SOA vereist nieuw type applicatie

Oracle BPEL PM en JEE in de praktijk

Veel (web-)applicaties binnen bedrijven zijn vooral gericht op het tonen en aanpassen van bedrijfsgegevens uit de database. Deze applicaties bestaan voornamelijk uit een navigatiemenu met schermen voor CRUD[1]-operaties. De formulieren voor de CRUD-operaties bestaan meestal uit een enkel scherm of zijn uiteengesplitst in wizards. In de toekomst zal er ook behoefte ontstaan aan een ander soort applicatie.

Met de opkomst van *SOA*-omgevingen, *BPEL* en de *Oracle BPEL Process Manager* (PM) gaan steeds meer bedrijven procesmatiger werken, waarbij de processen tevens een human workflow kunnen creëren voor interactie met de werknemers. Hierdoor wordt ook een nieuw type applicatie geïntroduceerd. Bedrijven die met een SOA omgeving gebaseerd op de Oracle BPEL Process Manager (PM) technologie procesmatiger willen gaan werken, hebben naast de traditionele administratieve applicatie tevens de behoefte aan een *werklijst-georiënteerde applicatie* waarin de gebruiker werkt vanuit een takenlijst en taken kan selecteren en uitvoeren die het proces voor de gebruiker heeft uitgezet.

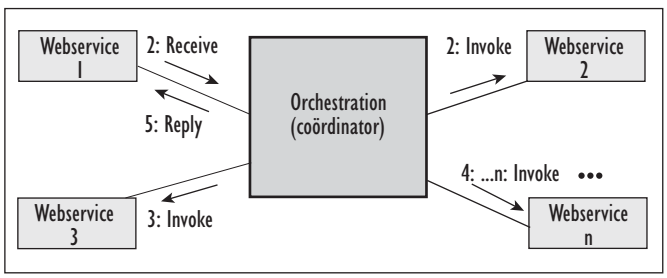
Een goed voorbeeld van een bedrijfsomgeving waarin een werklijst-georiënteerde applicatie gebruikt kan worden in combinatie met een Oracle BPEL proces is een laboratorium omgeving waarin monsters onderzocht worden op verschillende eigenschappen. Vanaf het moment dat een monster binnenkomt in het laboratorium gaat er een BPEL proces draaien voor het monster. Het BPEL proces zet vervolgens voor ieder deelonderzoek naar een specifieke eigenschap van het monster een gebruikerstaak uit voor een laboratorium medewerker. Een laboratorium medewerker werkt dus altijd vanuit een werklijst waarin de medewerker de onderzoekstaken ziet die hij of zij moet uitvoeren voor een specifiek monster.

Het doel van dit artikel is om u als lezer kennis te laten maken met de structuur van een werklijst-georiënteerde applicatie en hoe een dergelijke applicatie eenvoudig opgezet kan worden met het Struts- en het Tiles-framework. Allereerst laat ik u kort kennismaken met de globale ideeën van een SOA architectuur en de rol van de Oracle BPEL PM hierin. Daarna worden enkele concepten en technieken belicht van de Oracle BPEL PM[2] die nodig zijn om een werklijst-georiënteerde applicatie te implementeren. Tot slot wordt ingegaan op de structuur en de implementatie van een werklijst-georiënteerde applicatie.

SOA, BPEL en Oracle BPEL PM

Service Oriented Architecture (SOA) is op dit moment erg in opkomst als nieuw architectuurparadigma. Veel bedrijven bekijken de mogelijkheden om hun huidige IT architectuur om te vormen naar een SOA. In een SOA-omgeving stellen de verschillende systemen hun diensten beschikbaar aan andere systemen via *services*. Elke service binnen een SOA-omgeving maakt zijn gebruikerseisen kenbaar en doet geen enkele aanname ten opzichte van andere services binnen de SOA-omgeving (*loosely coupled*). Daarnaast werken de services binnen een SOA-omgeving samen om een specifieke taak te kunnen voltooien. Deze samenwerking wordt mede mogelijk gemaakt door een vastgelegd communicatieprotocol (bijvoorbeeld SOAP). De meeste SOA's gebruiken de webservicetechnologie om de services te implementeren, maar in principe kan iedere servicegebaseerde technologie gebruikt worden (bijvoorbeeld CORBA of DCOM).

De *Business Process Execution Language* (BPEL) wordt gebruikt om het concept van services uit te breiden met methoden om



Figuur 1. Orchestration-compositie van webservices.

de services met elkaar te laten verweven tot business services, die op hun beurt weer geïntegreerd kunnen worden tot workflows en business-processen. Kenmerk is dat de services geen kennis hebben van het feit dat ze gebruikt worden in een BPEL-proces. Dit heet ook wel *service orchestration* waarbij alleen BPEL weet heeft van het proces en de rol die de coördinator hierin speelt.

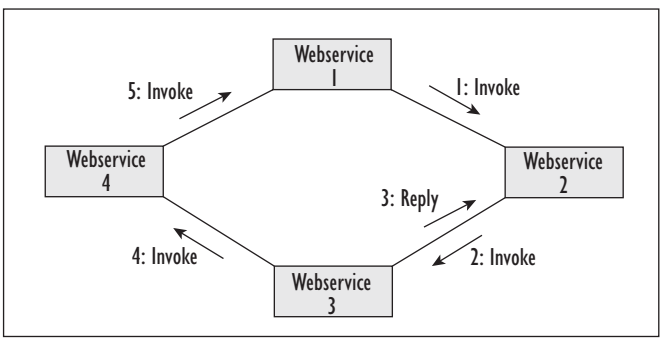
Het tegenovergestelde van het *service orchestration* paradigma is het minder flexibele *service choreography* paradigma. In een *service choreography* weten de webservices precies met welke andere webservices ze moeten communiceren. Het uitwisselen van berichten gebeurt in een *service choreography* dan ook direct en zonder tussenkomst van een coördinator.

BPEL is een *orchestration-language* ontstaan uit XLANG en WSFL en is een OASIS-standaard[3]. BPEL is een XML-gebaseerde taal en bevat vele (structurele) concepten die een gewone programmeertaal ook bevat, zoals *if-then-else*, *loop*, *sequence* (om acties in volgorde uit te voeren), *flow* (om acties in parallel uit te voeren) en lokale en globale variabelen. Er bestaan op dit moment verscheidene BPEL engines[4] om BPEL-processen te creëren, te deployen en te onderhouden.

Dit artikel gaat uit van de Oracle BPEL PM versie 10.1.2. De Oracle BPEL PM is een uitgebreid platform om BPEL processen op te deployen en te beheren. Daarnaast biedt Oracle plug-ins voor Eclipse en JDeveloper om BPEL processen te ontwikkelen. Met deze plug-ins kan de ontwikkelaar met behulp van wizards en grafische editors BPEL-processen ontwikkelen en deployen naar de Oracle BPEL PM.

Oracle BPEL Workflow en de BPEL Worklist API

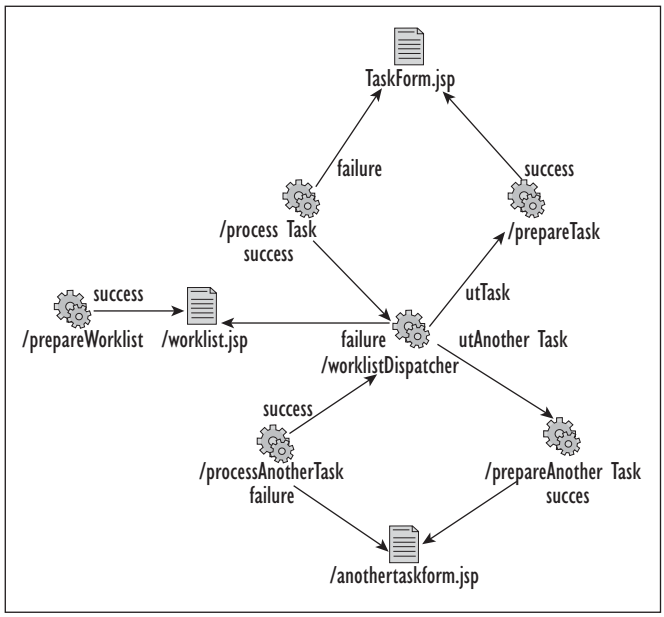
Binnen een Oracle BPEL-proces kunnen taken toegekend worden aan bepaalde gebruikers of groepen. Wanneer het BPEL-proces een taak toekent wacht het tot de taak is voltooid om weer door te gaan. Uitgezette taken verschijnen in een werklijst van de werklijstapplicatie, die standaard wordt meegeleverd met de BPEL PM-installatie. Wanneer de gebruiker inlogt op deze werklijstapplicatie worden de taken die zijn uitgezet voor de gebruiker getoond en kan de gebruiker vervolgens acties ondernemen om de taak te voltooien. Het nadeel van de meegeleverde werklijstapplicatie is dat het eigenlijk alleen bedoeld is om tijdens de ontwikkelfase van een BPEL-proces snel gebruikerstaken mee te kunnen testen. Oracle levert met de BPEL PM een *BPEL Worklist API* mee, waarmee een eigen werklijstapplicatie gebouwd kan worden in Java. Voordat ik de functionaliteit van deze API nader toelicht, wil ik graag eerst de werking van een human workflow in BPEL processen nader toelichten zodat er een beter zicht op werking ervan wordt geschapen.



Figuur 2. Choreography-compositie van webservices

Binnen een BPEL-proces kan het voorkomen dat het proces input van een gebruiker nodig heeft om een vervolgstap in het proces te kiezen. Met het uitzetten van taken wordt een human workflow gecreëerd, waarbij interactie met de gebruiker is vereist. In Oracle BPEL-jargon heet deze interactie met gebruikers *Workflow services*. Een goed voorbeeld van een proces waarbij BPEL een human workflow creëert, is het laboratorium voorbeeld omschreven in de inleiding. Figuur 3 illustreert de werking en de gebruikte componenten van het worklflowmechanisme in Oracle BPEL.

Uit figuur 3 valt af te leiden dat een gebruikerstaak in Oracle BPEL veel functionaliteit omvat. Zo kan de authenticatie/autorisatie van de gebruikers aan wie de taak wordt toegewezen geregeld worden via verschillende raamwerken. Daarnaast kan een taak worden geëscaleerd of aan een andere gebruiker worden toegewezen. Ook is er de mogelijkheid om mensen te mailen of sms-en wanneer dit noodzakelijk is. In dit artikel focus ik mij vooral op de *Worklist Service* en de *TaskActionHandler.*
Om een gebruikerstaak in Oracle BPEL uit te zetten, wordt een ander standaard BPEL-proces aangeroepen vanuit het BPEL-proces, namelijk de TaskActionHandler. Het aanroepen van de TaskActionHandler gebeurd (zoals bij iedere aanroep van een BPEL-proces) door een webservice-aanroep. Het XML-document bestaat uit een *TaskMessage* gedeelte en een *Payload* element. In het *TaskMessage* gedeelte worden alle eigenschappen van de taak opgeslagen, zoals onder andere de titel van de taak, aan wie het is toegewezen, de naam van het proces, datum waarop de taak is toegekend en taakidentificatiecode. De elementen in de TaskMessage zijn voor iedere taak hetzelfde en er kunnen dus geen taak specifieke elementen aan toe worden gevoegd. Om aan het XML-bericht taakspecifieke parameters toe te voegen, moet het Payload element gebruikt worden. Dit element kan een eigen gedefinieerde XML-structuur als inhoud verwachten. Deze XML-structuur bevat vaak de elementen voor de invoerparameters van de taak, maar kan daarnaast ook elementen bevatten voor specifieke taakinformatie, zoals een instructietekst.



Figuur 3. Communicatie over en weer tussen de verschillende componenten van de Workflow mechanisme

Waar de TaskActionHandler wordt gebruikt voor directe communicatie met het BPEL-proces, wordt de Worklist Service gebruikt voor communicatie met een gebruiker via een webapplicatie. De BPEL Worklist API verschijnt in twee implementaties:

- Local Worklist API
- Remote BPEL Worklist API

Verschil tussen beide implementaties is dat Local Worklist API een lokale variant is en dus alleen gebruikt kan worden in een webapplicatie die in dezelfde webcontainer draait als de BPEL PM, terwijl Remote Worklist API gebruikt wordt om met de Worklist Service te communiceren vanuit een andere webcontainer dan degene waarin de BPEL PM draait. De Worklist API is geïmplementeerd met EJB 2.1 session beans en de Remote Worklist API communiceert dus over RMI-IIOP met de Worklist Service.

De functionaliteit die de Remote BPEL Worklist API aanbiedt is een subset van de functionaliteit die de lokale variant aanbiedt. De Remote API heeft als grote beperking dat het alleen tekstgebaseerde payload structuren aan kan ontvangen. Daarnaast kan er geen tekstgebaseerde payload gewijzigd of gecreëerd worden[5]. Ik heb dit in de praktijk als een groot nadeel ervaren. Deze beperking stelt de ontwikkelaar bijvoorbeeld niet in staat om een XML-structuur als payload te kiezen, waarin de invoerwaarden van een taakformulier ingevuld worden voordat de taak weer terug wordt gegeven aan de Worklist Service. Dit was voor mij dan ook de reden om over te stappen naar de lokale variant. Tevens adviseert Oracle ook om de lokale variant te gebruiken in plaats van de remote variant, tenzij vereist is dat de applicatie in een andere container draait dan de BPEL PM. Ik zal de uitleg van de functionaliteit van de Worklist API dan ook baseren op de lokale variant.

Om van de Worklist Service gebruik te kunnen maken moet er eerst een geautoriseerde *Worklist Context* instantie aangemaakt worden. Voor de lokale variant wordt een Worklist Context op de volgende manier gecreëerd:

```
try{
String user = "Tom";
String password = "Hofte";
WorklistService client = WorklistService.getInstance();
WorklistContext ctx    = client.authenticateUser(user, password);
}
catch(Exception e){
          e.printStacktrace ();
}
```

Het is ook mogelijk om de Worklist Context direct te koppelen aan een geautoriseerd *HttpServletRequest* object. De `createContext(HttpServletRequest request)` van de Worklist Service klasse wordt gebruikt om een Worklist Context instantie te creëren op basis van een geautoriseerd *HttpServletRequest* object. De `createContext` gaat er vanuit dat het *RemoteUser* object is aanwezig is in het request object. Met de `createContext` methode kan de autorisatie gekoppeld worden aan bijvoorbeeld Oracle OID of JAZN.
Nu er een Worklist Context is aangemaakt kunnen de taken voor de gebruiker opgehaald worden. De Worklist API biedt uitgebreide functionaliteit om de taken te gefilterd en gesorteerd op te halen met de Worklist Service. De API biedt de gebruiker naast de standaardfilter en sorteermogelijkheden ook de mogelijkheid om eigen sorteer- en filteropties te specificeren. Deze speciale mogelijkheden van de API liggen buiten de scope van dit artikel. In de praktijk zal er echter wel degelijk gebruik gemaakt worden van deze mogelijkheid, omdat de standaard sorteer- en filter opties niet alle wensen van de eindgebruiker kunnen vervullen. In het onderstaande code-fragment worden alle taken van de groep opgehaald, waarin de gebruiker zit. Tevens worden alle opgehaalde taken gesorteerd op titel.

```
Map filterMap = new HashMap();

//Haal alleen de taken op van de groep waarin de gebruiker zit.
filterMap.put(IWorklistService.FILTER_TYPE_TASK_FILTER,
                IWorklistService.TASK_FILTER_MY_AND_GROUP);
               )

List tasks = null;
tasks      = client.getWorklistTasks(ctx, filterMap,
                    IWorklistService.SORT_FIELD_TASK_TITLE,
                    IWorklistService.SORT_ORDER_ASCENDING);
```

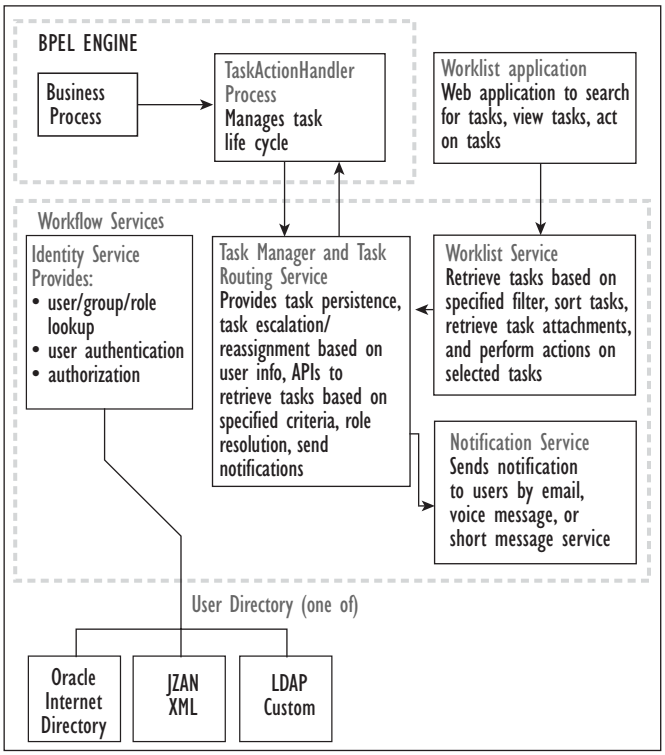
Nu alle gewenste taken zijn opgehaald kan een specifieke taak geselecteerd worden om vervolgens de inhoud van de taak te tonen op een detailscherm met eventueel een formulier voor de vereiste invoervelden voor de elementen van de payload. De elementen van de Task Message van een BPEL-taak worden door de API meteen uit te lezen via het `WorklistTask` object dat de BPEL-taak representeert. De payload bestaat uit een XML-document die ook als een XML-document in de `WorklistTask` staat. Om de gebruiker niet op te zadelen met ingewikkelde XML ontleed methoden om toch vanuit Java de elementen van de Payload te kunnen benaderen, heeft Oracle *schemac* meegeleverd in de BPEL PM distributie. Schemac is een tool om Java façaden-klassen voor XML-elementen te genereren, waarmee XML-elementen op een OO-manier gemanipuleerd kunnen worden. Schemac verwacht als input een XML schemadefinitie-file die de structuur van de payload definieert en het genereert daarvoor vervolgens de façades, maar ook de factory-klassen om de specifieke façades instanties te kunnen maken. In het volgende code-fragment wordt de payload van een taak omgezet naar façade klasse. De payload wordt gewijzigd door middel van de façade-klasse en vervolgens wordt de façade-klasse weer als een XML-document teruggezet in het taakobject. Als laatste wordt op de taak een wijziging doorgevoerd en wordt aan de Worklist Service door gegeven dat de taak is volbracht.

```
//Haal de payload uit de taak
 Element payload = (Element) taak.getPayload();

/*Voorbeeld van een payload facade
SomeTaskPayload ut = SomeTaskPayload Factory.createFacade(payload);

ut.setSomeProperty("SomeValue");

//Zet de gewijzigde payload als XML terug in de taak
Element root = ut.getRootElement();
taak.setPayload(ut.getRootElement());

//Zet de uitvoerder
taak.setUpdatedBy((String) httpSession.getAttribute("gebruikerX"));

//Action type
String actionType = "DONE";

//Verander de status van het  veld
client.updateTask(ctx, taak);

//Laat het process een tijdje slapen voor correcte afhandelen van de
taak
Thread.sleep(2000);
client.customTaskOperation(ctx, taak.getTaskId(), actionType);
```

Wat opvalt in het bovenstaande code-fragment is dat er een expliciete sleep van 2000 milliseconden staat tussen de methode `updateTask` en `customTaskOperation`. Uit ervaring blijkt dat dit noodzakelijk is om onverwacht gedrag te voorkomen. In de praktijk is gebleken dat het weglaten van de wachtroutine kan resulteren in een incorrecte afhandeling van de taak. Dit komt door het feit dat er twee wijzigacties op de BPEL-taak kort na elkaar worden uitgevoerd. De Worklist Service heeft per wijzigactie wat tijd nodig om het te verwerken. Oracle zelf adviseert zelfs een wachttijd van 5000 milliseconden, maar ik kom er in de praktijk tot nu toe altijd met een wachttijd van 2000 milliseconden vanaf. In deze sectie is de werking van de Workflow Service en de Worklist API van Oracle BPEL nader toegelicht. Het dient te zeggen dat lang niet alle functionaliteit is belicht, meer informatie is te vinden in de al eerder genoemde referenties van Oracle BPEL. Nu er enig gevoel voor de werking van de Workflow Service en de Worklist API is gecreëerd, is het tijd om in de volgende twee secties in te gaan op de structuur van de werklijstapplicatie en hoe deze flexibel en efficiënt met Struts en Tiles geïmplementeerd kan worden.

Structuur van de applicatie

De structuur van een eigen werklijstapplicatie is vrij eenvoudig. De werklijst is het centrale gedeelte van de applicatie en daaromheen 'zweven' de taak schermen. Aan de hand van de geselecteerde taak in de werklijst moet het juiste scherm getoond worden en na het voltooien (of annuleren) van een taak komt de gebruiker weer terug de takenlijst. Een individueel taak-



Figuur 4. Struts diagram voor een simpele worklist applicatie met twee taken.

scherm zweeft dus daadwerkelijk in de applicatie en valt niet binnen een vaste flow van schermen, zoals in een wizard. Het is het BPEL proces die bepaald wat de volgende taak voor de gebruiker is en dus wat het volgende scherm is dat de gebruiker te zien krijgt.
In deze sectie laat ik zien hoe een dergelijke structuur wordt geïmplementeerd in Struts[6]. Struts is een populair en volwassen MVC-framework van de Apache Software Foundation, waarmee webapplicaties opgezet kunnen worden. Het *Struts Action Framework*[7] vormt het controller gedeelte van de applicatie. Bij ieder request op een pagina wordt door de controller een Action klasse aangeroepen die de juiste data (bijvoorbeeld alle taken voor een persoon) ophaalt uit het model en in het juiste formaat klaarzet om op de pagina getoond te worden. Een Action klasse wordt niet alleen gebruikt om data op te halen uit het model en te prepareren voor gebruik. Het kan ook worden gebruikt om data, bijvoorbeeld ingevuld in een formulier op een pagina, te verwerken en op te slaan in het model. Voor een simpele werklijst applicatie met twee taken ziet de Struts-flow er nu uit als in figuur 4.

De applicatie start met de *prepareWorklist* Action klasse die alle taken voor een gebruiker ophaalt en eventueel gefilterd en gesorteerd klaar zet in het juiste formaat om op de pagina getoond te worden. De *worklistDispatcher* zorgt ervoor dat na een taakkeuze op het scherm het bijbehorende scherm van de taak ook daadwerkelijk getoond wordt. Iedere taak bestaat uit een *prepare*- en *process* Action klasse en een JSP-pagina. In de *prepare* Action worden de data uit de payload van de taak gehaald en klaargezet voor gebruik op de pagina. Wanneer de gebruiker het taakformulier op de pagina invult, wordt de *process* Action uitgevoerd en worden de ingevoerde data gevalideerd en in de juiste velden van de payload gezet. Als laatste wordt de Worklist Service geattendeerd op het feit dat de taak is volbracht.

De opzet in Struts lijkt op het eerste gezicht vrij eenvoudig. De enige uitdaging zit in het dynamisch bepalen van het detailscherm dat getoond moet worden na een taakselectie in de werklijst. In het geval van twee taken is dit nog eenvoudig op te lossen. De worklistDispatcher kijkt dan gewoon naar de titel van de taak en roept aan de hand van de titel de juiste vervolgactie aan. Deze oplossing heeft als nadeel dat de worklistDispatcher weet moet hebben van alle taken die het systeem bevat. Dit schept een extra afhankelijkheid die problematisch kan worden wanneer de applicatie veel meer dan twee taken bevat. Om ervoor te zorgen dat de worklistDispatcher geen weet heeft van welke taken er allemaal bestaan maar er toch voor zorgt dat de juiste stap wordt gekozen, wordt een veld uit de TaskMessage gebruikt om de vervolgactie te bepalen. Met andere woorden: het BPEL-process bepaalt aan de hand van de uitgezette taken de flow van schermen binnen de Struts-applicatie. Het *processName* veld van de TaskMessage is hier prima geschikt voor en zal in het onderstaande voorbeeld ook gebruikt worden. In figuur 4 heeft de worklistDispatcher twee vervolgacties naar de twee taken, namelijk *utTask* en *utAnotherTask*. In de struts-config.xml file zijn deze stappen als volgt gedefinieerd:

```
<action path="/worklistDispatcher" type="test.WorklistDispatcherAction">
      <forward name="failure" path="/worklist.jsp"/>
      <forward name="utTask" path="/prepareTask.do"/>
      <forward name="utAnotherTask" path="/prepareAnotherTask.do"/>
</action>
```

De taken Task en AnotherTask hebben als processName resp. utTask en utAnotherTask. Deze corresponderen nu met de gedefinieerde vervolgstappen van de worklistDispatcher actie. Op deze manier is de werking van de worklistDispatcher onafhankelijk van de taken die het systeem kan hebben. Het toevoegen van een nieuwe taak is nu ook eenvoudig te realiseren8. Het toevoegen van een vervolgactie voor de worklistDispatcher aan de strut-config.xml is voldoende om ervoor te zorgen dat de nieuwe taak wordt herkend door de worklistDispatcher. Nu de structuur van de werklijstapplicatie is uitgelegd en hoe BPEL taak de flow binnen de Struts-applicatie bepaalt wordt in de volgende sectie uitgelegd hoe je eenvoudig de schermen voor de taken kunt implementeren door gebruik te maken van het Tiles Framework.

Generieke taakschermen

Wat je in de praktijk vaak tegen komt bij het maken van taakschermen voor de taken van een BPEL-process is dat de schermen dezelfde lay-out hebben. In het laboratoriumvoorbeeld uit de inleiding zullen de vaste deelgebieden van een taakscherm bijvoorbeeld bestaan uit een titel, monsterinformatie, instructie en het invulformuliergedeelte. De deelgebieden voor de titel, monsterinformatie en de instructie zijn statisch van structuur, vanwege het feit dat deze deelgebieden voor iedere taak dezelfde velden zullen bevatten. Dit gaat echter niet op voor het invulformulier, waardoor dit een dynamisch deelgebied is. Dit betekent dat er voor de titel, monsterinformatie en het instructie gedeelte één keer een pagina gemaakt wordt die steeds wordt hergebruikt met verschillende inhoud voor de velden, afhankelijk van de taak. Voor iedere aparte taak is dan alleen nog een pagina nodig die het invulformulier voor de taak bevat.
Met template-mechanisme van het Struts Tiles Framework is het bovenstaande eenvoudig en flexibel te realiseren. Het Tilesframework biedt ondersteuning om de presentatielaag van een website op te delen in deelgebieden (tiles) en deze run-time

pas van inhoud te voorzien. Op deze manier wordt een lay-out herbruikbaar en kunnen jsp-pagina's voor de deelgebieden ook hergebruikt worden. Daarnaast wordt de lay-out gescheiden van de inhoud en wordt de lay-out van de gehele applicatie op een centrale plek geregeld, waarmee de flexibiliteit wordt bevorderd. Dit template- mechanisme maakt het gebruik van de `<jsp:include>` overbodig. De ontwikkelaar maakt een template-pagina aan die de lay-out en de verschillende deelgebieden beschrijft. Vervolgens registreert de ontwikkelaar de template bij het Tiles Framework en definieert alvast de inhoud van de statische deelgebieden binnen de template. Dit alles gebeurt in een XML-configuratiefile. Hierna refereert de ontwikkelaar in een nieuwe jsp-pagina aan de template en geeft alleen nog aan hoe de dynamische deelgebieden ingevuld dienen te worden.
Hoe werkt dit nu voor het laboratorium voorbeeld met statische deelgebieden voor de titel, monsterinformatie en de instructie en een dynamisch deelgebied voor het invulformulier? We gaan gemakshalve uit van een simpele lay-out, waarbij de deelgebieden onder elkaar worden uitgelijnd, maar het is voor te stellen dat er gemakkelijk met de HTML `<div>` tag een meer flexibele en meer complexe lay-out is te definiëren. In het volgende code-fragment is met behulp van de `<tiles:insert>` tag uit Tiles tag-library de template voor het laboratoriumvoorbeeld gedefinieerd.

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html>
 <head>
  <link href="css/screencss.css" rel="stylesheet" media="screen">
</script>
 </head>
<body>
<div class="titel">
<tiles:insert attribute="titel"/>
</div>
<div class="informatie">
<tiles:insert attribute="informatie"/>
</div>
<div class="instructie">
<tiles:insert attribute="instructie"/>
</div>
<div class="formulier"
<tiles:insert attribute="formulier"/>
</div>
</body>
</html>
```

Bovenstaande template kan nu in ieder taakscherm worden hergebruikt, zodat alle schermen van dezelfde lay-out zijn voorzien. Het bovenstaande template moet nu geregistreerd worden. Dit gebeurt declaratief in een Tiles XML-configuratie file die meestal `tiles-config.xml` is genaamd. Hieronder staat de Tiles-configuratiefile voor ons voorbeeld.

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<!DOCTYPE tiles-definitions PUBLIC
  "-//Apache Software Foundation//DTD Tiles Configuration//EN"
  "http://jakarta.apache.org/struts/dtds/tiles-config.dtd">

<tiles-definitions>
 <definition name="index.default" path="/templates/taakscherm_template.
jsp">
      <put name="titel"      value="/include/titel.jsp" />
      <put name="informatie" value="/include/informatie.jsp" />
      <put name="navigatie"  value="/include/navigatie.jsp" />
</definition>
</tiles-definitions>
```

In de bovenstaande XML-configuratiefile zijn ook al alle statische deelgebieden ingevuld. Voor een nieuwe taak moet nu alleen nog een pagina gedefinieerd worden met daarin het invulformulier en een pagina die aangeeft dat er gebruikt wordt gemaakt van de zojuist gedefinieerde template en wat de inhoud van het dynamische gedeelte moet zijn. Deze laatste pagina ziet er dan als volgt uit:

```
<%@ page contentType="text/html;charset=windows-1252"%>
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>

<!--
De statische tiles voor de titel informatie en instructies worden auto-
matisch    geladen m.b.v. XML koppeling
-->

  <tiles:insert definition="index.default">
      <tiles:put name="formulier" value="/include/taken/eersteTaak.
jsp"/>
  </tiles:insert>
```

Conclusie

Met de komst van het SOA-architectuurparadigma en de Oracle BPEL PM als platform om een SOA te implementeren, is er ook behoefte aan een nieuw soort webapplicatie: een werklijst-georiënteerde applicatie waarmee de taken binnen een door het BPEL-proces gecreëerde human workflow afgehandeld kunnen worden. De gebruiker van een werklijst-georienteerde applicatie werkt altijd vanuit de takenlijst en het BPEL-proces bepaald de flow van de schermen aan de hand van de volgorde van de taken die worden uitgezet voor een specifieke gebruiker. De schermen voor de taken zweven dus in de applicatie en verschijnen pas wanneer de bijhorende taak verschijnt in de takenlijst en de gebruiker er op klikt.
Met de BPEL Worklist API levert Oracle een prima API om zelf een werklijstgeoriënteerde applicatie te implementeren. In dit artikel is aangetoond dat Struts en Tiles de flexibiliteit van dit soort applicaties erg kan bevorderen. Helaas raadt Oracle op

het moment van schrijven het gebruik van de Remote Worklist API nog af, vanwege onvoldoende functionaliteit ten opzichte van de lokale variant. Hierdoor moet een werklijst-applicatie altijd getest worden in de container waar de BPEL PM ook in draait. Dit kan in de praktijk tijdens het testen nog wel eens voor problemen zorgen, omdat iedere keer gedeployed moet worden naar een container op een server waarin de BPEL-ontwikkelomgeving draait[9]. Tijdens de productie zal de performance van de lokale variant echter wel beter uitvallen dan de remote-variant, aangezien de laatste over het RMI-IIOP protocol werkt wat voor de nodige extra overhead zorgt.

De inhoud van dit artikel is gebaseerd op de huidige versie van de Oracle BPEL PM 10.1.2.0.2. Op het moment van schrijven heeft Oracle net versie 10.1.3 van Oracle BPEL PM[10] als final release uitgebracht. In de 10.1.3. versie is het Human Workflow gedeelte uitgebreid en verbeterd. Hiermee is ook de Worklist API aangepast en hebben heeft de 10.1.2 api de status *deprecated* gekregen. Daarnaast zal JSP in de toekomst worden ingeruild voor JSF, maar de concepten in dit artikel zullen daarmee niet veranderen.

Tom Höfte is werkzaam bij IT-eye.

Voetnoten

1 *Create Read Update Delete*
2 *Voor meer informatie over de Oracle BPEL PM: http://www.oracle.com/technology/products/ias/bpel/index.html*
3 *OASIS WSBPEL TC homepage [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel]*
4 *Lijst van BPEL engines [http://en.wikipedia.org/wiki/List_of_BPEL_engines]*
5 *Zie voor meer info Oracle BPM developer guide: http://www.huihoo.com/oracle/docs/B14099_19/integrate.1012/b14448/worklist.htm*
6 *In dit artikel wordt uitgegaan van het Struts framework, maar het eenvoudig in te denken dat hetzelfde resultaat bereikt kan worden door een nieuw framework, zoals JSF, te gebruiken.*
7 *http://struts.apache.org/struts-action/index.html*
8 *Naast het implementeren van de prepare- en process Action klassen en de JSP pagina voor de taak*
9 *In de praktijk zal er binnen een SOA project vaak een apart BPEL ontwikkelteam en GUI ontwikkelteam zijn, waardoor het GUI ontwikkelteam gebruik dient te maken van de BPEL processen op een centrale ontwikkelomgeving.*
10 *De 10.1.3 versie van Oracle BPEL PM is onderdeel van de Oracle SOA suite.*