New & Cool

In dit artikel geeft Jan Vissers inzicht in de manier waarop EJB3/JPA (Java Persistence API) kan worden gebruikt ter ondersteuning van een op Adobe Flex gebaseerde Rich Internet Application (RIA). Met behulp van faciliteiten die geboden worden door Adobe LiveCycle Data Services (LCDS), kunnen sexy user-interfaces relatief eenvoudig aangesloten worden op Java/JEE backends. De auteur zal specifiek ingaan op het gebruik van Data Management (DM) – een onderdeel van LCDS. DM leent zich vooral voor het presenteren van en werken met data-intensieve onderdelen van een uiteindelijke oplossing.

Data-rich internet applications

Adobe Flex, LCDS en EJB3/JPA

Eén van de eerste dingen waar ik aan denk bij Adobe Flex is: strakke user-interfaces, voorzien van de nodige toeters-en-bellen die het bekijken van en werken met een dergelijke interface leuk maken. Waar Flash-objecten – de uiteindelijke runtime representatie van wat je met Adobe Flex maakt – in het begin vooral gebruikt werden om websites op te leuken, wordt deze technologie in toenemende mate ook ingezet voor het realiseren van internetapplicaties. Het is niet de bedoeling om in dit artikel alle voor- en nadelen van Flex ten opzichte van andere technieken te bespreken. Een groot deel van de vaak gehoorde nadelen van Flex berusten overigens op vooroordelen en foute/gedateerde informatie. Wel is het, zeker voor mensen die onvoldoende op de hoogte zijn van Adobe Flex, goed om deze technologie te plaatsen in de context van de *standaard*technieken om webapplicaties te bouwen.

Van statische pagina's naar RIA's

Op het gebied van (web) user-interfaces is het momenteel dringen geblazen. Voor wat betreft headless enterprise Java/JEE is de afgelopen jaren veel vooruitgang geboekt en wordt een solide platform geboden. Een sterke toevoeging aan het platform is de EJB3/JPA standaard. Dat ligt anders voor die onderdelen die zich richten op user-interfaces. Hoewel er veel initiatieven – zowel vanuit standaardisatie als de open source community – zijn op dit vlak, is er nog veel onduidelijkheid over waar het nu naar toe gaat. Dit is vooral aan de orde voor applicaties die ons op basis van internettechnologie bereiken, veelal via een browser. Het overgrote deel van de technieken genereert dynamisch HTML op de server, die uiteindelijk naar de browser wordt gestuurd. Idealiter wordt hierbij voor de diverse stijlelementen van een pagina gebruik gemaakt van Cascading StyleSheets (CSS) die door de browser geïnterpreteerd worden. Vanuit het perspectief van een applicatie wordt het pas echt interessant als ook daadwerkelijk interactiviteit met de eindgebruiker vereist is. In de meest eenvoudige vorm wordt deze interactiviteit gefaciliteerd door het HTML <FORM> element. Lange tijd was dit eigenlijk het enige wat we nodig dachten te hebben. Maar hoe langer hoe meer eindgebruikers kregen te maken met internet applicaties en vaak vroegen die zich toch af waarom bepaalde dingen die in hun oude client-server applicatie zo gewoon waren niet meer konden via het web. Met JavaScript kon een deel van de tekortkomingen opgelost worden. Met de invoering van AJAX technologie kan nog beter worden ingespeeld op de groter wordende vraag naar interactieve, responsieve user-interfaces.

Adobe Flex staat een compleet andere aanpak voor, één die niet meer gebonden is aan HTML. Flex maakt gebruik van de Flash Player, een plugin die vaak al aanwezig is in de browser van eindgebruikers. Deze player speelt (duh!) Flash objecten af. Hiermee wordt een clientside platform geboden waarop applicaties gerealiseerd worden die een sterke gelijkenis kunnen vertonen met de client-server applicaties van weleer. Adobe Flex applicaties worden gemaakt met behulp van MXML en ActionScript, om vervolgens (meestal

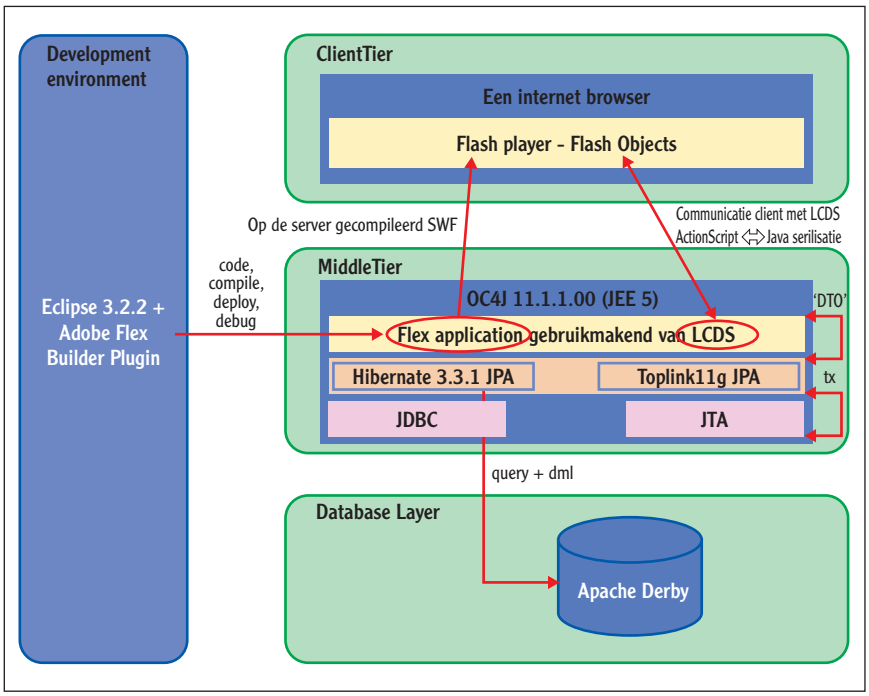
Jan Vissers
is Technology Manager bij Cumquat Information Technology. Reacties op dit artikel kunnen gemaild worden naar jan@cumquat.nl.
De broncode van de voorbeeldapplicatie kan gedownload worden van: http://www.cumquat.nl/technology.html

op de server) te worden gecompileerd naar het binary SWF formaat wat op zijn beurt naar de browser wordt gestuurd.
Moeten we dan nu altijd Adobe Flex gebruiken als we een user-interface realiseren? Zeker niet! Zoals zoveel dingen in ons vakgebied dient de uiteindelijke technologiekeuze aan te sluiten bij de geldende situatie. Wat hierbij onder meer van belang is, is de uiteindelijke aard van de toepassing. Veel of weinig *content*, aard van de *content*, veel of weinig *interactiviteit*, voor *intranet en/of extranet en/of internet, self-service* of *enterprise applicatie*, etc, etc. Indien de keuze uiteindelijk op Adobe Flex valt, is het goed om te weten dat deze technologie volwassen is en zeer goed in te passen is in een Java/JEE architectuur. Dit inpassen kan geoptimaliseerd worden door middel van Adobe LiveCycle Data Services (LCDS). Dit artikel toont daar een voorbeeld van. Alvorens op de LCDS in te gaan, is het nuttig om de 'spelers' voor te stellen.

Gebruikte producten en componenten

Het belangrijkste doel van dit artikel is om inzicht te geven in de manier waarop EJB3/JPA en LCDS gebruikt kan worden bij de realisatie van een *data*-rich internet application op basis van Adobe Flex. In het daar voorafgaande onderzoek zijn een behoorlijk aantal producten en componenten gebruikt die allen een bepaalde rol in het geheel spelen. Door het presenteren van deze onderdelen kan verderop in dit artikel wordt aan deze onderdelen gerefereerd, waarbij bovendien het speelveld van 'de oplossing' verduidelijkt wordt.

Figuur 1. Gebruikte componenten per laag.



- *OC4J11 – 11.1.1.0.0*
  Technology Preview van Oracle's Java EE 5.0 compliant applicatieserver.
- *Toplink11 – 11.1.1.0.0*
  Preview van Oracle Toplink met onder andere ondersteuning voor EJB3/JPA.
- Hibernate Entity Manager 3.3.1
  Hibernate's implementatie van de EJB3/JPA standaard[1].
- *Adobe Flex 2.0.1*
  Flex SDK en tools ter ondersteuning van het realiseren van Flex applicaties.
- *Adobe LiveCycle Data Services 2.5*
  J2EE component, ter ondersteuning van Adobe Flex, die ontsluiting van Java/JEE services en componenten vereenvoudigt en optimaliseert.
- *Apache Derby 10.2.2.0*
  Een volwaardige ANSI SQL relationele database, volledig geschreven in Java.
- *Eclipse 3.2.2 – FlexBuilder 2.0.1*
  De FlexBuilder is een Eclipse-plugin die het ontwikkelen van op Adobe Flex gebaseerde applicaties ondersteund.

Deze onderdelen zijn te rangschikken zoals getoond wordt in Figuur 1.

Waarschijnlijk is het onderdeel LiveCycle Data Services het minst bekend, maar het staat in dit artikel wel centraal. In de volgende paragraaf zal daarom uitvoeriger worden stilgestaan bij LCDS en de componenten waaruit het is opgebouwd.

LiveCycle Data Services

LCDS (voorheen Adobe Flex Data Services) bestaat uit een aantal componenten met als doel om de dynamiek van op Flex gebaseerde RIAs voor wat betreft het ontsluiten en uitwisselen van data, met het Java/JEE backend en tussen connected Flex applicaties onderling, te faciliteren. LCDS wordt gedeployed in een J2EE applicatieserver.
Zonder LCDS is het ook mogelijk om *data*-rich internet applications te maken met Flex, maar in bepaalde gevallen biedt LCDS voordelen, bijvoorbeeld op het gebied van nog geavanceerdere interactiviteit, performance en het gemak waarmee de data in een applicatie is in te bedden. LCDS bestaat uit de volgende componenten:

- RPC Services
  Deze component vertoont qua techniek overeenkomsten met Java/RMI. Het wordt mogelijk gemaakt om vanuit de Flex omgeving *remote services* aan te roepen. Dit kunnen bijvoorbeeld ook EJB componenten zijn.
- Data Management Service (DM)
  Anders dan bij RPC Services is de band die de Data Management Service heeft met het back-end en de connected clients onderling

'strakker'. DM speelt feitelijk de rol van een proxy die zich ophoudt tussen clients enerzijds en het back-end anderzijds. Vanuit die rol kan de DM geavanceerde datasynchronisatie uitvoeren, bijvoorbeeld het *pushen* van wijzigingen naar alle connected clients.

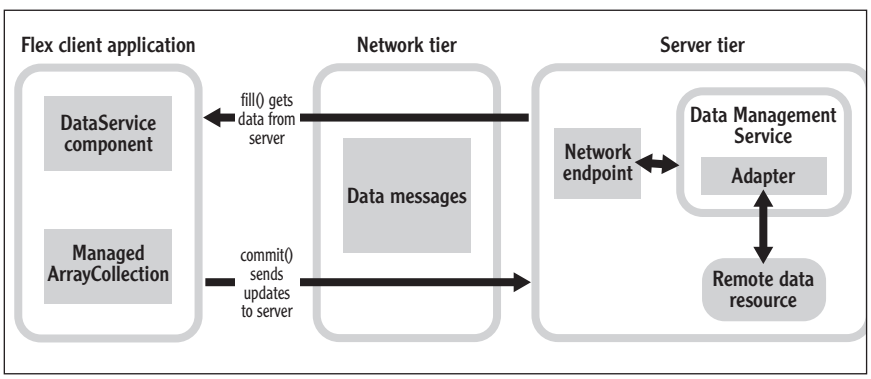
- Message Service
  Deze component maakt het mogelijk om applicatieonderdelen te realiseren, waarmee clients onderling met elkaar kunnen communiceren. Een voorbeeld hiervan is een *chat* component. Tevens kan de messaging gekoppeld worden aan Java Message Service (JMS) applicaties.

Het is belangrijk om te onthouden dat het inbedden van Java/JEE services en data in een Adobe Flex applicatie niet gebaseerd hoeft te zijn op een rigide keuze voor de ene of de andere techniek van LCDS. Een mix van technieken is heel goed mogelijk, waarbij de uiteindelijke keuze ook af dient te hangen van de te realiseren use case. Het is daarnaast zeer wel mogelijk om voor een bepaald deel helemaal geen LCDS te gebruiken, maar de standaardondersteuning van Flex voor het aanroepen van HTTP-services en/of SOAP te gebruiken.
Bij het onderzoek dat ten grondslag ligt aan dit artikel is specifiek gekeken naar het Data Management Service deel van LCDS, om te bestuderen of en zo ja hoe een EJB3/JPA laag hiermee te ontsluiten is. Verderop in dit artikel zal gedetailleerd worden ingegaan op de concrete configuratie en codeerwerkzaamheden die hierbij uitgevoerd zijn. Voor een basisbegrip van DM is het noodzakelijk om ook dit onderdeel wat van naderbij te bekijken.

LCDS Data Management Service

Feitelijk bestaat DM uit twee samenwerkende onderdelen. Aan de ene kant wordt data op de client gemanaged door een Adobe Flex *DataService* – een ActionScript component. Op de server zorgt de DM Service ervoor dat data gesynchroniseerd wordt met het back-end en de wijzigingen worden gedistribueerd naar de diverse gekoppelde clients = data push. Het server onderdeel van de DM verzorgt de uitwisseling van data met het back-end met behulp van zogenaamde *assemblers*. Deze assemblers communiceren met de daadwerkelijke data stores – bijvoorbeeld op basis van EJB3/JPA – om de data ook daadwerkelijk te persisteren en op te halen.

In Figuur 2 wordt geschetst op welke wijze de data tussen client en server stroomt. Zo wordt op het *DataService* component aan de client de *fill()* methode gebruikt om een dataset. Naast *fill()* worden nog tal van andere operaties geboden door de *DataService* component – waaronder voor het synchroniseren van data, het vastleggen en verwijderen van data. Op de server worden dergelijke *calls* op basis van *destination* configuratie gerouteerd naar de juiste *assembler*(routine). Een *destination* is hierbij een duidelijk afgebakende data-eenheid die in de Flex-applicatie gebruikt wordt. Vanuit het perspectief van de Java/JEE bestaat het ontsluiten van het back-end uit het implementeren van een Java Flex Assembler interface. Naast de basishandelingen die de DM voor ons uitvoert voor wat betreft het ophalen en vastleggen van data wordt ook functionaliteit geboden om bepaalde conflictsituaties te kunnen detecteren en oplossen.
DM zal zijn taken op basis van een bepaalde vastgelegde standaardconfiguratie uitvoeren. Zo wordt bijvoorbeeld vanuit het *DataService* component standaard gebruik gemaakt van *autoCommit*. Voor de duidelijkheid: dit is niet de *autoCommit* aan de serverzijde van JDBC, maar een ActionScript *autoCommit*. Een ander voorbeeld van de standaardconfiguratie van een *DataService* is de *auto-sync-enabled* die standaard *aan* staat. Dit houdt in dat de component direct op de hoogte wil worden gesteld als er een wijziging in de DM Service is opgetreden. Concreet houdt dit in dat een browser met Flex-applicatie geen *refresh* hoeft uit te voeren om een elders gemaakte datawijziging meteen te zien – mits die wijziging 'gezien' wordt door DM. Het zal lang niet altijd het geval zijn dat deze standaardconfiguratie voldoet.
Naast de standaardconfiguratie van de *DataService* voert ook de DM Service zijn taken volgens een bepaalde standaardconfiguratie uit. Ook hiervoor geldt dat hiervan kan worden afgeweken, door het aanpassen van de destination configuratie. Zo maakt de DM Service bijvoorbeeld standaard gebruik van JTA transacties. Indien dat niet wenselijk is, en transactionaliteit op een andere manier gecoördineerd moet worden dan kan de configuratie voor *use-transactions* op *false* gezet worden. Daarnaast kan bijvoorbeeld het *paging* gedrag – het uitserveren van subsets van data, ter bevordering van de schaalbaarheid van de applicatie – worden geconfigureerd, evenals declaratieve auto-



Figuur 2. Data flow in Data Management.

risatie over bepaalde DM Service operaties.
Het voert te ver om in dit artikel alle mogelijkheden – en het zijn er veel – op het gebied van gebruik en parameterisering van DM te bespreken. In ieder geval biedt het voldoende mogelijkheden om ook bij grote hoeveelheden data en veel gelijktijdige gebruikers op een efficiënte wijze data van client naar server, en omgekeerd te krijgen.

Voorbeeldapplicatie

Tot dusver is in dit artikel alleen nog maar op theoretisch niveau gekeken naar de aspecten die te maken hebben met LCDS en meer specifiek Data Management Service. Maar hoe werkt het nou in de praktijk? Om dat duidelijk te maken bespreken we in dit artikel een voorbeeldapplicatie. Alle details van deze voorbeeldapplicatie zijn te downloaden vanaf de website van dit blad, **www.javamagazine.nl**. Allereerst wordt het classmodel gepresenteerd dat via EJB3/JPA gepersisteerd zal worden in de Apache Derby database. Vervolgens wordt gedetailleerd beschreven welke handelingen nodig zijn om LCDS DM gereed te maken om de Java/JEE-laag te ontsluiten. Daarna wordt een blik geworpen op de ***assembler*** die feitelijk de link tussen DM en persistency laag/DAO vormt. Tenslotte wordt beschreven op welke manier vanuit de Eclipse FlexBuilder plugin de noodzakelijke MXML/ActionScript stukken gerealiseerd worden om uiteindelijk de gevraagde functionaliteit en user-interface te bieden.

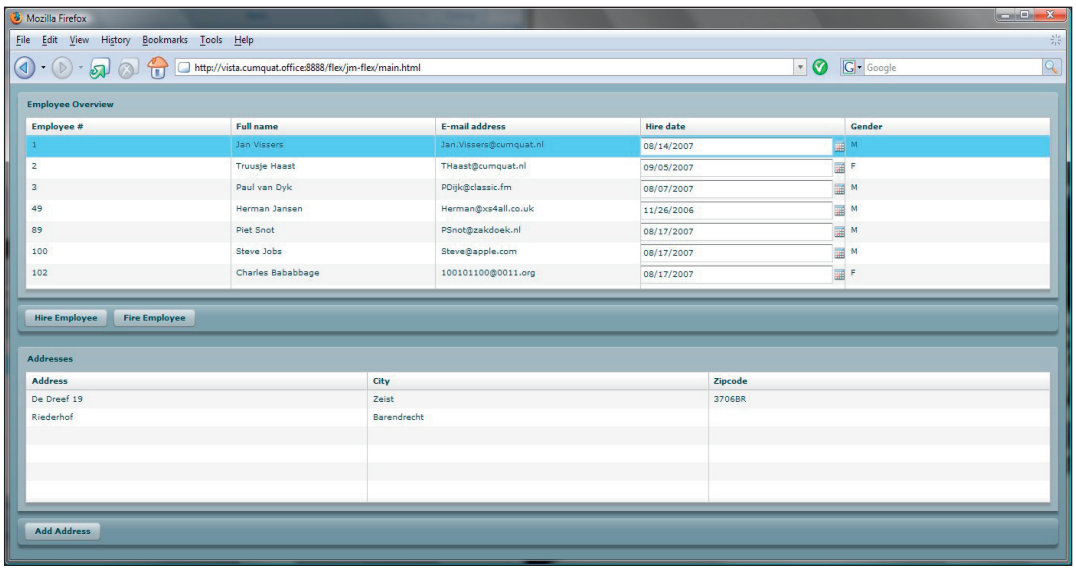
Samenvatting en conclusie

In dit artikel is beschreven op welke manier EJB3/JPA en LCDS gebruikt kunnen worden in een Adobe Flex *data*-rich internet application. Het zal duidelijk zijn dat hiermee slechts een klein deel van de functionaliteit besproken is. Zaken als *lazy loading* van associaties, het *pagineren* van resultaten of het afhandelen van fouten zijn onbesproken gebleven. Ik hoop dat ik in ieder geval een globaal beeld heb kunnen schetsen van wat deze techniek inhoudt en daarmee een bepaald referentiepunt gegeven heb.
Adobe Flex en LCDS bieden veel mogelijkheden om goed ogende user-interfaces te maken en sluiten goed aan op enterprise Java/JEE. Specifiek wat betreft Data Management en EJB3/JPA is in dit artikel aangetoond dat deze twee onderdelen kunnen samenwerken. Natuurlijk kan er nog wel het nodige geoptimaliseerd worden. Zeker ten aanzien van de hoeveelheid *plumbing* die noodzakelijk is op het vlak van *assembler* en *ActionScript* entity classes valt er nog veel te verbeteren. Mijn gedachten gaan hierbij uit naar een *generieke EJB3/JPA* assembler die op basis van (data-management) configuratie relatief autonoom zijn werk zou moeten kunnen doen. Feitelijk zou hierdoor niet meer per *destination* een specifieke *assembler* noodzakelijk zijn, maar kan de ene generieke *assembler* op basis van de configuratie zijn werk doen. Actionscript entity classes zouden daarnaast eventueel ook te genereren zijn. Mijn interesse is in ieder geval versterkt!

Referenties

1. Gedurende het onderzoekstraject ben ik overgestapt naar deze provider vanwege een blokkerende, (nog) niet-bevestigde bug in Toplink11.

Adobe Flex staat een aanpak voor die niet meer gebonden is aan HTML



Figuur 3. Killer app.