Ondanks het economische tij blijft het gebruik van internet groeien. "In 1999 had slechts 19 procent van de Europeanen ouder dan zestien jaar regelmatig toegang tot internet. Dit aandeel is in een paar jaar tijd verdubbeld tot 39 procent in 2001" aldus analist Forrester. Onder internetgebruik wordt door de meeste mensen het opvragen van statische HTML pagina's verstaan. In de begindagen van internet werd dan ook voornamelijk statische informatie verspreid. Later kwamen er steeds meer websites die de gebruiker gegevens lieten terug sturen naar de webserver, het zogenaamde posten van HTML formulieren.

User interface ontkoppeling

Aanpassingen met behulp van custom JSP tags

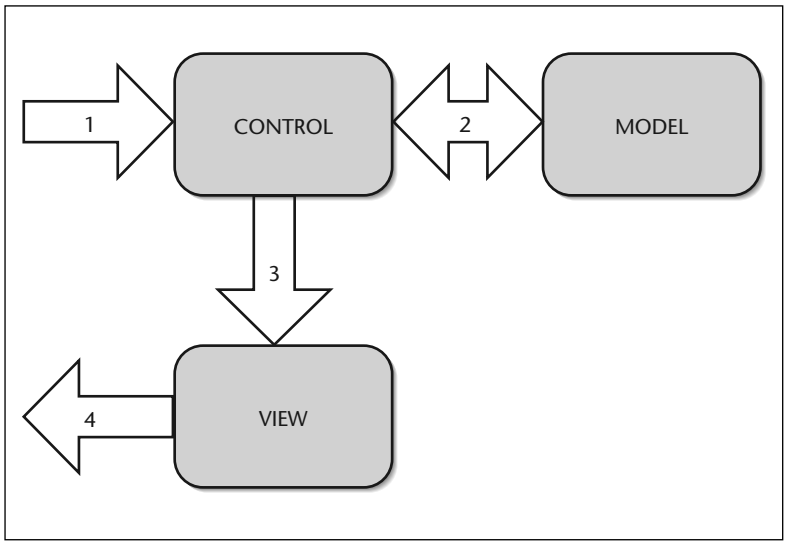
Deze interactiviteit maakte websites aantrekkelijker en functioneler. Langzaamaan ontstonden de eerste webapplicaties, waarbij een serverapplicatie voorzien werd van een client HTML user interface. HTML is van nature echter volstrekt niet geschikt om als applicatie user interface te fungeren. Het enige datatype dat door HTML forms wordt ondersteund is de string 'properties' van form. Elementen zoals read-only en visible worden in tekstformaat naar de browser gestuurd en daar geïnterpreteerd. Door de eeuwige 'browseroorlogen' kunnen ontwikkelaars er nooit zeker van zijn dat hun HTML forms op alle typen browsers exact gelijk worden weergegeven. Daarnaast bevat HTML de nodige inconsistenties zoals het gebruik van de key-words 'selected' en 'checked' om voor verschillende typen selectie-elementen de geselecteerde waarde aan te duiden. Het belangrijkste probleem is echter de beveiliging van de applicatie. HTML forms zijn altijd als platte tekst aan de browser client kant beschikbaar: een eenvoudige rechtermuisklik '*view source*' op de pagina en de onderliggende HTML code wordt zichtbaar gemaakt. Voor (statische) websites is dit geen probleem, maar een applicatie moet gegevens voor een specifieke gebruiker onzichtbaar kunnen maken. Met HTML forms kan een ervaren gebruiker deze gegevens altijd zichtbaar maken. Omgekeerd kan een (zeer ervaren) gebruiker gegevens altijd naar de server sturen ook indien het HTML form deze opdracht niet had gekregen.

In het algemeen zijn HTML forms net goed genoeg om statische pagina's te voorzien van de mogelijkheid om de gebruiker gegevens terug te laten sturen naar de server. Als user interface voor applicaties blijkt een aantal toevoegingen noodzakelijk.

Form Processoren Voor de server side verwerking van geposte forms wordt meestal gebruik gemaakt van een Form Processor. Server side talen als PHP en Perl kennen FormProc, phpFormGenerator, EasyForm en QuickForm [zie referentielijst onderaan dit artikel] als mogelijke form processoren. Voor Java is Struts [zie referentielijst] de meest gebruikte form processor maar veel web applicatie ontwikkelaars ontwerpen hun eigen form processor of breiden een bestaande uit. Dit artikel gaat uit van webgebaseerde applicaties bij het uitbreiden van Java form processoren.

> Het belangrijkste voordeel van het MVC pattern is de scheiding van de logica en de representatie van de applicatie

Java form processoren maken meestal gebruik van het model-view-control paradigma (zie figuur 1) ook wel aangeduid als het MVC pattern. Dit pattern gaat uit van een inkomende client request (pijl 1 in de figuur) welke door de controller component verwerkt wordt. Dit kan bijvoorbeeld het opvragen van product informatie zijn. De controller zal soms het business model



FIGUUR 1. MVC Pattern

component (pijl 2) moeten raadplegen, hier wordt bijvoorbeeld de product informatie uit een database gehaald. Vervolgens forward de controller de verzamelde product informatie naar de view component (pijl 3) welke de product informatie op de juiste manier rendered en het volgende scherm aan de gebruiker toont (pijl 4). Bij een web applicatie bestaat pijl 1 uit het sturen van een HTML form naar de server en pijl 4 van het terugsturen van de volgende HTML pagina naar de browser. Alle drie de MVC componenten bevinden zich dus aan de 'server' kant. De control-componenten worden meestal uitgevoerd als Servlets en de view-componenten als JSP pagina's. Zowel control als view bevinden zich bij Java dus in een servlet container. De model-componenten kunnen zich ook in een servlet container bevinden of (de meest uitgebreide mogelijkheid) in een EJB container.

> De control-componenten worden meestal uitgevoerd als servlets en de view-componenten als JSP pagina's

Het belangrijkste voordeel van het MVC pattern is de strikte scheiding van applicatie logica (control component, en model) en de representatie van de applicatie (view component).

Zoals de naam Form Processor al aangeeft zijn de meeste processoren goed ingericht op pijl 1, het verwerken van verstuurde HTML forms naar de server. De belangrijkste functionaliteiten die hier geboden worden zijn het omzetten van tekst Strings in de interne control datatypen, het valideren van de binnengekomen data tegen validatieregels (zoals lengte en formaat) en het doorsturen van verschillende typen request naar specifieke methoden om deze requests uit te voeren (in form processor-jargon wordt dit vaak 'actions' genoemd).

VERWEVENHEID Form processoren zijn vaak minder goed in pijl 3 en 4, het representeren van de gegevens. De belangrijkste oorzaak is de verwevenheid van form element properties met de applicatie logica. Een form element is een veld op een HTML pagina dat zich anders dient te presenteren afhankelijk van de properties van het veld, zoals bijvoorbeeld read-only of editable. Het best is dit duidelijk te maken in een voorbeeld. Stel je de volgende situatie voor:

In een applicatie kennen wij de entiteit ORDER welke het veld KORTINGSPERCENTAGE bevat. We onderscheiden drie typen gebruikers rollen:

- MANAGER, bevoegd om kortingspercentages van orders aan te passen. In dit geval moet het veld kortingspercentage eenvoudig zijn aan te passen.
- VERKOPER, bevoegd om kortingspercentages van een orders te zien. In dit geval moet het veld dus als 'platte' tekst kunnen worden weergegeven.
- KLANT, bevoegd om orders te plaatsen maar niet het kortingspercentage van hun orders te zien. In plaats van de werkelijke kortingspercentage worden bijvoorbeeld sterretjes (***) getoond.

In een Java webgebaseerde applicatie zal de presentatie van een orderscherm bij voorkeur worden weergegeven door één JSP pagina welke onder andere het veld kortingspercentage bevat. De hiernaast afgebeelde JSP code, waarbij gebruik wordt gemaakt van zogenaamde scriptlets (vet weergegeven), is één van de mogelijke manieren om dit te bereiken.

```
...
<tr>
  <td>Kortings percentage</td>
  <td>
    <input type="text" name="korting"
value="<%=
order.getKorting();%>">
  </td>
</tr>
...
```

CODE SNAPSHOT 1. Order JSP zonder autorisatie.

De JSP pagina bestaat uit veel gewone HTML code met als toevoeging de scriptlet <%= order.getKorting()

%> welke de methode getKorting() aanroept en de geretourneerde waarde plaatst in een HTML <input> element. Het label wordt weergegeven in de table cell ervoor. De pagina wordt echter nog door elk type gebruiker op dezelfde manier weergegeven. We breiden deze code daarom uit met de eerder genoemde gebruikers autorisaties (zie snapshot 2).

Afhankelijk van het type gebruikers rol wordt nu door de JSP pagina een input veld, read-only tekst of sterretjes weergegeven. Het probleem nu is dat we autorisatie logica hebben gecodeerd in de JSP pagina, ofwel in de view component. Dit maakt het beheer en onderhoud van het systeem op langere termijn lastig. De verwevenheid van business logica en presentatie is een veel voorkomend probleem bij webgebaseerde applicaties. Eigenlijk zouden we gebruik willen maken van user interface componenten op dezelfde wijze als dat we van deze componenten in niet-webgebaseerde clients gebruik kunnen maken.

CUSTOM JSP TAGS Een mogelijke oplossing is het gebruik van Custom JSP Tags, een uitbreiding op de normale JSP tag Library. Normale JSP tags zijn bijvoorbeeld de hierboven ook gebruikte scriptlets, <%= %>. Datgene wat tussen de scriptlet tags staat wordt uitgevoerd als Java code. Custom Tags geven ontwikkelaars de mogelijkheid de normale JSP tag library uit te breiden. De Custom Tags in JSP Pages Tutorial [zie referentielijst achteraan dit artikel] geven een goed overzicht met betrekking tot custom tags.

Stel dat we het orderscherm willen ontdoen van alle business logica. Slechts de presentatie blijft dan over. De pagina zou er dan bijvoorbeeld uit kunnen zien zoals in snapshot 3.

Hierin komt de custom tag <ui:textveld> voor dat het attribuut; name="korting" meekrijgt. Textveld is hier de naam van de tag, ui is de logische naam van de tag library waartoe textveld behoort. Deze custom tag library dient aan de JSP pagina bekend te worden gemaakt

```
...
<%@ taglib uri="/user-interface" prefix="ui" %>
...
<tr>
  <td>Kortings percentage</td>
  <td>
    <ui:textveld name="korting" />
  </td>
</tr>
...
```

CODE SNAPSHOT 3. Order JSP met custom tag

```
...
<% if(role.isManager()) { %>

<tr>
  <td>Kortings percentage</td>
  <td>
    <input type="text" name="korting" value="<%=
order.getKorting();%>">
  </td>
</tr>

<% } else if(role.isVerkoper()) { %>

<tr>
  <td>Kortings percentage</td>
  <td>
    <%= order.getKorting();%>
  </td>
</tr>

<% } else { %>

<tr>
  <td>Kortings percentage</td>
  <td>***</td>
</tr>

<% } %>
...
```

CODE SNAPSHOT 2. Order JSP met autorisatie

door middel van de directive <%@ taglib %>. De tag library zelf is een XML definitie file welke de servlet engine verteld welke custom tags in JSP pagina's voor mogen komen. De custom tag library user-interface bevat de definitie zoals in snapshot 4.

Deze vertelt de servlet container dat de tag; textveld gebruikt mag worden in JSP pagina's en verwerkt wordt

```
...
<tag>
  <name>textveld</name>
  <tagclass>view.TextVeldTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>tag om text velden mee weer te
geven.</info>
  <attribute>
     <name>name</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
  </attribute>
  ...
</tag>
...
```

CODE SNAPSHOT 4. User-interface custom tag library

```
...
public class TextVeldTag implements Tag
{
  private String _name;

  /**
   * Process start tag
   */
  public int doStartTag() throws JspException
{

    JspWriter out = pageContext.getOut();
    HttpSession session =
pageContext.getSession();
    UI ui =
(UI)session.getAttribute(USER_INTERFACE);
    UIElement veld = (UIElement)ui.get(_name);

    String html = "";

    if(!veld.isHidden()) {
      if(veld.isReadOnly()) {

        html += "<td>";
        html += veld.getValue();
        html += "</td>";
      } else {

        html += "<td>";
        html += "<input type=\"text\"
name=\""+ veld.getName() +"\" ";
        html += "value=\""+ veld.getValue()
+"\">";
        html += "</td>";
      }
    } else {

      html += "<td>";
      html += "***";
      html += "</td>";
    }

    try {

      out.print(html);

    } catch (IOException ioe) { ... }

    return Tag.EVAL_BODY_INCLUDE;
  }

  /**
   * Setter for _name.
   */
  public void setName(String name) {

    _name = name;
  }
}
```

CODE SNAPSHOT 5. Custom TextVeldTag class

```
...
public interface UIElement
{
  // rendering attributes
  public boolean isHidden();
  public void setHidden(boolean hidden);
  public boolean isReadOnly();
  public void setReadOnly(boolean readOnly);

  // value attributes
  public String getValue();
  public void setValue(String value);
  public String getName();
  public void setName(String name);
```

CODE SNAPSHOT 6. UIElement interface

door de Java class TextVeldTag. Deze class moeten we nog gaan coderen maar een aantal kenmerken kunnen we alvast weergeven. Zo zal de textveld tag in staat moeten zijn om:

- Een veld weer te geven als HTML <input> element.
- Een veld weer te geven als HTML text.
- Een veld niet weer te geven maar in plaats daarvan te vervangen door drie sterretjes ***

Hiernaast wordt een (sterk vereenvoudigd) voorbeeld gegeven van de class TextVeldTag. De tag maakt gebruik van het attribuut 'name' om te bepalen welk veld hij moet weergeven (zie snapshot 5).

In dit voorbeeld wordt de string 'korting' welke als het attribuut name via de textveld tag wordt meegegeven gebruikt om een class van het type UIElement uit de sessie (HttpSession object) te lezen. In deze (vereenvoudigde) weergave om de scheiding tussen control modules en view modules aan te geven wordt gebruik gemaakt van een container class UI om de veldelementen vanuit de servlet control logica door te geven naar de view modules. De interface UIElement gebruiken we om de overgang van de business control modules naar de presentatie view modules te definiëren (zie snapshot 6).

Classes die deze interface implementeren kunnen in de control-modules worden gebruikt. Op deze wijze kunnen business regels worden gecodeerd daar waar zij thuishoren: in de control logica. De methode displayOrder(int id) geeft hiervan een voorbeeld (zie snapshot 7).

De methode displayOrder(int id) maakt in dit geval onderdeel uit van de business logica in de control modules. De methode retourneert een class UI welke wordt gebruikt als container voor user interface elementen. Merk op dat tussen control en view modules

```
public UI displayOrder(int id) {

  ...
  Order order = database.getOrder(id);
  ...
  UIElement uiOrder = new UITextVeld();
  uiOrder.setName("korting");
  uiOrder.setValue(order.getValue);

  if(role.isManager()) {

    uiOrder.setHidden(false);
    uiOrder.setReadOnly(false);
  } else if(role.isVerkoper()) {

    uiOrder.setHidden(false);
    uiOrder.setReadOnly(true);
  } else {

    uiOrder.setHidden(true);
    uiOrder.setReadOnly(true);
  }
  ...
  UI ui = new UI();
 ui.addelement(uiOrder);

  ...
  return ui;
}
```

CODE SNAPSHOT 7. Business control logica voor order display

Referenties

1 Internetgebruik in Nederland - Aantal Europese surfers stijgt gestaag, http://www.be-wired.com/info/gebruik2.html
2 The CGI Resource Index – Programs & Scripts: Perl - Form Processing, http://cgi.resourceindex.com/Programs_and_Scripts/Perl/Form_Processing/
3 The CGI Resource Index – Programs & Scripts: PHP - Form Processing, http://php.resourceindex.com/Complete_Scripts/Form_Processing/
4 Apache.org – Jakarta - The Struts Web Application Framework, http://jakarta.apache.org/struts/index.html
5 The Java Web Services Tutorial - Custom Tags in JSP Pages, Stephanie Bodoff, http://java.sun.com/webservices/docs/ea1/tutorial/doc/JSPTags.html

enkel in termen van deze User Interface classes wordt gecommuniceerd hetgeen de scheiding van control en view logica compleet maakt.

Marco Borst is senior Java consultant en actief lid van de Java Community of Practice binnen Cap Gemini Ernst & Young.