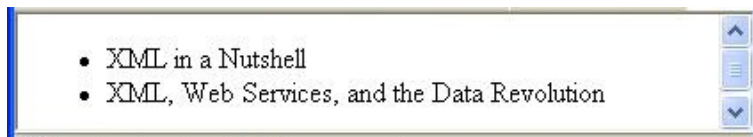
```

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

```

Het resultaat van dit stukje XSLT is dat het sjabloon voor het element *books* wordt uitgevoerd: de tag `` uitschrijven, vervolgens de sjablonen van alle kindelementen (lees: alle *book*-elementen) uitvoeren en dan de tag `` uitschrijven.

Voor elke *book*-element wordt het sjabloon uitgevoerd: de tag `` uitschrijven, vervolgens de sjablonen van alle *title*-elementen uitvoeren en dan de tag `` uitschrijven. Afhankelijk van het sjabloon voor het *title*-element kan het resultaat bv. het volgende zijn. Deze HTML-pagina bevat een lijstje met de titels van de boeken uit het oorspronkelijke XML-document.



Figuur 1.4: Uitvoer stukje XSLT

Als een sjabloon verschillende keren uitgevoerd moet worden voor verschillende elementen of knopen, dan is de volgorde waarin dat gebeurt de volgorde van de knopen in het document. Om de sjablonen in een andere volgorde uit te voeren kan je gebruik maken van het *xsl:sort*-element (§1.3.2). Dit element is dan een kindelement van het *xsl:apply-templates*-element.

xsl:for-each

Met behulp van het *xsl:foreach*-element is het mogelijk om een zelfde actie voor een reeks knopen uit te voeren. De knopen waarop deze actie uitgevoerd wordt, worden bepaald door het *select*-attribuut. De waarde van dit attribuut is een XPath-uitdrukking die een aantal knopen identificeert.

De geselecteerde knopen worden behandeld in de volgorde waarin ze in het XML-document voorkomen. Om deze volgorde te veranderen kan je gebruik maken van het *xsl:sort* (zie §1.3.2)-element.

In het volgend voorbeeld kan een *author*-element meerdere *firstname*-elementen bevatten. Het sjabloon voor een *author*-element voert het sjabloon voor de verschillende *firstname*-elementen uit en schrijft tussen de uitvoer van twee sjablonen een spatie naar de uitvoer.

```

<xsl:template match="author">
    <xsl:for-each select="firstname">
        <xsl:apply-templates select="."/>
        <xsl:text> </xsl:text>
    </xsl:for-each>
</xsl:template>

```

Merk op dat ‘.’ een XPath-uitdrukking is voor de huidige knoop. In het geval van het voorbeeld is dat een *firstname*-element.

xsl:sort

Het *xsl:sort*-element is een kindelement van een *xsl:apply-templates* (zie §1.3.2)-element of een *xsl:for-each* (zie §1.3.2)-element. Het bepaalt de volgorde waarin de sjablonen of de acties uitgevoerd worden. Deze sortering kan alfabetisch of numeriek zijn. Wil je op meerdere aspecten sorteren, dan moet je meerdere *xsl:sort*-elementen gebruiken.

Dit element heeft o.a. de volgende attributen.

select bepaalt de knoop, het element, het attribuut, ... waarop gesorteerd wordt.

data-type bepaalt het type van de knoop waarop gesorteerd wordt. Standaard wordt er alfabetisch gesorteerd (waarde is dan *text*). Om numeriek te sorteren moet dit attribuut de waarde *number* hebben.

order bepaalt de sorteervolgorde. Standaard is dat oplopend (waarde van het attribuut is dan *ascending*). Om aflopend te sorteren moet dit attribuut de waarde *descending* hebben.

In het volgend voorbeeld wordt voor elke kindelement *book* van het element *books* het sjabloon uitgevoerd. De volgorde waarin de sjablonen worden uitgevoerd is afhankelijk van het *isbn*-element. Het sjabloon wordt eerst toegepast op het boek met het (alfabetisch) kleinste isbn-nummer, enz.

```
<xsl:template match="books">
  <ul>
    <xsl:apply-templates select="book">
      <xsl:sort select="isbn"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>
```

xsl:if

De XSLT-syntax heeft ook een aantal selectiestructuren. Het instructie-element *xsl:if* is er één van. Deze constructie is vergelijkbaar met een *if*-structuur zonder *else*-clausule uit een programmeertaal. De algemene syntax is de volgende.

```
<xsl:if test="voorwaarde">
  opdrachten
</xsl:if>
```

Hierbij is de waarde van het attribuut *test* (*voorwaarde*) een logische uitdrukking in XPath die al dan niet vervuld is. De XSLT-opdrachten *opdrachten* worden uitgevoerd als de voorwaarde vervuld is.

Stel dat je een XML-document hebt waarbij de structuur bepaald wordt door de volgende DTD.

```
<!ELEMENT catalog (cd*)>
<!ELEMENT cd (price, title, artist)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

Het volgende stukje XSLT zorgt er dan voor dat enkel voor de CD's waarvan de prijs groter is dan 10, de inhoud van het *title*- en het *artist*-element uitgeschreven wordt.

```
<xsl:for-each select="catalog/cd">
  <xsl:if test="price &gt; 10">
    <tr><td>
      <xsl:value-of select="title"/>
    </td><td>
      <xsl:value-of select="artist"/>
    </td></tr>
  </xsl:if>
</xsl:for-each>
```

Merk op dat de voorwaarde gebruik maakt van *>* ipv van *>* omdat het XSLT-document een goed-gevormd XML-document moet zijn.

xsl:choose

Een andere selectie-structuur is het element *xsl:choose*. Deze structuur selecteert nul of één geval van een reeks alternatieven. De syntax is als volgt.

```
<xsl:choose>
  <xsl:when test="geval1">opdrachten1</xsl:when>
  <xsl:when test="geval2">opdrachten2</xsl:when>
  ...
  <xsl:when test="gevalN">opdrachtenN</xsl:when>
  <xsl:otherwise>opdrachtenAnders</xsl:otherwise>
</xsl:choose>
```

Het *xsl:choose*-element bestaat uit één of meerdere *xsl:when*-elementen eventueel gevolgd door één *xsl:otherwise*-element. Elk *xsl:when*-element heeft een *test*-attribuut. De waarde van dit attribuut (respectievelijk *geval1*, *geval2*, ...) is een logische XPath-uitdrukking.

De opdrachten van het eerste *when*-element waarvan de voorwaarde vervuld is, worden uitgevoerd. Indien voor geen enkele *when*-element de voorwaarde waar is, dan worden de opdrachten in het *xsl:otherwise*-element uitgevoerd indien dat element aanwezig is.

xsl:text

Met het *xsl:text*-element kan je tekst naar de uitvoer schrijven. Dit kan handig zijn om spaties of andere witte ruimte (*whitespace*) op te nemen in het resultaat. Aangezien een XSLT-document een XML-document is, worden spaties, ... normaal genegeerd.

In het volgend voorbeeld wordt het *xsl:text*-element gebruikt om een spatie te plaatsen tussen de inhoud van verschillende *firstname*-elementen.

```
<xsl:for-each select="firstname">
  <xsl:apply-templates select="."/>
  <xsl:text> </xsl:text>
</xsl:for-each>
```

xsl:variable

Het element *xsl:variable* wordt gebruikt om een naam met een bepaalde waarde te verbinden. Deze waarde kan dan elders in het XSLT-document opgeroepen worden met behulp van de gegeven naam. Merk op dat het geen variabele is zoals in een programmeertaal. De waarde kan namelijk niet aangepast worden en dus eigenlijk meer het equivalent van een constante in een programmeertaal. De waarde kan van gelijk welk XPath-type zijn: string, getal, verzameling knopen,

De syntax van het *xsl:variable*-element:

```
<xsl:variable name="naam" select="waarde"/>
```

of

```
<xsl:variable name="naam">
  ...
</xsl:variable>
```

In het eerste geval heeft het *xsl:variable*-element een *select*-attribuut en moet het een leeg element zijn. De waarde is dan de waarde van het *select*-attribuut (een XPath-uitdrukking). In het tweede geval is de waarde de inhoud van het *xsl:variable*-element.

De waarde van de ‘variabele’ kan in een uitdrukking elders in het XSLT-document opgeroepen worden. Hiervoor wordt een uitdrukking van de vorm *\$naam* gebruikt. *naam* is de waarde van het *name*-attribuut.

In het volgend voorbeeld worden twee ‘variabelen’ gedeclareerd. De eerste is het aantal *book*-elementen in het element *books*. De tweede is het getal *10*. Het stukje XSLT schrijft uit hoeveel *book*-elementen er zijn. Als minder zijn dan 10 dan wordt dit ook gemeld.

```
<xsl:variable name="aantal" select="count (books/book)"/>
```

```

<xsl:variable name="min">10</xsl:variable>
<xsl:text>Er zijn </xsl:text>
<xsl:value-of select="$aantal"/>
<xsl:text> boeken in de catalogus. </xsl:text>
<xsl:if test="$aantal < $min">
  <xsl:text>Er zijn te weinig boeken.</xsl:text>
</xsl:if>

```

xsl:param en xsl:with-param

Met de elementen *xsl:param* en *xsl:with-param* is het mogelijk om een parameter mee te geven aan een sjabloon. Het element *xsl:param* declareert de parameter in het sjabloon en geeft hem een standaardwaarde. Het element *xsl:with-param* wordt gebruikt om de parameter mee te geven bij het oproepen van het sjabloon.

De syntax van het *xsl:param*-element is analoog aan de syntax van het *xsl:variable*-element.

```
<xsl:param name="naam" select="waarde"/>
```

of

```

<xsl:param name="naam">
  ...
</xsl:param>

```

De waarde van het attribuut *name* bepaalt de naam van de parameter. De standaardwaarde van de parameter is in het eerste geval de waarde van het *select*-attribuut en in het tweede de inhoud van het *xsl:param*-element. Als het *xsl:param*-element een *select*-attribuut heeft, dan moet het een leeg element zijn. Dit attribuut is een XPath-uitdrukking. De waarde van de parameter is het resultaat van deze uitdrukking.

Het *xsl:param*-element is een kindelement van een *xsl:template*-element of eventueel van het basis-element. In het laatste geval is de parameter beschikbaar voor meerdere sjablonen.

Om een parameter mee te geven aan een sjabloon wordt het *xsl:with-param*-element gebruikt.

```

<xsl:apply-templates ...>
  <xsl:with-param name="naam" select="waarde"/>
</xsl:apply-templates>

```

Hierbij is *naam* de naam van de parameter die gedeclareerd werd in het sjabloon (m.b.v. het *xsl:param*-element). De waarde van de parameter, de XPath-uitdrukking *waarde* in het voorbeeld, vervangt de standaardwaarde van de parameter die vastgelegd werd door het *xsl:param*-element.

Ook hier moet het *xsl:with-param*-element leeg zijn als het een *select*-attribuut heeft. Een alternatief voor bovenstaande code is.

```
<xsl:apply-templates ...>
  <xsl:with-param name="naam">
    ...
  </xsl:with-param>
</xsl:apply-templates>
```

De inhoud van het element *xsl:with-param* bepaalt nu de waarde van de parameter.

Wordt een sjabloon opgeroepen zonder parameter door te geven, dan heeft de parameter de standaardwaarde bepaald door het *xsl:param*-element.

De waarde van de parameter wordt in het sjabloon opgeroepen door een uitdrukking van de vorm *\$naam*. Hierbij is *naam* de naam van de parameter (de waarde van het *name*-attribuut van het element *xsl:param*).

In het volgend voorbeeld veronderstellen we dat er verschillende servers met foto's beschikbaar zijn via het internet. De gegevens van deze servers worden bewaard in een XML-bestand van de volgende vorm.

```
<?xml version="1.0"?>
<servers>
  <server>
    <url>http://www.mijnfoto.be</url>
    <foto bron="IMG001">Mijn huis</foto>
    <foto bron="IMG002">Kinderen</foto>
    <foto bron="IMG003">Poezen</foto>
    <foto bron="IMG004">Tuin</foto>
    <foto bron="IMG005">Vakantie in Praag</foto>
    <foto bron="IMG006">Scoutskamp</foto>
    <foto bron="IMG007">Sneeuwpret</foto>
  </server>
  <server>
    <url>http://www.bedreigdedieren.org</url>
    <foto bron="IMG001">Bengaalse tijger</foto>
    <foto bron="IMG002">Panda</foto>
    <foto bron="IMG003">Koala</foto>
    <foto bron="IMG004">Wolf</foto>
  </server>
</servers>
```

De bijhorende DTD is dan

```
<!ELEMENT servers (server*)>
<!ELEMENT server (url,foto*)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT foto (#PCDATA)>
<!ATTLIST foto bron NMTOKEN #REQUIRED
```

De bedoeling is om voor XML-bestanden van bovenstaande vorm een HTML-pagina te genereren die een overzicht geeft van de beschikbare foto's en waarop je kan doorklikken naar de foto's op de

servers. In de twee kolom wordt de URL van de foto nog eens vermeld. De link in de eerste kolom klikt door naar de URL getoond in de tweede kolom. (Zie figuur 1.5)

http://www.mijnfoto.be	
Mijn huis	http://www.mijnfoto.be/TMG001
Kinderen	http://www.mijnfoto.be/TMG002
Poezen	http://www.mijnfoto.be/TMG003
Tuin	http://www.mijnfoto.be/TMG004
Vakantie in Praag	http://www.mijnfoto.be/TMG005
Scoutskamp	http://www.mijnfoto.be/TMG006
Sneeuwpret	http://www.mijnfoto.be/TMG007

http://www.bedreigdedieren.org	
Bengaalse tijger	http://www.bedreigdedieren.org/TMG001
Panda	http://www.bedreigdedieren.org/TMG002
Koala	http://www.bedreigdedieren.org/TMG003
Wolf	http://www.bedreigdedieren.org/TMG004

Figuur 1.5: Voorbeeld HTML-uitvoer

De volgende sjablonen voor *server* en *foto* realiseren deels de omzetting van het XML-bestand naar het HTML-bestand.

```
<xsl:template match="server">
  <xsl:variable name="host" select="url"/>
  <table border="1">
    <tr><th colspan="2">
      <xsl:value-of select="$host"/>
    </th></tr>
    <xsl:apply-templates select="foto">
      <xsl:with-param name="host"
        select="$host"/>
    </xsl:apply-templates>
  </table> <br></br>
```

```
</xsl:template>

<xsl:template match="foto">
  <xsl:param name="host"/>
  <xsl:variable name="url">
    <xsl:value-of select="$host"/>
    <xsl:text>/</xsl:text>
    <xsl:value-of select="@bron"/>
  </xsl:variable>
  <tr><td>
    <a>
      <xsl:attribute name="href">
        <xsl:value-of select="$url"/>
      </xsl:attribute>
      <xsl:value-of select="."/>
    </a>
  </td><td>
    <xsl:value-of select="$url"/>
  </td></tr>
</xsl:template>
```

Het *attribute*-element in het bovenstaande voorbeeld worden gebruikt om een attribuut toe te voegen aan het HTML-element *a*.

Hoofdstuk 2

XML Path Language (XPath)

XPath is een taal om bepaalde stukken van een XML-documenten te selecteren. XPath is zelf geen XML-taal. XPath beschouwt een XML-document als een boomstructuur met knopen. Aan de hand van allerlei criteria, zoals de naam van een element, zijn positie, zijn inhoud, de waarde van een attribuut, ... is het mogelijk om knopen in die boom te identificeren.

Zoals reeds besproken in hoofdstuk 1 gebruikt XSLT XPath-uitdrukkingen om bijvoorbeeld elementen te selecteren waarop een sjabloon toegepast moet worden. Ook andere technologieën zoals XForms en XPointer maken gebruik van XPath.

Naast knopen van een XML-document kan het resultaat van een XPath-uitdrukking een getal, een string of een logische waarde (*boolean*) zijn. Eenvoudige rekenkundige uitdrukkingen en stringmanipulaties zijn dus ook mogelijk.

2.1 De boomstructuur van een XML-document

Voor XPath is een XML-document een boom van knopen. Sommige knopen kunnen andere bevatten. Er is juist één basisknoop (*root node*) die alle andere bevat. XPath is een taal om knopen en verzamelingen van knopen te selecteren in die boom. XPath kent zeven types knopen.

- De basisknoop
- Elementen
- Tekst
- Attributen
- Commentaar
- Verwerkingsinstructies

- Namenruimtes

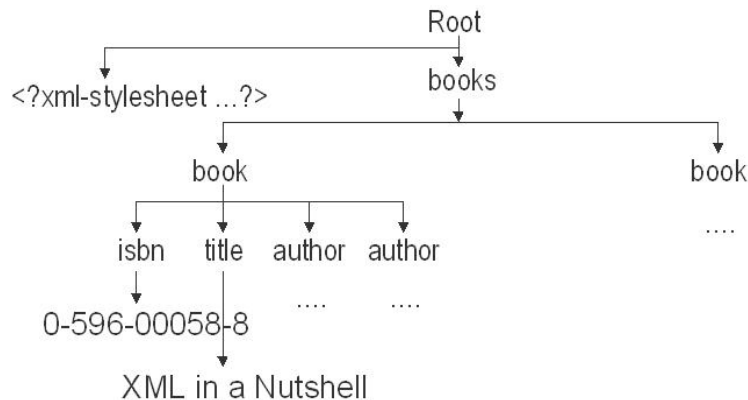
DTD's, entiteiten en CDATA-stukken zijn geen knopen voor XPath en kunnen dus niet geïdentificeerd worden met een XPath-uitdrukking. In de volgende voorbeelden wordt de boomstructuur van een aantal XML-documenten getoond.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="boeken.xsl"?>
<!DOCTYPE books SYSTEM "boeken.dtd">
<books>
  <book>
    <isbn>0-596-00058-8</isbn>
    <title>XML in a Nutshell</title>
    <author>
      <name>Harold</name>
      <firstname>Elliotte</firstname>
      <firstname>Rusty</firstname>
    </author>
    <author>
      <name>Means</name>
      <initial>W.</initial>
      <firstname>Scott</firstname>
    </author>
  </book>
  <book>
    <isbn>0-201-77641-3</isbn>
    <title>XML, Web Services, and the
      Data Revolution</title>
    <author>
      <name>Coyle</name>
      <initial>P.</initial>
      <firstname>Frank</firstname>
    </author>
  </book>
</books>
```

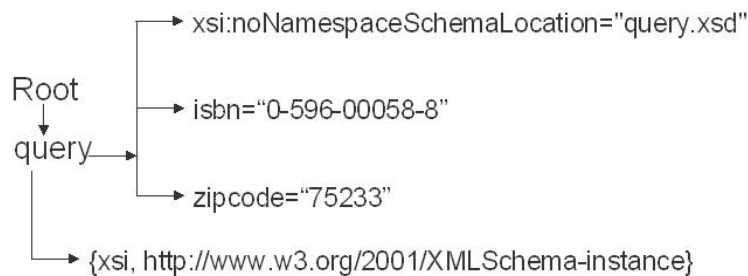
```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="query.xsd"
  isbn="0-596-00058-8" zipcode="75233"/>
```

Merk op dat de basisknoop *NIET* het basiselement van het XML-document is. De basisknoop omvat het volledige XML-document: basiselement, eventuele verwerkingsinstructies of commentaar voor of na het basiselement. De verwerkingsinstructie *xml-stylesheet* uit het eerste voorbeeld is een kind van de basisknoop.

De attributen *xmlns* en *xmlns:prefix* worden niet beschouwd als attribuutknopen. De boom bevat ook de namenruimte die gekend is in het *query*-element.



Figuur 2.1: XPath-boom books



Figuur 2.2: XPath-boom query

2.2 Paden

Een zeer belangrijk XPath-uitdrukking is een pad (*location path*). Een pad identificeert nul, één of meerdere knopen in een XML-document. De types van de geselecteerde knopen zijn één of meerdere types zoals beschreven in §2.1. Een pad bestaat uit verschillende stappen (*location steps*) verbonden met een /. Elke stap wordt uitgevoerd t.o.v. een bepaalde knoop, de contextknoop (*context node*) genoemd. Deze paden zijn op het eerste zicht vergelijkbaar met paden in een bestandssysteem.

2.2.1 Basispad (*Root Location Path*)

Het eenvoudigste pad is '/' en selecteert de basisknoop van het XML-document. Dit is een absoluut pad. In welke context je het basispad ook opgeeft, het identificeert steeds de basisknoop van het XML-document. (Vergelijkbaar met de *root* van een Unix-bestandssysteem.)

In het volgend voorbeeld gebruikt XSLT het basispad om een sjabloon te maken voor het volledige XML-document.

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

```

<html>
  <head>
    <title>Overzicht boeken</title>
  </head>
  <body>
    <xsl:apply-templates/>
  </body>
</html>
</xsl:template>

```

2.2.2 Kindelement-stap (*Child Element Location Steps*)

De kindelement-stap bestaat uit de naam van een element en selecteert alle elementen met deze naam in de huidige context. Bijvoorbeeld de stap *book* identificeert alle kindelementen *book* van de context-knoop. Welke elementen geselecteerd worden hangt af van de contextknoop. Een kindelement-stap is dus steeds een relatief pad.

Bijvoorbeeld, in de contextknoop *books* in het eerste voorbeeld van §2.1, zal het pad *book* verwijzen naar twee *book*-elementen. Was de contextknoop echter */*, de basisknoop van het document, dan selecteert het pad *book* niets.

In XSLT is de contextknoop voor een XPath-uitdrukking, bv. als waarde van een *select*-attribuut, die knoop die afgehandeld wordt. In het onderstaande voorbeeld is de contextknoop voor het pad *name*, het *author*-element waarvoor het sjabloon wordt uitgevoerd.

```

<xsl:template match="author">
  <xsl:apply-templates select="name" />
</xsl:template>

```

2.2.3 Attribuut-stap (*Attribute Location Steps*)

Om het attribuut van een element te selecteren gebruik je een *@* gevolgd door de naam van het attribuut. Het element moet dan de contextknoop zijn. Om in het tweede voorbeeld van §2.1 het attribuut *isbn* van het element *query* te selecteren kan je de volgende paden gebruiken.

- *@isbn* als de context het element *query* is
- */query/@isbn* in elke context

Met het volgende stukje XSLT is het mogelijk om de waarde van het *isbn*-attribuut van het *query*-element uit te schrijven.

```

<xsl:template match="query">
  <xsl:value-of select="@isbn" />
</xsl:template>

```

2.2.4 Stap naar commentaar, tekst of verwerkingsinstructie

Aangezien er naast de basisknoop, elementen en attributen ook knopen in de boomstructuur van een XML-document voorkomen die commentaar, tekst of verwerkingsinstructies voorstellen, moeten die ook geselecteerd kunnen worden met een XPath-uitdrukking. Tekst- en commentaar-knopen hebben geen naam en worden respectievelijk met de volgende stappen geïdentificeerd.

- *text()*
- *comment()*

Het volgende pad selecteert alle ISBN-nummers, als tekstknopen van de XML-boom, in het eerste voorbeeld van §2.1.

```
/books/book/isbn/text()
```

Bij verwerkingsinstructies is het mogelijk om alle verwerkingsinstructies in een bepaalde context te selecteren of om enkel die verwerkingsinstructies te identificeren met een bepaalde naam.

- *processing-instruction()*
- *processing-instruction('target')* (*target* is de naam van de te selecteren verwerkingsinstructies)

2.2.5 Andere padelementen

Naast de stappen besproken in de voorgaande paragrafen kunnen paden nog uit andere aspecten bestaan zoals bv. jokertekens.

Jokertekens (*wildcards*)

Met jokertekens is het mogelijk om verschillende element- of andere knopen tegelijk te selecteren. De drie mogelijke jokertekens zijn.

- ***: selecteert elke elementknoop in de context. De naam van het element is willekeurig.
- *node()*: selecteert alle knopen in de context. Deze knopen kunnen van gelijk welk type zijn: elementen, attributen, tekst, commentaar, namenruimtes of verwerkingsinstructies.
- *@**: selecteert alle attributen van de huidige context.

Het is ook mogelijk om alleen elementen of attributen uit een bepaalde namenruimte te selecteren. In dat geval maakt ook de afkorting (*prefix*) van de namenruimte deel uit van het pad.

- *prefix:**: selecteert elke elementknoop van een bepaalde namenruimte in de contextknoop.
- *@prefix:**: selecteert alle attributen van een bepaalde namenruimte in de huidige contextknoop.

Meerdere mogelijkheden

Soms wil je meerdere types elementen of attributen identificeren, maar niet alle types. Bijvoorbeeld, je wilt een XSLT-sjabloon maken voor de elementen *isbn* en *title*, maar niet voor de elementen *author*, *firstname*,... In dat geval kan je paden combineren met een rechte streep (*|*). In de volgende voorbeelden worden alle *isbn*- en *title*-elementen geselecteerd.

```
isbn|title

/books/book/isbn|title
```

Speciale tekens

Zoals bij paden in bestandssystemen hebben één punt (.) en twee punten (..) een speciale betekenis voor paden in XPath. Ook twee schuine strepen (//) hebben een speciale betekenis.

De contextknoop selecteren met . Een punt (.) betekent de contextknoop. In XSLT wordt die vaak gebruikt om de waarde van de huidige knoop te bepalen. In het volgend voorbeeld wordt een sjabloon beschreven voor *title*-elementen. De inhoud van de *title*-elementen wordt naar de uitvoer geschreven als kind van het HTML-element *i*. Het zal dus als schuine tekst getoond worden in een browser.

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

Nakomelingen selecteren met // Twee schuine strepen (//) selecteren knopen uit alle nakomelingen van de contextknoop en de contextknoop zelf. In het begin van XPath-uitdrukking selecteert het alle nakomelingen van de basisknoop (*root node*).

De uitdrukking *//book* selecteert alle *book*-elementen in het XML-document. Alle *id*-attributen in een XML-document identificeren kan met de uitdrukking *//@id*. Het volgende voorbeeld selecteert alle *name*-elementen die voorkomen in de huidige knoop (als kind, kleinkind, achterkleinkind, ...)

```
./name
```

Selecteer het ouderelement met .. Twee punten (..) betekenen de ouder van de huidige knoop. In het volgende voorbeeld worden alle elementen met een attribuut *id* geïdentificeerd.

```
//@id/..
```

2.2.6 Voorwaarden

Een XPath-uitdrukking kan meer dan één knoop identificeren. Soms is het nodig om deze selectie nog te verfijnen. Dit kan met een voorwaarde. Elke stap in een pad kan een voorwaarde hebben die de selectie knopen, bepaald door de stap, kan verkleinen. Deze voorwaarde is een logische uitdrukking en wordt gecontroleerd voor elke knoop uit de selectie. Is de voorwaarde niet voldaan (het resultaat is *false*), dan wordt de knoop verwijderd uit de selectie.

In het volgend voorbeeld wordt het basiselement *query* geïdentificeerd waarvan de waarde van het attribuut *zipcode* de waarde 75233 heeft.

```
/query[@zipcode=75233]
```

```
//book[title="XML"]  
//name[.='Ongenae']
```

In de bovenstaande voorbeelden worden respectievelijk alle *book*-elementen geselecteerd waarvan het kindelement *title* de inhoud *XML* heeft en alle *name*-elementen waarvan de inhoud *Ongenae* is.

Merk op dat de waarde van een element de inhoud van het element is en dat voor strings zowel enkele (') als dubbele (") aanhalingstekens gebruikt kunnen worden. Dit is vaak zinvol als een XPath-uitdrukking gebruikt wordt als waarde van een attribuut, die tussen dubbele aanhalingstekens staat.

```
<xsl:apply-templates select="//name[.='Ongenae']" />
```

In de bovenstaande voorbeelden wordt de relationele operator *=* gebruikt. Naast deze operator ondersteunt XPath ook nog de operatoren *<*, *>*, *<=*, *>=*, *=*, *or* en *and*. De uitdrukking *//person[@born<=1976]* identificeert alle *person*-elementen waarvan de numerieke waarde van het *born*-attribuut kleiner is dan of gelijk aan 1976. Merk op dat als je deze uitdrukking gebruikt in een XML-document (bv. XSLT), je het *<*-teken moet vervangen door *<*.

Elke stap in een pad kan een voorwaarde hebben. De uitdrukking *//person[@born<1950]/name[firstname="Alan"]* selecteert eerst alle *person*-elementen waarvan de waarde van het attribuut *born* kleiner is dan 1950. Van al deze *person*-elementen selecteert ze de *name*-elementen die een kindelement *firstname* hebben met inhoud *Alan*.

2.2.7 Lange padnamen

Tot nu toe hebben we verkorte paden gebruikt. Deze zijn makkelijker om te schrijven, korter en vertrouwd. Dit zijn ook de paden die het best zijn voor XSLT. Er bestaat echter ook een niet-verkorte syntax voor paden in XPath. Deze syntax is langer, maar minder cryptisch en flexibeler.

Elke stap in een pad heeft twee vereiste delen, een as en een test om knopen te identificeren, en één optioneel stuk, een voorwaarde. De as bepaalt de richting waarin knopen geselecteerd worden

vanuit de contextknoop. De test bepaalt welke knopen uit die bepaalde richting geïdentificeerd moeten worden. De voorwaarde kan de selectie nog verfijnen.

In een verkort pad worden de as en de knopentest gecombineerd. In de niet-verkorte vorm worden as en test gescheiden door twee dubbele punten (::). De uitdrukkingen

```
books/book/isbn
query/@isbn
```

worden in niet-verkorte vorm

```
child::books/child::book/child::isbn
child::query/attribute::isbn
```

De as *child* bepaalt dat er enkel naar kindelementen van de contextknoop gekeken wordt. Met de as *attribute* worden alle attributen van de huidige knoop geselecteerd. Naast deze twee assen zijn er nog drie die we reeds in verkorte vorm bespraken.

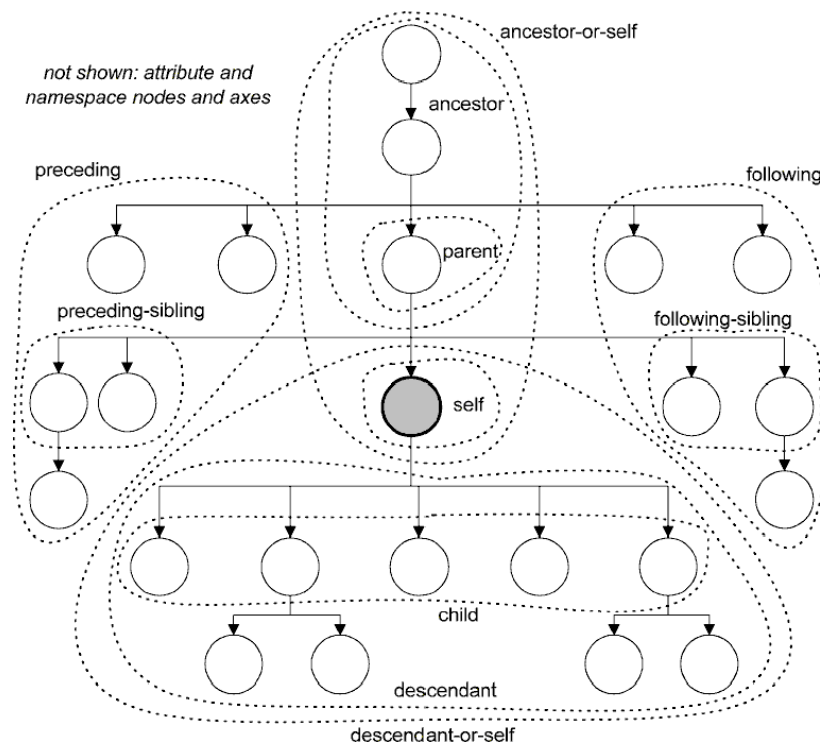
- *parent* (..)
- *self* (.)
- *descendant-or-self* (//)

De andere assen hebben geen verkorte vorm. (zie ook figuur 2.3)

<i>ancestor</i>	Alle voorouders (elementen) van de contextknoop (niet inbegrepen).
<i>ancestor-or-self</i>	Alle voorouders van de contextknoop en de contextknoop zelf.
<i>namespace</i>	Alle namenruimtes in de huidige knoop. Deze namenruimtes kunnen gedefinieerd zijn in de huidige knoop of in één van zijn voorouders.
<i>descendant</i>	Alle nakomelingen van de huidige knoop (kinderen, kinderen van kinderen, ...), behalve de huidige knoop zelf.
<i>following-sibling</i>	Alle knopen na de huidige knoop en kinderen van dezelfde ouder als de contextknoop. Attributen en namenruimtes hebben geen siblings (broers of zussen).
<i>preceding-sibling</i>	Alle knopen voor de huidige knoop en kinderen van dezelfde ouder als de contextknoop. Attributen en namenruimtes hebben geen siblings (broers of zussen).
<i>following</i>	Alle knopen na de huidige knoop, maar geen attributen of namenruimtes.
<i>preceding</i>	Alle knopen voor de huidige knoop, maar geen attributen of namenruimtes.

2.3 Andere XPath-uitdrukkingen

Tot nu toe hebben we ons geconcentreerd op paden. Paden zijn XPath-uitdrukkingen die een verzameling knopen van een XML-documenten selecteren. Naast deze paden zijn er ook XPath-uitdrukkingen die strings, getallen of logische waarden voorstellen. Bijvoorbeeld,



Figuur 2.3: Overzicht assen in XPath

```
@born < 1950
'Ongenae'
75332
@zipcode=75332
```

Deze uitdrukkingen zijn geen verzamelingen van knopen en kunnen dus niet gebruikt worden in het *match*-attribuut van een *xsl:template*-element. Ze kunnen echter wel gebruikt worden als waarde van het *select*-attribuut van een *xsl:value-of*-element of in voorwaarden.

2.3.1 Datatypes

XPath kent drie andere datatypes naast knopen: getallen, strings en logische waarden.

Getallen

Er zijn geen echte gehele getallen in XPath. Alle getallen worden voorgesteld door de IEEE 64-bit codering (equivalent met het type *double* in Java).

Voor dit gegevenstype zijn vijf rekenkundige operatoren beschikbaar: + (optelling), - (verschil), * (product), *div* (deling) en *mod* (rest bij deling). Deze operatoren gedragen zich zoals in Java.

De volgende XSLT-opdracht schrijft de eeuw waarin iemand geboren is naar de uitvoer. De waarde van het attribuut *born* is het geboortejaar.

```
<xsl:value-of select="' '(@born-(@born mod 100)) div 100 + 1'/'>
```

Strings

Strings in XPath bestaan uit reeksen van Unicode-lettertekens. Strings staan tussen enkele of dubbele aanhalingstekens. Deze tekens mogen geen deel uitmaken van de string. Als een string dubbele aanhalingstekens bevat moet hij tussen enkele aanhalingstekens staan en omgekeerd.

Met de operatoren = en != kan je twee strings vergelijken.

Logische waarden

Een logische waarde heeft twee mogelijke statussen: *true* en *false*. Deze twee sleutelwoorden zijn niet beschikbaar in XPath. Wel zijn er twee functies die de rol van deze constanten overnemen: *true()* en *false()*.

Meestal verkrijg je logische waarden als resultaat van een vergelijking tussen twee getallen of strings. Deze kunnen gecombineerd worden met de operatoren *and* en *or*. De negatie wordt gerealiseerd met de functie *not(...)*

Logische waarden komen meestal voor in de voorwaarden van paden en in het *test*-attribuut van sommige XSLT-elementen (bv. *xsl:if*).

2.3.2 Functies

XPath heeft een aantal functies. Deze sectie geeft hiervan een beperkt overzicht. Een meer uitgebreid overzicht vind je op [XPa] Er bestaan geen procedures in XPath. Het resultaat van de functie is van één van de bestaande types: logische waarde, getal, strings of verzameling van knopen.

Knopenfuncties

Met knopenfuncties worden functies bedoeld die ofwel als argument ofwel als resultaat een verzameling van knopen hebben.

- position()* bepaalt het volgnummer van de huidige knoop in de context.
- last()* bepaalt het aantal knopen in de huidige context. (Het volgnummer van de laatste knoop)
- count(...)* telt het aantal knopen in het argument. Het argument is een verzameling knopen.
- id(...)* bepaalt een verzameling knopen. Deze verzameling bevat alle elementen in het XML-document met één van de gespecificeerde ID's. Het argument is een string bestaande uit ID's gescheiden door spaties.

In het onderstaand voorbeeld wordt het sjabloon voor een element *jaar* beschreven. Een jaar kan meerdere *vak*-elementen hebben. Het *xsl:for-each*-element overloopt deze elementen. De functie *position()* geeft het volgnummer van het huidige *vak*-element weer dat afgehandeld wordt door de lus. Het bepaalt met andere woorden het hoeveelste *vak*-element dit element is binnen het *jaar*-element.

```
<xsl:template match="jaar">
  <xsl:for-each select="vak">
    <xsl:value-of value="position()" />
  </xsl:for-each>
</xsl:template>
```

String-functies

XPath bevat een aantal functies voor eenvoudige stringmanipulaties zoals de lengte bepalen, samenvoegen, ...

<i>concat(...,...)</i>	voegt zijn argumenten samen tot één string. Deze functie verwacht tenminste twee argumenten.
<i>contains(...,...)</i>	gaat na of het tweede argument een deel (substring) is van het eerste. Deze functie is hoofdlettergevoelig.
<i>starts-with(...,...)</i>	controleert of het eerste argument begint met het tweede argument.
<i>string(...)</i>	converteert het argument naar een string.
<i>string-length(...)</i>	bepaalt de lengte van het argument.

Logische functies

<i>true()</i>	geeft steeds de waarde <i>true</i> terug. De symbolische constante <i>true</i> bestaat niet in XPath.
<i>false()</i>	geeft steeds de waarde <i>false</i> terug. De symbolische constante <i>false</i> bestaat niet in XPath.
<i>not(...)</i>	neemt de negatie van het argument. <i>true</i> wordt <i>false</i> en omgekeerd.
<i>boolean(...)</i>	converteert het argument naar een logische waarde. Alle getallen behalve 0 en NaN worden <i>true</i> . Lege verzamelingen van knopen worden <i>false</i> . Alle strings behalve de lege string zijn <i>true</i> .

Numerieke functies

XPath voorziet een aantal eenvoudige numerieke functies.

<i>ceiling(...)</i>	bepaalt de kleinste gehele waarde groter dan of gelijk aan het argument.
<i>floor(...)</i>	bepaalt de grootste gehele waarde kleiner dan of gelijk aan het argument.
<i>number(...)</i>	converteert het argument naar een getal.
<i>round(...)</i>	bepaalt de gehele waarde dichtst bij het argument.

Hoofdstuk 3

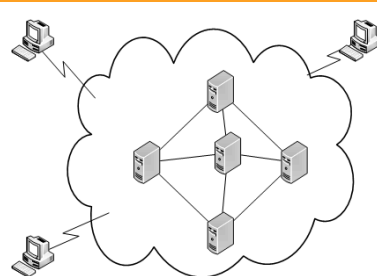
Webservices

WEBSERVICES

Veerle Ongenae

Gedistribueerde applicaties

2



<https://msdn.microsoft.com/en-us/library/cc239737.aspx>
Industrieel Ingenieur Informatica, Universiteit Gent

Gedistribueerde applicaties


3

- EDI

Industrieel Ingenieur Informatica, Universiteit Gent

EDI

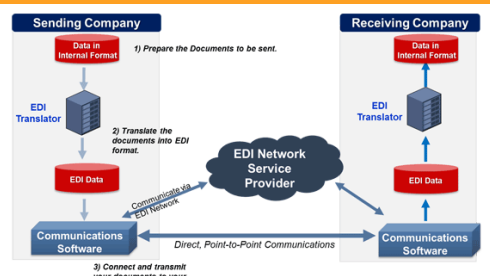
4



http://www.scaoneurope.nl/wp/?page_id=396
Industrieel Ingenieur Informatica, Universiteit Gent

EDI - architectuur

5



1) Prepare the Documents to be sent.
2) Translate the documents into EDI format.
3) Connect and transmit your documents to your business partner.

<http://www.edibasics.com/what-is-edi/how-does-edi-work/>
Industrieel Ingenieur Informatica, Universiteit Gent

Gedistribueerde applicaties

6

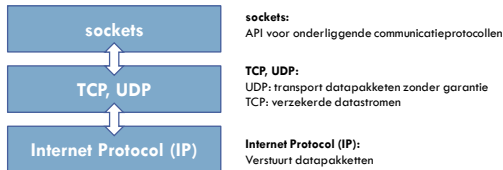
- EDI
- RPC

Industrieel Ingenieur Informatica, Universiteit Gent

Middleware

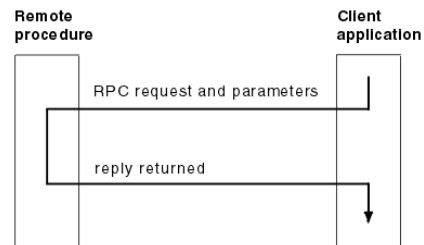
Softwarelaag

- Abstractie en afschermen
 - Netwerk, hardware, OS, ...
- Uniforme interface



Industrieel Ingenieur Informatica, Universiteit Gent

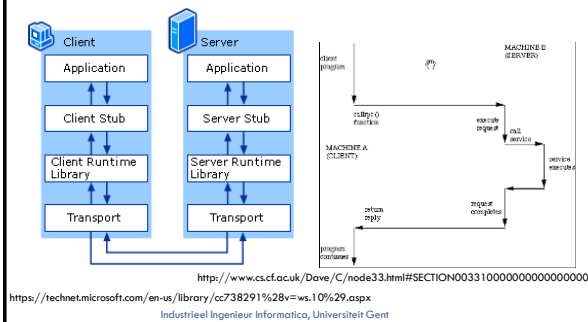
RPC



http://www-01.ibm.com/support/knowledgecenter/SSGMCP_4.1.0/com.ibm.cics.ts.doc/dftm/topics/dftm00188.html

Industrieel Ingenieur Informatica, Universiteit Gent

Werking RPC

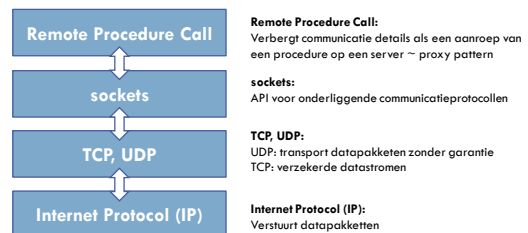


<https://www.cs.cit.ac.uk/Dave/C/node33.html#SECTION00033100000000000000>

<https://technet.microsoft.com/en-us/library/cc738291%28v=ws.10%29.aspx>

Industrieel Ingenieur Informatica, Universiteit Gent

Communicatie tussen applicaties



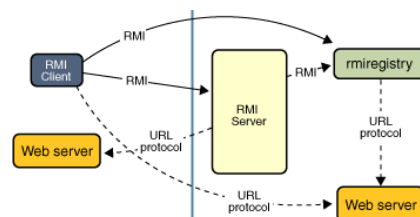
Industrieel Ingenieur Informatica, Universiteit Gent

Gedistribueerde applicaties

- EDI
- RPC
- RMI, DCOM, CORBA

Industrieel Ingenieur Informatica, Universiteit Gent

RMI

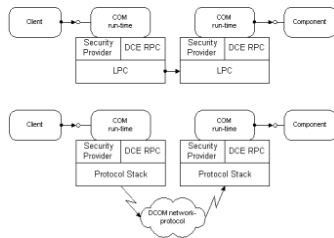


<https://docs.oracle.com/javase/tutorial/rmi/overview.html>

Industrieel Ingenieur Informatica, Universiteit Gent

DCOM

13

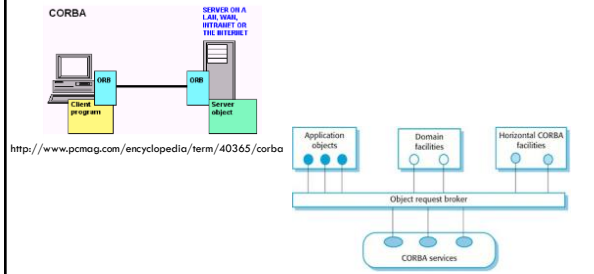


<http://www.active-undelete.com/dcom-overview.htm>

Industrieel Ingenieur Informatica, Universiteit Gent

CORBA

14



<http://www.pcmag.com/encyclopedia/term/40365/corba>

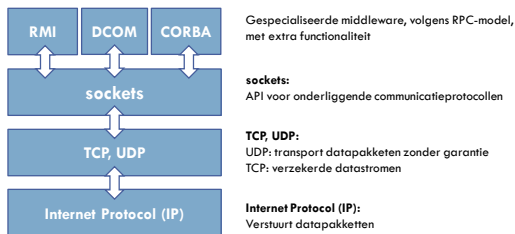
<https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/DistribSys/Corba.html>

Industrieel Ingenieur Informatica, Universiteit Gent

Communicatie tussen applicaties

15

- Gedistribueerd programmeren populairder → nood aan extra functionaliteit en flexibiliteit



Gespecialiseerde middleware, volgens RPC-model, met extra functionaliteit

sockets:
API voor onderliggende communicatieprotocollen

TCP, UDP:
UDP: transport datapakketten zonder garantie
TCP: verzekerde datastromen

Internet Protocol (IP):
Verstuurt datapakketten

Industrieel Ingenieur Informatica, Universiteit Gent

Gedistribueerde applicaties

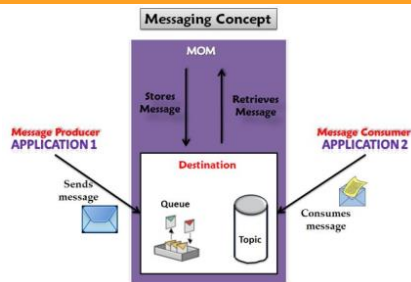
16

- EDI
- RPC
- RMI, DCOM, CORBA
- Message-Oriented Middleware (MOM)

Industrieel Ingenieur Informatica, Universiteit Gent

Message-Oriented Middleware

17



<http://theopentutorials.com/tutorials/java-ee/ejb3/mdb/message-driven-beans-introduction/>

Industrieel Ingenieur Informatica, Universiteit Gent

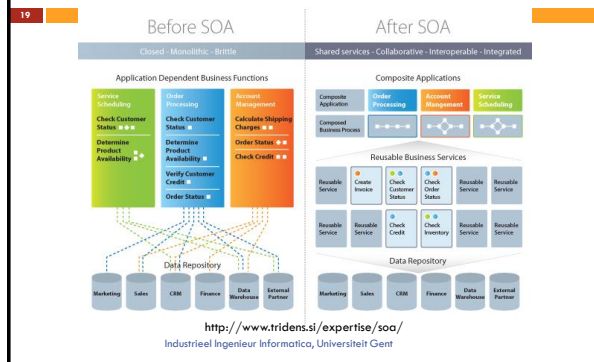
Gedistribueerde applicaties

18

- EDI
- RPC
- RMI, DCOM, CORBA
- Message-Oriented Middleware (MOM)
- Service-Oriented Architecture (SOA)

Industrieel Ingenieur Informatica, Universiteit Gent

Service Oriented Architecture (SOA)



SOA - kenmerken

- 20
- ☐ Logical view
 - ☐ Message orientation
 - ☐ Description orientation
 - ☐ Granularity
 - ☐ Network orientation
 - ☐ Platform neutral
- Industrieel Ingenieur Informatica, Universiteit Gent

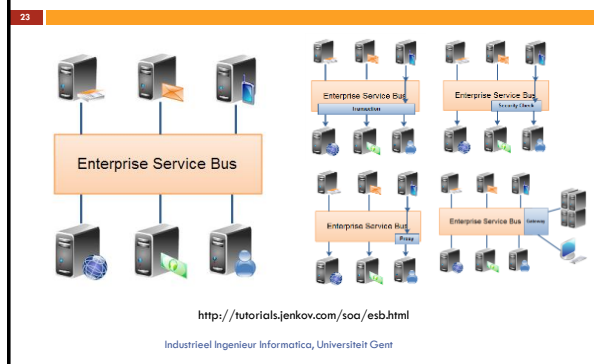
Uitdagingen en problemen SOA

- 21
- ☐ Traagheid en (on)betrouwbaarheid gebruikte transport
 - ☐ Geen gedeeld geheugen voor requester en provider
 - ☐ Wat indien een deel van de opdracht mislukt?
 - ☐ Gelijktijdige toegang tot bronnen
 - ☐ Wat als één partner incompatibel wordt?
- Industrieel Ingenieur Informatica, Universiteit Gent

COM/CORBA/... versus webservices

- 22
- ☐ Webservices
 - ☐ Platformonafhankelijk
 - ☐ Over internet
 - ☐ Snelheid minder belangrijk
 - ☐ Bestaande applicatie openstellen via internet
 - ☐ COM/CORBA/...
 - ☐ Performanter
- Industrieel Ingenieur Informatica, Universiteit Gent

Enterprise Service Bus (ESB)



Gedistribueerde applicaties

- 24
- ☐ EDI
 - ☐ RPC
 - ☐ RMI, DCOM, CORBA
 - ☐ Message-Oriented Middleware (MOM)
 - ☐ Service-Oriented Architecture (SOA)
 - ☐ Webservices
 - ☐ Definitie
- Industrieel Ingenieur Informatica, Universiteit Gent

Webservice - definitie

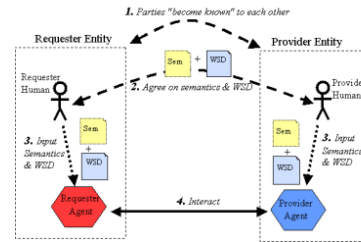
- Software service (dienst)
- Toegankelijk via internet
 - ▣ Via bestaande protocollen bv. HTTP



<http://tutorials.jenkov.com/web-services/message-formats.html>

Industrieel Ingenieur Informatica, Universiteit Gent

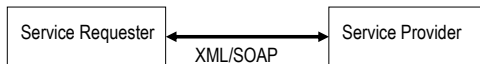
Webservice – entities



<http://www.w3.org/TR/ws-arch/>

Industrieel Ingenieur Informatica, Universiteit Gent

Webservice – andere terminologie



Industrieel Ingenieur Informatica, Universiteit Gent

Gedistribueerde applicaties

- EDI
- RPC
- RMI, DCOM, CORBA
- Message-Oriented Middleware (MOM)
- Service-Oriented Architecture (SOA)
- Webservices
 - ▣ Definitie
 - ▣ Technologieën
 - SOAP

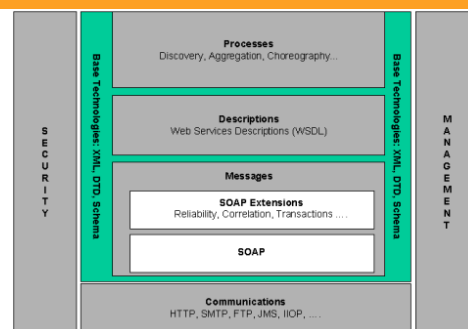
Industrieel Ingenieur Informatica, Universiteit Gent

Twee types webservices

- RESTfull webservices
 - ▣ Roy Fielding
 - ▣ JAX-RS (Java API for RESTful Web Services)
 - ▣ Web API (.NET)
- SOAP webservices

Industrieel Ingenieur Informatica, Universiteit Gent

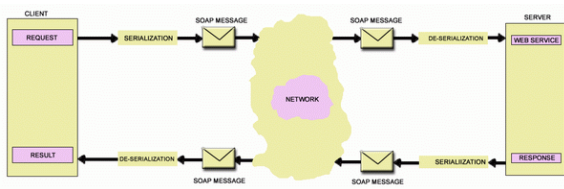
Technologieën



<http://www.w3.org/TR/ws-arch/>

SOAP webservices

31



<http://www.codeproject.com/Articles/6399/Securing-web-services-using-SOAP-extensions>
Industriële Ingenieur Informatica, Universiteit Gent

Voorbeeld SOAP-aanvraag

32

POST /ZwiftBooks HTTP/1.1
Host: www.zwiftbooks.com
Content-Type: text/xml
Content-Length: 134
SOAP Action: "Some-URI"

```
<Envelope xmlns="http://www.w3.org/2003/05/soap-envelope"
  encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  <Body>
    <zwiftbooks:GetGuaranteedDeliveryTime
      xmlns:zwiftbooks="http://www.zwiftbooks.com">
      <zwiftbooks:isbn>0-13-18188-222</zwiftbooks:isbn>
      <zwiftbooks:zipcode>75240</zwiftbooks:zipcode>
    </zwiftbooks:GetGuaranteedDeliveryTime>
  </Body>
</Envelope>
```

Industriële Ingenieur Informatica, Universiteit Gent

Voorbeeld SOAP-antwoord

33

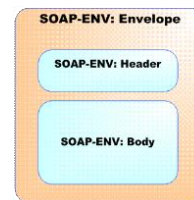
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 122

```
<Envelope xmlns="http://www.w3.org/2003/05/soap-envelope">
  <Body>
    <zwiftbooks:GetBestDeliveryTimeResponse
      xmlns:zwiftbooks="http://www.zwiftbooks.com">
      <zwiftbooks:Time>2 hours</zwiftbooks:Time>
    </zwiftbooks:GetBestDeliveryTimeResponse>
  </Body>
</Envelope>
```

Industriële Ingenieur Informatica, Universiteit Gent

Structuur SOAP Bericht

34



<https://mytechprep.wordpress.com/tag/soap-actor/>
Industriële Ingenieur Informatica, Universiteit Gent

Voorbeeld: SOAP-hoofding

35

```
<header>
  <n:alertcontrol
    xmlns:n="http://example.org/alertcontrol">
    <n:priority>1</n:priority>
    <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
</header>
```

Industriële Ingenieur Informatica, Universiteit Gent

SOAP-berichtenpad

36



<https://myshadesofgray.wordpress.com/category/programming/java/>
Industriële Ingenieur Informatica, Universiteit Gent

Voorbeeld: SOAP-hoofding

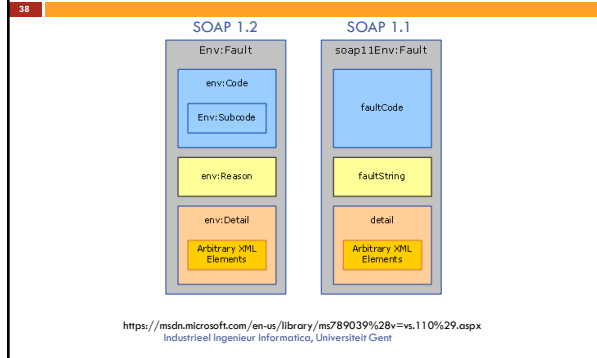
```

37 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Áke Jógvan Øyvind</n:name>
    </n:passenger>
  </env:Header>
  <env:Body>...</env:Body></env:Envelope>

```

Industrieel Ingenieur Informatica, Universiteit Gent

SOAP-fouten



Voorbeeld SOAP-fouten

```

39 <?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>rpc:BadArguments</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Processing error</env:Text>
      </env:Reason>
      <env:Detail>
        <e:myFaultDetails xmlns:e="http://travelcompany.example.org/faults">
          <e:message>Name does not match card number</e:message>
          <e:errorCode>999</e:errorCode>
        </e:myFaultDetails>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>

```

Industrieel Ingenieur Informatica, Universiteit Gent

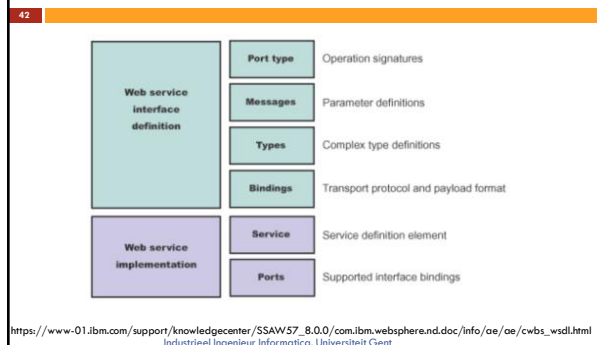
Gedistribueerde applicaties

- 40
- EDI
 - RPC
 - RMI, DCOM, CORBA
 - Message-Oriented Middleware (MOM)
 - Service-Oriented Architecture (SOA)
 - Webservices
 - Definitie
 - Technologieën
 - SOAP
 - WSDL
- Industrieel Ingenieur Informatica, Universiteit Gent

WSDL

- 41
- Web Service Description Language
 - XML-standaard W3C
 - Twee versies
 - WSDL 1.1 (<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>)
 - WSDL 2.0 (<http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/>)
- Industrieel Ingenieur Informatica, Universiteit Gent

WSDL 1.1



Voorbeeld WSDL 1.1

43

```
<?xml version="1.0"?>
<definitions name="StockQuote"
  targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="TradePriceRequest">
        <complexType base="xsd:string"/>
      </element>
      <element name="TradePrice">
        <complexType base="xsd:string"/>
      </element>
    </schema>
  </types>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Voorbeeld WSDL 1.1

44

```
<message name="GetLastTradePriceInput">
  <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
  <part name="body" element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
  </operation>
</portType>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Voorbeeld WSDL 1.1

45

```
<binding name="StockQuoteSoapBinding"
  type="tns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation
      soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Voorbeeld WSDL 1.1

46

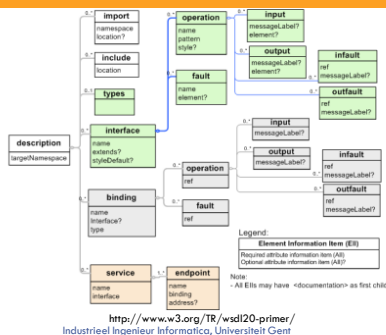
```
<service name="StockQuoteService">
  <documentation>My first service</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
  </port>
</service>

</definitions>
```

Industrieel Ingenieur Informatica, Universiteit Gent

WSDL 2.0 -structuur

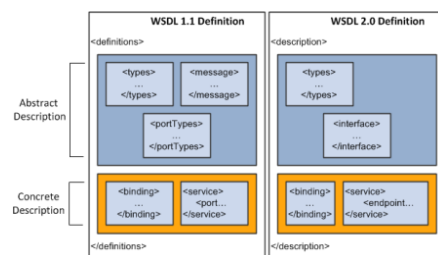
47



Industrieel Ingenieur Informatica, Universiteit Gent

WSDL 1.0 versus 2.0

48



http://docs.oracle.com/cd/E41633_01/pr853pbh1/eng/pt/tlbr/concept_UnderstandingProvidingWSDLDocuments-076201.htm

Industrieel Ingenieur Informatica, Universiteit Gent

Voorbeeld WSDL 2.0

49

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/ns/wsd1"
  targetNamespace="http://greath.example.com/2004/wsd1/resSvc"
  xmlns:tns="http://greath.example.com/2004/wsd1/resSvc" ...
</description>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Voorbeeld WSDL 2.0

50

```
<description ...> ...
<types>
  <xs:schema ...>
    <xs:element name="checkAvailability" type="tCheckAvailability"/>
    <xs:complexType name="tCheckAvailability">
      <xs:sequence>
        <xs:element name="checkInDate" type="xs:date"/>
        <xs:element name="checkOutDate" type="xs:date"/>
        <xs:element name="roomType" type="xs:string"/>
      </xs:sequence> </xs:complexType>
    <xs:element name="checkAvailabilityResponse" type="xs:double"/>
    <xs:element name="invalidDataError" type="xs:string"/>
  </xs:schema>
</types> ...
</description>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Voorbeeld WSDL 2.0

51

```
<description ...> ... <types> ... </types>
<interface name="reservationInterface" >
  <fault name="invalidDataFault" element="ghns:invalidDataError"/>
  <operation name="opCheckAvailability"
    pattern="http://www.w3.org/ns/wsd1/in-out"
    style="http://www.w3.org/ns/wsd1/style/iri" wsdl:safe="true">
    <input messageLabel="In" element="ghns:checkAvailability" />
    <output messageLabel="Out"
      element="ghns:checkAvailabilityResponse" />
    <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
  </operation>
</interface> ... </description>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Voorbeeld WSDL 2.0

52

```
<description ...> ... <types> ... </types>
<interface name="reservationInterface" > ... </interface>
<binding name="reservationSOAPBinding"
  interface="tns:reservationInterface"
  type="http://www.w3.org/ns/wsd1/soap"
  soap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
  <operation ref="tns:opCheckAvailability"
    soap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
  <fault ref="tns:invalidDataFault" soap:code="soap:Sender"/>
</binding>
...
</description>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Voorbeeld WSDL 2.0

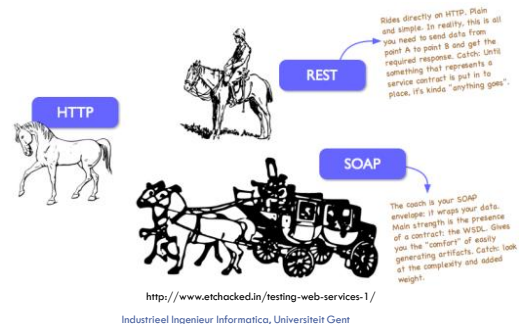
53

```
<?xml version="1.0" encoding="utf-8" ?>
<description ...> ... <types> ... </types>
<interface name="reservationInterface" > ... </interface>
<binding name="reservationSOAPBinding"
  interface="tns:reservationInterface" ...> ... </binding>
<service name="reservationService"
  interface="tns:reservationInterface">
  <endpoint name="reservationEndpoint"
    binding="tns:reservationSOAPBinding"
    address="http://greath.example.com/2004/reservation"/>
  </service>
</description>
```

Industrieel Ingenieur Informatica, Universiteit Gent

REST ↔ SOAP

54



Informatie

55

□ Boek

- "Distributed Systems Concepts and Design" Coulouris, Dollimore and Kindberg. Addison and Wesley, 2012
- "SOA in Practice", Nicloai M. Josuttis, O'Reilly, 2007
- "Introduction to Middleware" Letha H. Etzkorn, Chapman & Hall, 20170612. VitalBook file.

Industrieel Ingenieur Informatica, Universiteit Gent

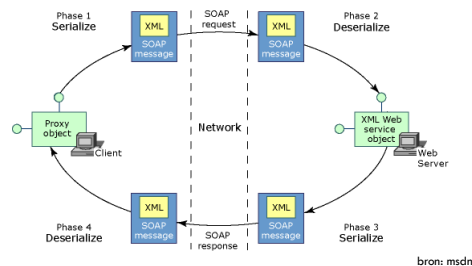
3.1 Webservices in Java



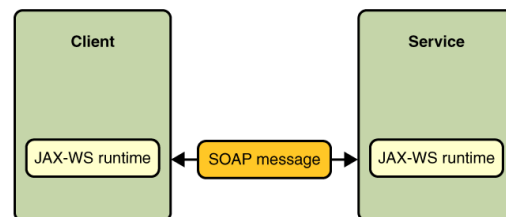
Informatie over webservices

- The JavaEE 7 Tutorial Part VI Web Services
(<https://docs.oracle.com/javaee/7/tutorial/partwebsvcs.htm#BNAYK>)
- Webservices in Netbeans
(<http://www.netbeans.org/kb/trails/web.html>)

Overzicht



JAX-WS Principe



Voorbeeld

- Boek-object opvragen aan de hand van ISBN

Web Services	
Endpoint	Information
Service Name: (http://web.boeken.iii.be/) Catalogus	Address: http://localhost:8080/WebCatalogus/Catalogus
Port Name: (http://web.boeken.iii.be/) CatalogusPort	WSDL: http://localhost:8080/WebCatalogus/Catalogus?wsdl
Implementation class: be.iii.boeken.web.Catalogus	

Catalogus Web Service Tester
This form will allow you to test your web service implementation (WSDL File)
To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.
Methods : public abstract be.iii.boeken.web.Boek be.iii.boeken.web.Catalogus.geefBoek(java.lang.String) :geefBoek (isbn2)
http://localhost:8080/WebCatalogus/Catalogus?wsdl

SOAP-aanvraag

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:geefBoek xmlns:ns2="http://web.boeken.iii.be/">
      <isbn>isbn2</isbn>
    </ns2:geefBoek>
  </S:Body>
</S:Envelope>
```

SOAP-antwoord

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:geefBoekResponse xmlns:ns2="http://web.boeken.iii.be/">
      <return>
        <isbn>isbn2</isbn>
        <prijs>36.0</prijs>
        <titel>Java Servlet Programming</titel>
      </return>
    </ns2:geefBoekResponse>
  </S:Body>
</S:Envelope>
```

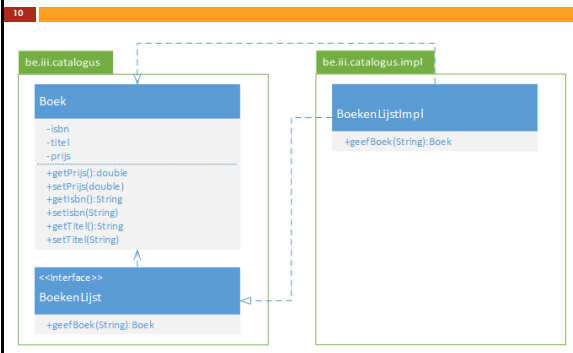
WSDL

- WSDL: Webservice Description Language
- <http://localhost:8080/WebCatalogus/Catalogus?WSDL>
- <http://localhost:8080/WebCatalogus/Catalogus?xsd=1>

Werkwijze in J2EE

- Maken webservice
 - Klassen, interface, ... met eigenlijke functionaliteit maken (bo en jdbc)
 - Webservice maken
 - Webservice publiceren
- Clientapplicatie
 - Op basis van WSDL proxy-klassen genereren (wsimport.exe)
 - Clientapplicatie maakt gebruik van proxy-klassen/objecten

Eigenlijke functionaliteit



Webservice

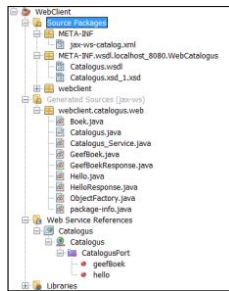
```
@WebService(serviceName = "Catalogus")
public class Catalogus {
    @WebMethod(operationName = "geefBoek")
    public Boek geefBoek(@WebParam(name = "isbn") String isbn) {
        try {
            BoekenLijst boeken = new BoekenLijstImpl();
            return boeken.geefBoek(isbn);
        } catch (Exception e) {
            return null;
        }
    }
}
```

Webservice publiceren

- Webservice deployen
 - War-bestand
 - Publiceren webservice
 - Wordt gewrap in een servlet
- WSDL automatisch gegenereerd
 - Wsdl af te halen via <http://localhost:8080/WebCatalogus/Catalogus?wsdl>
 - ws-gen: tool om dit handmatig te doen

Genereren proxy-klassen

13



WebClient

Java Servlet Programming
Java Server Programming

14

```
public static void main(String[] args) {
    for (int i = 2; i <= 3; i++) {
        String isbn = Integer.toString(i);
        Boek boek = geefBoek("isbn"+isbn);
        System.out.println(boek.getTitel());
    }
}

private static Boek geefBoek(String isbn) {
    webclient.catalogus.web.Catalogus_Service service
        = new webclient.catalogus.web.Catalogus_Service();
    webclient.catalogus.web.Catalogus port
        = service.getCatalogusPort();
    return port.geefBoek(isbn);
}
```

WebClientServlet

15

```
import be.iii.boeken.web.Catalogus_Service;
...
import javax.xml.ws.WebServiceRef;

@WebServlet(urlPatterns = {"/BoekServlet"})
public class BoekServlet extends HttpServlet {

    @WebServiceRef(wsdlLocation =
        "http://localhost:8080/WebCatalogus/Catalogus?WSDL")
    Catalogus_Service service;

    protected void processRequest(...) throws ... {
        ...
        Catalogus port = service.getCatalogusPort();
        Boek boek = port.geefBoek(isbn); ...}
}
```

3.2 Webservices in C#

WINDOWS COMMUNICATION FOUNDATION (WCF)

Veerle Ongenaë

WCF - overzicht

2

- Wat is WCF
- Architectuur WCF
 - Contracten, Gedrag, Berichten, Hosting
- Ontwikkeling WCF
 - Ontwerp servicecontract
 - Implementatie contract
 - Configuratie
 - Hosting
 - Ontwikkelen client

Industrieel Ingenieur Informatica, UGent

WCF

3

- Gedistribueerde applicaties
 - Communicatie tussen applicaties
 - Op dezelfde computer
 - Over een intranet
 - Over het internet
- Service-oriented applications (~SOA)
- Webservices
 - SOAP
 - REST
- Transport over
 - Named pipes
 - TCP
 - HTTP
 - MSMQ
- Berichten
 - Binair
 - Tekst

Industrieel Ingenieur Informatica, UGent

WCF Concepten

4

- Client-Services
- Communicatie
 - Berichten
 - Tussen "endpoints"
- Service
 - Bestaat uit verschillende "endpoints"
- Endpoint
 - Waarnaar berichten sturen
 - Hoe berichten sturen
 - Structuur berichten

Industrieel Ingenieur Informatica, UGent

WCF - overzicht

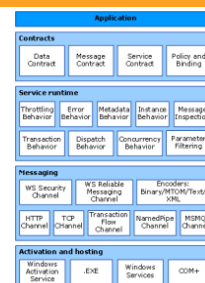
5

- Wat is WCF
- Architectuur WCF
 - Contracten, Gedrag, Berichten, Hosting
- Ontwikkeling WCF
 - Ontwerp servicecontract
 - Implementatie contract
 - Configuratie
 - Hosting
 - Ontwikkelen client

Industrieel Ingenieur Informatica, UGent

WCF-architectuur

6



<https://msdn.microsoft.com/en-us/library/ms733128%28v=vs.110%29.aspx>

Industrieel Ingenieur Informatica, UGent

WCF Architectuur - Contracten

7

- Data Contract
 - Parameters in bericht
 - XSD
- Message Contract
 - Structuur bericht
- Service Contract
 - Beschikbare interface
 - Signatuur methodes
- Bindings, Policies
 - Hoe communiceren met service?
 - Beveiliging?

Industrieel Ingenieur Informatica, UGent

WCF Architectuur – gedrag

8

- Service runtime
- Eigenlijke gedrag applicatie
 - Aantal berichten dat verwerkt kan worden
 - Wat bij fouten?
 - Welke metadata beschikbaar voor de buitenwereld?
 - Hoeveel instanties tegelijk van service?
 - Transacties
 - ...

Industrieel Ingenieur Informatica, UGent

WCF Architectuur - Berichten

9

- Verschillende kanalen verwerken berichten
 - Zowel hoofdingen als berichten
 - Bv. Kanaal voor authenticatie
- Twee type kanalen
 - Transport
 - Lezen en schrijven van berichten van en naar het netwerk
 - Protocol
 - Lezen en schrijven van hoofdingen
 - ...

Industrieel Ingenieur Informatica, UGent

WCF Architectuur - Hosting

10

- Een service is een programma
- Moet uitgevoerd worden
 - Als executable
 - Self-hosted service
 - Gehost in een externe omgeving
 - Webserver
 - Als Windows service

Industrieel Ingenieur Informatica, UGent

Voorbeeld

11

- Service op IIS
 - tijdService
 - tijdClient
 - tijdWebServiceImpl
- Service als console-applicatie
 - tijdServer
 - toonTijdVanConsole (Client)

Industrieel Ingenieur Informatica, UGent

Overzicht

12

- Ontwerp servicecontract
- Implementatie contract
- Configuratie
- Hosting
- Ontwikkelen client

Industrieel Ingenieur Informatica, UGent

Ontwerp servicecontract

13

- Service – dienst
 - Bestaat uit operations – opdrachten
 - Methode
 - AttribootOperationContract
 - Parameters en teruggeefwaarde
 - Serialiseerbaar
 - Geen referenties, maar kopies
 - Eigen datatypes
 - Contract
 - AttribootDataContract
 - AttribootDataMember
 - Interface
 - AttribootServiceContract
- Namespace: System.ServiceModel;

Industrieel Ingenieur Informatica, UGent

Servicecontract

14

□ Interface met attributen

```
[ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]
public interface ICalculator {
    [OperationContract]
    double Add(double n1, double n2);
    [OperationContract]
    double Subtract(double n1, double n2);
    [OperationContract]
    double Multiply(double n1, double n2);
    [OperationContract]
    double Divide(double n1, double n2);
}
```

Industrieel Ingenieur Informatica, UGent

Type communicatie

15

- Vraag/antwoord
 - Standaard
- Eénrichtingscommunicatie (one-way contract)
 - Client → server
 - Client verwacht geen antwoord
 - Methode zonder returnwaarde
 - Geen uitvoerparameters
- Tweerichtingscommunicatie (duplex contract)
 - Client → server
 - Client verwacht geen antwoord
 - Server → client
 - Events op client
 - Informatie opvragen van client

Industrieel Ingenieur Informatica, UGent

Eénrichtingscommunicatie

16

□ Methode

- Geen returnwaarde
- Geen uitvoerparameters
- AttribootOperationContract
 - Eigenschap IsOneWay op true

```
[OperationContract(IsOneWay=true)]
void Hello(string greeting);
```

Industrieel Ingenieur Informatica, UGent

Duplex communicatie

17

- Tweerichtingsverkeer
 - Geen vraag/antwoord
 - Communicatie kan in beide richtingen
 - Client start communicatie op (sessie)
 - Houdt kanaal open
 - Server kan later antwoorden
- Twee interfaces
 - Client → Server
 - Server → Client
 - Call back contract

Industrieel Ingenieur Informatica, UGent

Duplex communicatie - syntax

18

```
[ServiceContract(Namespace =
    "http://Microsoft.ServiceModel.Samples",
    SessionMode=SessionMode.Required,
    CallbackContract=typeof(ICalculatorDuplex
    Callback))]
public interface ICalculatorDuplex {
    [OperationContract(IsOneWay = true)]
    void Clear();
    [OperationContract(IsOneWay = true)]
    void AddTo(double n);
    [OperationContract(IsOneWay = true)]
    void SubtractFrom(double n);
    [OperationContract(IsOneWay = true)]
    void MultiplyBy(double n);
    [OperationContract(IsOneWay = true)]
    void DivideBy(double n);
}
```

```
public interface ICalculatorDuplexCallback
{
    [OperationContract(IsOneWay = true)]
    void Result(double result);
    [OperationContract(IsOneWay = true)]
    void Equation(string eqn);
}
```

Implementatie op client

Implementatie op server

Industrieel Ingenieur Informatica, UGent

Overzicht

19

- Ontwerp servicecontract
- Implementatie contract
- Configuratie
- Hosting
- Ontwikkelen client

Industrieel Ingenieur Informatica, UGent

Implementatie servicecontract

20

□ Klasse die interface implementeert

```
[ServiceContract]
public interface IMath {
    [OperationContract]
    double Add(double A, double B);
    [OperationContract]
    double Multiply (double A, double B);
}

public class MathService : IMath {
    public double Add (double A, double B) { return A + B; }
    public double Multiply (double A, double B) { return A * B; }
}
```

Industrieel Ingenieur Informatica, UGent

Overzicht

21

- Ontwerp servicecontract
- Implementatie contract
- Configuratie
- Hosting
- Ontwikkelen client

Industrieel Ingenieur Informatica, UGent

Configuratie

22

- Specificiëren endpoint
 - Welke service?
 - Interface
 - Implementatie
 - Waar?
 - URL
 - Hoe communiceren? (binding)
 - TCP? HTTP?
 - Tekst? Binair?
 - Beveiligd?
- Configuratie = belangrijk deel WCF-programmatie

Industrieel Ingenieur Informatica, UGent

Specifiëren endpoint

23

- Meestal in configuratie
- Kan ook in code

Industrieel Ingenieur Informatica, UGent

Specifiëren endpoint in code

24

```
Uri baseAddress = new Uri("http://localhost:8000/ServiceModelSamples/Service");
// Adres

ServiceHost ServiceHost selfHost = new ServiceHost(typeof(CalculatorService),
baseAddress);
// Implementatie service

try {
    selfHost.AddServiceEndpoint( typeof(ICalculator), new WSHttpBinding(),
    "CalculatorService");
    // Welke service? Binding
    ... // gedrag instellen, service runnen
} catch (CommunicationException ce) { Console.WriteLine("An exception occurred:
{0}", ce.Message); selfHost.Abort();
}
```

Industrieel Ingenieur Informatica, UGent

Specifiëren endpoint in configuratie

25

```
<system.ServiceModel>
  <services>
    <service name="Microsoft.ServiceModel.Samples.CalculatorService">
      <binding>
        <endpoint address="http://localhost:8000/ServiceModelSamples/Service"
          binding="basicHttpBinding" bindingConfiguration="myBindingConfiguration1"
          contract="Microsoft.ServiceModel.Samples.ICalculator" />
      </binding>
    </service>
  </services>
  <bindings>
    <basicHttpBinding>
      <binding name="myBindingConfiguration1" closeTimeout="00:01:00" />
    </basicHttpBinding>
  </bindings>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceMetadata httpGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.ServiceModel>
```

Klassenaam implementatie
URL
Binding
Interface
Opvragen metadata service
Gedrag service

Industrieel Ingenieur Informatica, UGent

Bestaande bindingen

- 26
- Voorzien in WCF
 - Een aantal voorbeelden
 - BasicHttpBinding
 - HTTP
 - Tekst/XML
 - Bv. Asmx
 - WSHttpBinding
 - Beveiligd, niet-duplex
 - NetTcpBinding
 - Beveiligd, communicatie tussen WCF-applicaties op verschillende machines
 - NetNamedPipeBinding
 - Beveiligd, geoptimaliseerd tussen WCF-applicaties op één machine

Industrieel Ingenieur Informatica, UGent

Overzicht

- 27
- Ontwerp servicecontract
 - Implementatie contract
 - Configuratie
 - Hosting
 - Ontwikkelen client

Industrieel Ingenieur Informatica, UGent

Hosting

- 28
- Publiceren
 - Runtime omgeving
 - Beheren levensloop en context
 - Windows proces
 - Ondersteunen managed code
 - Voorbeeld
 - IIS
 - Console-applicatie (zelf "hosten")
 - Windows service
 - ...
 - Code service onafhankelijk van host

Industrieel Ingenieur Informatica, UGent

Hosten via console-applicatie

29

```
Uri baseAddress = new Uri("http://localhost:8000/ServiceModelSamples/Service");
using (ServiceHost host = new ServiceHost(typeof(CalculatorService), baseAddress)) {

    host.AddServiceEndpoint(typeof(ICalculator), new WSHttpBinding(), "CalculatorService");
    ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
    smb.HttpGetEnabled = true;
    host.Description.Behaviors.Add(smb);

    host.Open();
    Console.WriteLine("The service is ready at {0}", baseAddress);
    Console.WriteLine("Press <Enter> to stop the service.");
    Console.ReadLine();

    host.Close();
}
```

Industrieel Ingenieur Informatica, UGent

Hosten als Windows service

30

```
public class CalculatorWindowsService : ServiceBase {
    public ServiceHost serviceHost = null;
    public CalculatorWindowsService() {
        ServiceName = "WCFWindowsServiceSample";
    }

    public static void Main() {
        ServiceBase.Run(new CalculatorWindowsService());
    }

    protected override void OnStart(string[] args) {
        if (serviceHost != null) { serviceHost.Close(); }
        serviceHost = new ServiceHost(typeof(CalculatorService));
        serviceHost.Open();
    }

    protected override void OnStop() {
        if (serviceHost != null) {
            serviceHost.Close();
            serviceHost = null;
        }
    }
}

[RunInstaller(true)]
public class ProjectInstaller : Installer {
    private ServiceProcessInstaller process;
    private ProjectInstaller service;
    public ProjectInstaller() {
        process = new ServiceProcessInstaller();
        process.Account = ServiceAccount.LocalSystem;
        service = new ServiceInstaller();
        service.ServiceName = "WCFWindowsServiceSample";
        Installers.Add(process);
        Installers.Add(service);
    }
}
```

Industrieel Ingenieur Informatica, UGent

Hosten in IIS

31

- Maak een service in IIS
 - TijdService.svc en TijdService.svc.cs
- Publiceer/Run
- WSDL
 - <http://localhost:49695/TijdService.svc?wsdl>

Industrieel Ingenieur Informatica, UGent

Overzicht

32

- Ontwerp servicecontract
- Implementatie contract
- Configuratie
- Hosting
- Ontwikkelen client

Industrieel Ingenieur Informatica, UGent

Client ontwikkelen

33

- Ophalen
 - Contract
 - Bindings
 - Adres
- Svcutil.exe
 - svcutil.exe /language:cs /out:generatedProxy.cs /config:app.config
<http://localhost:8000/ServiceModelSamples/service>
 - Genereert
 - Configuratiebestand
 - Proxy-klasse
 - Beiden toevoegen aan clientproject
 - Kan ook via WSDL indien gehost op IIS

Industrieel Ingenieur Informatica, UGent

Proxy gebruiken

34

- Proxy aanmaken
- Methodes proxy oproepen
- Proxy afsluiten

Industrieel Ingenieur Informatica, UGent

Proxy gebruiken

35

```
class Client {
    static void Main() {
        CalculatorClient client = new CalculatorClient();
        double value1 = 100.00D;
        double value2 = 15.99D;
        double result = client.Add(value1, value2);

        client.Close();
    }
}
```

Industrieel Ingenieur Informatica, UGent

Informatie

37

- WCF (<http://msdn.microsoft.com/en-us/library/dd456779.aspx>)

Industrieel Ingenieur Informatica, UGent

3.3 Externe API's

Hoofdstuk 4

Netwerkprogrammatie in Java

4.1 Inleiding

In client-serverapplicaties communiceren programma's met elkaar via netwerkverbindingen. Hierbij wordt gebruikt gemaakt van TCP (Transmission Control Protocol). Aan elke kant van de netwerkverbinding binden de programma's, die met elkaar willen communiceren, een socket. De programma's schrijven naar en lezen van die socket om met elkaar gegevens uit te wisselen. De Java API voorziet het pakket *java.net* om op een eenvoudige manier zo'n netwerkprogramma's te maken. De basis van hoe dit verloopt en welke klassen en interfaces hiervoor beschikbaar zijn worden in dit hoofdstuk besproken.

4.2 Basistechnieken

4.2.1 Een voorbeeld: het UFP-protocol

Als rode draad door dit hoofdstuk ontwikkelen we client- en serverprogrammatuur voor een zelfverzonnen protocol met de naam *UFP* (*Useless Facts Protocol*). Hiermee kan een client een willekeurig 'nutteloos feit' opvragen vanop de server. (Deze service wordt aangeboden op poort 4257.)

Het UFP-protocol verloopt op de volgende manier: de client stuurt een tekstlijn naar de server die een willekeurig geheel getal bevat in decimale vorm. De server antwoordt hierop met een aantal tekstlijnen, afgesloten met één lege lijn. De client mag op deze manier meerdere berichten na elkaar opvragen, totdat hij uiteindelijk zelf de verbinding afsluit.

De tekst die door de server teruggestuurd wordt (het 'nutteloos feit') kan zelf geen lege lijnen bevatten. Lijnen in dit protocol worden steeds afgesloten door een enkele linefeed, zonder voorafgaande carriage return.

De server beslist zelf welk nutteloos feit hij met welk getal laat overeenkomen. Er wordt enkel ver-

wacht dat hetzelfde getal steeds overeenstemt met dezelfde tekst.

De server die wij zullen implementeren, haalt zijn nutteloze feiten uit een tekstbestand waarin de verschillende tekstjes gewoon door lege lijnen van elkaar zijn gescheiden. Hiervoor maken we gebruik van een hulpklasse *ParagrafenLijst* met de volgende publieke interface:

```

class ParagrafenLijst
{
    public ParagrafenLijst (String bestandsnaam) throws IOException;
        // Maakt een nieuwe paragrafenlijst aan uit het bestand met de
        // gegeven naam

    public int size ();
        // Geeft het totaal aantal paragrafen in deze lijst terug.

    public String get (int i);
        // Geeft de paragraaf terug met het opgegeven volgnummer.
        // De afzonderlijke lijnen worden afgesloten met een linefeed.
}

```

4.2.2 Een eenvoudige client

Als eerste voorbeeld schrijven we het programma *UFPClient*, een eenvoudige clienttoepassing. Deze toepassing aanvaardt als opdrachtlijnparameters achtereenvolgens de naam van de server, het poortnummer en het ‘berichtnummer’ van het nutteloos feit:

```

$ java UFPClient gonzo.hogent.be 4257 13
Zo'n 50 procent van alle bankovervallen in de V.S. vinden
plaats op een vrijdag.
$

```

Worden er geen drie argumenten opgegeven, dan gebruikt het programma de volgende standaardwaarden: de UFP-server heet *localhost* en gebruikt poort 4257, en voor het nutteloos feit wordt een willekeurig getal gebruikt.

Deze waarden worden vastgelegd bij het begin van het programma:

```

String hostname;
if (args.length == 0)
    hostname = "localhost";
else
    hostname = args[0];

int port;
if (args.length <= 1)
    port = 4257;

```

```

    else
        port = Integer.parseInt (args[1]);

    int msgno;
    if (args.length <= 2)
        msgno = (int) (Math.random() * 1024*1024);
    else
        msgno = Integer.parseInt (args[2]);

```

Onze interesse gaat echter voornamelijk uit naar de rest van het programma, een eerste voorbeeld van het gebruik van sockets in Java.

```

import java.net.*;
import java.io.*;

class UFPClient
{
    static public void main (String[] args)
    {
        ... // Bepaal hostname, port en msgno

        try (
            Socket client = new Socket (hostname, port);
            BufferedReader reader =
                new BufferedReader(new InputStreamReader(
                    client.getInputStream()));
            PrintWriter printer =
                new PrintWriter(new OutputStreamWriter(
                    client.getOutputStream()), true);) {

            printer.println (msgno);

            String lijn = reader.readLine ();
            while (lijn != null && lijn.length() != 0) {
                System.out.println (lijn);
                lijn = reader.readLine ();
            }

        } catch (IOException e) {
            // problemen met schrijven of lezen naar socket
            ...
        }
    }
}

```

Alvast enkele opmerkingen:

- Alle netwerkklassen behoren tot het pakket *java.net*, vandaar de eerste *import*-opdracht. Omdat netwerkcommunicatie gebruik maakt van het streammechanisme uit Java, moet ook het pakket *java.io* worden geïmporteerd.
- Het omzetten van een *String* naar een geheel getal kan een uitzondering genereren. In het huidige programma wordt daar geen rekening mee gehouden, maar in een professioneel programma kan je dergelijke uitzonderingen beter opvangen in een *try/catch*-blok, dat voor een juiste afwikkeling van het programma zorgt.
- We gebruiken een object van de klasse *PrintWriter* om uitvoer over het netwerk te versturen. Merk op dat we bij de constructie van deze printwriter als tweede argument *true* hebben opgegeven. Hierdoor wordt het uitvoerkanaal automatisch ge'flushed' na elke *println*. Dit is noodzakelijk voor de goede afhandeling van het protocol.
- Het gebruik van een object van de klasse *PrintWriter* en een object van de klasse *BufferedReader* om te communiceren over het netwerk veronderstelt dat het protocol tussen de client en de server tekst-georiënteerd is. Een communicatie waarbij bv. binaire bestanden zoals tekeningen worden doorgestuurd zal gebruik maken van (gebufferde) input- en outputstreams.

De klasse *Socket* in Java stelt een TCP-socket voor zoals die in een normaal clientprogramma wordt gebruikt. Er bestaan verschillende constructors voor deze klasse. We vermelden er hier slechts twee:

```
public Socket(String host, int port) throws UnknownHostException, IOException;
```

```
public Socket(InetAddress address, int port) throws IOException;
```

In het eerste geval geef je de naam op van de server en het poortnummer van de netwerkdienst waarmee je wenst te converseren. In het tweede geval wordt de server aangeduid door een IP-adres (zie §4.2.3).

Bij het aanmaken van een socket wordt onmiddellijk de verbinding met de tegenpartij geopend. In Java hoeft je niet expliciet een *connect*-functie op te roepen zoals in C++. Lukt dit niet, dan genereert Java een uitzondering van het type *IOException*, of meestal meer specifiek van een deelklasse van *SocketException*.

```
$ java UFPCClient
Exception in thread "main" java.net.ConnectException:
    Connection refused
...
$
```

Met elke open socket zijn twee streams verbonden: een uitvoerstream waarmee je berichten naar de server kan sturen, en een invoerstream waarmee je gegevens kan ontvangen. Je gebruikt de volgende methoden om toegang te krijgen tot deze ingebouwde streams:

```
public InputStream getInputStream () throws IOException;
```

```
public OutputStream getOutputStream () throws IOException;
```

Merk op dat de socketstreams *byte*-georiënteerd zijn. Omdat we in het voorbeeldprogramma met een tekstgericht protocol werken, hebben we ze onmiddellijk omgezet naar readers en writers van een gepast type.

Om de verbinding met de server af te sluiten, gebruik je de methode *close* van de klasse *Socket*. Deze sluit meteen ook de twee socketstreams af. Het *try*-block zorgt voor het afsluiten van de streams en de socket, in omgekeerde volgorde van het openen.

4.2.3 De klasse *InetAddress*

IP-adressen in een Java-omgeving worden voorgesteld met behulp van de klasse *InetAddress*. Deze klasse bevat geen publieke constructor en daarom moet je in de plaats nieuwe IP-adressen (*InetAddress*-objecten) aanmaken met een van de volgende klassenmethoden:

```
public static InetAddress getLocalHost() throws UnknownHostException;

public static InetAddress getByName(String host) throws UnknownHostException;
```

De eerste methode geeft het IP-adres terug van de machine waarop het Java-programma draait, de tweede geeft het IP-adres terug van een machine met een gegeven naam. Er is ook een methode die *alle* IP-adressen van een bepaalde machine bepaalt.

```
public static InetAddress[] getAllByName (String host)
    throws UnknownHostException;
```

De methodes *getByName* en *getAllByName* aanvaarden ook strings in ‘dotted-decimal’notatie als argument, in plaats van een echte DNS-naam.

Het volgende programma drukt alle IP-adressen af van alle machines wiens naam op de opdrachtlijn werd opgegeven:

```
public static void main (String[] args) {
    for (int i=0; i < args.length; i++) {
        try {
            InetAddress [] names = InetAddress.getAllByName (args[i]);
            for (int j=0; j < names.length; j++)
                System.out.println (names[j]);
        } catch (UnknownHostException e) {
            System.err.println("Host " + args[i] + " onbekend");
            System.err.println(" (" + e + ")");
        }
    }
}
```

Merk op dat we de mogelijke uitzonderingen hebben opgevangen. Wanneer een onbekende machine-naam wordt ingegeven, resulteert dit in een passend foutbericht:

```
$ java Adressen gonzo.hogent.be honzo.hogent.be localhost
gonzo.hogent.be/193.190.172.16
gonzo.hogent.be/192.168.16.17
Host honzo.hogent.be onbekend
    (java.net.UnknownHostException: honzo.hogent.be)
localhost/127.0.0.1
$
```

Zoals je ziet resulteert de *toString*-methode van *InetAddress* (die door *println* achter de schermen wordt gebruikt), in een string die zowel de naam van de machine bevat als het IP-adres in decimale notatie. Met de volgende methoden kan je deze twee onderdelen *afzonderlijk* bekomen:

```
public String getHostName();

public String getHostAddress();
```

4.2.4 Een eenvoudige server

Java gebruikt twee verschillende socketklassen voor de twee soorten sockets die bij een TCP-server worden gebruikt. De *luisterende* socket is een object van de klasse *ServerSocket*, de *verbonden* socket is van hetzelfde type *Socket* als bij een TCP-client (zie §4.2.2).

Als eerste voorbeeld programmeren we een eenvoudige server voor het UFP-protocol: een server die slechts één client tegelijkertijd kan bedienen.

We kozen ervoor om de UFP-server te implementeren als object van een klasse *UFPServer* en niet als een traditioneel programma zoals in §4.2.2. In het hoofdprogramma worden de poort waarop de server zal luisteren en het bestand waarin de te tonen paragrafen staan bepaald. Vervolgens wordt een object *UFPServer* aangemaakt dat zal luisteren naar binnenkomende aanvragen. De constructor en het hoofdprogramma van de klasse *UFPServer* zien er als volgt uit:

```
import java.io.*;
import java.net.*;

public class UFPServer
{
    final ParagrafenLijst paragrafen;
    private ServerSocket server;

    UFPServer (String bestand, int poort) throws IOException
    {
        // paragrafenlijst en serversocket initialiseren
        paragrafen = new ParagrafenLijst (bestand);
        server = new ServerSocket (poort);
    }
}
```



```

public static void main (String[] args)
{
    // poort bepalen
    int poort;
    if (args.length == 0)
        poort = 4257;
    else
        poort = Integer.parseInt(args[0]);

    // bestand bepalen
    String bestand;
    if (args.length <= 1)
        bestand = "UFPdata.txt";
    else
        bestand = args[1];

    // server opstarten
    try {
        UFPServer server = new UFPServer(bestand,poort);
        server.luister();
        server.afsluiten();
    } catch (IOException e) {
        System.out.println("Probleem met UFPServer: " + e.getMessage());
    }

}

private void afsluiten() throws IOException
{
    server.close();
}

... // andere methoden van deze klasse
}

```

In de constructor van *UFPServer* wordt de paragrafenlijst met nutteloze feiten op voorhand ingelezen en daarna wordt de serversocket aangemaakt met behulp van de volgende constructor:

```

public ServerSocket (int port) throws IOException;

```

De opgegeven parameter bevat het poortnummer waaraan deze service moet worden gehecht.

Je kan ook nog een tweede parameter opgeven die het aantal clients bepaalt die tegelijkertijd met de server een verbinding kunnen leggen (m.a.w. de lengte van de *listen*-wachtrij) maar doorgaans is de defaultwaarde hier voldoende.

Het oproepen van de methode *afsluiten* zorgt ervoor dat, indien de oneindige lus in de methode *luister*

wordt beëindigd, de serversocket op correcte wijze wordt afgesloten.

Het hart van deze server is de methode *luister*:

```
private void luister ()
{
    while (true)
    { // wachten op aanvraag
        try (Socket client = server.accept())
        {
            InetAddress clientIP = client.getInetAddress();
            System.out.println("UFPServer: accepted "
                               + clientIP);
            // aanvraag verwerken
            handleClient(client);
            System.out.println("UFPServer: disconnect "
                               + clientIP);
        } catch (IOException e)
        {
            /* opvangen fout bij het afhandelen van
               de huidige client */
            ...
        }
    }
}
```

In essentie is dit een oneindige lus waarin telkens op een clientconnectie wordt gewacht die dan meteen wordt verwerkt door de methode *handleClient*.

De methode *accept* van de klasse *ServerSocket* wacht totdat er een client een verbinding met de server legt en ze geeft een nieuwe socket terug (de verbonden socket) die de connectie met deze server representeert. Deze verbonden socket is van het type *Socket* en wordt op dezelfde manier gebruikt als bij een TCP-client.

Merk op dat de methode *getInetAddress* van *Socket* het IP-adres van de tegenpartij teruggeeft. In dit voorbeeld gebruiken we dit adres enkel maar als informatie voor een logbestand. Het is namelijk de bedoeling dat de server wordt opgestart met een opdracht zoals deze:

```
$ java UFPServer >>UFPServer.log
```

Merk op dat we een *try/catch*-combinatie gebruiken om fouten op te vangen die optreden tijdens het verwerken van één enkele client, zonder dat de volgende client er last van heeft.

De methode *handleClient* verzorgt de verbinding met de client en zorgt voor juiste de afhandeling van het UFP-protocol.

```
private void handleClient (Socket client) throws IOException
{
```

```

try (// kanaal om van te lezen en naar te schrijven bepalen
    BufferedReader in = new BufferedReader(
        new InputStreamReader (client.getInputStream()));
    PrintWriter uit = new PrintWriter(
        new OutputStreamWriter (client.getOutputStream()),true);
{
    // aanvraag inlezen en antwoord aanmaken en doorsturen
    String vraag = in.readLine();
    while (vraag != null)
    {
        try
        {
            // nummer paragraaf bepalen
            int index = Integer.parseInt(vraag) * 36277287;
            if (index < 0) index = -index;
            index %= paragrafen.size();
            // paragraaf doorsturen
            uit.println(paragrafen.get(index));
        } catch (NumberFormatException e)
        {
            uit.println();
        }
        vraag = in.readLine();
    }
} catch (IOException e)
{
    System.out.println("UFPServer: " + e.getMessage());
}

```

(De verschillende *index*-berekeningen vertalen het ‘berichtnummer’ dat van de client komt in een volgnummer dat binnen de paragrafenlijst past. De factor 36277287 zorgt voor een min of meer ‘willekeurig’ verband tussen het opgegeven berichtnummer en het werkelijke volgnummer. Dit maakt het bericht iets minder voorspelbaar.)

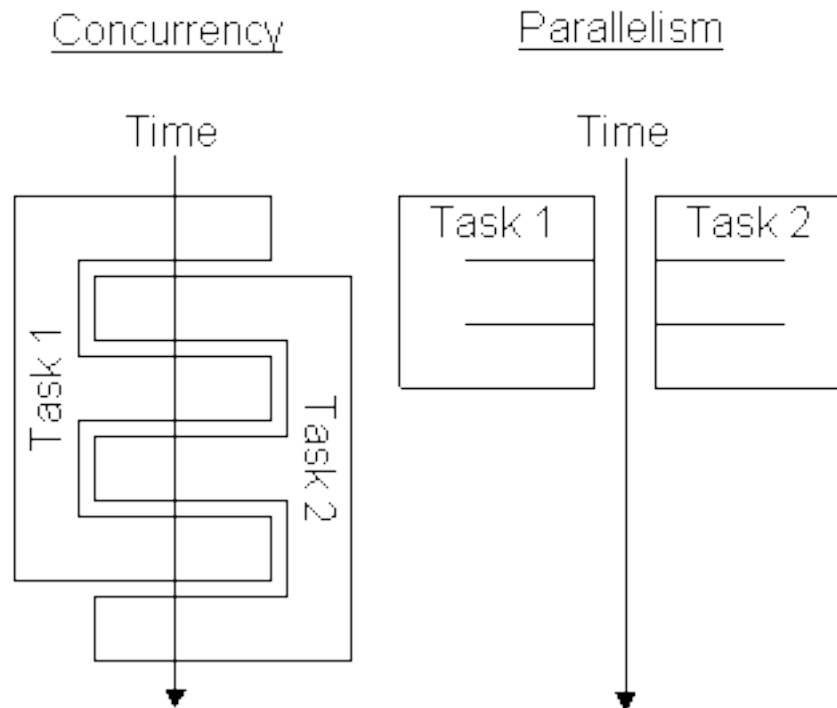
4.3 Concurrency

Om meerdere clients te gelijk te bedienen heeft een server *threads* nodig. In deze sectie wordt *concurrency* in Java besproken.

Concurrent software is software die (schijnbaar) gelijktijdig verschillende taken kan uitvoeren. Van bij de start werd het Java-platform ontwikkeld om concurrency te ondersteunen. Sinds versie 5.0 werden ook *high-level concurrency APIs* toegevoegd. We bespreken eerst basisondersteuning wat betreft concurrency en vervolgens behandelen we een aantal van de meer *high-level* aspecten.

4.3.1 Concurrency versus Parallelism

Zoals geïllustreerd in figuur 4.1 ([thra]) geeft een systeem dat concurrency implementeert de indruk dat verschillende taken gelijktijdig worden uitgevoerd. In feite worden de taken in stukken opgedeeld die elk om beurt afgewerkt worden door de processor met behulp van *time slicing*. In parallelle systemen worden twee of meer taken echt gelijktijdig verwerkt. Hiervoor is een systeem met meerdere CPU's nodig.

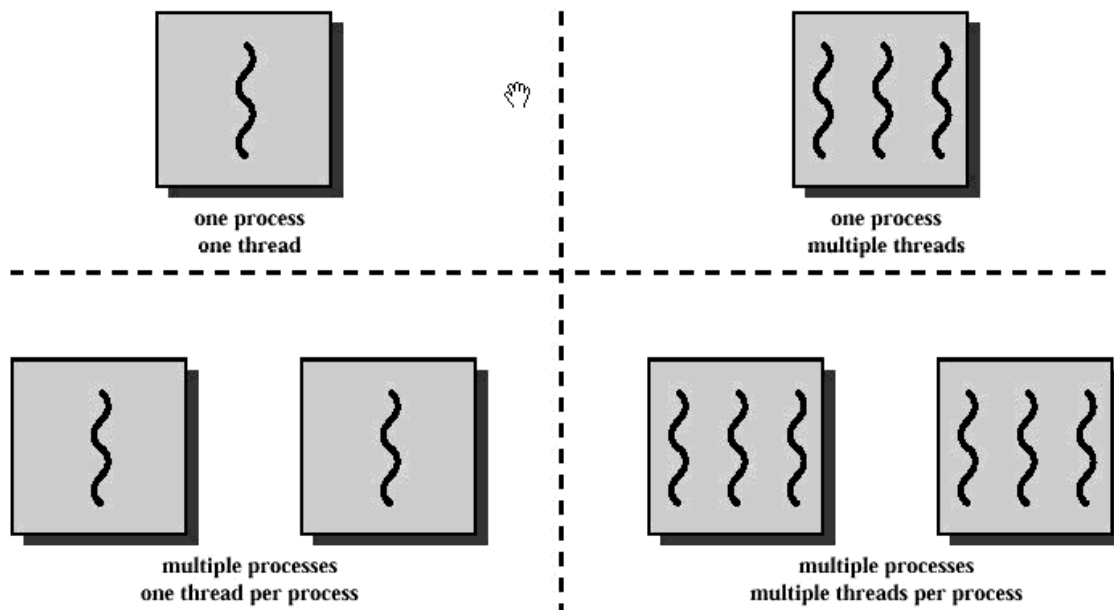


Figuur 4.1: Concurrency versus Parallelism

4.3.2 Processen en threads

In concurrency zijn er twee mogelijke opties om code gelijktijdig uit te voeren: processen en threads (zie figuur 4.2, [thrb]). In een Javaprogramma wordt concurrency bijna uitsluitend gerealiseerd met threads.

Processen Een proces is een op zich staande *execution environment*. Een proces heeft zijn eigen verzameling van bronnen zoals bijvoorbeeld geheugenruimte. Communicatie tussen processen verloopt via IPC (*Inter Process Communication*, bijvoorbeeld via *pipes*). De meeste Java virtuele machines worden uitgevoerd als één proces.



Figuur 4.2: Processen en threads

Threads Threads worden ook wel lichtgewicht processen genoemd omdat ze minder systeembronnen vereisen bij het aanmaken. Threads bestaan enkel binnen een proces. De verschillende threads binnen één proces delen de bronnen van het proces: geheugen, open bestanden, ... Dit maakt communicatie tussen threads efficient, maar ook foutgevoelig.

4.3.3 Threads in Java

Threads in java zijn instanties van de klasse `Thread`. Er zijn twee strategieën voor het gebruik van threads in java.

- Telkens als de applicatie een asynchrone taak moet uitvoeren, wordt een thread aangemaakt en gestart. Je controleert zelf het aanmaken en beheren van threads.
- Je delegeert het beheren van de threads aan een *executor*.

Om bepaalde functionaliteit in een thread te realiseren, moet je bij het aanmaken van de thread de code die thread moet uitvoeren meegeven. Hiervoor zijn er drie opties.

1. Geef een object mee dat de interface `Runnable` implementeert. Deze interface heeft één methode `run` die de uit te voeren code bevat. Je geeft het `Runnable` object mee in de constructor. (zie codevoorbeeld 4.1)

```

public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}

```

Codevoorbeeld 4.1: Implementeer de interface Runnable

2. Leid een klasse af van de klasse Thread, die zelf de interface Runnable implementeert. De methode run heeft in dit geval een lege implementatie. In de afgeleide klasse overschrijf je deze methode met de gewenste functionaliteit. (zie codevoorbeeld 4.2)

```

public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }

}

```

Codevoorbeeld 4.2: Afleiden van de klasse Thread

3. Gebruik een lambda-uitdrukking met de gewenste functionaliteit om een Runnable-object aan te maken. (zie codevoorbeeld 4.3)

```

public static void main(String args[]) {
    Runnable task = () -> {
        System.out.println("Hello from a thread!");
    };
    new Thread(task).start();
}

```

Codevoorbeeld 4.3: Gebruik een lambda-uitdrukking

De klasse Thread definieert een aantal methodes die handig zijn voor het beheer van threads, onder andere statische methodes om informatie over de huidige thread op te vragen of zijn toestand te veranderen. Verder zijn er ook methodes om een thread te pauzeren, te onderbreken of om te wachten op het aflopen van een thread.

4.3.4 Communicatie tussen threads

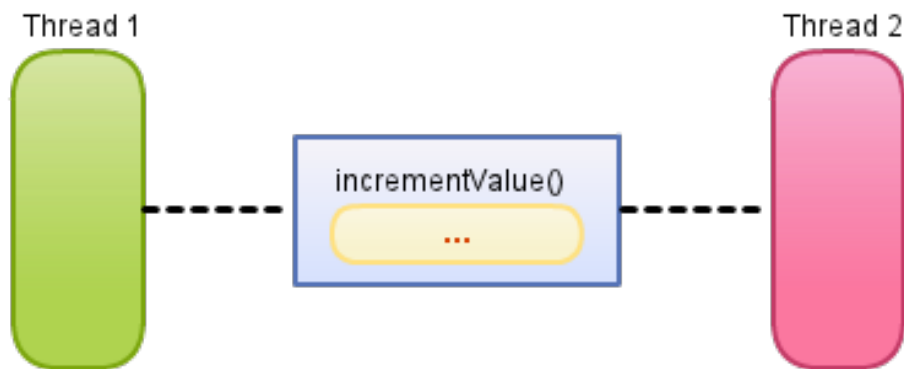
Threads communiceren vooral door gemeenschappelijke toegang tot bepaalde velden, objecten en methodes. Deze vorm van communicatie is zeer efficiënt, maar mogelijks foutgevoelig. Gemeenschappelijke toegang tot dezelfde objecten kan twee types fouten veroorzaken: *thread interference* en *memory consistency errors*.

Thread interference kan optreden wanneer verschillende threads gedeelde data veranderen. Memory consistency errors zijn fouten afkomstig van inconsistente views op gedeelde data. In het onderstaand voorbeeld wordt geïllustreerd hoe deze kunnen optreden.

Stel dat twee threads een referentie hebben naar een zelfde object van het type `SharedObject` met een methode `incrementValue()`. (zie code 4.4 en figuur 4.3, [thrc])

```
public class SharedObject{  
    int value = 0;  
  
    public void incrementValue(){  
        int temp = value;  
        temp = temp +1 ;  
        value = temp;  
    }  
}
```

Codevoorbeeld 4.4: Een gedeeld object



Figuur 4.3: Gemeenschappelijke toegang vanuit twee threads

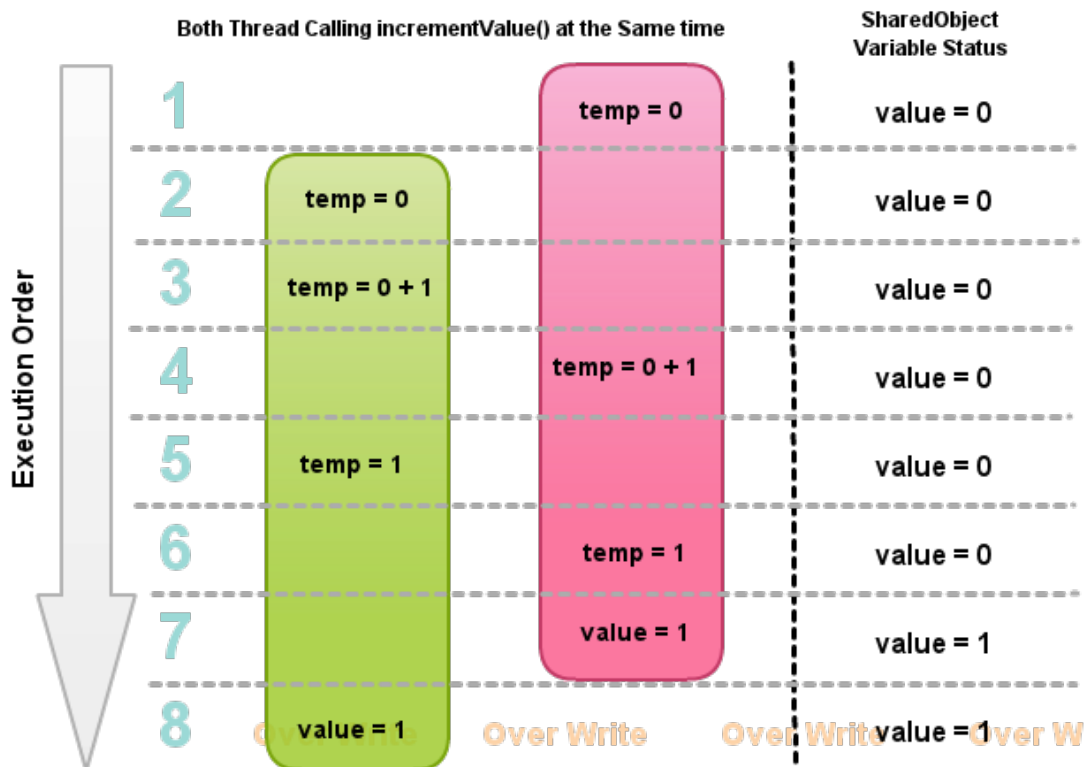
Stel dat bij het uitvoeren van de software de methode `incrementValue()` gelijktijdig wordt uitgevoerd vanuit de twee threads. Na het uitvoeren van deze twee methode-oproepen zou je verwachten dat de waarde van het veld `value` van het `SharedObject`-object met twee verhoogd is. Dit hoeft niet altijd zo te zijn omdat de twee threads de methode gelijktijdig uitvoeren! Wat er kan mis gaan:

- De ene thread kan de acties van de andere thread overschrijven.

- De ene thread weet niet van de veranderingen die de andere thread doorvoerde.

Alles hangt af van hoe de *Thread Scheduler* de uitvoering van de verschillende opdrachten in de methode `incrementValue()` plant. Aangezien we niet zeker weten dat de ene thread de volledige code uitvoert voor de andere thread toegang heeft tot hetzelfde object, kan het dat we na het uitvoeren van de beide methode-oproepen eindigen met inconsistente data in ons gedeeld object.

In figuur 4.4 ([thrc]) wordt getoond hoe het mogelijks kan mislopen. Stel dat de groene thread het uitvoeren van de methode start voordat de rode thread klaar is (faze 2 en 3 op de figuur). Dan kan het gebeuren dat de groene thread toegang heeft tot de waarde van `value` voordat de rode thread deze waarde heeft aangepast. Zo overschrijft de groene thread de waarde die aangepast werd door de rode thread (faze 7 en 8 op de figuur).



Figuur 4.4: Voorbeeld Thread interference

Met behulp van *synchronization* kan je deze problemen vermijden. Synchronisatie zelf introduceert mogelijks nieuwe problemen zoals *deadlock*, *starvation* en *livelock*. Voor meer informatie verwijzen we naar de cursus besturingssystemen.

4.3.5 Executors en concurrent collections

Tot nu toe focusten we op de *low level* API's voor concurrency die in elke versie van het Java-platform beschikbaar zijn. In deze paragraaf bekijken we de *high level* API's voor concurrency die geïntroduceerd werden in versie 5.0. Deze zijn API's zijn geschikter voor meer complexe taken en systemen met meerdere processors en cores. We focussen op *executors* en *concurrent collections*. Andere functionaliteit, die we niet bespreken, is o.a.

- *Lock objects* worden gebruikt om toegang tot gedeelde bronnen vanuit verschillende threads te controleren. Een lock zorgt ervoor dat maar één thread tegelijk de bron kan manipuleren.
- *Atomic variables classes* zijn klassen die primitieve types voorstellen en er tegelijk voor zorgen dat het aanpassen van de data van de variabele atomair verloopt. Dit wil zeggen dat de waarde van deze variabele vanuit verschillende threads aangepast kan worden zonder dat er een risico is op *memory consistency errors*.
- `ThreadLocalRandom` is een generator van random getallen voor verschillende threads.

Executor

In grootschalige programma's is het aan te raden om het aanmaken en beheren van threads te scheiden van de rest van de toepassing. Een *executor* maakt dit mogelijk. In de package `java.util.concurrent` zijn er drie executorinterfaces gedefinieerd. (zie figuur 4.5)

- `Executor` is een eenvoudige interface om nieuwe taken te starten.
- `ExecutorService` is een afgeleide interface van `Executor` die extra functionaliteit toevoegt om de levensloop van taken en van de executor te beheren.
- `ScheduledExecutorService` is een subinterface van `ExecutorService` die het mogelijk maakt om taken periodiek of op een later tijdstip te plannen.

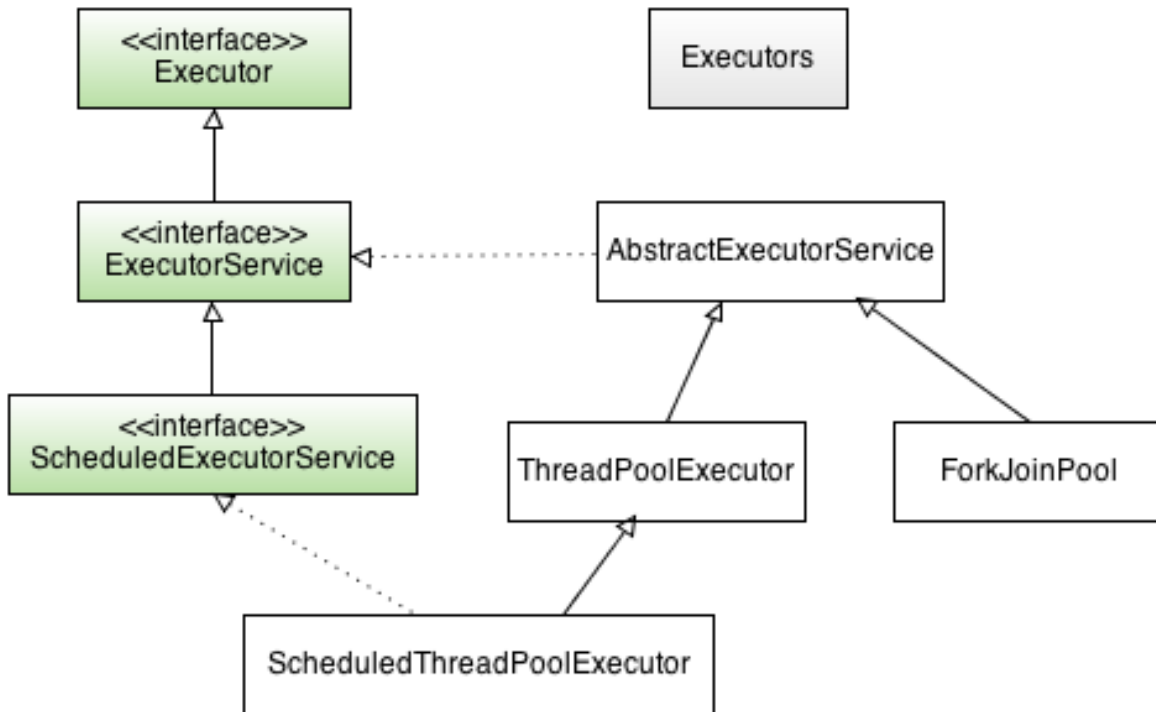
De interface `Executor` bestaat uit één methode `execute` waarmee je een taak kan starten. Deze methode vervangt het expliciet opstarten van een thread (zie codefragment 4.5 en 4.6).

```
Runnable r = ... ;
(new Thread(r)).start();
```

Codevoorbeeld 4.5: Thread starten

```
Runnable r = ... ;
Executor e = ... ;
e.execute(r);
```

Codevoorbeeld 4.6: Taak aan een executor geven



Figuur 4.5: Overzicht executorinterfaces [exe]

De code in 4.6 is minder specifiek. Afhankelijk van de implementatie van de executor kan de thread onmiddellijk gestart worden, zoals in voorbeeld 4.5, of toegekend worden aan een reeds bestaande werker-thread, of toegevoegd worden aan de wachtrij (*queue*) totdat een werker-thread beschikbaar is.

De methode `execute` heeft een `Runnable`-object als parameter en voert die taak asynchroon uit (zie voorbeeld 4.7). Het is niet mogelijk om een resultaat van de uitgevoerde taak te verkrijgen. Daarvoor moet je een `Callable`-object gebruiken. (zie voorbeeld 4.8)

```

Executor executorService = ... ;
executorService.execute(new Runnable() {
    @Override
    public void run() {
        System.out.println("Asynchronous task");
    }
});
  
```

Codevoorbeeld 4.7: Runnable-object uitvoeren

ExecutorService

De interface `ExecutorService` voegt de veelzijdigere `submit`-methode toe. Deze methode verwacht een `Runnable`-object of `Callable`-object als parameter. Met een `Callable`-object kan de

taak een waarde teruggeven. Het resultaat van de `submit`-methode is een `Future`-object waarmee de teruggeefwaarde opgevraagd kan worden.

Een `Future`-object stelt het resultaat van een asynchrone berekening voor en heeft methodes om na te gaan of de berekening voltooid is, om te wachten op het beëindigen van de taak en om het resultaat op te vragen.

```

ExecutorService executorService = ... ;
Future<String> future = executorService.submit(new Callable() {
    @Override
    public String call() throws Exception {
        System.out.println("Asynchronous Callable");
        return "Callable Result";
    }
});
try {
    System.out.println("future.get() = " + future.get());
} catch (InterruptedException | ExecutionException ex) {...}

```

Codevoorbeeld 4.8: Callable-object uitvoeren

Het resultaat van de asynchrone opdracht kan opgehaald worden met de methode `get`. Deze methode blokkeert indien nodig totdat de taak afgerond is.

Een opdracht annuleren gebeurt met de methode `cancel`. Dit kan enkel als de opdracht nog niet klaar is.

De `submit`-methode aanvaardt ook `Runnable`-objecten. In dat geval kan het `Future`-object gebruikt worden om te controleren of de opdracht afgelopen is.

```

executorService.submit(() -> {
    System.out.println("Asynchronous task");
});

```

Codevoorbeeld 4.9: Runnable-object indienen

Een `ExecutorService` kan aangemaakt worden met de factory `Executors` die in verschillende types executors voorziet. In codevoorbeeld 4.10 wordt het aanmaken van drie types executors geïllustreerd.

- Een executor met één werker-thread met een onbegrensde wachtrij.
- Een executor met 10 werker-threads met een onbegrensde wachtrij.
- Een executor met 10 werker-threads die taken uitgesteld of periodiek kan uitvoeren.

```

ExecutorService execServ1 = Executors.newSingleThreadExecutor();

ExecutorService execServ2 = Executors.newFixedThreadPool(10);

```

```
ExecutorService execServ3 = Executors.newScheduledThreadPool(10);
```

Codevoorbeeld 4.10: Aanmaken executorservice

Een executor waarbij het aantal werker-threads kan uitbreiden, maak je aan met de methode `newCachedThreadPool`.

Verder voorziet de `ExecutorService` methodes om een hele reeks `Callable`-objecten mee te geven en methodes om de executor af te sluiten.

De methode `invokeAll` voert alle `Callable`-objecten uit de collectie meegegeven als parameter uit (zie voorbeeld 4.11). Het resultaat is een lijst van `Future`-objecten met de resultaten van elke taak uit de verzameling. Merk op dat een taak ook kan eindigen ten gevolge van een exceptie. Dit kan je niet opmaken uit het `Future`-object.

```
ExecutorService executorService = Executors.newSingleThreadExecutor();

Set<Callable<String>> callables = new HashSet<>();
callables.add((Callable<String>) () -> "Task 1");
callables.add((Callable<String>) () -> "Task 2");
callables.add((Callable<String>) () -> "Task 3");

try {
    List<Future<String>> futures = executorService.invokeAll(callables);

    for(Future<String> future : futures)
        System.out.println("future.get = " + future.get());
} catch (InterruptedException | ExecutionException ex) {...}

executorService.shutdown();
```

Codevoorbeeld 4.11: Een reeks opdrachten uitvoeren

Als je de `executorservice` niet meer nodig hebt, dan moet je hem expliciet afsluiten met de methode `shutdown` zodat de threads afgesloten worden. De service wordt niet onmiddellijk afgesloten, maar hij zal geen nieuwe taken meer aannemen en van zodra alle ingediende opdrachten afgerond zijn zal hij afsluiten. Als je een programma start via de `main`-methode en dit is afgelopen, dan blijft het programma actief zolang de `executorservice` niet is afgesloten. De actieve threads in de `executorservice` verhinderen dat de JVM wordt afgesloten.

Als je een `executorservice` onmiddellijk wil afsluiten, dan kan je de methode `shutdownNow` aanroepen. Die probeert om alle actieve taken dadelijk te sluiten en slaat niet-uitgevoerde taken over.

De methode `invokeAny` aanvaardt een collectie van `Callable`-objecten. Het resultaat van deze methode is het resultaat van één van de meegegeven taken. Het is niet mogelijk om te weten van welke opdracht je het resultaat zal krijgen. Als één van de taken klaar is, dan worden de andere taken geannuleerd.

```
ExecutorService executorService = Executors.newSingleThreadExecutor();

Set<Callable<String>> callables = new HashSet<>();
for (int i = 1; i <= 100; i++) {
    final int j = i;
    callables.add((Callable<String>) () -> {
        String opdracht = "Task " + j;
        System.out.println(opdracht);
        return opdracht;
    });
}

try {
    String result = executorService.invokeAny(callables);
    System.out.println("result = " + result);
} catch (InterruptedException | ExecutionException ex) {
    Logger.getLogger(...)
}
```

Codevoorbeeld 4.12: Een aantal opdrachten uitvoeren

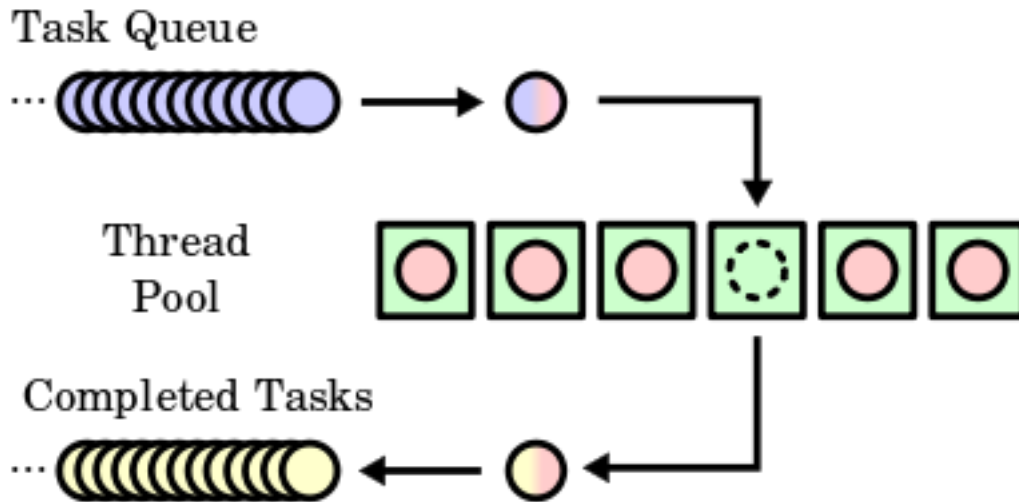
Thread Pools

De meeste implementaties van executors uit de package `java.util.concurrent` gebruiken een *thread pool* die bestaat uit een aantal werker-threads. Een werker-thread bestaat los van de taken die ze uitvoeren en worden meestal hergebruikt voor meerdere taken.

Het gebruik van werker-threads vermindert de overhead veroorzaakt door het aanmaken van threads. Thread-objecten verbruiken een significante hoeveelheid geheugen en in grootschalige applicaties zorgt het telkens toewijzen en vrijmaken van geheugen voor significante *memory management overhead*.

Eén type thread pool is een *fixed thread pool*. Dit type bestaat uit een vastgesteld aantal actieve threads. Als een thread om de een of andere reden zou afsluiten, dan wordt hij automatisch vervangen door een nieuwe. Taken worden toegevoegd aan een thread pool via een interne wachtrij (*queue*). Deze queue bevat de extra taken indien er meer taken dan threads zijn. (zie figuur 4.6 ([thrd])

Eén voordeel van een fixed thread pool is dat het een toepassing toelaat om elegant te vertragen. Beschouw een webserver die elke HTTP-aanvraag in een aparte thread afhandelt. Indien de webapplicatie voor elke aanvraag een nieuwe thread creëert en het systeem ontvangt meer threads dan het kan afhandelen, dan zal de applicatie plots stoppen met het beantwoorden van alle threads. Met een fixed thread pool zal de applicatie de aanvragen trager afhandelen, maar ze blijft wel draaien.



Figuur 4.6: Thread pool [exe]

ScheduledExecutorService

De interface `ScheduledExecutorService` breidt de interface `ExecutorService` verder uit met methodes om taken te plannen met een zekere vertraging of op periodieke tijdstippen.

Concurrent Collections

De package `java.util.concurrent` voegt een aantal collecties toe aan het Java Collections Framework die helpen om data te delen tussen verschillende threads en zo memory consistency errors voorkomen. In deze sectie beschrijven we drie interfaces uit deze packages.

- `ConcurrentMap` is een afgeleide interface van `java.util.Map` die atomaire opdrachten definieert. Atomaire opdrachten zorgen ervoor dat je methodes niet *synchronized* moet maken. Deze opdrachten verwijderen een *key-value*-paar alleen als de sleutel bestaat. Omgekeerd wordt een *key-value*-paar enkel toegevoegd als de sleutel nog niet aanwezig is.
- `ConcurrentNavigableMap` is een thread-safe equivalent van een `TreeMap`.
- `BlockingQueue` is een *first-in-first-out* datastructuur die blokkeert of een timeout geeft, als je iets probeert toe te voegen aan een volle wachtrij of als je iets probeert te lezen van een lege wachtrij. De methodes van een `BlockingQueue` zijn beschikbaar in vier varianten die elk op een andere manier reageren wanneer een opdracht niet uitgevoerd kan worden.
 - een exceptie gooien
 - een speciale waarde (**null** of **false**) teruggeven
 - oneindig blokkeren

- een tijdje blokkeren

Een overzicht van deze methodes vind je in tabel 4.1.

	Exceptie gooien	Speciale waarde	Blokkeren	Timeout
Insert	<code>add(e)</code>	<code>offer(e)</code>	<code>put(e)</code>	<code>offer(e, time, unit)</code>
Remove	<code>remove()</code>	<code>poll()</code>	<code>take()</code>	<code>poll(time, unit)</code>
Examine	<code>element()</code>	<code>peek()</code>	niet mogelijk	niet mogelijk

Tabel 4.1: Overzicht methodes `BlockingQueue`

Atomaire acties worden als één geheel uitgevoerd. Deze acties worden volledig uitgevoerd of niet. De actie kan niet ergens in het midden stoppen. Er zijn geen zijeffecten zichtbaar totdat de actie afgelopen is.

De Java-taal definieert een *happens-before* relaties op opdrachten die lezen en schrijven naar het geheugen vanuit verschillende threads. De resultaten van een schrijfo opdracht door één thread zijn gegarandeerd zichtbaar voor een leesopdracht door een andere thread als de schrijfoperatie de *happens-before* relatie heeft met de leesopdracht.

Zo zal een schrijfactie naar een **volatile** veld de *happens-before* relatie hebben met een leesactie naar datzelfde veld.

Alle methodes van klassen in de package `java.util.concurrent` brengen deze garantie op een hoger niveau. Bijvoorbeeld, de acties om een object toe te voegen aan een concurrent collection *happen-before* de volgende actie om dat element op te vragen of te verwijderen.

4.4 Netwerkprogrammatie met behulp van draden

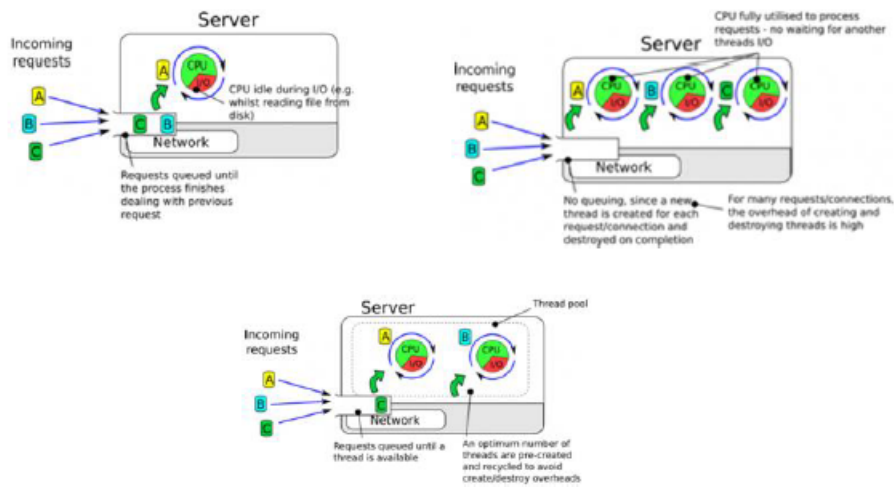
Op dit ogenblik kan de UPF-server maar één aanvraag van een client tegelijk afhandelen. Met behulp van draden (*threads* in het engels) passen we de server aan zodat hij meerdere clients tegelijkertijd kan afwerken.

Figuur 4.7 illustreert de opties voor een server die meerdere clients heeft.

- één thread die alle clients afhandelt
- één thread per client
- een thread pool

4.4.1 Structuur server

In §4.2.4 zagen we dat de kern van een server die slechts één client tegelijk bedient de volgende structuur heeft in pseudocode.



<https://myshadesofgray.wordpress.com/2014/04/13/java-executor-framework/>

Figuur 4.7: Server met meerdere clients

```
while (true) {
    aanvaardt connectie
    behandelt client
}
```

Een server die meerdere clients tegelijk wil afhandelen zal voor elke clientaanvraag een nieuwe draad starten. Deze draad zal de communicatie met client verder voeren en de server kan ondertussen nieuwe aanvragen ontvangen waarvoor hij dan opnieuw een draad opstart. De structuur van de server ziet er dus zo uit:

```
while (true) {
    aanvaardt connectie
    start een draad die de clientaanvraag behandelt
}
```

Twee nadelen van deze werkwijze zijn:

- Het telkens aanmaken en afsluiten van threads genereert een overhead als er veel gelijktijdige clientaanvragen zijn.
- Als het aantal clients groter is dan de capaciteit van de server, dan kan de server crashen.

Een threadpool bestaat uit een aantal threads. Deze threads worden telkens hergebruikt. Dit voorkomt de overhead van het telkens aanmaken en afsluiten van threads en het mogelijks uitvallen van de server bij te veel gelijktijdige aanvragen. Als er te veel aanvragen zijn, dan worden deze in een wachtrij geplaatst.

De structuur van de server is dus als volgt:

```
while (true) {  
    aanvaardt connectie  
    maakt een draad voor de clientaanvraag en bezorgt die aan een executor  
}
```

De klasse *UFP*Server heeft in grote lijnen dezelfde structuur als voorheen. De methode *handleClient* is verdwenen. De functionaliteit van deze methode zal overgenomen worden door een draad.

```
import java.io.*;  
import java.net.*;  
  
public class UFPServer  
{  
    final ParagrafenLijst paragrafen;  
    private ServerSocket server;  
  
    UFPServer (String bestand, int poort) throws IOException  
    {  
        ... // paragrafenlijst en serversocket initialiseren  
    }  
  
    private void luister ()  
    {  
        ... // wachten op aanvragen; een draad opstarten  
    }  
  
    private void afsluiten() throws IOException  
    {  
        server.close();  
    }  
  
    public static void main (String[] args)  
    {  
        ... // poort en bestand bepalen; server opstarten  
    }  
}
```

De constructor en de hoofdmethode hebben dezelfde structuur als in §4.2.4, enkel de methode *luister* is aangepast.

```
private void luister ()  
{  
    ExecutorService execServ = Executors.newFixedThreadPool(10);
```

```

    while (true)
    {    // wachten op aanvraag
        try
        {
            Socket client = server.accept();
            // aanvraag verwerken
            execServ.submit(new UFPThread(client,paragrafen));
        } catch (IOException e)
        {
            System.out.print("UFPServer: ");
            System.out.println(e.getMessage());
        }
    }
}

```

4.4.2 De draad

De eigenlijke communicatie met de client wordt nu afgehandeld in een aparte draad. De klasse *UFPThread* is een afgeleide klasse van *Thread* en implementeert zijn *run*-methode. Deze methode bestaat uit bijna dezelfde code als de methode *handleClient* van de UFP-server uit §4.2.4. Het enige verschil is dat we de uitzondering die kan optreden door het afsluiten van de socket lokaal moeten behandelen omdat de methode *run* geen uitzonderingen kan gooien.

```

import java.net.*;
import java.io.*;

public class UFPThread extends Thread
{
    Socket client;
    ParagrafenLijst paragrafen;

    UFPThread(Socket client, ParagrafenLijst paragrafen)
    {
        // client en paragrafenlijst initialiseren
        this.client = client;
        this.paragrafen = paragrafen;
    }

    public void run()
    {    // kanaal om van te lezen en naar te schrijven bepalen
        try (BufferedReader in = new BufferedReader(
            new InputStreamReader (client.getInputStream()));
            PrintWriter uit = new PrintWriter(
            new OutputStreamWriter (client.getOutputStream()),true);
        )
        {

```

```

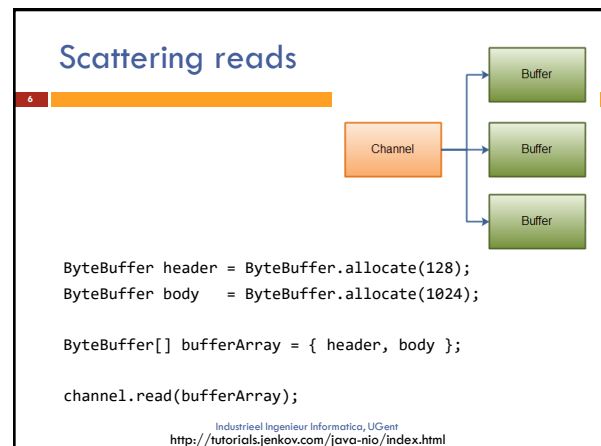
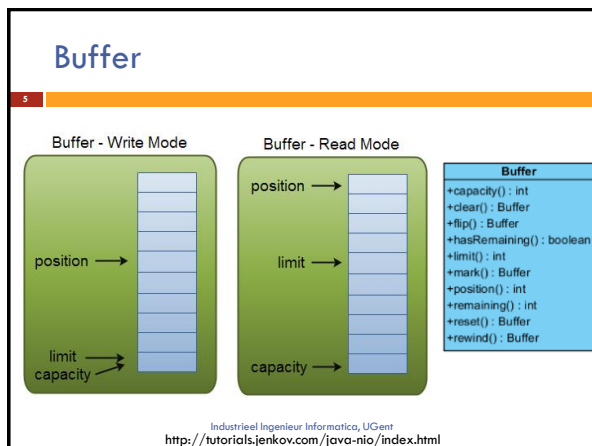
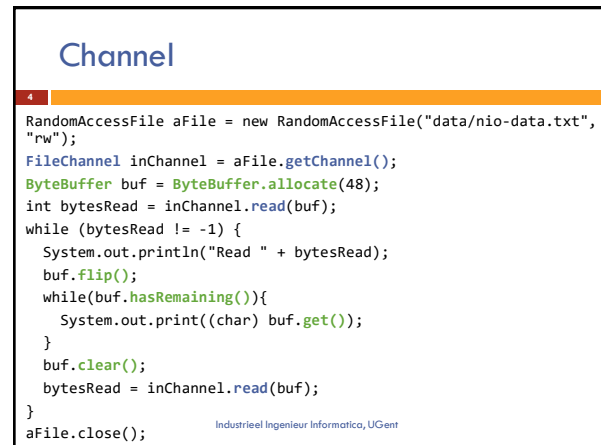
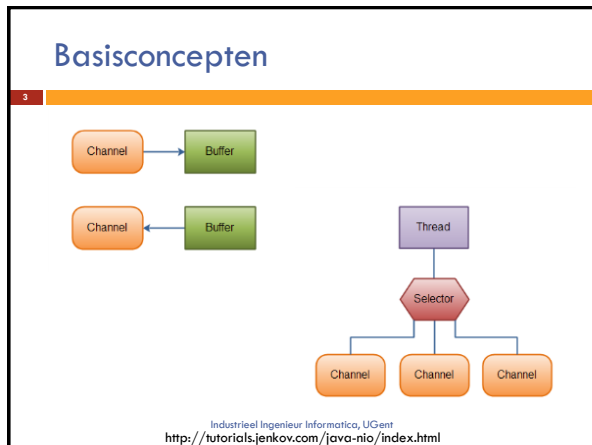
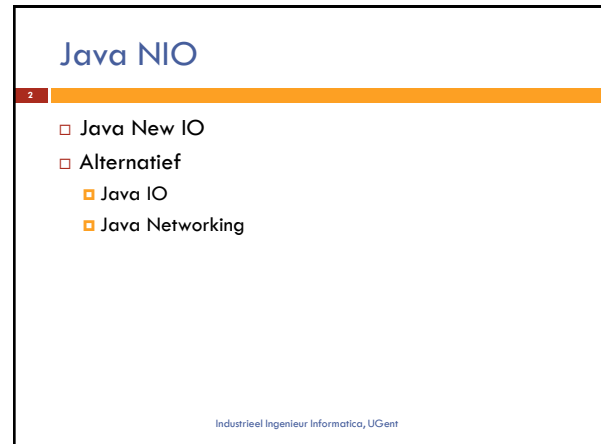
        // aanvraag inlezen en antwoord aanmaken en doorsturen
        String vraag = in.readLine();
        while (vraag != null)
        {
            try
            {
                // nummer paragraaf bepalen
                int index = Integer.parseInt(vraag)
                    * 36277287;
                if (index < 0) index = -index;
                index %= paragrafen.size();
                // paragraaf doorsturen
                uit.println(paragrafen.get(index));
            } catch (NumberFormatException e)
            {
                uit.println();
            }
            vraag = in.readLine();
        }
    } catch (IOException e)
    {
        System.out.println("UFPServer, probleem met client: "
            + e.getMessage());
    } finally {
        // client afsluiten
        try
        {
            client.close();
        } catch (IOException e)
        {
            System.out.print("UFPServer, probleem afsluiten");
            System.out.println(" client: " + e.getMessage());
        }
    }
}
}

```

Met de constructor worden een object *Socket* en een object *ParagrafenLijst* meegegeven. Elke draad zal dus van hetzelfde object *ParagrafenLijst* gebruik maken.

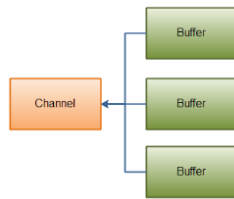
Hoofdstuk 5

Java NIO



Gathering writes

7



```
ByteBuffer header = ByteBuffer.allocate(128);
ByteBuffer body = ByteBuffer.allocate(1024);

//write data into buffers

ByteBuffer[] bufferArray = { header, body };

channel.write(bufferArray);
```

Industrieel Ingenieur Informatica, UGent
<http://tutorials.jenkov.com/java-nio/index.html>

Channel to channel transfer

8

```
RandomAccessFile fromFile
    = new RandomAccessFile("fromFile.txt", "rw");
FileChannel fromChannel = fromFile.getChannel();

RandomAccessFile toFile
    = new RandomAccessFile("toFile.txt", "rw");
FileChannel toChannel = toFile.getChannel();

long position = 0;
long count = fromChannel.size();

toChannel.transferFrom(fromChannel, position, count);
```

Industrieel Ingenieur Informatica, UGent

Channel to channel transfer

9

```
RandomAccessFile fromFile
    = new RandomAccessFile("fromFile.txt", "rw");
FileChannel fromChannel = fromFile.getChannel();

RandomAccessFile toFile
    = new RandomAccessFile("toFile.txt", "rw");
FileChannel toChannel = toFile.getChannel();

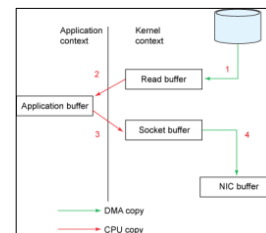
long position = 0;
long count = fromChannel.size();

fromChannel.transferTo(position, count, toChannel);
```

Industrieel Ingenieur Informatica, UGent

Zero copy

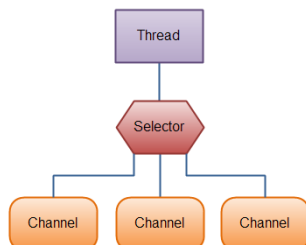
10



Industrieel Ingenieur Informatica, UGent
<http://www.ibm.com/developerworks/library/j-zerocopy/>

Selector

11



Industrieel Ingenieur Informatica, UGent
<http://tutorials.jenkov.com/java-nio/index.html>

Selector – aanmaak en registratie

12

```
// aanmaak selector
Selector selector = Selector.open();

// registratie één of meerdere kanalen
channel.configureBlocking(false);
SelectionKey key
    = channel.register(selector, SelectionKey.OP_READ);
```

Industrieel Ingenieur Informatica, UGent

Selector - gebruik

13

```
while(true) {
    // wachten
    int readyChannels = selector.select();
    if(readyChannels == 0) continue;

    for (SelectionKey key : selector.selectedKeys()) {
        .. // kanaal klaar om te ...
    }
    selector.selectedKeys().clear();
}
```

Industrieel Ingenieur Informatica, UGent

Selector - communicatie

14

```
if(key.isAcceptable()) {
    // a connection was accepted by a ServerSocketChannel.
    Channel channel = key.channel(); ...
} else if (key.isConnectable()) {
    // a connection was established with a remote server.
} else if (key.isReadable()) {
    // a channel is ready for reading
} else if (key.isWritable()) {
    // a channel is ready for writing
}
...
```

Industrieel Ingenieur Informatica, UGent

SocketChannel: aanmaak - lezen

15

```
// aanmaken
SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(
    new InetSocketAddress("http://jenkov.com", 80));

// lezen
ByteBuffer buf = ByteBuffer.allocate(48);
int bytesRead = socketChannel.read(buf);
```

Industrieel Ingenieur Informatica, UGent

SocketChannel: schrijven - sluiten

16

```
// buffer met info
String newData = "New String to write to file...";
ByteBuffer buf = ByteBuffer.allocate(48);
buf.clear();
buf.put(newData.getBytes());
// data schrijven
buf.flip();
while(buf.hasRemaining()) {
    socketChannel.write(buf);
}
// afsluiten
socketChannel.close();
```

Industrieel Ingenieur Informatica, UGent

ServerSocketChannel: aanmaak

17

```
ServerSocketChannel serverSocketChannel =
    ServerSocketChannel.open();

serverSocketChannel.socket().bind(new InetSocketAddress(9999));

while(true){
    SocketChannel socketChannel = serverSocketChannel.accept();

    //do something with socketChannel...
}

serverSocketChannel.close();
```

Industrieel Ingenieur Informatica, UGent

Voorbeeld

18

- ChatServerProgram.java
- ChatServer.java

Industrieel Ingenieur Informatica, UGent

Hoofdstuk 6

JMS

JAVA MESSAGE SERVICE (JMS)

Veerle Ongenaë

Overzicht

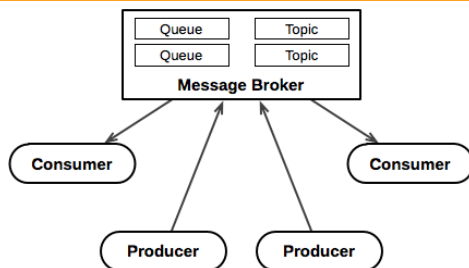
2

- Wat is JMS?
- JMS API
 - Configuratie
 - Producers
 - Consumers
 - Listeners
 - Subscriptions
 - Messages
 - Browsers

Industrieel Ingenieur Informatica, UGent

Messaging

3



<http://torquebox.org/builds/html-docs/messaging.html>

Industrieel Ingenieur Informatica, UGent

JMS

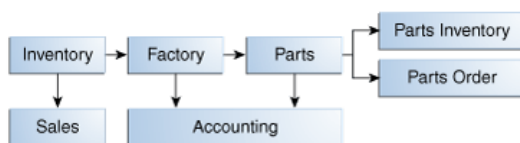
4

- Java Message Service
 - Berichten
 - Asynchroon
 - Betrouwbaar

Industrieel Ingenieur Informatica, UGent

Gebruik JMS

5



<https://docs.oracle.com/javase/7/tutorial/jms-concepts001.htm#BNCDR>

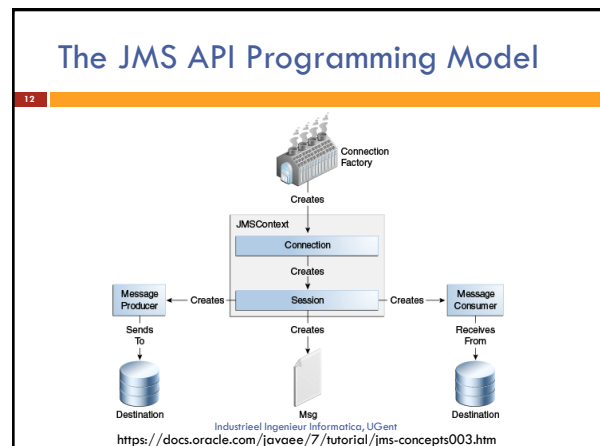
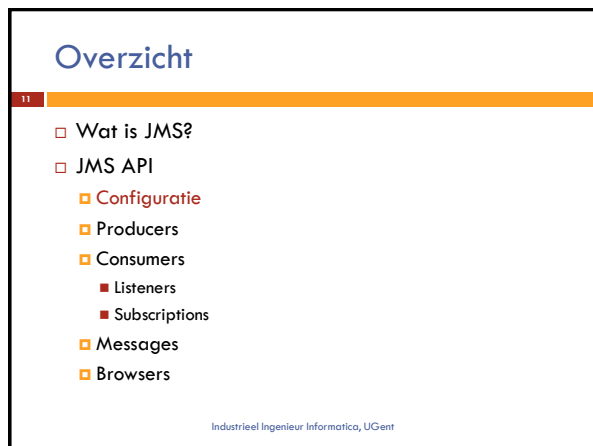
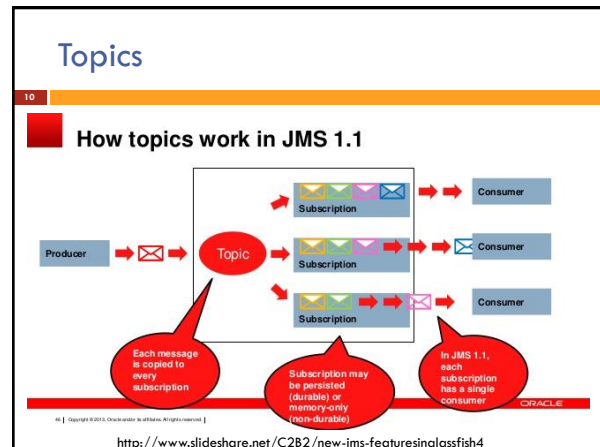
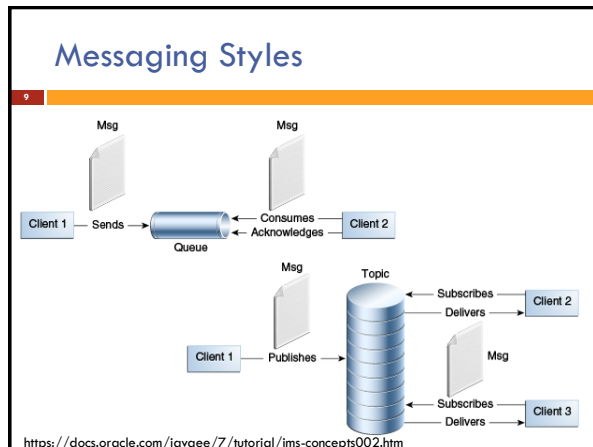
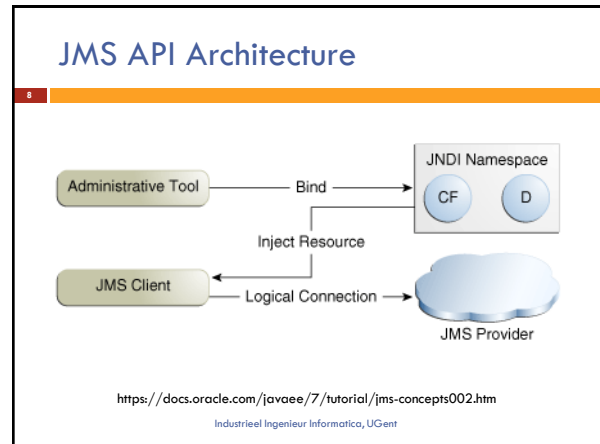
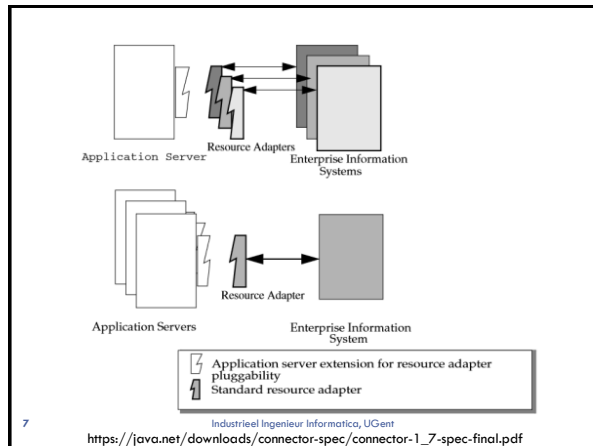
Industrieel Ingenieur Informatica, UGent

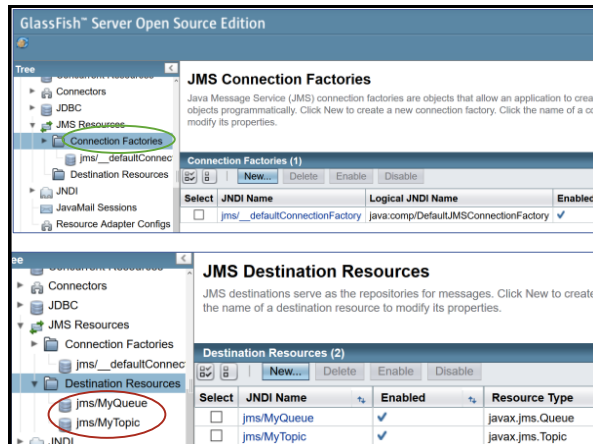
JMS versus J2EE

6

- JMS API deel van J2EE
- Verschillende componenten kunnen berichten sturen en ontvangen
- Clients kunnen luisteren naar berichten
- Combineerbaar met JTA (Java Transaction API)

Industrieel Ingenieur Informatica, UGent





JMS Administered Objects

```

import javax.annotation.Resource;
import javax.jms.ConnectionFactory;
import javax.jms.Topic;
import javax.jms.Queue;

public class Producer {
    @Resource(lookup = "java:comp/DefaultJMSConnectionFactory")
    private static ConnectionFactory connectionFactory;
    @Resource(lookup = "jms/MyQueue")
    private static Queue queue;
    @Resource(lookup = "jms/MyTopic")
    private static Topic topic;
}

```

Overzicht

- Wat is JMS?
- JMS API
 - Configuratie
 - Producers
 - Consumers
 - Listeners
 - Subscriptions
 - Messages
 - Browsers

Industrieel Ingenieur Informatica, UGent

JMSContext

```

try (JMSContext context = connectionFactory.createContext()); {
    ... // berichten maken, browsen, sturen of ontvangen
} catch (JMSRuntimeException e) {
    ...
}

```

JMS Message Producers

```

try (JMSContext context = connectionFactory.createContext()); {
    int count = 0;
    for (int i = 0; i < NUM_MSGS; i++) {
        String message = "This is message " + (i + 1)
            + " from producer";
        context.createProducer().send(dest, message);
        count += 1;
    }
    //Send a non-text control message indicating end of messages.
    context.createProducer().send(dest, context.createMessage());
} catch (JMSRuntimeException e) {
    ...
}

```

Producer

```

C:\Users\vongena\Documents\Gedistribueerde toepassingen\voorbeelden\jms\Producer
r>appliance -client dist\producer.jar queue 3
okt 26, 2015 2:13:18 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
okt 26, 2015 2:13:18 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9
-b) Compile: July 29 2014 1229
okt 26, 2015 2:13:18 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMO
TE, connection mode is TCP
okt 26, 2015 2:13:18 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Destination type is queue
Sending message: This is message 1 from producer
Sending message: This is message 2 from producer
Sending message: This is message 3 from producer
Text messages sent: 3

```

Industrieel Ingenieur Informatica, UGent

Overzicht

19

- Wat is JMS?
- JMS API
 - Configuratie
 - Producers
 - Consumers
 - Listeners
 - Subscriptions
 - Messages
 - Browsers

Industrieel Ingenieur Informatica, UGent

JMS Message Consumers

20

```
try (JMSContext context = connectionFactory.createContext()); {
    JMSConsumer consumer = context.createConsumer(dest);
    while (true) {
        Message m = consumer.receive(1000);
        if (m != null) {
            if (m instanceof TextMessage) {
                System.out.println("Reading message: " +
                    m.getBody(String.class));
            } else {break;}
        }
    }
} catch (JMSRuntimeException e) {
    ...
}
```

Consumer: voorbeeld

21

```
C:\Users\vongenae\Documents\gedistribueerde toepassingen\voorbeelden\jms\SynchCo
nsumer>appclient -client dist\SynchConsumer.jar queue
pct 26, 2015 2:17:08 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
pct 26, 2015 2:17:09 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9
-b) Compile: July 29 2014 1229
pct 26, 2015 2:17:09 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMO
TE, connection mode is TCP
pct 26, 2015 2:17:09 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Destination type is queue
Reading message: This is message 1 from producer
Reading message: This is message 2 from producer
Reading message: This is message 3 from producer
Messages received: 3
```

Industrieel Ingenieur Informatica, UGent

Queue met 2 consumers

22

```
C:\Users\vongenae\Documents\gedistribueerde toepassingen\voorbeelden\jms\SynchCo
nsumer>appclient -client dist\SynchConsumer.jar queue
pct 26, 2015 2:49:26 PM org.hibernate.validator.int
INFO: HV000001: Hibernate Validator 5.0.0.Final
pct 26, 2015 2:49:26 PM com.sun.messaging.jms.ra.R
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9
-b) Compile: July 29 2014 1229
pct 26, 2015 2:49:26 PM com.sun.messaging.jms.ra.R
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMO
TE, connection mode is TCP
pct 26, 2015 2:49:26 PM com.sun.messaging.jms.ra.R
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Destination type is queue
Reading message: This is message 1 from producer
Reading message: This is message 2 from producer
Reading message: This is message 3 from producer
Reading message: This is message 4 from producer
Reading message: This is message 5 from producer
Reading message: This is message 6 from producer
Reading message: This is message 7 from producer
Reading message: This is message 8 from producer
Reading message: This is message 9 from producer
Reading message: This is message 10 from producer
Reading message: This is message 11 from producer
Reading message: This is message 12 from producer
Messages received: 12
```

Industrieel Ingenieur Informatica, UGent

Overzicht

23

- Wat is JMS?
- JMS API
 - Configuratie
 - Producers
 - Consumers
 - Listeners
 - Subscriptions
 - Messages
 - Browsers

Industrieel Ingenieur Informatica, UGent

Asynchronous consumers

24

```
try (JMSContext context = connectionFactory.createContext()); {
    JMSConsumer consumer = context.createConsumer(dest);
    TextListener listener = new TextListener();
    consumer.setMessageListener(listener);
} catch (JMSRuntimeException e) {
    ...
}
```

JMS Message Listeners

```

25
public class TextListener implements MessageListener {
    @Override
    public void onMessage(Message m) {
        try {
            if (m instanceof TextMessage) {
                ...
            } else {
                ...
            }
        } catch (JMSException | JMSRuntimeException e) {
            ...
        }
    }
}

```

Consumer: voorbeeld

```

26
C:\Users\vongenae\Documents\Gedistribueerde toepassingen\voorbeelden\jms\AsynchC
onsumer>apclient -client dist\AsynchConsumer.jar topic
okt 26, 2015 2:29:37 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
okt 26, 2015 2:29:37 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9
-b) Compile: July 29 2014 1229
okt 26, 2015 2:29:37 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMO
TE, connection mode is TCP
okt 26, 2015 2:29:37 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Destination type is topic
To end program, enter Q or q, then <return>

```

Industrieel Ingenieur Informatica, UGent

```

To end program, enter Q or q, then <return>
Reading message: This is message 1 from producer
Reading message: This is message 2 from producer
Reading message: This is message 3 from producer
Reading message: This is message 4 from producer
Reading message: This is message 5 from producer
Reading message: This is message 6 from producer
Reading message: This is message 7 from producer
Reading message: This is message 8 from producer
Reading message: This is message 9 from producer
Reading message: This is message 10 from producer
Reading message: This is message 11 from producer
Reading message: This is message 12 from producer
Reading message: This is message 13 from producer
Reading message: This is message 14 from producer
Reading message: This is message 15 from producer
Reading message: This is message 16 from producer
Reading message: This is message 17 from producer
Message is not a TextMessage
Reading message: This is message 1 from producer
Reading message: This is message 2 from producer
Reading message: This is message 3 from producer
Reading message: This is message 4 from producer
Reading message: This is message 5 from producer
Reading message: This is message 6 from producer

```

27

Message Properties en selectors

```

28
Message msg = ...

// Setting message properties
msg.setStringProperty("title", "Thriller");
msg.setIntProperty("releaseYear", 1982);

JMSContext ctx = ...
MessageConsumer consumer
    = ctx.createConsumer(queue, "releaseYear < 1980");

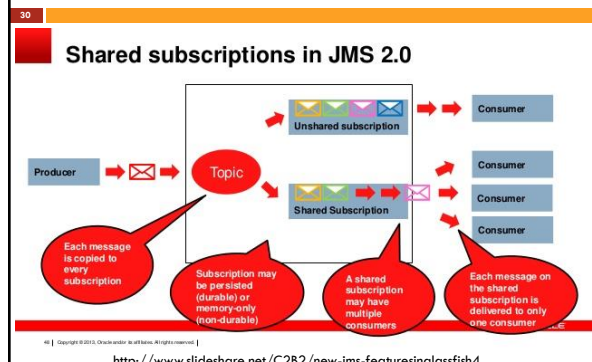
```

Overzicht

- Wat is JMS?
- JMS API
 - Configuratie
 - Producers
 - Consumers
 - Listeners
 - Subscriptions
 - Messages
 - Browsers

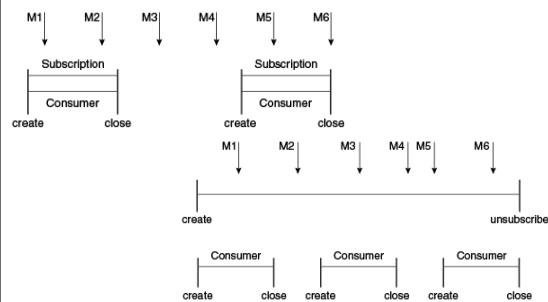
Industrieel Ingenieur Informatica, UGent

Consuming Messages from Topics



Durable Subscriptions

31



Durable Subscriptions - aanmelden

32

```

try (
    JMSContext context = durableConnectionFactory.createContext()
    {
        consumer = context.createDurableConsumer(topic, "MakeItLast");
        listener = new TextListener();
        consumer.setMessageListener(listener);
        ...
    } catch (JMSRuntimeException e) {...}
  
```

Durable Subscriptions - uitschrijven

33

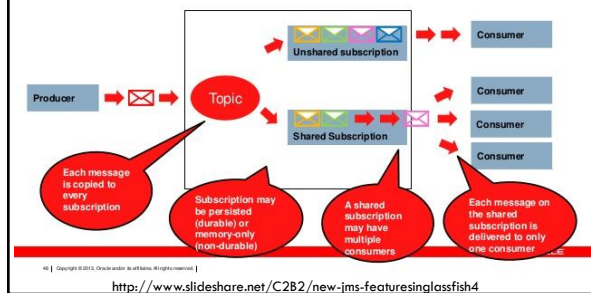
```

try (
    JMSContext context = durableConnectionFactory.createContext()
    {
        context.unsubscribe("MakeItLast");
    } catch (JMSRuntimeException e) {...}
  
```

Shared Subscriptions

34

Shared subscriptions in JMS 2.0



Shared Subscriptions

35

```

try (JMSContext context = connectionFactory.createContext()) {
    consumer = context.createSharedConsumer(topic, "SubName");

    listener = new TextListener();
    consumer.setMessageListener(listener);

    ...
} catch (JMSRuntimeException e) { ... }
  
```

Shared Durable Subscriptions

36

```

try (JMSContext context = connectionFactory.createContext()) {
    consumer
        = context.createSharedDurableConsumer(topic, "MakeItLast");

    listener = new TextListener();
    consumer.setMessageListener(listener);

    ...
} catch (JMSRuntimeException e) { ... }
  
```

Overzicht

37

- Wat is JMS?
- JMS API
 - Configuratie
 - Producers
 - Consumers
 - Listeners
 - Subscriptions
 - Messages
 - Browsers

Industrieel Ingenieur Informatica, UGent

Header Field	Description	Set By
JMSDestination	Destination to which the message is being sent	JMS provider <code>send</code> method
JMSDeliveryMode	Delivery mode specified when the message was sent (see Specifying Message Persistence)	JMS provider <code>send</code> method
JMSDeliveryTime	The time the message was sent plus the delivery delay specified when the message was sent (see Specifying a Delivery Delay)	JMS provider <code>send</code> method
JMSExpiration	Expiration time of the message (see Allowing Messages to Expire)	JMS provider <code>send</code> method
JMSPriority	The priority of the message (see Setting Message Priority Levels)	JMS provider <code>send</code> method
JMSMessageID	Value that uniquely identifies each message sent by a provider	JMS provider <code>send</code> method
JMSTimestamp	The time the message was handed off to a provider to be sent	JMS provider <code>send</code> method
JMSCorrelationID	Value that links one message to another; commonly the <code>JMSMessageID</code> value is used	Client application
JMSReplyTo	Destination where replies to the message should be sent	Client application
JMSType	Type identifier supplied by client application	Client application
JMSRedelivered	Whether the message is being redelivered	JMS provider prior to delivery

Message Body

39

Message Type	Body Contains
TextMessage	A <code>java.lang.String</code> object (for example, the contents of an XML file).
MapMessage	A set of name-value pairs, with names as <code>String</code> objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.
ByteMessage	A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.
StreamMessage	A stream of primitive values in the Java programming language, filled and read sequentially.
ObjectMessage	A <code>Serializable</code> object in the Java programming language.
Message	Nothing. Composed of header fields and properties only. This message type is useful when a message body is not required.

Berichten aanmaken

40

```
TextMessage message = context.createTextMessage();
message.setText(msg_text);    // msg_text is a String
context.createProducer().send(message);
```

```
String message = "This is a message";
context.createProducer().send(dest, message);
```

```
context.createProducer().send(dest,
    context.createMessage());
```

Industrieel Ingenieur Informatica, UGent

Berichten ontvangen

41

```
Message m = consumer.receive();
if (m instanceof TextMessage) {
    String message = m.getBody(String.class);
    System.out.println("Reading message: " + message);
} else {
    // Handle error or process another message type
}
```

```
String message = consumer.receiveBody(String.class);
```

Overzicht

42

- Wat is JMS?
- JMS API
 - Configuratie
 - Producers
 - Consumers
 - Listeners
 - Subscriptions
 - Messages
 - Browsers

Industrieel Ingenieur Informatica, UGent

JMS Message Browsers

```

43
try (JMSContext context = connectionFactory.createContext()); {
    QueueBrowser browser = context.createBrowser(queue);
    Enumeration msgs = browser.getEnumeration();

    if (!msgs.hasMoreElements()) {
        System.out.println("No messages in queue");
    } else {
        while (msgs.hasMoreElements()) {
            Message tempMsg = (Message) msgs.nextElement();
            System.out.println("\nMessage: " + tempMsg);
        }
    }
} catch (JMSRuntimeException e) { ... }

```

```

44
C:\Users\vongenae\Documents\Gedistribueerde toepassingen\voorbeelden\jms\Message
Browser>appclient -client dist\MessageBrowser.jar
okt 26, 2015 2:37:23 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
okt 26, 2015 2:37:23 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9
-b) Compile: July 29 2014 1229
okt 26, 2015 2:37:23 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMO
TE, connection mode is TCP
okt 26, 2015 2:37:23 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
No messages in queue

```

Industrieel Ingenieur Informatica, UGent

```

45
C:\Users\vongenae\Documents\Gedistribueerde toepassingen\voorbeelden\jms\Message
Browser>appclient -client dist\MessageBrowser.jar
okt 26, 2015 2:38:28 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
okt 26, 2015 2:38:28 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9
-b) Compile: July 29 2014 1229
okt 26, 2015 2:38:28 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMO
TE, connection mode is TCP
okt 26, 2015 2:38:28 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE

Message:
Text: This is message 1 from producer
Class: com.sun.messaging.jmq.jmsclient.TextMessageImpl
getJMSMessageID(): ID:9-192.168.56.1(b2:3:ea:b1:f2:e5)-7388-1445866678820
getJMSTimestamp(): 1445866678820
getJMSCorrelationID(): null
JMSReplyTo: null
JMSDestination: PhysicalQueue
getJMSDeliveryMode(): PERSISTENT
getJMSRedelivered(): false
getJMSType(): null
getJMSExpiration(): 0

```


Bibliografie

- [Casa] Cascading style sheets. <http://www.w3.org/Style/CSS/>.
- [Casb] Cascading style sheets level 2 revision 1 (css 2.1) specification, full property table. <http://www.w3.org/TR/CSS21/propidx.html>.
- [exe] Java: Executor framework. <https://myshadesofgray.wordpress.com/2014/04/13/java-executor-framework/>.
- [HM01] Elliotte Rusty Harold and W. Scotte Means. *XML in a Nutshell, A Desktop Quick Reference*. O'Reilly, January 2001.
- [thra] Programming java threads in the real world, part 1. <https://www.javaworld.com/article/2076774/java-concurrency/programming-java-threads-in-the-real-world--part-1.html>.
- [thrb] Threads: Basic theory and libraries. <https://users.cs.cf.ac.uk/Dave.Marshall/C/node29.html>.
- [thrc] Java threading and concurrency introduction. <http://blog.jbaysolutions.com/2011/10/06/java-threading-and-concurrency-introduction/>.
- [thrd] Thread pool. https://en.wikipedia.org/wiki/Thread_pool.
- [XML] Extensible markup language (xml). <http://www.w3.org/XML/>.
- [XPa] Xpath functies. https://www.w3schools.com/xml/xsl_functions.asp.
- [XSL] The extensible stylesheet language family (xsl). <http://www.w3.org/Style/XSL/>.