HOGESCHOOL GENT
LID VAN DE ASSOCIATIE UNIVERSITEIT GENT

# Computernetwerken IV:
# de vergeten hoofdstukken

door

Sander Vrijders

Academiejaar 2011-2012

Only a life lived for others is a life worthwhile. - Albert Einstein

# Voorwoord

Deze 'cursus' kwam tot stand aangezien de cursus van het academiejaar 2011-2012, voor het vak computernetwerken IV, gegeven in de master 'Industrieel Ingenieur Informatica', voor een groot deel bestond uit tutorials, afgeprint van het internet. De 'cursus' omvat de vergeten hoofdstukken, hierdoor start de hoofdstuknummering vanaf 3.9.

Dit werd volledig in LaTeX geschreven, de broncode is te vinden op `https://github.com/sandervrijders/Computernetwerken-IV`. Het gebruikte LaTeX template, en een zeer interessante cursus kunnen gevonden worden op `http://latex.ugent.be/templates.php`.

Sander Vrijders, november 2011

# Inhoudsopgave

## 3.9 JSTL

JSTL staat voor JSP Standard Tag Library. Het draagt bij tot een verdere scheiding van view en model. Met JSTL kan men bijvoorbeeld conditionele uitdrukkingen formuleren, XML documenten gebruiken, internationalisatie toepassen, met databanken connecteren, en nog veel meer. JSTL is geen deel van de JSP specificatie. Het is een aparte library. Indien men in Netbeans werkt, moet deze dus toegevoegd worden aan het project.

Core is het meest gebruikte onderdeel van JSTL. Core omvat onder andere het gebruik van variabelen, flow control, het behandelen van een URL.

Bovenaan elke jsp pagina voegt men het volgende toe indien men hiervan wil gebruik maken:

Listing 3.1: Verwijzing naar de library

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

Men zegt hier dus dat men van het core onderdeel van JSTL[1] zal gebruik maken, en dit door in een tag telkens de prefix 'c' te gebruiken.

### 3.9.1 Variabelen

Onder het gebruik van variabelen verstaan we het gebruik van de tags remove en set. Met set kan men een EL variabele op 1 van de scopes zetten (page, request, session, of application). Met remove wordt ze er weer afgehaald.

Listing 3.2: Voorbeeld van remove en set

```
<c:set var="foo" scope="session" value="param.remove"/>
<c:remove var="cart" scope="session"/>
```

[1] Andere mogelijkheden zijn XML, i18n, database, functions

### 3.9.2 Flow control

Flow control is het gebruik van conditionele uitdrukkingen. Zo kan je gebruik maken van: if, foreach, choose. Choose is eigenlijk een switch-case die we onder andere kennen uit de programmeertaal C++.

Listing 3.3: Voorbeeld van choose

```
<c:choose>
  <c:when test="${customer.category ==   t r i a l  }" >
    ...
  </c:when>
  <c:when test="${customer.category ==   m e m b e r  }" >
    ...
  </c:when>
  <c:otherwise>
    ...
  </c:otherwise>
</c:choose>
```

Hieronder nog een voorbeeld van if en foreach.

Listing 3.4: Voorbeeld van if

```
<c:if test="${!empty param.Add}">
  ...
</c:if>
```

Listing 3.5: Voorbeeld van foreach

```
 <c:forEach var="item" items="${sessionScope.cart.items}">
  ...
  <tr>
  <td align="right" bgcolor="#ffffff">
    ${item.quantity}
  </td>
  ...
</c:forEach>
```

### 3.9.3 URLs

Men kan hiermee documenten inladen, redirecten, URLs herschrijven. In de volgende listing wordt de XML file books.xml ingeladen in de variabele xml, die dan geparst wordt.

Listing 3.6: Voorbeeld van inladen van een xml file

```
<c:import url="/books.xml" var="xml" />
<x:parse doc="${xml}" var="booklist" scope="application" />
```

Als men een URL wil schrijven in een JSP pagina kan men gebruik maken van de tag url. Indien cookies niet ondersteund worden, wordt het JSESSIONID automatisch toegevoegd.

Listing 3.7: Voorbeeld van c:url

```
<c:url var="url" value="/catalog" >
  <c:param name="Add" value="${bookId}" />
</c:url>
```

Men kan van de tag redirect gebruik maken om de gebruiker naar een andere pagina te sturen.

### 3.9.4 Andere functies

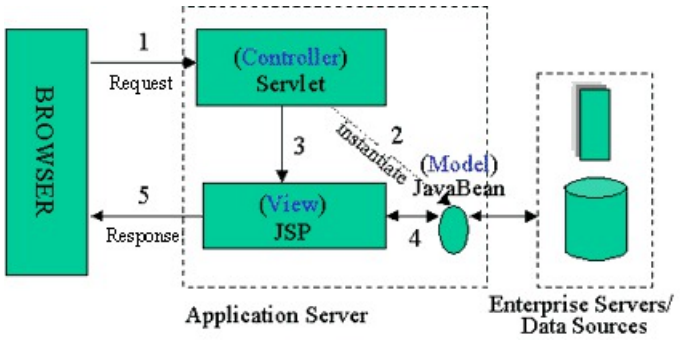
Verder kan men ook nog gebruik maken van c:catch en c:out. c:catch zorgt voor foutafhandeling. De exceptie die gegooid werd, kan men in de variable die in var staat stockeren. c:out negeert HTML karakters, zodanig dat er niet aan cross-site scripting gedaan kan worden. Het is dus aan te raden dat men dit gebruikt, telkens men een EL variable gebruikt.

# Hoofdstuk 4

# Model-View-Controller

## 4.1 Inleiding

MVC ofte Model-View-Controller kennen we reeds uit de cursus "GUIs". Ook in webapplicaties wordt dit toegepast. Hierbij schrijft men het model in Java, de View wordt geconstrueerd met behulp van JSP. De controller die zorgt voor de afhandeling van requests implementeert men met een servlet.




## 4.2 Implementatie met RequestDispatcher

Aan de hand van de meegestuurde request parameters bepaalt men welke view moet opgeroepen worden. Met een object van de klasse RequestDispatcher stuurt men de aanvraag dan effectief door naar de juiste JSP pagina. Het is mogelijk, en zelfs aan te raden, de eigenlijke JSP pagina's te plaatsen in de WEB-INF map. Hierdoor is het niet mogelijk om de JSP pagina's direct op te vragen, de webserver kan echter wel in deze map. Een voorbeeld is te zien in listing 4.1.

Listing 4.1: Een voorbeeld van ResquestDispatcher

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException {
  String operation = request.getParameter("operation");
  if (operation == null) {
    operation = "unknown";
  }
  String address;
  if (operation.equals("order")) {
    address = "/WEB-INF/Order.jsp";
  } else if (operation.equals("cancel")) {
    address = "/WEB-INF/Cancel.jsp";
  } else {
    address = "/WEB-INF/UnknownOperation.jsp";
  }
  RequestDispatcher dispatcher = request.getRequestDispatcher(address);
  dispatcher.forward(request, response);
}
```

Soms is het wenselijk om in plaats van RequestDispatcher, sendRedirect te gebruiken. Dit is interessant wanneer de gebruiker de JSP pagina ook rechtstreeks moet kunnen raadplegen. Dit betekent wel dat de pagina ook zonder extra invoer de juiste data bevat, bijvoorbeeld dankzij default waarden. Let er wel op dat je de JSP pagina dan niet in de WEB-INF map plaatst.

Om er voor te zorgen dat de aanvragen door het controller servlet worden afgehandeld, voegt men enkele regels toe aan web.xml. In listing 4.2 worden bijvoorbeeld alle URL's van de vorm *.do doorgestuurd naar het controlerservlet.

Listing 4.2: web.xml

```
<servlet>
  <servlet-name>ControleServlet</servlet-name>
  <servlet-class>be.hogent.iii.steden.web.ControleServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ControleServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

## 4.3 Gevolgen voor JSP

### 4.3.1 Gebruik van Beans

Aangezien de JSP pagina's slechts de view voorstellen, is het niet de bedoeling dat er hier nog objecten aangemaakt worden. Dit is de taak van het controllerservlet. Gebruik dus:

```
<jsp:usebean ... type="package.class" scope="request"/>
```

In plaats van:

```
<jsp:usebean ... class="package.class" scope="request"/>
```

De waarden van opgehaalde objecten mogen ook niet aangepast worden. Dankzij het scope attribuut van useBean kan men ook kiezen vanwaar men de Bean afhaalt (request, session of application).

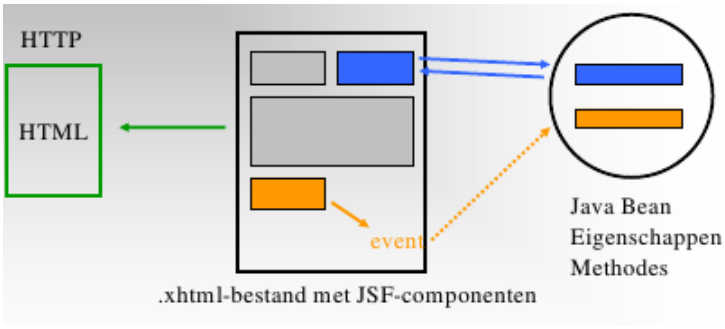
### 4.3.2 Relatieve padnamen

Als men in de JSP pagina gebruik maakt van relatieve padnamen (bv. om een stylesheet op te halen), moet men er rekening mee houden dat het gebruikte startpad het pad is van het servlet. Het request object verandert immers niet. Om problemen te vermijden, gebruikt men dus best steeds een absoluut pad.

# Hoofdstuk 5

# Java Server Faces

JSF staat voor Java Server Faces. Het is gebouwd bovenop Servlets. Men poogt hetzelfde te doen als bij een combinatie van JSP en JSTL, namelijk een scheiding van model en view. JSF is echter uitgebreider. Vroeger werd dit ingebed in JSP pagina's, maar nu gebruikt men facelets. Deze hebben de extensie .xhtml.




## 5.1 Componenten en beans

### 5.1.1 Componenten

De facelets bestaan uit gewone HTML, JSF componenten (en eventueel JSTL componenten). JSF componenten worden opgedeeld in 3 categoriën: core JSF, HTML JSF en facelets tags. Deze categoriën hebben elk hun eigen xml namespace. Als men ze dus wil gebruiken, moet men bovenaan de pagina het volgende toevoegen.

Listing 5.1: XML namespaces van JSF

```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:f="http://java.sun.com/jsf/core"
 xmlns:ui="http://java.sun.com/jsf/facelets">

</html>
```

De facelets tags gebruikt men als men met templates werkt. De HTML JSF omvat alle grafische componenten. Core JSF wordt gebruikt om bepaalde acties uit te voeren, om te valideren, om mogelijkheden te specifiëren.

Enkele veel voorkomende tags zijn:

- column: kolom in tabel
- commandButton: submit-knop
- commandLink: link
- dataTable: tabel
- form: formulier
- graphicImage: figuur
- inputHidden: verborgen veld
- inputSecret: wachtwoordveld
- inputText: tekstveld
- inputTextarea: invoer meerdere lijnen
- message: bericht
- messages: berichten
- outputLabel: label
- outputLink: link (zonder actie)
- outputFormat: bericht

- outputText: 1 lijn tekst
- panelGrid: groepeert componenten in tabelvorm
- selectBoolean: checkbox
- selectItem: 1 item in lijst
- selectItems: meerdere items in lijst
- selectManyCheckbox: lijst checkboxen
- selectManyListbox: lijst, meerdere items selecteren mogelijk
- selectManyMenu: keuzelijst, meerdere items selecteren mogelijk
- selectOneListbox:lijst, 1 item selecteren
- selectOneMenu: keuzelijst, 1 item selecteren
- selectOneRadio: lijst radiobuttons

Net zoals bij JSTL kan men gebruik maken van EL (Expression Language). Dit kan men op twee manieren doen.

1. Indien men gebruik maakt van ${...} wordt de EL uitdrukking meteen uitgevoerd, wanneer de pagina wordt geladen.
2. Anderzijds kan men ook gebruik maken van #{...}. Hierdoor wordt de uitdrukking niet meteen uitgevoerd, maar op het gepaste moment. JSF gebruikt dit, omdat de levensloop meerdere cycli heeft. Om er zeker van te zijn dat de uitdrukking pas opgeroepen wordt wanneer de waarden beschikbaar zijn, is dit dus vereist. Men past twee soorten binding toe.
    - value-binding: De eigenschappen van een bean (zie 5.1.2) worden toegekend.
    - method-binding: Een methode wordt toegekend (bijvoorbeeld bij het klikken op een knop zal de methode dan uitgevoerd worden)

De tags worden op de server omgezet in UI-componenten. Deze zorgen voor de eigenlijke interactie met de gebruiker. Als de pagina moet getoond worden zullen deze (eventueel) data uit

beans halen, en als de pagina ingediend wordt, kunnen ze de data opslaan in beans. Bij deze UI-componenten kan men event listeners (5.3), validators (5.7) en convertors (5.8) registreren.

Een overzicht van de beschikbare UI-componenten:

- UIColumn: kolom in UIData
- UICommand: actie uitvoeren
- UIData: tonen groep data
- UIForm: groep componenten, HTML-form
- UIGraphic: figuur
- UIInput: invoer, afgeleid van UIOutput
- UIMessage: bericht tonen
- UIMessages: berichten tonen
- UIOutput: uitvoer op pagina
- UIPanel: zorgt voor layout kindcomponenten
- UIParameter: parameter
- UISelectBoolean: invoer logische waarde, afgeleid van UIInput
- UISelectItem: een item
- UISelectItems: meerdere items
- UISelectMany: invoer meerdere items uit keuze, afgeleid UIInput
- UISelectOne: invoer item uit keuze, afgeleid UIInput
- UIViewRoot: basis componentboom

De UI-component bevat niet de manier waarop hij gerendered moet worden. Hiervoor is er een aparte rendererklasse vereist.

### 5.1.2 Beans

JavaBeans zijn herbruikbare softwarecomponenten voor Java. Het zijn Javaklassen die een bepaalde conventie volgen. Ze worden gebruikt om meerdere objecten in 1 object te bundelen (de bean). Een JavaBean is een Java object dat serializable is, een constructor zonder argumenten heeft, en zijn dataleden beschikbaar maakt via getters en setters. Ook kan men er methodes aan toevoegen.

Er zijn twee manieren om te kennen te geven dat de klasse een JavaBean is. Men kan gebruik maken van annotaties, of men kan de configuratie declareren in faces-config.xml.

Een voorbeeld met annotaties:

Listing 5.2: annotaties

```
@ManagedBean
@SessionScoped
public class GebruikersInfo {     }
```

Met ManagedBean maakt men duidelijk dat het hier om een JavaBean gaat. SessionScoped zegt dat de Bean op de sessie scope moet gezet worden. Andere mogelijkheden zijn:

- Application (@ApplicationScoped)
- Session (@SessionScoped)
- View (@ViewScoped)
- Request (@RequestScoped)
- None (@NoneScoped)
- Custom (@CustomScoped)

Als men gebruik maakt van faces-config dan moet men daar alles in declareren, een voorbeeld hiervan is te zien in 5.3

Listing 5.3: faces-config.xml

```
 <managed-bean>
  <managed-bean-name>customer</managed-bean-name>
  <managed-bean-class>   </managed-bean-class>
  <managed-bean-scope> request </managed-bean-scope>
  <managed-property>
    <property-name>mailingAddress</property-name>
    <value>#{addressBean}</value>
  </managed-property>
</managed-bean>
<managed-bean>
  <managed-bean-name>addressBean</managed-bean-name>
  <managed-bean-class>     </managed-bean-class>
  <managed-bean-scope> none </managed-bean-scope>
</managed-bean>
```

Zoals men in 5.3 ook kan zien, kan men aan een property meteen een waarde geven. Dit kan een constante zijn, of dus zoals in 5.3 een EL uitdrukking. Dit moet dan uiteraard wel een object in dezelfde, of een ruimere, scope zijn. Als het object nieuw moet aangemaakt worden, wordt de scope 'none' verondersteld.

### 5.1.3 Configuratie

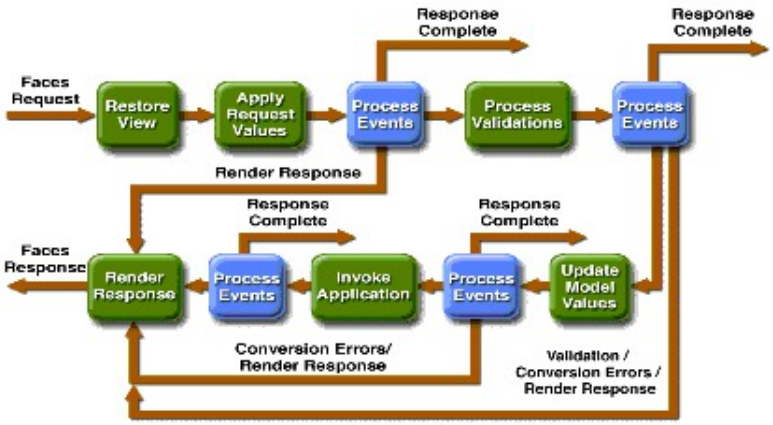
Als dit allemaal gebeurd is, moet men enkel nog in web.xml de juiste wijzigingen aanbrengen. Indien men NetBeans gebruikt, wordt er hier al een deel automatisch geconfigureerd. Een voorbeeld web.xml implementatie voor JSF 2.0 is te zien in 5.4

Listing 5.4: web.xml voor JSF 2.0

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">
```

```
    <display-name>JavaServerFaces</display-name>

    <!-- Change to "Production" when you are ready to deploy -->
    <context-param>
      <param-name>javax.faces.PROJECT_STAGE</param-name>
      <param-value>Development</param-value>
    </context-param>

    <!-- Welcome page -->
    <welcome-file-list>
      <welcome-file>faces/hello.xhtml</welcome-file>
    </welcome-file-list>

    <!-- JSF mapping -->
    <servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Map these files with JSF -->
    <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>

  </web-app>
```

## 5.2 Levensloop




De levensloop van een JSF applicatie bestaat uit 6 fases:

- Restore View Phase
- Apply Request Values Phase
- Process Validations Phase
- Update Model Values Phase
- Invoke Application Phase
- Render Response Phase

### 5.2.1 Restore View Phase

Wanneer een aanvraag voor een JavaServer Faces pagina is gedaan, bijvoorbeeld wanneer er op een link of een knop wordt geklikt, begint de JavaServer Faces implementatie met de Restore View Phase.

Tijdens deze fase bouwt de JavaServer Faces implementatie de weergave van de pagina, worden er event handlers en validators bij de componenten geregistreerd, en slaat de huidige view in de FacesContext op, dat alle informatie die nodig is om een aanvraag te verwerken bevat. Alle de component tags, event handlers, converters en validators hebben toegang tot de FacesContext

instantie.

Indien het verzoek voor de pagina een eerste aanvraag is, zal de JavaServer Faces implementatie zorgen voor een lege view tijdens deze fase en zal er voortgegaan worden naar de Render Response fase, waarin de lege view is gevuld met de onderdelen waarnaar wordt verwezen door de tags op de pagina .

Indien het verzoek voor de pagina een postback is, bestaat er reeds een view van deze pagina. Tijdens deze fase zal de JavaServer Faces implementatie de vorige view herstellen met behulp van de informatie over de toestand die op de client of de server te vinden is.

### 5.2.2 Apply Request Values Phase

Nadat de boomstructuur van componenten is hersteld, elke component in de boom haalt zijn nieuwe waarde van de request parameters met behulp van de decode methode. De waarde wordt vervolgens lokaal opgeslagen op de component. Indien de conversie van de waarde mislukt, wordt er een foutmelding in verband met de component gegenereerd en op de FacesContext geplaatst (in een wachtrij). Dit bericht wordt weergegeven tijdens de Render Response Phase, samen met eventuele validatie fouten als gevolg van de Process Validations Phase.

Als er decode methodes of event listeners renderResponse oproepen op de huidige FacesContext dan zal JSF meteen naar de Render Response Phase springen.

Als er foutmeldingen staan in de wachtrij van de FacesContext, zal JSF dit melden aan geïnteresseerde luisteraars.

Als sommige onderdelen op de pagina hebben hun directe attributen (zie De onmiddellijke Attribute) ingesteld op true, dan is de validatie, conversie en afspraken verbonden aan deze onderdelen zullen worden verwerkt tijdens deze fase.

Als er onderdelen op de pagina zijn die true hebben staan voor hun Immediate attribuut, zal de validatie, conversie, en events met betrekking tot deze componenten in deze fase afgehandeld worden.

Op dit punt kan de applicatie FacesContext.responseComplete oproepen, indien de aanvraag geredirect moet worden, of indien er geen componenten op de pagina zijn.

Aan het eind van deze fase hebben de onderdelen hun nieuwe waarden, en berichten en events staan in de wachtrij op de FacesContext.

### 5.2.3 Process Validations Phase

Tijdens deze fase zal de JavaServer Faces implementatie alle validators geregistreerd op de componenten in de boom verwerken. Het onderzoekt de component attributen dat de regels voor de validatie opleggen en vergelijkt deze regels met de lokale waarde opgeslagen voor de component.

Als de lokale waarde ongeldig is, zal de JavaServer Faces implementatie een foutmelding op de FacesContext zetten. Er wordt meteen doorgegaan naar de Render Response Phase, zodat de pagina opnieuw wordt weergegeven met de weergegeven foutmeldingen. Als er conversie fouten waren uit de Apply Request Values Phase, worden deze ook getoond.

Als er validatie methoden of eventlisteners renderResponse aanriepen op de huidige FacesContext, zal er meteen naar de Render Response Phase gegaan worden.

Op dit punt kan de applicatie FacesContext.responseComplete oproepen, indien de aanvraag geredirect moet worden, of indien er geen componenten op de pagina moeten zijn.

Als er meldingen staan in de wachtrij van de FacesContext, zal JSF dit melden aan geïnteresseerde luisteraars.

### 5.2.4 Update Model Values Phase

Als blijkt dat de gegevens geldig zijn, kan JSF de component boom overlopen en de bijbehorende server-side objecteigenschappen instellen op de lokale waarden van de componenten. De JavaServer Faces implementatie zal de properties van de JavaBean alleen updaten als de waarde afkomstig is van het attribuut van een input component. Als de lokale data niet kan worden geconverteerd naar het type van de property, zal er meteen naar de Render Response Phase worden voortgegaan. Zodat de pagina opnieuw wordt weergegeven met corresponderende foutboodschappen. Dit is vergelijkbaar met wat er gebeurt met validatie errors.

Als er updateModels methoden of eventlisteners renderResponse aanriepen op de huidige FacesContext, zal er meteen naar de Render Response Phase gegaan worden.

Op dit punt kan de applicatie FacesContext.responseComplete oproepen, indien de aanvraag geredirect moet worden, of indien er geen componenten op de pagina moeten zijn.

Als er meldingen staan in de wachtrij van de FacesContext, zal JSF dit melden aan geïnteresseerde luisteraars.

### 5.2.5 Invoke Application Phase

Tijdens deze fase zal JSF alle events op applicatieniveau behandelen, zoals het indienen van een formulier of een koppeling naar een andere pagina.

Op dit punt kan de applicatie FacesContext.responseComplete oproepen, indien de aanvraag geredirect moet worden, of indien er geen componenten op de pagina moeten zijn.

Als de view wordt gereconstrueerd uit een voorgaande aanvraag en als een component een event heeft afgevuurd zal JSF dit melden aan geïnteresseerde luisteraars.

### 5.2.6 Render Response Phase

Tijdens deze fase zal JSF autoriteit geven aan de JSP container indien de applicatie JSP pagina's bevat. Als dit een eerste aanvraag is, zullen de componenten toegevoegd worden aan de componentboom, terwijl de JSP container de pagina uitvoert. Als dit niet een eerste aanvraag is, zijn de componenten al toegevoegd aan de boom, zodat ze niet opnieuw hoeft worden toegevoegd. In beide gevallen zullen de componenten zichzelf renderen wanneer ze door de JSP container tegengekomen worden in de pagina.

Indien de aanvraag een postback is, en er fouten optraden tijdens apply request values phase, process validations phase, of update model values phase, zal de originele pagina getoond worden. Als de pagina's berichten of berichten tags bevat, worden eventuele foutmeldingen weergegeven op de pagina.

Nadat de inhoud van de weergave wordt weergegeven, wordt de status van de antwoord opgeslagen, zodat latere verzoeken er toegang tot hebben. Het is dus beschikbaar in de restore view phase.

## 5.3 Events

Zoals hiervoor reeds gezien, worden events door componenten gegenereerd. Als de geregistreerde luisteraars verwittigd worden, wordt er informatie over de gebeurtenis meegegeven. Er zijn in totaal 3 soorten events.

1. Value-change events: veranderen waarde component
2. Action events: klikken op knoppen, links,
3. Data-model events: selecteren rij,

Om te luisteren naar een event zijn er twee mogelijke manieren. Men kan een methode van een managed bean oproepen, of men kan een luisterklasse schrijven.

Indien men een managed bean gebruikt, moet men dit als volgt duidelijk maken op de JSF pagina.

Listing 5.5: ManagedBean

```
<h:commandButton actionListener="#{gebruikersInfo.bepaalKeuze}" />
```

Men vult dus telkens het attribuut actionListener in van een bepaalde component. In de bean zelf moet men de methode dan als volgt implementeren.

Listing 5.6: ManagedBean implementatie

```
public void bepaalKeuze(ActionEvent e) {
  keuze = "Hallo " + naam + ", u koos " + thema + ".";
}
```

Hier is er sprake van een ActionEvent, een andere mogelijkheid is bijvoorbeeld ValueChangeEvent. Deze methodes kunnen een AbortProcessingException opgooien, als er iets gebeurt dat niet mag gebeuren.

Als men een luisterklasse schrijft moet men de interface javax.faces.event.ValueChangeListener of javax.faces.event.ActionListener implementeren.

Listing 5.7: Luisterklasse

```
public void processValueChange(ValueChangeEvent event) throws
    AbortProcessingException
public void processAction(ActionEvent event) throws AbortProcessingException
```

Nadien moet men deze luisterklasse dan nog meegeven met de component. Dit kan met de tags valueChangeListener of actionListener, die onderdeel zijn van het 'core' onderdeel van JSF.

Listing 5.8: "Luisterklasse meegeven"

```
<h:inputText    >
  <f:valueChangeListener type="listeners.NameChanged" />
</h:inputText>
```

## 5.4 UIData componenten

Net zoals in ASP.NET met bijvoorbeeld GridView bestaat er voor JSF een equivalente datastructuur die veel data moet kunnen bevatten. h:dataTable is zo de corresponderende tag van deze UIComponent. Met h:column kan men de layout van de kolom beschrijven. Met f:facet beschrijft men dan weer de headers (h:column tag) en de footers (na h:column tags).

## 5.5 i18n

Om taalonafhankelijkheid te bekomen, maakt men zoals zo vaak in Java, gebruik van .properties bestanden. Een .properties bestand is een gewoon tekstbestand dat per lijn een sleutel-waarde paar bevat. Commentaar kan men toevoegen na een #-teken.

In een JSP pagina moet men dan de bundle inladen, en daarna kan men er waarden uit ophalen met behulp van de sleutel (zie 5.9).

Listing 5.9: ï18n”

```
<f:loadBundle basename="be.hogent.tiwi.jsf.beans.labels" var="labels"/>
...
<f:facet name="header">
  <h:outputText value="#{labels.titel}"/>
</f:facet>
```

## 5.6 Navigatie

In JSF worden er strikte regels opgelegd in verband met de navigatie. Aan de hand van het resultaat van een methode (die bijvoorbeeld opgeroepen wordt na het klikken op een knop), kan er bepaald worden naar welke pagina men mag gaan.

Listing 5.10: Knop om door te gaan

```
<h:commandButton value= S u b m i t  action="#{cashier.submit}" />
```

Deze navigatieregels plaatst men in faces-config.xml.

Listing 5.11: Navigatie

```
<navigation-rule>
  <from-view-id>/greeting.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/response.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

Men kan in principe kiezen wat men teruggeeft met de methode (sowieso wel een string). Veelgebruikte strings zijn: Success, Failure, Login, No Results. Dit plaatst men dan in from-outcome.

## 5.7 Validatie

Er zijn meerdere manieren om te valideren. Als men simpelweg wil eisen dat een component een waarde bevat, kan men het attribuut required op true zetten.

Listing 5.12: required component

```
<h:inputText value="#{personBean.personName}" required="true"/>
```

Men kan ook het attribuut validator invullen met een bepaalde methode die voor validatie zorgt, die men zelf heeft geschreven. In de JSP pagina verwijst men hier dan naar via EL ( #(eenKlasse.controleerNaam) ).

Listing 5.13: Validator attribuut

```
public void controleerNaam(FacesContext context, UIComponent toValidate, Object
    value) {     }
```

Naast deze validatietechnieken kan men ook nog speciale tags gebruiken, binnenin de tag. Dit kan gebeuren met reeds ingebouwde validatie, of met een zelf geschreven validatiemethode.

Beschikbare ingebouwde validatietags zijn:

- validateLength-tag: valideert de lengte van een string
- validateLongRange-tag: bevat attributen Minimum en Maximum
- validateDoubleRange-tag: bevat attributen Minimum en Maximum
- validateRegEx
- validateRequired

Indien men zelf een validator wil schrijven, moet men de Validator interface implementeren.

Listing 5.14: Validator interface

```
public void validate(FacesContext context, UIComponent component, Object
    toValidate)
```

Men kan deze koppelen aan de component op de volgende manier:

Listing 5.15: Validator in JSF

```
<h:inputText ... >
  <f:validator validatorId="customValidator" /> ...
</h:inputText>
```

Men moet dan uiteraard wel in faces-config.xml aangeven waarvoor customValidator staat.

Listing 5.16: faces-config.xml voor validator

```
<validator>
  ...
  <validator-id>customValidator</validator-id>
  <validator-class>
```

```
      validators.CustomValidator
    </validator-class>
  </validator>
```

## 5.8 Conversie

Aangezien de manier van presentatie van een bepaald gegeven kan verschillen in de view en in het model, moet dit geconverteerd worden. Soms gebeurt dit impliciet, bijvoorbeeld als een int moet geconverteerd worden naar een waarde voor een h:inputText. Achter de schermen wordt hiervoor eigenlijk een IntegerConverter gebruikt.

Soms is dit echter niet wat men wilt. Men kan zelf bepalen welke converter er gebruikt wordt met de f:converter tag.

Listing 5.17: converter

```
<h:inputText value="#{LoginBean.age}" />
  <f:converter converterId="Integer" />
</h:inputText>
```

Of met het attribuut converter:

Listing 5.18: converter attribuut

```
<h:inputText converter="javax.faces.convert.IntegerConverter     />
```

Er bestaan ook nog andere tags, die bijvoorbeeld een getal omzetten naar een datum, of naar een geldeenheid.

Listing 5.19: Andere conversies

```
<h:outputText value="#{cashier.shipDate}">
 <f:convertDateTime pattern="EEEEEEEE, MMM dd, yyyy" />
</h:outputText>
</h:outputText>
<h:outputText value="#{cart.total}" >
  <f:convertNumber type="currency /> 
</h:outputText>
```

Indien men een conversie wilt die niet beschikbaar is, kan men tenslotte een eigen converter schrijven. Hiervoor implementeert men de Converter interface. Deze bevat 2 methodes, de ene voor omzetting van de presentatie naar het model en de andere vice versa.

Listing 5.20: Converter interface

```
Object getAsObject(FacesContext context, UIComponent component, String value)
String getAsString(FacesContext context, UIComponent component, Object value)
```

Men registreert deze converter in faces-config.xml.

Listing 5.21: faces-config.xml voor converter

```
<converter>
  <description>     </description>
  <converter-id>CreditCardConverter</converter-id>
  <converter-class>
    converters.CreditCardConverter
  </converter-class>
</converter>
```

Hierna kan men deze oproepen in een JSF pagina.

Listing 5.22: Gebruik eigen validator

```
<h:inputText converter="CreditCardConverter    >
  ...
</h:inputText>
```

## 5.9 Componenten

Men kan nog veel meer doen met componenten dan hiervoor reeds is gezegd. Indien men bijvoorbeeld binnenin een tag het attribuut rendered gebruikt, kan men bepalen of de component al dan niet gerendered wordt.

Listing 5.23: Rendered

```
<h:commandLink id="check"
  ...
  rendered="#{cart.numberOfItems > 0}">
  <h:outputText value="#{bundle.CartCheck}"/>
</h:commandLink>
```

In bovenstaande listing wordt er slechts tekst getoond als het winkelwagentje meer dan nul items bevat.

Met het immediate attribuut kan men dan weer bepalen in welke fase events, validaties en dergelijke worden uitgevoerd. Als dit op true staat wordt dit gedaan in de Apply Request Values Phase. Dit wordt bijvoorbeeld bij commandLink gebruikt om te voorkomen dat formuliergegevens verwerkt worden.

Met het binding attribuut kan men de component meegeven aan de JavaBean. Van hieruit kan men hem dan bewerken. Als we bijvoorbeeld een UIInput component willen meegeven kan dit als volgt. In de JSF pagina:

Listing 5.24: inputText meegeven

```
<inputText binding="#{UserNumberBean.userNoComponent}" />
```

In de JavaBean komt dan volgende methode:

Listing 5.25: Component krijgen

```
UIInput userNoComponent = null;
public void setUserNoComponent(UIInput userNoComponent) {
  this.userNoComponent = userNoComponent;
}
public UIInput getUserNoComponent() {
 return userNoComponent;
}
```

## 5.10 Template

Het is vaak wenselijk om meerdere pagina's met dezelfde structuur te hebben. Men kan een template[1] gebruiken. 1 pagina bevat de template, en de andere pagina's vullen dit template in met hun content.

In de template komt er dan op de plaats waar de eigenlijke content (hier de hoofding) moet komen:

Listing 5.26: template

```
<ui:insert name="hoofding">Hoofding</ui:insert>
```

Op de eigenlijke pagina gaat men dan als volgt te werk, als de template template.xhtml heet:

Listing 5.27: ingevulde template

```
<ui:composition template="template.xhtml">
  <ui:define name="hoofding">De titel</ui:define>
  <ui:define name="links">Menu</ui:define>
  <ui:define name="rechts">Inhoud</ui:define>
</ui:composition>
```

## 5.11 Samengestelde componenten

Indien men dezelfde componenten in een veel voorkomende combinatie gebruikt, kan men overwegen om een herbruikbare component te schrijven. Op deze manier kan men meer modulair werken.

Men definieert hiervoor eerst een interface, waarin men specifieert welke attributen, events, ... moeten opgenomen worden. Hierna komt de implementatie. Hier specifieert men de componenten die men gaat gebruiken, via attrs specifieert men wat men van de interface implementeert. Een voorbeeld is te zien hieronder.

[1] Merk de analogie op met masterpages in ASP.NET

Listing 5.28: Composite component

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:cc="http://java.sun.com/jsf/cc">

  <!-- INTERFACE -->
  <cc:interface>
    <cc:attribute name="image" required="false"/>
    <cc:attribute name="actionMethod"
             method-signature="java.lang.String action()"/>
  </cc:interface>

  <!-- IMPLEMENTATION -->
    <cc:implementation>
    <h:form>
      <h:commandLink action="#{cc.attrs.actionMethod}" immediate="true">

      <h:graphicImage value="#{cc.attrs.image}"
                styleClass="icon"/>

      </h:commandLink>
    </h:form>
  </cc:implementation>
</html>
```

De componenten worden bewaard in de map resources/ezcomp/. Indien men de component wil gebruiken moet men bovenaan de namespace 'http://java.sun.com/jsf/composite/ezcomp' toevoegen. Hierna kan men een component bijvoorbeeld als volgt oproepen.

Listing 5.29: Component email

```
<em:email id="email" value="link met bean"></em:email>
```

# Hoofdstuk 6

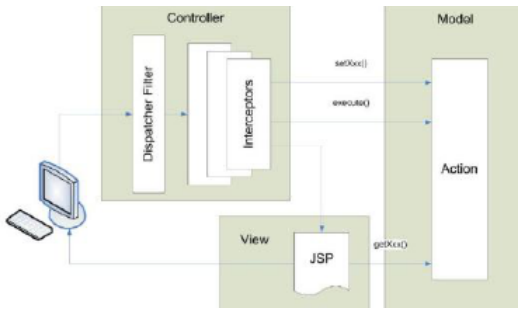
# Struts

## 6.1 Wat is struts2?

Struts is een open-source web application framework, dat voortbouwt op servlets. Het moedigt ontwikkelaars aan om de Model-View-Controller architectuur te gebruiken.

Struts 2 is een implementatie van MVC/Model 2 (Eigenlijk MVC2 of pull-MVC), waarbij acties deel uitmaken van het model. Struts 2 is een samengaan van Struts en WebWork (een ander framework).

Zoals we zien in figuur 6.2 bestaat Struts uit een controller, model, en view. De controller bestaat op zich nog eens uit een filter en interceptors. De interceptors stellen de eigenschappen in van de actieklassen en roepen dan een actie op. Het model bevat actieklassen die de actie uitvoeren, dit is meestal data opvragen en instellen. De view toont dan de huidige staat van het




Figuur 6.1: Principe van Struts

model, de view kan geschreven zijn in JSP.

### 6.1.1 Controller

Een filter van de controller genereert, in tegenstelling tot bij servlets, geen antwoord. Wat het wel doet, is een request bekijken en manipuleren (headers, parameters en attributen). Ook kan het de response aanpassen (headers en body). Het is mogelijk om een ganse reeks van filters te hebben, een ketting van filters. Men moet dit configureren in web.xml.

Listing 6.1: Configuratie filters

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class>
    org.apache.struts2.dispatcher.FilterDispatcher
  </filter-class>
</filter>

<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

### 6.1.2 Actie

Een actieklasse is afgeleid van com.opensymphony.xwork2.ActionSupport. In de methode execute vindt men de eigenlijke actie. Deze heeft als teruggeefwaarde een string, die het resultaat van de actie beschrijft. (bv. SUCCESS) De eigenschappen van deze klasse bevatten de in te stellen data door de interceptors.

Een andere standaardmethode van ActionSupport is getText. Hiermee kan men informatie opvragen uit een properties bestand, dat zich in dezelfde package als de klasse bevindt. Als argument geeft men een sleutel mee. Deze properties bestanden kunnen in verschillende talen beschikbaar zijn. bv. package.properties en package_nl.properties.

Listing 6.2: Beschikbare standaard teruggeefwaarden van ActionSupport

```
String SUCCESS = "success";
String NONE = "none";
String ERROR = "error";
String INPUT = "input";
String LOGIN = "login";
```

Om een URL te verbinden aan een bepaalde actieklasse, en het resultaat hiervan aan een bepaalde view, moet men dit configureren in struts.xml. Men plaatst struts.xml in het class path, dus in de basis van het Java project. Als men het project deployed, moet dit staan in de WEB-INF/classes map.

Struts.xml overschrijft de standaardconfiguratie. Het kan ook opgesplitst worden in verschillende bestanden, die dan met een include opdracht geïncludeerd worden.

Listing 6.3: standaardconfiguratie struts.xml

```
<struts>
  <constant name="struts.enable.DynamicMethodInvocation" value="false" />
  <constant name="struts.devMode" value="false" />
</struts>
```

Listing 6.4: Include opdracht

```
<struts>
  <include file="example.xml"/>
</struts>
```

Listing 6.5: Voorbeeldconfiguratie van struts.xml

```
<struts>
  <package name="example" extends="struts-default">
    <action name="HelloWorld" class="example.HelloWorld">
      <result>/HelloWorld.jsp</result>
      <result name="input">/Login.jsp</result>
      <result type="redirectAction">Menu</result>
    </action>
  </package>
```

```
</struts>
```

### 6.1.3 View

Om de view te construeren kan men gebruik maken van de struts library.

Listing 6.6: Struts taglib

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```

Enkele interessante tags zijn de volgende:

- property: eigenschap van de bijhorende actieklasse tonen
- url: maakt URL op basis van action-attribuut, men kan parameters toevoegen met param-tag, men gebruikt dit in een a-tag
- a: genereert een link
- text: haalt tekst uit het properties-bestand

Men heeft toegang tot de data via OGNL. OGNL staat voor Object Graph Navigation Language. Men plaatst een OGNL uitdrukking tussen % . OGNL maakt gebruik van een context map.

Deze context map bestaat uit:

- application
- session
- value stack(root): een stack met objecten: tijdelijke objecten (bv. van een foreach), en het action-object
- request
- parameters
- attributen (doorzoekt, in volgorde, page, request, session, en application scopes)

Men kan aan deze objecten in OGNL aan via # ervoor te zetten. bv #session. Van de objecten op de value stack kan men ook meteen de eigenschappen opvragen, zonder verwijzing naar het object.

Listing 6.7: OGNL voorbeelden

```
#parameters["thema"] of #parameters.thema (~getParameter)
#request["view"] of request.view (~getAttribute)
#session[  n a a m  ] of #session.naam
#application["catalogus"] of #application.catalogus
#attr["view"] of #attr.view
```

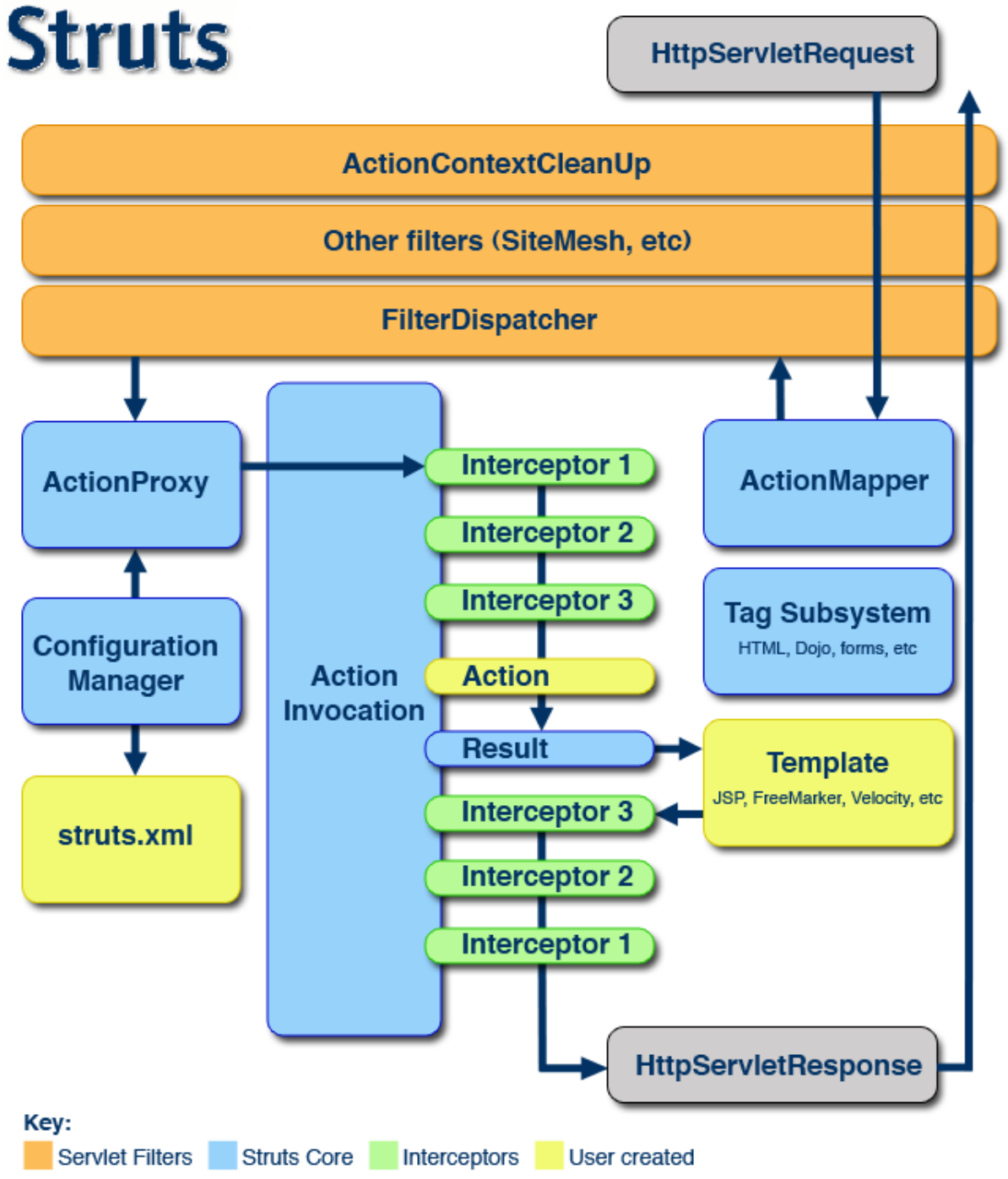
## 6.2 Localization

Localization betekent dat men pagina's taalonafhankelijk gaat maken. Men gebruikt labels in de JSP pagina's om met deze labels het bijhorende woord op te vragen. De labels en bijhorende woorden zitten per taal in .properties bestanden. Men kan de properties bestanden per package maken (package.properties) of per klasse (klassenaam.properties). Men kan de taal instellen via een parameter. Als we bijvoorbeeld naar het russisch willen omwisselen.

Listing 6.8: Localization

```
<a href="<s:url value="Welcome.do">
            <s:param name="request_locale" value="'ru'" />
            </s:url>">Russian</a>


<s:url id="url" action="HelloWorld">
<s:param name="request_locale">en</s:param>
</s:url>
<s:a href="%{url}">English</s:a>
```

Dit wordt opgevangen door een interceptor, de i18n interceptor. Deze plaatst de locale op de ActionContext.




Figuur 6.2: Principe van Struts 2

## 6.3 Configuratie acties

Indien men voor een pagina geen actie specifieert wordt standaard ActionSupport als actieklasse gebruikt. In struts.xml komt er zo:

Listing 6.9: Struts.xml zonder actieklasse

```
<action name="Welcome">
  <result>/Welcome.jsp</result>
</action>
```

Listing 6.10: Alle acties matchen

```
<action name="*" >
  <result>/tutorial/{1}.jsp</result>
</action>
```

Men kan ook alle mogelijke acties matchen, zoals te zien is in listing 6.10/ De {1} in die listing komt overeen met de gematchte actienaam. Men kan ook de lege actieklasse ExampleSupport meegeven.

Listing 6.11: ExampleSupport

```
<action name="*" class="example.ExampleSupport">
  <result>/{1}.jsp</result>
</action>
```

Struts.xml kan meerdere elementen bevatten, het eerste gematchte patroon wordt aanvaard. Men zet dus steeds laatst, als men dit gebruikt.

## 6.4 Formulier

Voor formulieren te construeren kan men gebruik maken van de tags s:form, s:textfield, s:password en s:submit. Men voorziet getters en setters in de actieklasse voor de ingevulde waarden, zodanig dat de eigenschappen ingesteld kunnen worden. Indien men bijvoorbeeld een tekstvak met value user heeft, maakt men een setter setUser in de actieklasse.

## 6.5 Validatie

Validatie is controle van de invoer van een gebruiker. Men maakt hiertoe een XML document aan. Het volgt de naamconventie Naamactie-validation.xml. De eerste keer dat een loginformulier getoond wordt, mag er geen validatie uitgevoerd worden.

Hiertoe overschrijft men in de actieklasse de methode input, en men laat deze uitvoeren in plaats van validate en execute.

Listing 6.12: Formulier configuratie struts.xml

```
<action name="Login*" method="{1}" class="example.Login">
  <result name="input">/Login.jsp</result>
  <result type="redirect-action">Menu</result>
</action>
```

Men roept de pagina de eerste keer op met Logininput. Hierdoor wordt de methode ook ingevuld met input. Deze geeft de waarde “input” terug. Hierdoor wordt Login.jsp geladen. Als dit formulier ingevuld wordt en opgestuurd wordt, wordt de methode validate en execute uitgevoerd. Als alles goed gaat wordt de gebruiker doorgeloodst naar Menu. Anders wordt Login.jsp opnieuw geladen.

Er is verschillende validatie beschikbaar.

Listing 6.13: Voorbeeld validatie xml file

```
<validators>
<!-- Field Validators for email field -->
  <field name="email">
    <field-validator type="required">
      <message>...</message>
    </field-validator>
    <field-validator type="email">
      <message>...</message>
    </field-validator>
```

```
  </field>
  <field name="bar">
    <field-validator type="required">
      <message>...</message>
    </field-validator>
    <field-validator type="int">
      <param name="min">6</param>
      <param name="max">10</param>
      <message>bar must be between
${min} and ${max}, current value is ${bar}.</message>
    </field-validator>
  </field>
  <field name="bar2">
    <field-validator type="regex">
      <param name="expression">[0-9],[0-9]</param>
      <message>   </message>
    </field-validator>
  </field>
  <field name="date">
    <field-validator type="date">
      <param name="min">12/22/2002</param> <param
  name="max">12/25/2002</param> <message>...</message>
    </field-validator>
  </field>
  <field name="myField">
     <field-validator type="fieldexpression">
       <param name="expression"><![CDATA[#myCreditLimit > #myGirfriendCreditLimit
           ]]></param>
       <message>My credit limit should be MORE than my girlfriend</message>
     </field-validator>
  </field>
</validators>
```

Enkele mogelijke types validatie zijn:

- required: Niet null?
- requiredstring: Niet null? Niet allemaal spaties?
- int: int? In interval?

- date: datum? In interval?
- expression: Logische OGNL-uitdrukking
- fieldexpression: Logische OGNL-uitdrukking, gekoppeld aan een veld
- email: email?
- url: url?
- conversion: Conversie gelukt?
- stringlength: Lengte in interval?
- regex: Voldoet het aan een reguliere expressie?

## 6.6 ActionContext

De ActionContext is de context waarin een actie wordt uitgevoerd.

Listing 6.14: ActionContext

```
import com.opensymphony.xwork2.ActionContext;
ActionContext context = ActionContext.getContext();
```

Hiermee krijgt men toegang tot de sessie, parameters, ...

Acties die toegang willen tot de sessie van een gebruiker moeten de SessionAware interface implementeren. Dit geeft hen toegang tot een map waarin ze objecten kunnen plaatsen, die ook in een volgende aanvraag beschikbaar moeten zijn. Analoog heeft men de interface ApplicationAware.

Listing 6.15: SessionAware

```
public class Actie extends ActionSupport implements SessionAware {
  protected Map sessieMap;

  public void setSession(Map arg0) {
    this.sessieMap = arg0;
  }
}
```

# Hoofdstuk 7

# Javascript

ECMAScript is de scriptingtaal gestandardiseerd door Ecma International in the ECMA-262 specification and ISO/IEC 16262. ECMAScript wordt in browsers geïmplementeerd door JavaScript. Of in het geval van Internet Explorer door JScript. JavaScript en JScript zijn eigenlijk dezelfde taal, ze hebben gewoon een andere naam omwille van toenmalige trademark problemen tussen Sun en MicroSoft.

JavaScript is een scripttaal ontworpen om HTML-pagina's interactiever en dynamischer te maken. Het is vaak ingebed in HTML pagina's. Hoe de JavaScript wordt geïnterpreteerd hangt vaak af van browser tot browser. Men moet hier dus rekening mee houden bij het maken van een website.

JavaScript heeft een lage aanleerdrempel, men heeft geen kennis van object-geörienteerd programmeren nodig. De syntax lijkt op die van de C-gebaseerde talen zoals C++, Java, C#.

De grootste concurrent is VBScript, van MicroSoft, dat een Active Scripting taal is, en gebaseerd is op Visual Basic.

JavaScript heeft niets te maken met Java. Ze hebben echter wel beide een C-achtige syntax, en JavaScript volgt veel van de naamgevingsconventies van Java. De originele naam van JavaScript was LiveScript, maar werd hernoemd door een co-marketing deal tussen Netscape en

Sun. Netscape mocht in ruil Java bundelen met hun op dat moment domminante browser.

## 7.1 JavaScript als programmeertaal

### 7.1.1 Types en variabelen

Er is geen sprake van een sterke typering zoals bij C++ of Java. Types worden dynamisch (at runtime) toegekend en eventueel veranderd. Er is geen declaratie nodig, dit kan eventueel met var[1]. De ingebouwde types zijn Number, Boolean, String, null en undefined.

Listing 7.1: Voorbeeld declaratie

```
var antwoord = 75;
antwoord = "Dank u wel";
antwoord = "Het resultaat is " + 75; // "Het resultaat is 75"
antwoord = "37" - 7; // 30
antwoord = "37" + 7; // "377"
```

### 7.1.2 Objecten

Objecten zijn containers voor waarden (properties). Dit kunnen variabelen zijn, of andere objecten. Objecten kunnen ook methodes bevatten. Met behulp van '.' en '[ ]' heeft men toegang tot de properties en de methodes.

Listing 7.2: Voorbeeld van objecten

```
var persoon = new Object();
persoon.naam =  Ongenae ;
persoon.voornaam =  Veerle ;
persoon.schoenmaat = 38;
persoon["naam"] =  De Smedt ;
persoon["voornaam"] =  Thomas ;
persoon["schoenmaat"] = 45;
persoon = {naam: "Janssens", voornaam: "Nele",schoenmaat: 40};
```

[1] Vergelijk met 'my' in Perl

```
var foo = {a: "alpha", 2: "two"};
var test = foo.a; // "alpha"
test = foo[2]; // "two"
test = foo["a"]; // alpha
test = foo["2"]; // two

var namen = ["Veerle", "Thomas", "Joris"];
var kleuren = ["rood", ,"geel"]; //tweede variabele is undefined
kleuren[1] = "groen";
kleuren[3] = "blauw";
```

### 7.1.3 Functies

Functies zijn uit te voeren opdrachten. De syntax is als volgt.

Listing 7.3: Syntax functie

```
function naam(argument) {
  ...
}
```

Een functie kan een waarde teruggeven, met de opdracht return. Men kan functies ook anoniem declareren.

Listing 7.4: Anonieme functie

```
var eatCakeAnon = function(){
     alert("So delicious and moist");
};
eatCakeAnon();
```

Elke functie is ook een object. Functies die men eigenlijk als objecten gaat gebruiken roept men vaak aan met de new operator. Als men in een functie zelf naar de functie (en dus het object) wil verwijzen kan men gebruik maken van de operator this. Als een functie geassocieerd is met

een object noemt men dit een methode.

Listing 7.5: Gebruik van functies

```
function verdubbel(getal) {
  return 2*getal;
}
function wijzigNaam(persoon, nieuweNaam) {
  persoon.naam = nieuweNaam;
}

var getal = 10.2;
var dubbel = verdubbel(10.2); // 20.4
var persoon = {naam: "Janssens", voornaam: "Nele", schoenmaat: 40};
wijzigNaam(persoon,"Van Canneyt"); // {naam:   V a n  Canneyt", voornaam:    }

function langeNaam() {
  return this.naam + " " + this.voornaam;
}

var persoon = {naam: "Janssens", voornaam: "Nele", schoenmaat: 40,
    getmaat() {return this.schoenmaat;}, set maat(grootte) {this.schoenmaat =
        grootte;}};

persoon.langeNaam = langeNaam;
var naam = persoon.langeNaam();
var maatNele = persoon.maat;
persoon.maat = 39;

function pasFunctieToeOpArray(functie,lijst) {
  var resultaat = new Array;
  for (var i = 0; i < lijst.length; i++)
    resultaat[i] = functie(lijst[i]);
  return resultaat;
}

var resultaat = map(function(x) {return x * x * x}, [0, 1, 2, 5, 10]);
```

Standaard zijn er ook een aantal ingebouwde objecten met interessante functies. Deze zijn Math, Number, String en Date.

Aangezien JavaScript een taal is die gebaseerd is op prototypes, is er geen onderscheid tussen de term klasse en object. Men kan objecten als template gebruiken om nieuwe objecten te maken. Dit laat toe om dynamisch eigenschappen toe te voegen aan 1 of meerdere objecten. Op deze manier kan men aan overerving doen.

### 7.1.4 Controlestructuren

De beschikbare controlestructuren zijn if(en eventueel else) , while en for.

Om te debuggen kan men firebug gebruiken in firefox. JSLint laat dan weer syntactische controle toe van de JavaScript code. Er zijn ook JavaScript libraries beschikbaar, de meest gebruikte zijn JQuery, YUI en Prototype.

## 7.2 JavaScript in HTML-paginas

JavaScript scripts worden client side uitgevoerd in de browser. Men kan het voor verscheidene zaken gebruiken. Validatie van een formulier, dynamische menu's, reageren op events, info over de browser ophalen, ...

Om JavaScript toe te voegen aan een HTML pagina, voegt men het volgende toe.

Listing 7.6: script in HTML

```
<script type="text/javascript">
  ...// javascript code
</script>
<script type="text/javascript" src="timer.js"></script>
```

In de header voegt men op te roepen functies toe, en in de body onmiddelijk uit te voeren opdrachten.

De HTML DOM is een onderdeel van DOM (Document Object Model, W3C). Het geeft toegang tot de HTML code, en geeft de mogelijkheid om HTML code te veranderen. Men kan bijvoorbeeld elementen ophalen (document is een voorgedefinieerd object dat de huidige pagina voorstelt).

Listing 7.7: Elementen ophalen

```
document.getElementById(...);
document.getElementsByTagName(...);
```

Men kan ook attributen en CSS-kenmerken ophalen met de simpele puntnotatie.

Listing 7.8: Ophalen attributen en CSS

```
document.getElementById('titel').style //attribuut ophalen
document.getElementById('titel').style.color = ... //CSS kenmerk aanpassen
```

De browser laadt documenten en figuren. De gebruiker kan interageren met zijn muis en toetsenbord. Deze events kan men opvangen met JavaScript. De belangrijkste events zijn de volgende.

- onload, onunload
- onfocus, onblur, onchange
- onsubmit: return-waarde bepaalt of het formulier ingediend wordt. Men moet best expliciet return schrijven. Dit maakt het voorwaardelijk, anders wordt de functie gewoon uitgevoerd. Zie listing 7.9.
- onmouseover, onmousemove, onmouseout
- onkeydown, onkeypress, onkeyup
- onclick: return-waarde bepaalt of de actie uitgevoerd wordt

Listing 7.9: Form met event

```
<form action="..." onsubmit="return controleer_invoer();">
```

Men kan deze functies aan deze events hechten, rechtstreeks via het HTML attribuut. Een alternatief is echter om dit in de JavaScript code te doen. Hiertoe registreert men de event handler in de code. Het member element (event property) moet men instellen, dit heeft dezelfde naam als de reeds vermelde attributen (dus bv. onchange). Men kent hieraan een functie toe. Men doet dit het beste op het einde van het body element. Bij onsubmit moet men nu niet expliciet return toevoegen.

Window is een standaard beschikbaar object. De belangrijkste eigenschap ervan is location. Het heeft ook enkele methodes.

- open()
- close()
- focus()
- alert(): popupvenster met info, 1 knop
- confirm(): popup-venster met vraag voor bevestiging, 2 knoppen (ok en cancel), geeft een returnwaarde terug (true of false).
- prompt(): info vragen, tekstvak en twee knoppen (ok en cancel), dit returnt de ingevoerde tekst.

Navigator is een ander beschikbaar standaard object. Het geeft info over de browser. De eigenschappen ervan zijn de volgende.

- appCodeName
- appName
- appVersion

- cookieEnabled
- platform
- userAgent

Men kan ook gebruik maken van timers. Hiertoe heeft men de methodes setTimeout en setInterval van window ter beschikking. Ze hebben als parameters een verwijzing naar een functie en een tijdsinterval. Ze hebben als teruggeefwaarde het id van de timer. SetTimeout voert de functie eenmalig uit, terwijl setInterval dit met intervallen doet. Om de timer uit te schakelen zijn er equivalente functies clearTimeout en clearInterval. Deze nemen als parameter het id van de ingestelde timer.

Men heeft ook reguliere expressies ter beschikking in JavaScript.

Listing 7.10: RegExps

```
re = /ab+c/
re = new RegExp("ab+c")
```

RegExp heeft bepaalde functies. exec gaat op zoek naar een match, en heeft als resultaat de eerste gevonden string. test is een andere functie en geeft alleen aan of de regexp gevonden is.

De klasse String heeft ook enkele nuttige methodes. Match bijvoorbeeld, zoekt naar exacte matches van de string, de returnwaarde is een array met resultaten. Search zoekt dan weer de index van de te zoeken string. Als de string niet gevonden werd, wordt er -1 teruggegeven. Verder zijn er nog de functies replace en split.

Men kan rechtstreeks HTML toevoegen in JavaScript met de methode write van document. Dit is echter af te raden, aangezien dit geen strikte XHTML is.

## 7.3 DHTML

DHTML staat voor Dynamic HTML en slaat op de combinatie van HTML, JavaScript, de HTML DOM en CSS. Op deze manier kan men dynamische en interactieve pagina's maken.

## 7.4 Nadelen JavaScript

Een groot nadeel van JavaScript is de browserafhankelijkheid. De namen van objecten, methodes, eigenschappen kunnen namelijk verschillen. Men moet de code dus testen in verschillende browsers.

Een ander nadeel is dat debuggen moeilijk is. Men kan naar de foutmeldingen van de browser kijken. In firefox opent men hiertoe de foutconsole. In Internet Explorer dubbelklikt men hiertoe op het gevaarbordje. In firefox kan men ook firebug gebruiken, zoals reeds eerder gezegd. Men kan ook zien hoever de code geraakt door gebruik te maken van popups.

Een voorbeeld van een browserafhankelijkheid is het event object. In Internet Explorer wordt dit niet meegegeven met de eventhandler, men moet dit verkrijgen via window.event. In een andere browser wordt dit wel meegegeven.

Listing 7.11: Browseronafhankelijke code

```
function schrijf(e) {
  if (!e) e = window.event; // IE
  ...
}
```

Aan het event kan men bijvoorbeeld vragen welke toets ingedrukt werd. Dit met behulp van e.keyCode en e.which. Men kan ook nagaan in welk element het event is gebeurd. Dit met de eigenschappen target en srcElement. Voor een overzicht van de eigenschappen van events kan men kijken op `http://www.quirksmode.org/dom/w3c_events.html`

Men kan het karakter horende bij een bepaalde keyCode opvragen met String.fromCharCode(keyNum).

Als laatste moet het element innerHTML vermeld worden. Dit geeft rechtstreekse toegang tot de HTML code binnenin een element. Dit is een gewone string. Het is echter wel te vermijden, omdat dit niet propere code geeft.

# Hoofdstuk 8

# AJAX

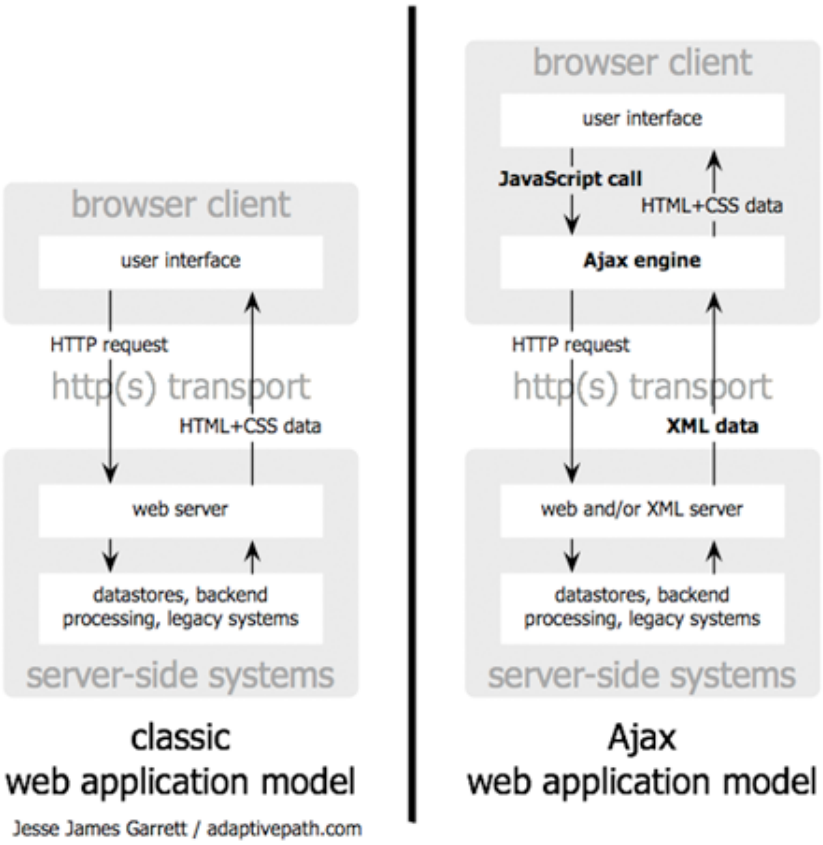
AJAX staat voor Asynchronous JavaScript and XML. Men gaat de bestaande standaarden anders gebruiken. Hiervoor gebruikt men in JavaScript het XMLHttpRequest-object, om vanuit JavaScript HTTP berichten te sturen en te ontvangen. Het gebruik van XML is niet vereist hoewel dit in de naam zit. Vaak wordt JSON gebruikt. Men krijgt de indruk dat men lokaal werkt omdat webpagina's sneller geladen worden, men gebruikt asynchrone aanvragen.

Bij een klassieke webpagina stuurt de client een HTTP bericht naar de server. De browser wacht tot hij een antwoord heeft gekregen van de server. De browser toont het antwoord. Het nieuwe document vervangt het vorige volkomen.

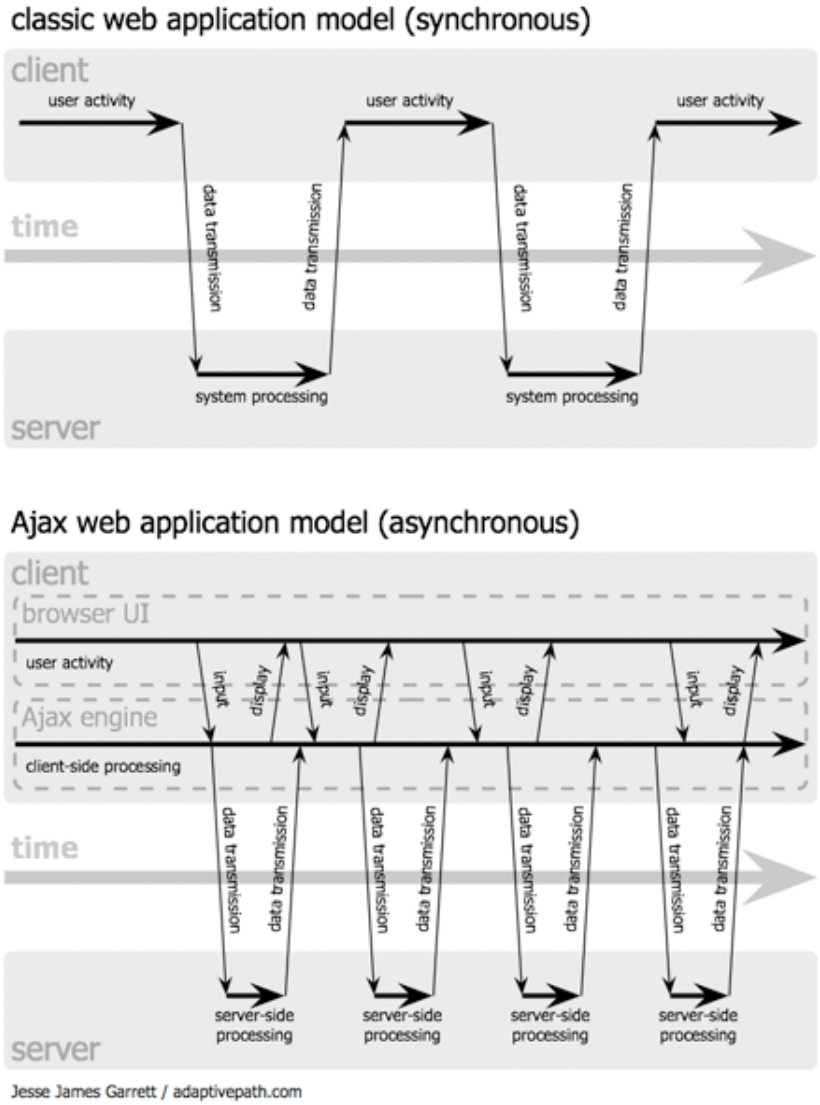
Bij AJAX kunnen delen van het HTML document vervangen worden. Ondertussen kan de gebruiker verder werken. Er is dus sprake van een asynchrone communicatie met de server. Hoe dit in zijn werk gaat zien we in figuren 8.1 en 8.2 .

De AJAX engine doet zijn werk in de browser. Hij krijgt een HTTP antwoord terug in XML vorm.

Zoals eerder reeds gezegd, steunt AJAX dus vooral op het XMLHttpRequest object. Hoe men dit construeert zien we in listing 8.1. Merk op dat men dus een ActiveXObject moet aanmaken indien een gebruiker de browser IE versies 5 en 6 gebruikt.




Figuur 8.1: Principe van AJAX




Figuur 8.2: Principe van AJAX in de tijd

Listing 8.1: Create XMLHttpRequest object

```
new XMLHttpRequest(); // In IE 7+, Safari 1.2, Mozilla1.0/Firefox, Opera 8+,
    Netscape 7

new ActiveXObject("Microsoft.XMLHTTP");
new ActiveXObject("Msxml2icrosoft.XMLHTTP"); //Deze twee zijn beschikbaar in IE
    5+, het zijn ActiveX objecten.
```

Op deze objecten kan men de methode open() aanroepen.

Listing 8.2: open() methode

```
open(aanvraagMethode, URL)
open(aanvraagMethode, URL, asynchroon)
open(aanvraagMethode, URL, asynchroon, gebruikersnaam)
open(aanvraagMethode, URL, asynchroon, gebruikersnaam,wachtwoord)
```

Het laat toe om te connecteren met een server. Aanvraagmethode is GET, POST, PUT of DELETE. Bij URL vult men de URL in van de bron. Asynchroon staat default op true.

Daarna moet men van het object de eigenschap onreadystatechange instellen. Dit moet de functie zijn die uitgevoerd moet worden als de readystatechange verandert. Mogelijke waarden van de readystatechange zijn.

- 0: request not initialized
- 1: server connection established
- 2: request received
- 3: processing request
- 4: request finished and response is ready

Men moet dus controleren of het request gelukt is en er een antwoord beschikbaar is. Vaak controleert men ook de status van het HTTP antwoord.

Listing 8.3: Voorbeeld implementatie onreadystatechange

```
xmlhttp.onreadystatechange=function()
 {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
   {
   document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
   }
 }
```

Om een aanvraag ook daadwerkelijk te sturen naar de server gebruikt men de functie send(corpus). In corpus zet men de inhoud van het HTTP bericht. Dit zijn ofwel parameters (bij POST) ofwel null. Men roept deze functie pas op na het instellen van de callback-functie (onreadystatechange).

Als men een antwoord heeft ontvangen, zijn enkele eigenschappen van het XMLHttpRequest object ingesteld. Zo kan men status opvragen, dit geeft de ontvangen status terug. responseText geeft het corpus van het antwoord weer. Dit is enkel als men een tekst krijgt. Alternatieven zijn responseXML, responseStream en responseBody.

Het antwoord kan afhankelijk zijn van parameters, zo kan men bijvoorbeeld een formulier invullen.

De parameters volgen wel een codering. Indien er speciale tekens in voorkomen, gebruikt men in de plaats % gevolgd door 2 hexadecimale cijfers die de ASCII-code voorstellen van het teken. Voor een spatie kan men in plaats van de vermelde methode een + gebruiken.

Listing 8.4: Structuur parameters

```
naam=waarde&naam1=waarde1&...
```

Als men een GET-aanvraag doet, moet men de parameters na de URL toevoegen.

Listing 8.5: GET met parameters

```
url?naam=waarde&...
```

Bij een POST zijn de parameters eigenlijk het corpus. Men moet hiertoe de header van het XMLHttpRequest object instellen, en daarna de parameters meegeven met de send functie.

Listing 8.6: Parameters bij POST

```
aanvraag.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
aanvraag.send( naam=waarde&      );
```

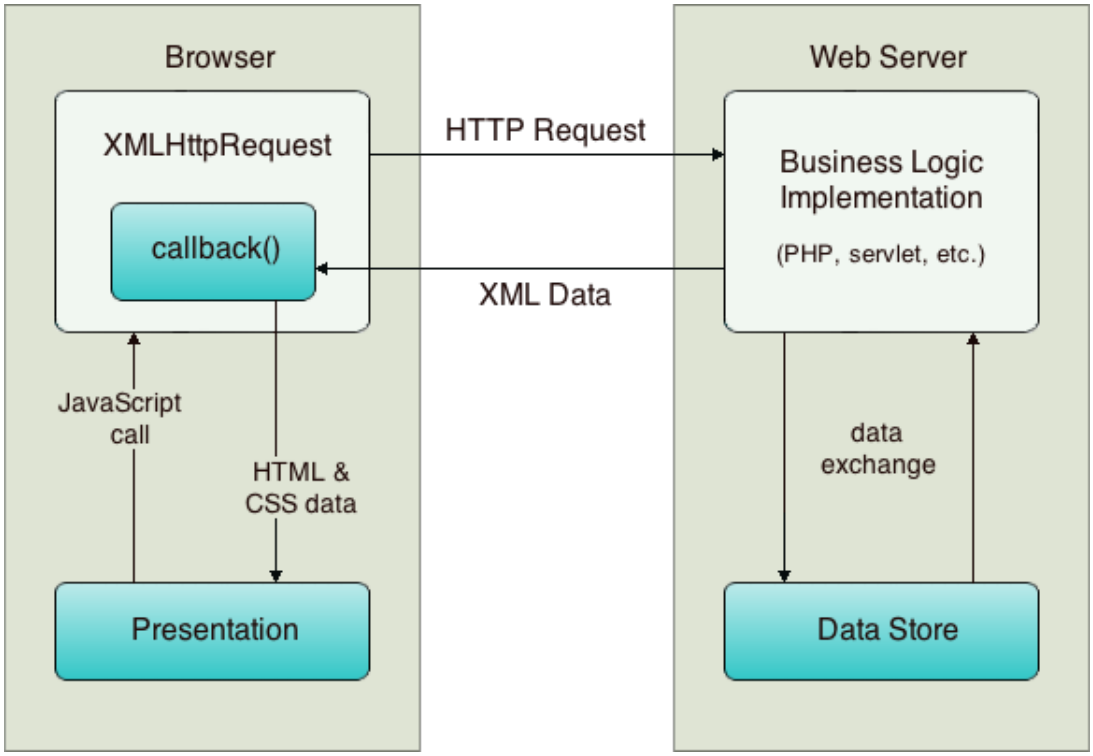
Als men in ASP.NET werkt kan men op de serverkant aan deze parameters via het Request-object.

Men kan geen AJAX aanvragen doen naar een ander domein dan het domein van de huidige webpagina. In Internet Explorer zijn eventueel wel expliciete uitzonderingen mogelijk. Een mogelijke oplossing is een server side proxy. De server zal de aanvraag dan doorsturen naar de het andere domein, en het ontvangen antwoord terug doorsturen naar de client.

Ajax Frameworks vereenvoudigen het schrijven van websites met AJAX. Normaal heeft men vrij veel kennis nodig om complexe applicaties te maken. Frameworks bieden echter voorgedefinieerde componenten, die men kan gebruiken. Frameworks moet men ofwel installeren, ofwel gewoon naar het script verwijzen in de HTML code.

Enkele veelvoorkomende frameworks zijn de volgende.

- Prototype
- Script.aculo.us: uitbreiding prototype
- JQuery
- Yahoo! UI Library
- Dojo Toolkit
- Mootools




Figuur 8.3: Flow van AJAX

Verder heeft men nog taalspecifieke frameworks.

- PHP: PHP-script, Xajax, sajax, ...
- ASP.NET: Installatie vereist, ASP.NET Ajax, DynAjax, ...
- Java: .jar-bestand, DWR, Google Web Toolkit, Thinwire, ...

# Hoofdstuk 9

# Object Relational Mapping

Object-relational mapping (ORM, O / RM, en O / R mapping) is een programmeer techniek voor het omzetten van gegevens ussen onverenigbare type systemen in object-oriented programmeertalen. Dit creert in feite een "virtuele object database"die gebruikt kan worden vanuit de programmeertaal. Er zijn zowel gratis en commercile pakketten beschikbaar die object-relational mapping realiseren, hoewel sommige programmeurs kiezen om hun eigen ORM tools te creren.

Als men ORM wil toepassen op een .NET applicatie data en een databank, kan men gebruik maken van nHibernate. Men moet dan de DLLs downloaden en toevoegen aan het project (in bin-map, bij References). In web.config moet men informatie plaatsen in verband met de databank. (provider, connectiestring)

## 9.1 Objecten afbeelden

### 9.1.1 Eigenschappen

Men kan de afbeelding op twee manieren realiseren.

- Met behulp van annotaties
- Via een xml file (.hbm.xml bestand)

Als we even een voorbeeld bekijken met een xml file.

Listing 9.1: XML file voor ORM

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">
  <class name="ProductOrder, __Code" table="ProductOrder">
    <id name="ProductOrderID" column="ProductOrderID" type="Int32" unsaved-value
        ="null">
      <generator class="native" />
    </id>
    <property name="CustomerID" column="CustomerID" type="Int32" />
    <property name="ProductName" column="ProductName" type="String" />
    <property name="Quantity" column="Quantity" type="Int32" />
    <property name="TotalCost" column="TotalCost" type="Decimal"/>
  </class>
</hibernate-mapping>
```

Listing 9.2: De bijhorende klasse

```
public partial class ProductOrder {
  private int _productOrderID;
  private int _customerID;
  private string _productName;
  private int _quantity;
  private decimal _totalCost;
  ...
  property declarations
  ...
}
```

We zien dus dat elke property gemapt wordt op een kolom in de database.

Shadow information is data dat objecten nodig hebben, naast hun gewone data, om te bestaan. De objecten weten bijvoorbeeld niet wat de primary key in de tabel is, dit moet dus ook gemapt worden.

Listing 9.3: ID mapping

```
<id name="ProductOrderID" column="ProductOrderID" type="Int32" unsaved-value="
    null">
  <generator class="native" />
</id>
```

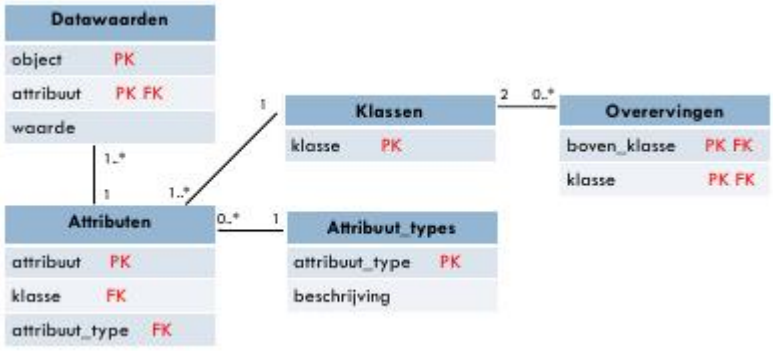
nHibernate ondersteunt ook samengestelde sleutels. Generator bepaalt hoe ids moeten aangemaakt worden bij nieuwe objecten. Native kiest de juiste methode uit de volgende drie: Identity, Hilo, Sequence. Deze methodes zijn databankspecifiek. Oracle gebruikt bijvoorbeeld Sequence, MS SQL Server gebruikt Identity.

Zoals reeds gezegd kunnen we ook met annotaties werken, in plaats van een xml file. Hieronder ziet men het equivalent van listing 9.1, maar dan met annotaties.

Listing 9.4: ORM met annotaties

```
using NHMA = NHibernate.Mapping.Attributes;

[NHMA.Class(Table = " ProductOrder")]
public class ProductOrder {
  private int _productOrderID;
  [NHMA.Id(Name = "ProductOrderID")]
  [NHMA.Generator(1, Class = "native")]
  public virtual int ProductOrderID {
  get { return _productOrderID; }
  set {_productOrderID = value; }
  }
  ...
}
```

Bij de annotatie generator zien we een volgnummer staan. Dit is gewoon de volgorde van het attribuut moesten we een xml file schrijven.




Figuur 9.1: Generieke tabel

### 9.1.2 Overerving

Er zijn vier verschillende manieren om overerving te implementeren in nHibernate.

1. Volledige hirarchie naar 1 tabel
2. Elke concrete klasse naar eigen tabel
3. Elke klasse naar een tabel
4. Alle klassen naar een generieke tabel

Als we de volledige hirarchie naar n tabel brengen zitten we met het probleem dat dit moeilijk uitbreidbaar is. Als we elke concrete klasse naar zijn eigen tabel mappen is het dan weer moeilijk aanpasbaar. Elke klasse naar een tabel is nog een van de beste oplossingen. Op het niveau van databanken werkt men dan het systeem van foreign keys die naar primary Keys verwijzen.

De laatste mogelijkheid is om alle klassen naar een generieke tabel te mappen. Dit wordt enkel toegepast bij complexe applicaties met weinig data. Het volgt het schema van afbeelding 9.1.

Hieronder een voorbeeld van een configuratie voor ORM met overerving.

Listing 9.5: ORM met overerving

```
 <?xml version="1.0" encoding="utf-8" ?>
<class name="Person, __Code" table="Person">
  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">
    <id name="PersonID" column="PersonID" type="Int32  unsaved-value="null">
      <generator class="native" />
    </id>
    <property name="FirstName" column="FirstName" type="String"/>
    <property name="LastName" column="LastName" type="String"/>
    <joined-subclass name="Customer, __Code" table="Customer">
      <key column="PersonID"/>
      <property name="Address" column="Address" type="String"/>
      <property name="City" column="City" type="String"/>
      <property name="State" column="State" type="String"/>
      <property name="ZipCode" column="ZipCode" type="String"/>
    </joined-subclass>

    <joined-subclass name="SalesPerson, __Code  table="SalesPerson">
      <key column="PersonID"/>
      <property name="EmployeeNumber" column="EmployeeNumber  type="String"/>
    </joined-subclass>
  </hibernate-mapping>
</class>
```

### 9.1.3 Relaties

In object georïenteerd programmeren hebben objecten relaties. Men onderscheidt drie soorten.

- associatie: er is een verband tussen de objecten
- aggregatie: het object heeft een ander object als datalid
- compositie: het object bestaat uit andere objecten, het deel bestaat niet zonder het geheel.

Men kan relaties indelen op verschillende criteria.

- Indeling volgens multipliciteit
  - 1-op-1
  - 1-op-veel

  - veel-op-veel

- Indeling volgens richting
  - eenrichtingsrelatie (half-duplex)
  - tweerichtingsrelatie (full-duplex)

Bij objecten realiseert men deze relaties met getters en setters, en met add en remove operaties, in het geval van 1 of veel-op-veel relaties. Als deze operaties slechts in 1 object bestaan, is het een eenrichtingsrelatie, anders een tweerichtingsrelatie.

In databanken wordt dit gerealiseerd met behulp van foreign keys. Met de foreign key verwijst men naar de primary key van een andere tabel. Indien we een veel-op-veel relatie moeten realiseren, gebruikt men een associatietabel, bestaande uit twee kolommen met de foreign keys die verwijzen naar de primary keys van de correspondere tabellen. Deze relaties zijn altijd tweerichtingsrelaties.

#### 1-op-1 relatie

Listing 9.6: XML file met 1-op-1 relatie

```
<one-to-one name= curriculum" class=  CV " />
```

Bij een 1-op-1 relatie in nHibernate geeft men met welke klasse er een relatie is in de xml file. Aangezien de ids ingesteld zijn, weet nHibernate welk object net bedoelt wordt. Indien het een tweerichtingsrelatie is, moet dit uiteraard ook in de configuratie van het ander object gebeuren.

Om het object dan daadwerkelijk op te halen, gaat men als volgt te werk.

- Object inlezen
- Kolom met foreign key gebruiken om object waar er een relatie mee is te identificeren
- Zoeken in tabel naar object waar er een relatie mee is
- Eventueel dit object inlezen (indien geen lazy loading)
- Indien tweerichtingsrelatie: omgekeerde referentie instellen

### 1-op-veel relatie

Listing 9.7: 1 op veel relatie

```
[NHibernate.Mapping.Attributes.Bag]
[NHibernate.Mapping.Attributes.Key(1, Column =  CustomerID")]
[NHibernate.Mapping.Attributes.OneToMany(2,ClassType=typeof(ProductOrder))]
public virtual IList<ProductOrder> Orders { get; set; }
```

Een 1-op-veel relatie ziet er bijvoorbeeld als volgt uit. Er wordt aangegeven wat de kolom met de foreign key in de andere tabel is. In dit geval dus CustomerID. Ook wordt er aangegeven hoe de andere klasse heet met ClassType.

Een IList in C# komt overeen met een bag in nHibernate, een ISet met een set.

Als we het voorbeeld volgen, moet er in de klasse ProductOrder een soortgelijke definitie gebeuren. Daar is er echter wel sprake van een ManyToOne relatie in plaats van een OneToMany relatie.

Om dit object daadwerkelijk op te halen gaat men als volgt te werk.

- Object inlezen
- Object waar een relatie mee is identificeren:
  - Al object in applicatie?
  - Moet men het object nog aanmaken?
- Referentie leggen
- Object toevoegen aan object waar er een relatie mee is

### veel-op-veel relatie

Een veel-op-veel relatie is gelijkaardig aan een 1-op-veel relatie.

Listing 9.8: veel-op-veel in XML

```
<set name="AssignedCustomers" table="AssignedCustomers" cascade="save-update">
  <key column="SalesPersonID"/>
  <many-to-many class="Customer,__Code" column="CustomerID"/>
 </set>
```

Aangezien cascade op save-update staat ingesteld, bewaart nHibernate ook Customer-objecten die toegevoegd zijn aan de Iset. Standaard worden enkel properties bewaard. Dit is handig, omdat men dan niet meer manueel de objecten moet updaten in de database.

Een standaard manier om objecten op te halen gaat als volgt.

- SELECT-opdracht uitvoeren om objecten waar er een relatie mee is te bepalen
- Object waar er een relatie mee is aanmaken of bepalen
  - Nieuw object aanmaken?
  - Als het een bestaand object is: Vernieuwen?
- Dit object toevoegen aan de verzameling

## 9.2 Objecten ophalen, bewaren

### 9.2.1 Session en SessionFactory

Indien men de objecten daadwerkelijk wil ophalen, moet men met de databank connecteren. Hiervoor zorgt het Session object (NHibernate.ISession). Dit bevat de functionaliteit van de databank en de connectie ermee. Men kan de volgende atomaire opdrachten uitvoeren.

- Create
- Read
- Update
- Delete

De sessie is lightweight. Men moet voor en na elke reeks opdrachten (transactie), het object aanmaken en vernietigen. Nadeel van het object is dat het niet thread safe is.

Om een sessie te krijgen gebruikt men de SessionFactory. Hiermee kan men de sessieobjecten aanmaken. Vooraleer men dit doet moet men uiteraard de configuratie inlezen, die nodig is om te connecteren met de databank. Men moet (uiteraard) ook de .hbm.xml bestanden inlezen.

De SessionFactory is heavyweight, maar gelukkig thread safe. Men maakt 1 object per applicatie aan.

### 9.2.2 Ophalen en bewaren

Een voorbeeld van het ophalen van een object kan men hieronder zien. Er zijn twee mogelijkheden: aan de hand van het ID, of met criteria.

Listing 9.9: Ophalen mbv ID

```
public static Customer getCustomer(int customerID) {
  // Open the session
  ISession session = NHibernateHelper.GetCurrentSession();
  // Load the order from the database
  Customer customer = (Customer)session.Load(typeof(Customer), customerID);
  //Close session
  NHibernateHelper.CloseSession();
  return customer;
}
```

Listing 9.10: Ophalen mbv criteria

```
public static List<ProductOrder> getProductOrdersForCustomer(int customerID) {
  List<ProductOrder> orders = new List<ProductOrder>();
  // Get the session
  ISession session = NHibernateHelper.GetCurrentSession();
  // Load the order from the database
  ICriteria criteria = session.CreateCriteria(typeof(ProductOrder));
  criteria.Add(Expression.Eq("CustomerID", customerID));
```

```
    criteria.List(orders);
    NHibernateHelper.CloseSession();
    return orders;
}
```

Men kan om een object te bewaren de operatie SaveOrUpdate toepassen. Hiermee hoeft de programmeur niet te kiezen welke operatie hij net bedoelt. nHibernate kiest automatisch de juiste. Om een object te verwijderen gebruikt men delete. Het is aan te raden om steeds transacties te gebruiken.

### 9.2.3 Lazy Loading

Lazy loading betekent dat de objecten pas opgehaald worden wanneer dit nodig is. Hiervoor moet de sessie wel nog actief zijn. Men kan overigens op een sessie object de methode Reconnect oproepen. Men kan dit aan- of uitschakelen in de hbm.xml bestanden. Men vult hiertoe het attribuut lazy in met de gewenste methode.

Listing 9.11: Lazy loading

```
<set name="Classes" cascade="all" lazy="true">
  ...
</set>
```

# Hoofdstuk 10

# Webservices

## 10.1 Communicatie tussen applicaties

Gedistribueerde applicaties zijn verschillende stukken software die allen samenwerken om een bepaalde taak uit te voeren. Men heeft dus verschillende computers en omgevingen. Er moet dus data uitgewisseld worden tussen deze programma's op verschillende computers.

Voor er webservices bestonden maakte men gebruik van EDI, CORBA, RMI, DCOM.

### 10.1.1 EDI

EDI staat voor Electronic Data Interchange. Het is een WAN voor een geografische regio, waar iedereen kan inpluggen. Enkel het protocol en de booschappen liggen vast, de netwerk details worden overgelaten aan de implementatie.

Er zijn verschillende netwerken. Ze zijn echter duur om in te stappen, daarbovenop is het dan nog is moeilijk en duur om de andere netwerken te bereiken. Een beter alternatief is dus vereist.

### 10.1.2 CORBA, RMI, DCOM

CORBA staat voor Common Object Request Broker Architecture. RMI voor Remote Method Invocation en DCOM voor Distributed Component Object Model.

Ze zijn allen gedistribueerde object-infrastructuren die over het internet communiceren. Het zijn transportprotocollen die bovenop TCP geïmplementeerd zijn. Er zijn afspraken omtrent de communicatie tussen objecten.

Het voordeel is dat servers niet tot hetzelfde netwerk hoeven te behoren, wat bij EDI wel het geval is (zelfde WAN). Het nadeel is dan weer dat ze enkel met dezelfde infrastructuur kunnen communiceren, tenzij men extra lagen implementeerd op de reeds ingewikkelde structuur.

## 10.2 Webservices

### 10.2.1 Terminologie

Een webservice is een softwaresysteem dat communicatie voorziet tussen computers. Het heeft een interface die beschreven wordt in WDSL (Web Services Description Language). Andere machines interageren met de webservice via SOAP berichten. Deze worden vaak via HTTP verstuurd, in XML formaat.

We kunnen twee verschillende rollen onderscheiden, die een computer kan hebben.

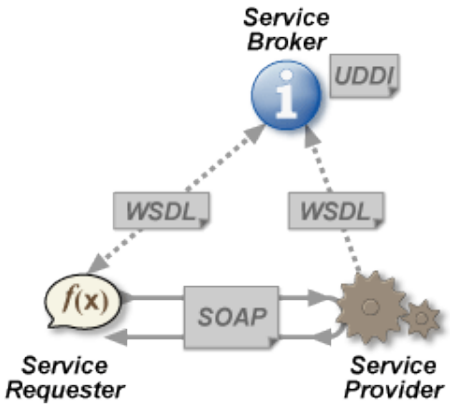
- Service provider: Biedt een webservice aan
- Service requester: Maakt gebruik van een webservice

In figuur 10.1 zien we hoe deze communiceren.

### 10.2.2 SOA

SOA staat voor Service-oriented Architecture. Het is een architectuur (het legt dus regels op) voor gedistribueerde systemen. Het wordt gekenmerkt door de volgende kenmerken.

- Logical view: Een abstracte definitie van de webservice.




Figuur 10.1: Webservices

- Message orientation: Hier worden de berichten gedefinieerd. Welk antwoordbericht wordt er gestuurd op een vraagbericht?
- Description orientation: Een beschrijving van de service die leesbaar is door een computer. Enkel wat nodig is om de service te gebruiken hoort hier thuis.
- Granularity: De service mag maar een beperkt aantal berichten ondersteunen. Hierdoor krijgen we grote en complexe berichten.
- Network orientation: De service wordt aangesproken over het netwerk. Dit kenmerk is niet noodzakelijk.
- Platform neutral: De berichten moeten in een gestandardiseerd formaat zijn. (vaak XML) De service moet onafhankelijk zijn van OS, taal, ...

SOA is een goed idee. Het heeft echter ook verschillende problemen.

- 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 moet opgevangen worden
- Wat als n partner incompatibel wordt?

### 10.2.3 REST

REST staat voor Representational State Transfer. Net zoals SOA legt het bepaalde kenmerken vast. Als men hieraan voldoet is de service RESTful. De webservice kan eender wat zijn. Het werd vastgelegd door Roy Fielding. REST werd eerst gedefinieerd in de context van HTTP, maar eender welk protocol is mogelijk (HTTP, SOAP, XML, ...).

REST is eenvoudig in gebruik. Ook is het bron georiënteerd, de service wordt geïdentificeerd door een URI (Uniform resource identifier).

Zoals reeds eerder gezegd legt REST bepaalde kenmerken vast. De architecturale principes zijn de volgende:

- Gebruik HTTP-methodes expliciet
- Wees statusloos
- URIs hebben een structuur analoog aan directorys
- Wissel XML of JSON uit

De HTTP methodes laat men overeenstemmen met hun eigenlijke bedoeling. Men kan dus een analogie maken met CRUD (create, read, update en delete). Create wordt dan gelijk aan POST, read aan GET, update aan PUT, en delete aan DELETE.

- POST: bron maken op server
- GET: bron ophalen van server
  - Geen neveneffecten (veranderingen op server)
  - Idempotent
- PUT: status van een bron aanpassen
- DELETE: bron verwijderen

REST is statusloos. Elke aanvraag staat dus los van de vorige, er worden geen sessiegegevens bijgehouden door de server (Dus wel door de cliënt indien nodig).

De voordelen hiervan zijn dat men de aanvraag aan een willekeurige server kan doen. Men doet dus aan load-balancing omdat de last kan gespreid worden over servers heen. (Een server kan een aanvraag doorsturen indien nodig) Ook is de betrouwbaarheid hierdoor verhoogd. Men kan aan failover doen. Wat dus het wisselen van server is, als de andere malfunctioneert.

Uit de URI bij REST moet men kunnen afleiden waarvoor de bron staat, en waarop de actie moet toegepast worden, het is een zelfstandig naamwoord. De actie leidt men af uit de gebruikte HTTP methode, dit is het werkwoord in de zin. Zie hiervoor.

Listing 10.1: Voorbeeld van een REST aanvraag

```
POST /users HTTP/1.1
```

De URIs volgen een directorystructuur, ze zijn dus hiërarchisch opgebouwd, en zijn te beschouwen als een boomstructuur. De URI moet ook eenvoudig, verstaandbaar en voorspelbaar zijn.

Extra tips bij het opstellen van de URI zijn de volgende.

- Verberg extensies (.jsp, .php, .asp, ): URIs blijven bij veranderende technologie
- Gebruik kleine letters
- Vervang spaties door - of _
- Vermijd querystrings
- Vermijd 404 Not Found
- Statisch (onveranderlijke) links: bladwijzers

De typische inhoud van een antwoordbericht bevat de huidige toestand van de bron. Het HTTP-body is eenvoudig, leesbaar en bestaat vaak uit XML of JSON (Javascript Object Notation)[1].

Rest is minder afhankelijk van applicatieservers, en combineerbaar met Ajax. Het is echter niet altijd de juiste keuze.

### 10.2.4 Technologieën

#### SOAP

SOAP staat voor Simple Object Access Protocol (soms ook Service Oriented Architecture Protocol). Het was origineel van Microsoft, ondertussen is het een W3C standaard. Het protocol is gebaseerd op XML. Er gebeurt een uitwisseling van informatie in een gedecentraliseerde, gedistribueerde omgeving (XML berichten). Transport gebeurt opnieuw via het web, dus meestal via HTTP en SMTP.

Indien SOAP via HTTP verloopt, zullen de eerste lijnen en de headerlijnen blijven staan. Het corpus echter, wordt in de vraag en ook in het antwoord vervangen door XML. Een zelfde verloop vindt plaats bij FTP en SMTP.

De XML corpus bestaat uit XML-tags die een SOAP-bericht beschrijven en die een framework geven voor de inhoud van het bericht. Binnenin deze tags staat er XML inhoud. Er zijn een aantal regels voor servers die SOAP berichten ontvangen.

Listing 10.2: Voorbeeld van een SOAP bericht

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"
```

[1] Merk de analogie op met AJAX

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Zoals we kunnen zien bestaat elk bericht uit een soap-envelop, een soap-hoofding en een soap-body[2].

De soap-envelop heeft als optioneel attribuut encodingStyle, dit geeft aan hoe het geserialiseerd wordt. Het is het rootelement van een soap bericht, en dus verplicht.

De soap-hoofding bevat extra richtlijnen en informatie voor de SOAP-server. Zoals transacties, beveiliging, filters, ... Dit element is optioneel.

Het soap-body bevat de uit te wisselen info in XML formaat. Dit XML formaat is onafhankelijk van de SOAP structuur, en dus domein-specifiek. De info kan verschillende vormen aanmeten. Het kan data zijn, of een resultaat van een methode of net de aanroep van een methode. Het soap-body element is verplicht.

De SOAP berichten volgen een bepaald berichtenpad. Het bestaat uit SOAP-knopen. Het begint bij de SOAP-zender, en passeert nul of meerdere tussenliggende SOAP-servers, op weg naar de SOAP-ontvanger. Een tussenliggende SOAP-server stuurt het bericht door, nadat hij de voor hem bedoelde SOAP-hoofdingen heeft verwerkt.

Elke SOAP-knoop heeft 1 of meerdere rollen. Ze zijn ofwel door de applicatie zelf gespecifieerd als proxy, beveiliging, caching, ... Ofwel door SOAP gespecifieerd. De mogelijkheden zijn dan

[2]Als ik douche heb ik dit ook. Flauwe mop? Schrijf dan zelf eens een cursus, dan kan je er betere maken!

de volgende.

- none: niet bedoeld voor een specifieke knoop
- next: ontvanger en alle tussenliggende knopen
- ultimateReceiver: de ontvanger

De SOAP-hoofding kan aangeven wat de rol van een bepaalde server is, dit in het attribuut actor. De waarde hiervan is een rol gespecifieerd door SOAP. Bijvoorbeeld voor een next knoop: http://schemas.xmlsoap.org/soap/actor/next.

De server moet zich aan bepaalde regels houden, zoals reeds eerder gezegd.

- Identificatie van de stukken van het SOAP-bericht die voor de huidige server bedoeld zijn
- Nagaan of deze server alle delen van de headers die bedoeld zijn voor deze server en die het attribuut mustUnderstand hebben, ondersteunt. Indien niet fout genereren
- Uitvoeren van de stukken van de headers bedoeld voor deze server
- Indien niet de SOAP-ontvanger: alle headers bedoeld voor deze server verwijderen en bericht doorsturen

Het loopt fout met SOAP wanneer een bericht niet te begrijpen valt, of er een fout gebeurt bij het te behandelen bericht. Als dit gebeurt, wordt er een element bijgevoegd in het body-element, namelijk het fault-element. Dit element bevat een foutcode, een faultstring (leesbare verklaring), en een detail attribuut met info over het probleem.

Men kan een vergelijking maken tussen REST en SOAP.

REST is volledig statusloos, er gebeurt ook caching waardoor de performantie verhoogd. De service en de gebruiker kennen de context. Er is echter geen formele beschrijving zoals bij WSDL (zie **??**). Ook is REST handig bij beperkte bandbreedte, zoals bij mobiele toestellen het geval

is. Men kan het makkelijk invoegen in een bestaande website met AJAX.

SOAP anderzijds, definieert een formeel contract via WSDL. Ook heeft een complexere functionaliteit, en ondersteunt het dus meer dan CRUD. Beveiliging, transacties, ... zijn mogelijk. Er is asynchrone afhandeling.

### WSDL

WSDL staat voor Web Service Description Language. Het is een XML-standaard van het W3C. Het beschrijft de webservice door middel van en interface en implementatiedetails. Het bepaalt hoe de service requester de service provider moet aanspreken. Momenteel zijn er twee gangbare 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/)

De specificatie van de interface van de webservice is leesbaar door computers. De interface definieert het formaat van de berichten, de datatypes, de transportprotocollen, de serialisatieformaten voor transport, de netwerklocaties en het patroon om berichten uit te wisselen.

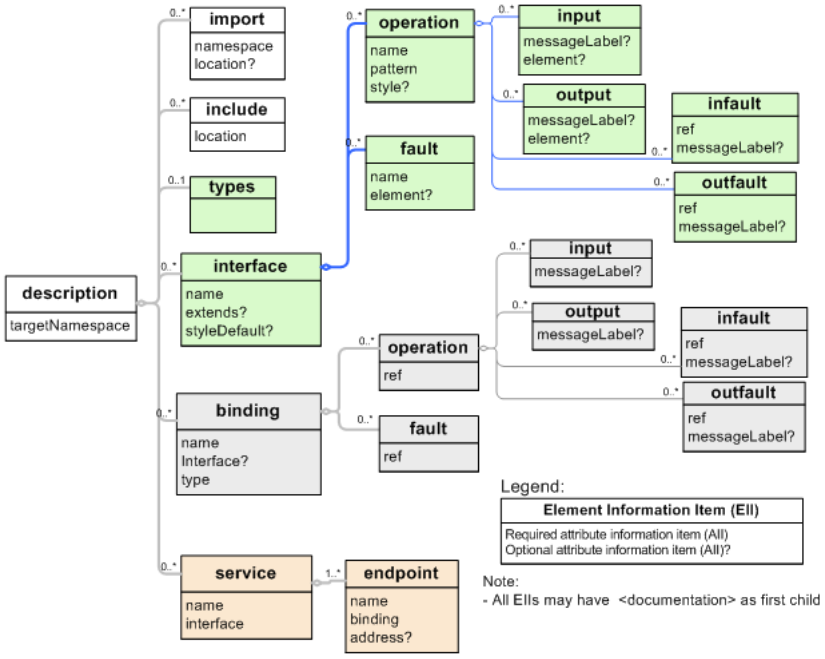
Listing 10.3: Een voorbeeld van WSDL 1.1

```
<definitions name= BoekAgent" targetNamespace="http://www.zwiftbooks.com/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://www.zwiftbooks.com/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" >

  <message name= GetDeliveryInfoRequest">
    <part name= zipcode" type="xsd:integer"/>
    <part name= isbn" type="xsd:string"/>
  </message>
  <message name= GetDeliveryInfoResponse">
    <part name="result" type="xsd:duration"/>
  </message>
```

```
<portType name= BoekAgent">
  <operation name= GetDeliveryInfo" parameterOrder= zipcode isbn">
    <input message="tns:GetDeliveryInfoRequest" name="GetDeliveryInfoRequest
        "/>
    <output message="tns:GetDeliveryInfoResponse " name="
        GetDeliveryInfoResponse"/>
  </operation>
  <!-- andere diensten -->
</portType>

<binding name= BoekAgentBinding" type="tns:BoekAgent">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name= GetDeliveryInfo">
<soap:operation soapAction= http://zbooks.org/action/ZBooks.Get-DeliveryInfo
    "/>
  <input>
    <soap:body use="encoded" namespace= http://zbooks.org/message/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>

  <output>
    <soap:body use="encoded" namespace="http://zbooks.org/message/"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
    </operation>
 </binding>
</definitions>
```

De structuur is als volgt voor WSDL 1.1.

- Types: gebruikte types beschreven in XML Schema
- Messages: mogelijke berichten
- Port Types: verzameling van alle opdrachten van een webservice
- Bindings: beschrijving concreet formaat van de berichten en het gebruikte protocol
- Services: beschrijving van een webservice bestaande uit verschillende 'port types'




Figuur 10.2: WSDL 2.0 structuur

De structuur van WSDL 2.0 (ook te zien in figuur **??**) is als volgt.

- types: beschrijving berichten in XML Schema
- interface: abstracte funcionaliteit
- binding: hoe toegang tot diensten
- service: waar toegang tot diensten

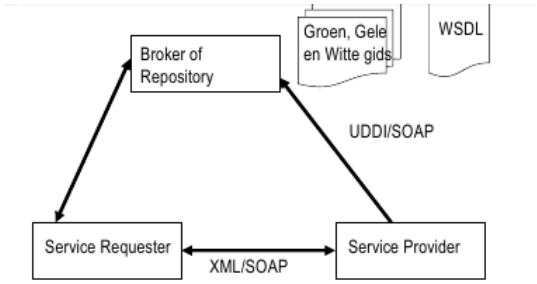
Listing 10.4: Een voorbeeld van WSDL 2.0

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

  <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>

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

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

    <service name="reservationService" interface="tns:reservationInterface">
      <endpoint name="reservationEndpoint" binding="tns:reservationSOAPBinding"
        address ="http://greath.example.com/2004/reservation"/>
    </service>

  </description>
```

Er zijn verschillende soorten communicatie mogelijk.

- Vraag/antwoord: het vaakst gebruikt.




Figuur 10.3: Discovery service

- Enrichtingscommunicatie (one-way contract): van cliënt naar server, cliënt verwacht geen antwoord.
- Solicit-response: Van server naar cliënt, server verwacht antwoord.
- Notification: Van server naar cliënt, server verwacht geen antwoord.

**UDDI**

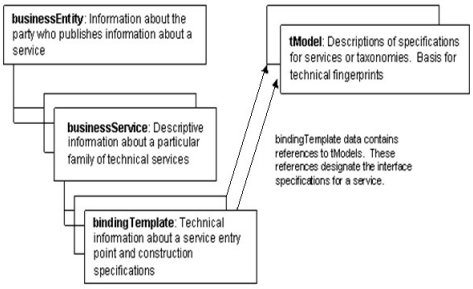
Om een webservice te vinden heeft men twee mogelijkheden. Ofwel kent de service requester de service provider. Ofwel via een discovery service.

Een discovery service wordt ook wel een repository, of een broker genoemd. Men kan het vergelijken met een telefoonboek (Witte, gele en groene gids). De broker publiceert verschillende diensten. De beschikbare info vindt men terug onder de vorm van een technische beschrijving en een functionele beschrijving (vaak in UDDI).

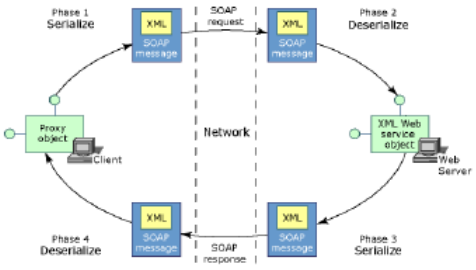
UDDI staat voor Universal Description, Discovery and Integration. Het is een XML protocol, dat men gebruikt om webservices te beschrijven en te registreren bij een broker.

De broker bevat de volgende info in zijn register.

- Witte gids: contactinformatie: naam, adres, webstek, . . . , alsook registratie van de provider
- Gele gids: categorien van diensten, hoe vindt een client een webservice?
- Groene gids: technische informatie, hoe kan een client gebruik maken van een webservice?




Figuur 10.4: UDDI elementen




Figuur 10.5: SOAP in Java

UDDI bestaat uit de volgende elementen om deze door te geven aan de broker.

- businessEntity: Informatie over bepaalde provider
- businessService: Beschrijving van een bepaalde webservice bestaande uit n of meerdere diensten
- bindingTemplate: Technische informatie, informatie voor implementatie
- tModel: Technische informatie per dienst

### 10.2.5 In Java

In Java kan men JAX-WS (Java API for XML Web Services) gebruiken als men met SOAP wil werken, en JAX-RS als men met REST wil werken.

Als we het voorbeeld in figuur 10.5 bekijken, zien we hoe JAX-WS werkt. We zien dat er een proxy-object is. Dit is een lokaal object voor de clientapplicatie die de webservice 'voorstelt'.

De clientapplicatie roept de methodes van het proxyobject op, waarna de infrastructuur dit serialiseert tot een SOAP bericht. Het bericht wordt verstuurd over het netwerk naar de ontvanger via HTTP.

Op de server wordt het bericht gedeserialiseerd. Als er nog geen webservice object bestaat wordt dit aangemaakt. De methode van de webservice wordt uitgevoerd, waarna dit geserialiseerd wordt en met SOAP over het netwerk wordt verstuurd.

De client ontvangt het SOAP-antwoord, dit wordt gedeserialiseerd en doorgegeven aan het proxyobject. Het proxyobject tenslotte, genereert een resultaat voor de clientapplicatie.

Voor een voorbeeld kan men in de slides kijken, ik denk echter dat het principe duidelijk is.

De informatie voor het proxyobject wordt met WSDL (zie **??**) gedefinieerd. Dit wordt door de server gepubliceerd, nadat hij de webservice gemaakt heeft (de webservice wordt gewrapt in een servlet). Op basis van de WSDL worden proxy-klassen gegenereerd, waarvan de client gebruik maakt.

Merk op dat het op de server nodig is om gebruikte klassen serialiseerbaar te maken, anders kan de data niet met SOAP verstuurd worden.
Op de server gebruikt men annotaties om de service te definieëren.

Listing 10.5: Webservice op server

```
@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;
    }
  }
```

```
}
```

Men kan de proxy klassen automatisch laten genereren met Netbeans. Hiervoor is enkel de URI van de WSDL vereist. Als men dan een methode wil oproepen van de service kan men zelfs makkelijk met code insertion werken. Als men Call Web Service Operation selecteert, wordt automatisch de nodige code gegenereerd.

Als men een REST service wil maken, moet men opnieuw men annotaties werken. Men moet het pad naar de klasse annoteren, alsook de methodes (deelpad, http-methode, invoer-uitvoer, parameters).

Enkele voorbeelden hieronder.

Listing 10.6: REST in Java

```
@Path("/Agenda")
public class AgendaResource {     }

package org.netbeans.rest.application.config;
@javax.ws.rs.ApplicationPath("resources")
public class ApplicationConfig extends javax.ws.rs.core.Application {
}

@GET
@Produces("application/xml")
public String getAgenda() {    }

@PUT
@Consumes("application/xml")
public void setAgendaItem(String inhoud) {    }

@Path("{datum}/{id}")
@DELETE
public void deleteAgendaItem(@PathParam("datum") String
datumString, @PathParam("id") String idString) {    }
```

# Hoofdstuk 11

# Windows Communication Foundation

WCF staat voor Windows Communication Foundation, en is ontwikkeld door Microsoft. Het maakt gedistribueerde applicaties mogelijk, en laat dus communicatie tussen verschillende applicaties toe. WCF laat toe om service-oriented applicaties te maken (zie 10.2.2). Het maakt dus gebruik van webservices, en gebruikt SOAP of REST. Transport gebeurt over named pipes, TCP en HTTP.

Er is een client-server relatie tussen endpoints. Een service bestaat uit verschillende endpoints. Een endpoint is een plaats naar waar men berichten stuurt. Hoe men dit doet en de structuur van de berichten legt men vast in de architectuur

## 11.1 Architectuur WCF

### 11.1.1 Contracten

- Data Contract: De parameters in het bericht, om dit te definiëren gebruikt men XSD (XML schema definition)
- Message Contract: De structuur van het bericht
- Service Contract: De beschikbare interface en de signatuur van de methodes

- Bindings, Policies: Hierin legt men vast hoe men met de service moet interageren, alsook de beveiliging

### 11.1.2 Gedrag

Het gedrag van de WCF webservice noemt men Behaviour service runtime. Het is het eigenlijke gedrag van de applicatie. Dit omvat het aantal berichten dat kan verwerkt worden, wat er bij fouten gedaan moet worden, welke metadata beschikbaar moet zijn voor de buitenwereld, hoeveel instanties er tegelijk kunnen zijn van een service, hoe transacties verlopen, ...

### 11.1.3 Berichten

Er zijn verschillende kanalen die berichten verwerken. Deze kanalen zijn op elkaar gestacked, en elk bericht moet erdoor passeren. Zo heeft men bijvoorbeeld een kanaal voor authenticatie. Er zijn twee types kanalen.

- Transport: Lezen en schrijven van berichten van en naar het netwerk (via TCP, UDP, ...)
- Protocol: Lezen en schrijven van hoofdingen, ... (met behulp van SOAP)

Het onderste kanaal op de stack is steeds een transport type kanaal.

### 11.1.4 Hosting

Een webservice is een programma, dat dus ook uitgevoerd moet worden. Men kan dit op verschillende manieren doen. Als executable (self-hosted service), of gehost in een externe omgeving (op een webserver, of als windows service).

## 11.2 Ontwikkeling WCF

### 11.2.1 Ontwerp servicecontract

De service bestaat uit operaties. Bij elke methode komt er het attribuut OperationContract. De methode moet serialiseerbare parameters en een serialiseerbare teruggeefwaarde hebben, en deze moeten ook kopiën zijn in plaats van referenties.

Indien men eigen datatypes gebruikt, moet men hier ook een contract voor opstellen, met attributen DataContract en DataMember

.

De interface zelf annoteert men met ServiceContract. Dit alles zit in de namespace System.ServiceModel.

Listing 11.1: Webservice

```
[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);
}
```

Listing 11.2: Eigen datatypes

```
[DataContract]
public class CompositeType
{
    bool boolValue = true;
    string stringValue = "Hello ";

    [DataMember]
    public bool BoolValue
    {
        get { return boolValue; }
        set { boolValue = value; }
    }

    [DataMember]
    public string StringValue
    {
```

```
        get { return stringValue; }
        set { stringValue = value; }
    }
}
```

Er zijn twee soorten communicatie mogelijk: Eenrichtingscommunicatie (one-way contract) en tweerichtingscommunicatie (duplex contract).

Bij een one-way contract stuurt de client een bericht naar de server, en verwacht geen antwoord terug. De methode heeft dus geen returnwaarde, geen uitvoerparameters. Men moet de eigenschap IsOneWay dan wel op true zetten.

Listing 11.3: one-way contract

```
[OperationContract (IsOneWay=true)]
```

Bij tweerichtingscommunicatie kan de communicatie in beide richtingen gebeuren, het is echter geen vraag/antwoord systeem. De client start de communicatie op en houdt een kanaal open. De server kan hierlangs later berichten sturen. Een duplex contract kan dus event-like gedrag vertonen, aangezien de server berichten kan terugsturen naar de client. Men moet echter wel twee interfaces definiëren. Een gewone, en een call back contract.

Listing 11.4: Duplex contract

```
[ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples", SessionMode
    =SessionMode.Required,
                CallbackContract=typeof(ICalculatorDuplexCallback))]
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 Equals(double result);
    [OperationContract(IsOneWay = true)]
    void Equation(string eqn);
}
```

### 11.2.2 Implementatie contract

Nadat een interface is gedefinieerd, moet de service ook geïmplementeerd worden.

Listing 11.5: Implementatie interface

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
public class CalculatorService : ICalculatorDuplex
{
    double result = 0.0D;
    string equation;

    public CalculatorService()
    {
        equation = result.ToString();
    }

    public void Clear()
    {
        Callback.Equation(equation + " = " + result.ToString());
        equation = result.ToString();
    }

    public void AddTo(double n)
    {
        result += n;
        equation += " + " + n.ToString();
        Callback.Equals(result);
```

```
        }

        public void SubtractFrom(double n)
        {
            result -= n;
            equation += " - " + n.ToString();
            Callback.Equals(result);
        }

        public void MultiplyBy(double n)
        {
            result *= n;
            equation += " * " + n.ToString();
            Callback.Equals(result);
        }

        public void DivideBy(double n)
        {
            result /= n;
            equation += " / " + n.ToString();
            Callback.Equals(result);
        }

        ICalculatorDuplexCallback Callback
        {
            get
            {
                return OperationContext.Current.GetCallbackChannel<
                    ICalculatorDuplexCallback>();
            }
        }
    }
```

### 11.2.3 Configuratie

Configuratie is een belangrijk onderdeel van WCF. Men moet enkele details over het endpoint opgeven. Welke service bedoelt men? Waar vindt men de interface en de implementatie ervan? Wat is het URL ervan? Op welke manier gaat men communiceren? TCP? HTTP? Tekst? Bi-

nair? Al dan niet beveiligd?

De configuratie steekt men meestal in een aparte XML file, maar het kan ook in de code gebeuren.

Listing 11.6: Configuratie in code

```
Uri baseAddress = new Uri("http://localhost:8000/ServiceModelSamples/Service");
ServiceHost ServiceHost selfHost = new ServiceHost(typeof(CalculatorService),
    baseAddress);
try {
    SelfHost.AddServiceEndpoint( typeof(ICalculator), new WSHttpBinding()," 
        CalculatorService");
    ... // gedrag instellen en service runnen
} catch (CommunicationException ce) {
    Console.WriteLine("An exception occurred:{0}", ce.Message); selfHost.Abort();
}
```

Listing 11.7: Configuratie in XML file

```
 <?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>    <services>
      <!-- This section is optional with the new configuration model
           introduced in .NET Framework 4. -->
      <service name="Microsoft.ServiceModel.Samples.CalculatorService"
               behaviorConfiguration="CalculatorServiceBehavior">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/ServiceModelSamples/service
                "/>
          </baseAddresses>
        </host>
        <!-- this endpoint is exposed at the base address provided by host: http
            ://localhost:8000/ServiceModelSamples/service  -->
        <endpoint address=""
                  binding="wsHttpBinding"
                  contract="Microsoft.ServiceModel.Samples.ICalculator" />
        <!-- the mex endpoint is exposed at http://localhost:8000/
            ServiceModelSamples/service/mex -->
        <endpoint address="mex"
```

```
                binding="mexHttpBinding"
                contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="CalculatorServiceBehavior">
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="False"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Men kan een eigen binding specifieren, maar er zijn reeds voorgedefinieerde bindingen.

- 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 n machine

### 11.2.4 Hosting

Zoals reeds eerder gezegd kan men de applicatie op verschillende manieren hosten. De code service is onafhankelijk van de host. Het windows proces (IIS, Console, service) ondersteunt de managed code.

Listing 11.8: Hosten in console

```
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();
  }
```

Listing 11.9: Host als windowsservice

```
    public class CalculatorWindowsService : ServiceBase
      {
          public ServiceHost serviceHost = null;
          public CalculatorWindowsService()
          {
              // Name the Windows Service
              ServiceName = "WCFWindowsServiceSample";
          }

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

          // Start the Windows service.
          protected override void OnStart(string[] args)
          {
              if (serviceHost != null)
              {
                  serviceHost.Close();
              }

              // Create a ServiceHost for the CalculatorService type and
              // provide the base address.
              serviceHost = new ServiceHost(typeof(CalculatorService));
```

```
            // Open the ServiceHostBase to create listeners and start
            // listening for messages.
            serviceHost.Open();
        }

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

    // Provide the ProjectInstaller class which allows
    // the service to be installed by the Installutil.exe tool
    [RunInstaller(true)]
    public class ProjectInstaller : Installer
    {
        private ServiceProcessInstaller process;
        private ServiceInstaller service;

        public ProjectInstaller()
        {
            process = new ServiceProcessInstaller();
            process.Account = ServiceAccount.LocalSystem;
            service = new ServiceInstaller();
            service.ServiceName = "WCFWindowsServiceSample";
            Installers.Add(process);
            Installers.Add(service);
        }
    }
}
```

Een service in IIS is vrij makkelijk, men implementeerd de service en laat hem gewoon 'runnen'.

### 11.2.5 Ontwikkelen client

Als men de service wil gebruiken, moet men het contract, de bindings en het adres van de service ophalen. Daarvoor bestaat de handige tool svcutil.exe. Deze genereert het configuratiebestand en de proxy klassen (zie webservices). Men voegt deze beide toe aan het project. Daarna kan men de klasse instantieren (de proxy aanmaken), hem gebruiken, en na afloop weer vernietigen.

Listing 11.10: svcutil.exe

```
svcutil.exe /language:cs /out:generatedProxy.cs /config:app.config
  http://localhost:8000/ServiceModelSamples/service
```

Listing 11.11: Gebruik van de voorbeeld webservice

```
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();
  }
}
```

# Bibliografie

[1] http://en.wikipedia.org/

[2] http://netbeans.org/kb/docs/web/ajax-quickstart.html

[3] http://www.simple-talk.com/dotnet/asp.net/calling-cross-domain-web-servi

[4] http://stackoverflow.com/questions/218399/ajax-get-requests-use-paramete

[5] http://www.tizag.com/ajaxTutorial/ajaxxmlhttprequest.php

[6] http://www.w3schools.com/ajax/ajax_xmlhttprequest_
onreadystatechange.asp

[7] http://www.w3schools.com/jsref/met_win_settimeout.asp

[8] http://helephant.com/2008/08/23/javascript-anonymous-functions/

[9] http://stackoverflow.com/questions/245062/whats-the-difference-between-j

[10] http://www.htmlgoodies.com/beyond/javascript/article.php/3470971/
Java-vs-JavaScript.htm

[11] http://msdn.microsoft.com/

[12] http://stackoverflow.com/questions/303013/whats-the-point-of-a-datacontr

[13] http://keithelder.net/2008/01/17/exposing-a-wcf-service-with-multiple-bi

[14] http://stackoverflow.com/questions/2188909/generate-datacontract-from-xs

[15] http://www.tutorialspoint.com/uddi/uddi_elements.htm

[16] http://java.boot.by/wsd-guide/ch03.html

[17] http://graal.ens-lyon.fr/~cperez/web/_media/ens:2_services.pdf

[18] http://www.global-computing.org/pres/ws.pdf

[19] http://www.mkyong.com/hibernate/hibernate-cascade-example-save-update-de

[20] http://weblogs.asp.net/ricardoperes/archive/2009/06/21/nhibernate-mappings-one-to-one.aspx

[21] http://nhforge.org/wikis/howtonh/lazy-loaded-one-to-one-with-nhibernate.aspx

[22] http://ayende.com/blog/2381/nhibernate-one-to-one

[23] http://design-antony.blogspot.com/2007/07/aggregation-vs-composition.html

[24] http://www.agiledata.org/essays/mappingObjects.html

[25] http://nhforge.org/wikis/howtonh/your-first-nhibernate-based-application.aspx

[26] http://www.ibm.com/developerworks/java/library/j-jsf2fu2/index.html

[27] http://www.theserverside.com/news/thread.tss?thread_id=35900

[28] http://docs.oracle.com/javaee/5/tutorial/doc/bnavg.html

[29] http://webmoli.com/2008/08/12/arbitrary-validation-in-jsf/

[30] http://exadel.com/web/portal/jsftutorial-validation

[31] http://johnderinger.wordpress.com/2007/10/11/validators-and-converters/

[32] http://docs.oracle.com/javaee/5/tutorial/doc/bnaxb.html

[33] http://docs.oracle.com/javaee/5/tutorial/doc/bnaqq.html

[34] http://www.oracle.com/technetwork/java/index.html

[35] http://www.mkyong.com/jsf2/jsf-2-0-hello-world-example/

[36] http://www.roseindia.net/jsf/

[37] http://myfaces.apache.org/core12/myfaces-api/apidocs/javax/faces/component/UICommand.html

[38] http://www.ibm.com/developerworks/java/library/j-jsf4/

[39] http://blogs.oracle.com/learnwithpavan/entry/difference_between_and_in_adfj

[40] http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

[41] Cursus Computernetwerken IV

[42] Slides Computernetwerken IV

[43] http://struts.apache.org/2.0.6/struts2-core/apidocs/com/opensymphony/xwork2/ActionSupport.html

[44] http://stackoverflow.com/questions/870279/where-to-put-struts-xml

[45] http://struts.apache.org/2.0.11.1/docs/ognl.html

[46] http://struts.apache.org/2.0.6/docs/how-do-we-change-locales.html

[47] http://struts.1045723.n5.nabble.com/Fw-RE-ActionSupport-input-what-s-it-html

[48] http://struts.apache.org/2.0.14/docs/fieldexpression-validator.html