

Die Erwartungen der Business-Anwender sind mehr und mehr geprägt durch das, was sie im Web vorfinden: Anwendungen wie Google Maps sind überall verfügbar und hochgradig interaktiv. Daraus ergibt sich ein spürbarer Markt-Druck zur Migration vorhandener Anwendungen in Richtung Java und Web 2.0.

Smart Migration: schrittweise Umstellung von Oracle- auf Java-Open-Source-Technologien

Thomas Haskes und Oliver Zandner, Triestram & Partner GmbH

Klassische Business-Anwendungen können die Erwartungen bezüglich Web 2.0 in der Regel nicht erfüllen, besonders dann, wenn es um Standard-Software geht. Was ist aber, wenn es sich bei solcher Standard-Software um komplexe Systeme handelt, in denen aufwändige Entwicklungsarbeit steckt? Das Risiko einer „Big-Bang-Umstellung“ wagen? Oder eher den Weg einer „sanften“, schrittweisen Umstellung einschlagen?

Dieser Artikel liefert einen „Werkstatt-Bericht“ eines solchen Migrationsprojekts: Das Unternehmen des Autors entwickelt und vertreibt seit 1991 das Software-Produkt *lisa.lims*, ein Labor-Informations- und Managementsystem. Es basiert bis zur Version 9 auf den klassischen Oracle-Technologien, also der Datenbank sowie Forms und Reports. Das System ist nach den Architektur-Empfehlungen von Oracle entworfen worden: Die Domänen beziehungsweise die Business-Logik steckt in Form von PL/SQL-Logik in der Datenbank. Dies ergibt folgendes Mengengerüst:

- 400 Forms
- 250 Reports
- 400 PL/SQL-Datenbank-Packages

Die migrierte Anwendung soll folgende Schlüssel-Anforderungen erfüllen:

1. Die vorhandene Business-Logik in PL/SQL in der Datenbank wiederverwenden, um die Investition zu schützen, die bislang dort hineingeflossen ist
2. Eine individualisierbare Benutzeroberfläche haben

3. Auf Java als Basis-Plattform basieren
4. Sowohl als Desktop- als auch als Internet-Anwendung verfügbar sein
5. In ihrer Internet-Version mindestens mit dem Microsoft Internet Explorer und Mozilla Firefox kompatibel sein
6. Mit möglichst vielen Applikations-Servern kompatibel sein

Der Artikel konzentriert sich auf die Punkte 1, 3 und 6. Die Anforderungen 2, 4 und 5 sind im Paper „Single Sourcing in Java: Desktop-Anwendung & Web-Applikation aus einer Quelle“ dargestellt (siehe www.t-p.com).

PL/SQL-Logik in der Datenbank bei der Migration

Bei der Auswahl der Migrations-Strategie spielte der immense Umfang der vorhandenen PL/SQL-Logik in der Datenbank eine entscheidende Rolle, da sie aufgrund dessen eine beachtliche Investition darstellt. Um diese zu schützen, ist es im Falle von *lisa.lims* nicht möglich, die Anwendung einer Big-Bang-Migration zu unterziehen – was bedeutet hätte, sie „auf der grünen Wiese“ unter Verwendung anderer Technologien wie etwa JEE vollständig neu zu erstellen. Dies wäre wirtschaftlich und zeitlich nicht sinnvoll gewesen. Daher steht die Wiederverwendung des vorhandenen PL/SQL ganz oben in der Liste der technischen Anforderungen an die migrierte Anwendung.

Darüber hinaus sind in der bereits vorhandenen PL/SQL-Logik auch Schreibzugriffe auf die Datenbank implementiert, zum Beispiel solche, die von Subsystemen wie etwa Laborgeräten erfolgen, die direkt an die Daten-

bank angebunden sind. Damit stellt die PL/SQL-Logik auch die Konsistenz von Schreibzugriffen auf die Datenbank sicher, die nicht direkt über die Applikation erfolgen. Die Geräte an die Mittelschicht anzubinden und damit die entsprechende Logik dort neu zu implementieren, hätte bedeutet, die gesamte Architektur der Geräteanbindung zu überarbeiten.

Relevanz der Mehrbenutzer-Fähigkeit und des pessimistischen Sperrverhaltens

lisa.lims ist ein Mehrbenutzer-System, in dem der Nutzer dediziert an der Datenbank angemeldet wird. Die PL/SQL-Logik in der Datenbank verwendet den Session-Kontext, unter anderem, um mittels Package-Variablen Datenbank-Änderungen nachzuverfolgen, die der Nutzer durchgeführt hat. Das bedeutet für das migrierte System, dass es ebenfalls in der Lage sein muss, jedem angemeldeten Benutzer eine individuelle Datenbank-Verbindung zuzuweisen.

Schließlich müssen im Arbeitsumfeld eines Labors exklusive Schreibzugriffe und lang laufende Transaktionen gewährleistet sein: Der Benutzer muss sich darauf verlassen können, dass für die Dauer seiner Untersuchung die entsprechenden Datensätze von niemand anderem geändert werden können, auch wenn die Untersuchungen an einer Probe länger dauern.

Technische Anforderungen an die migrierte Anwendung

Zusammengefasst muss die migrierte Anwendung folgende technische Anforderungen erfüllen:

1. Pessimistisches, exklusives Sperren von Datensätzen ermöglichen, auch bei lang laufenden Transaktionen
2. In der Datenbank gespeichertes PL/SQL aufrufen können
3. Jedem Applikationsbenutzer eine exklusive Datenbank-Verbindung zuteilen
4. Den Benutzer unter seinem spezifischen Account in der Datenbank anmelden

Technologiestudie: Spring vs. JEE

In einer Technologiestudie wurden zunächst die Java Enterprise Edition sowie das Spring Framework evaluiert. Es stellte sich heraus, dass die Umsetzung der Kernanforderungen mittels der Java-Enterprise-Plattform aus folgenden Gründen nicht möglich war:

- *Inkompatibilität der Schichten-Trennung*

Der JEE-Ansatz sieht vor, dass die Geschäftslogik grundsätzlich auf der Mittelschicht implementiert wird – und zwar im EJB-Container mittels Session oder Entity Beans, deren Geschäftsdaten mittels Container Managed Transactions in die Datenbank zurückgeschrieben werden. Dieser mittelschichtzentrierte Architektur-Ansatz steht im grundsätzlichen Gegensatz zum datenbankzentrierten, bei dem Geschäftslogik ganz oder teilweise in der Datenbank-Schicht implementiert ist. Dennoch wurde die Machbarkeit der Migration unter Verwendung der JEE-Plattform geprüft, nicht zuletzt deshalb, weil JEE bereits weit verbreitet ist und es sich um einen etablierten Standard handelt. Die weiteren Probleme führten jedoch letztendlich zu der Entscheidung, JEE für die Migration unserer Software nicht zu verwenden.

- *Container Managed Persistence*

JEE sieht für die Geschäftslogik in Session-Beans standardmäßig vor, dass Datenbank-Transaktionen durch den EJB-Container verwaltet werden. Ganz anders jedoch in lisa.lims: Verändert der Benutzer Daten, so sperrt die Anwendung die ent-

sprechenden Datensätze so lange, bis der Benutzer die Transaktion zurückrollt oder durch Speichern abschließt. Die Kontrolle der Transaktion liegt also beim Nutzer bzw. der Anwendung. Bei den meisten Applikations-Servern war es zum Zeitpunkt der Technologiestudie möglich, die automatische Transaktionsverwaltung des Servers abzuschalten und sogenannte „User-Transactions“ zu verwenden. Jedoch war diese Konfiguration meist herstellerepezifisch, was die Portierbarkeit stark einschränkt.

- *Connection Pooling*

In der JEE-Welt basiert die Verbindung zur Datenbank auf dem Prinzip des Connection Pooling: Der Applikations-Server startet eine definierte Anzahl von Verbindungen zur Datenbank, die ständig offen gehalten werden, um den Overhead des Verbindungsaufbaus zu minimieren. Führt der Benutzer in der Anwendung eine Datenbank-bezogene Aktion aus, so nutzt der Applikations-Server dazu eine der Verbindungen aus dem Pool und gibt die Verbindung am Ende der Aktion wieder in den Pool zurück, wo sie dem nächsten Applikations-Benutzer zu Verfügung steht. Dieses Konzept hat zwei Folgen:

- a. Alle Benutzer der Applikation nutzen ein und denselben Datenbank-Benutzer. Die Datenbanklogik, die den Session-Kontext des Datenbankbenutzers verwendet oder Änderungsverfolgung betreibt, ist damit nicht mehr verwendbar.
- b. Um die vorhandene PL/SQL-Logik in der Datenbank wiederverwenden zu können, kommen für lisa.lims nur solche Applikations-Server in Frage, die in der Lage sind, eine Datenbank-Funktion in der gerade aktiven Datenbank-Session des Benutzers auszuführen. Jedoch nicht alle evaluierten Server waren dazu in der Lage, was von der Implementierung der Java Persistence API (JPA) des jeweiligen Servers abhängt. Ein

Austausch der JPA-Implementierung ist zwar oft möglich, wird jedoch dann nicht mehr durch den Hersteller-Support abgedeckt.

- *Herstellerspezifische Erweiterungen des Applikations-Servers*

Die Spezifikation der Java-Enterprise-Architektur bezieht sich auch auf den Applikations-Server. Sun hat mit GlassFish eine Referenz-Implementierung vorgelegt. Andere Hersteller von Applikations-Servern weichen von diesem Standard ab, teilweise dadurch, dass sie spezifische Erweiterungen der Funktionalität bieten. Für die migrierte An-

Unsere Inserenten

ABOCRAFT www.abocraft.com	Seite 43
BASIS Europe Distribution GmbH www.basis-europe.eu	Seite 15
esentri consulting GmbH www.esentri.de	Seite 33
exensio GmbH www.exensio.de	Seite 11
Hunkler GmbH & Co. KG www.hunkler.de	Seite 3
imining GmbH www.imining.de	Seite 13
Libelle AG www.libelle.com	Seite 59
MuniQsoft GmbH www.muniqsoft.de	Seite 27
Neue Mediengesellschaft Ulm mbH www.databasepro.de	Seite 41
Nexlet GmbH www.nexlet.de	Seite 23
OPITZ CONSULTING GmbH www.opitz-consulting.de	U 2
ORACLE Deutschland GmbH www.oracle.com	U 3
PITSS GmbH www.pitss.com	Seite 57
Trivadis GmbH www.trivadis.com	U 4
WIN-Verlag GmbH & Co. KG www.digital-business-magazin.de	Seite 35

wendung relevante Spezifika fanden sich beim JBoss Application Server zum Beispiel im Bereich der Datensatzsperrern. Da die migrierte Applikation jedoch auf möglichst vielen Applikations-Servern lauffähig sein soll, ist die Nutzung solcher Hersteller-Spezifika strikt zu vermeiden.

Am Ende der Technologie-Studie blieb von der Vielzahl der Features, die die JEE-konformen Applikations-Server bieten, nur noch eines übrig, das für die Anforderungen relevant war: die Möglichkeit, die Datenbank-Zugriffe im Applikations-Server in JavaBeans zu kapseln. Die dedizierte Verbindung zwischen einer solchen Bean und dem Client hätte in diesem Fall dadurch erreicht werden können, dass dem Client seine jeweilige Session-Bean mittels JNDI per Remote-Zugriff verfügbar gemacht wird. Dies hätte auch ermöglicht, die Datenbanklogik zu einem späteren Zeitpunkt schrittweise in der Mittelschicht in Java neu zu implementieren. Allerdings bietet das

Spring Framework für das Remoting eine wesentlich leichtgewichtige Alternative. Aufgrund der beschriebenen Ergebnisse der Technologie-Studie hat sich das Unternehmen der Autoren für den nachfolgenden Ansatz entschieden (siehe http://www.t-p.com/fileadmin/template/Upload/PDF/itsolutions_news/Oracle_Forms_goes_Web_2.0.pdf).

Verwendung von OSGi als Basistechnologie

Eine der Anforderungen lautete, dass die migrierte Anwendung sowohl als Desktop- als auch als Web-Client laufen soll. Deshalb fiel die Entscheidung auf Eclipse RCP/RAP als eine der Basis-Plattformen, um Web- und Desktop-Client auf derselben Code-Basis zu entwickeln. Eclipse basiert auf OSGi. Bei OSGi handelt es sich um eine auf Java aufbauende Spezifikation, die dazu dient, Anwendungen durch ein Komponentenmodell zu modularisieren und zu verwalten. Durch die Entschei-

dung für Eclipse RCP/RAP war die Verwendung von OSGi für die Client-Seite bereits gesetzt. Im Sinne einer durchgängigen Technologiebasis bot es sich an, auch für den Server OSGi-fähige Lösungen einzusetzen und diesen damit ebenfalls modular gestalten zu können. Das entspricht dem äußerst modularen Aufbau von lisa.lims. Das Eclipse-Projekt „Equinox“ stellt eine quelloffene OSGi-Implementierung zur Verfügung. Diese Laufzeit-Umgebung ist äußerst flexibel einsetzbar: Sie kann einerseits als eigenständige Serveranwendung laufen; andererseits kann sie in ein Servlet eingebettet werden und somit auf jeden Servlet-fähigen Web-Server als normale Webanwendung deployed werden (als WAR- oder EAR-Archiv).

Verwendung des Spring Frameworks

Das Spring Framework bietet mittels OSGi die gesuchte Möglichkeit, auch die Server-Seite modular aufzubauen. Damit basieren Client- und Server-Sei-

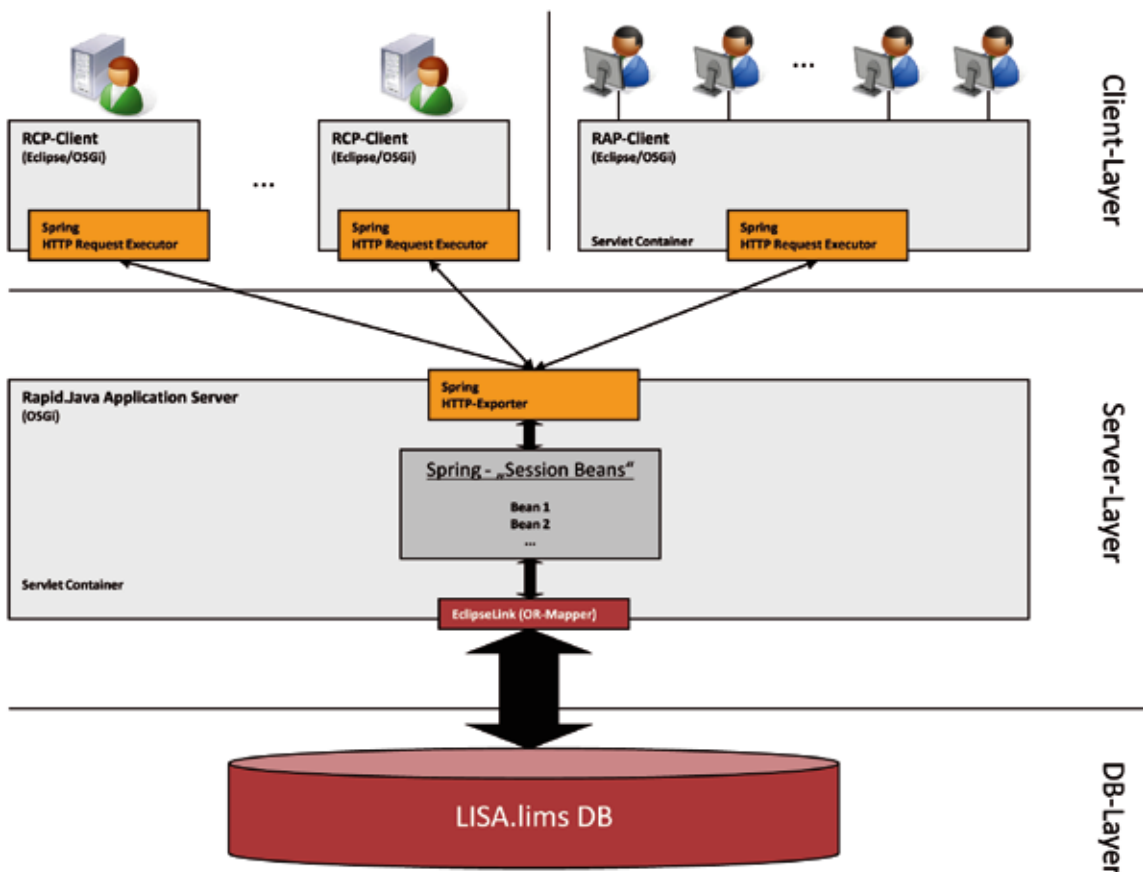


Abbildung 1: Remoting mithilfe des Spring Frameworks

te in der migrierten Anwendung auf derselben Technologie. Mit Spring ergibt sich auch die oben angesprochene, leichtgewichtige Alternative zu JNDI: In diesem Ansatz implementieren die Spring-Beans auf der Mittelschicht die Zugriffe auf die Datenbank. Das Dispatcher-Servlet des Spring Frameworks bindet die einzelnen Spring-Beans dediziert an die einzelnen HTTP-Sessions, die in einem Servlet-fähigen Webserver laufen. Der jeweilige Client greift auf seine Web-Session mittels des Spring-HTTP-Invokers zu.

Verwendung des Objekt-relationalen Mappers EclipseLink

EclipseLink (derzeit in der Version 1.3 in Verwendung) ermöglicht es, Datensätze in der Datenbank pessimistisch zu sperren, Stored-Functions und -Procedures in der Datenbank aufzurufen, jedem Applikationsbenutzer eine exklusive Datenbank-Verbindung zuzuteilen und den Benutzer unter seinem spezifischen Account in der Datenbank anzumelden. Gleichzeitig implementiert EclipseLink die JPA-Spezifikation, sodass auch viele nützliche JPA-Features genutzt werden können. Für die JPA-Spezifikation 2.0 ist EclipseLink die Referenz-Implementierung. Da das EclipseLink-Projekt auf dem O/R-Mapper „Toplink“ von Oracle basiert, ist es

sogar möglich, einige herstellereigenspezifische Funktionen der Oracle-Datenbank zu nutzen. Damit war EclipseLink unsere erste Wahl. EclipseLink steht ebenso wie das Spring Framework in einer OSGi-kompatiblen Variante zur Verfügung.

Fazit

Der gewählte Ansatz bietet folgende Vorteile:

1. Er verleiht der migrierten Java-Anwendung die datenbankrelevanten Eigenschaften, die in der Welt von Oracle-Forms etablierter und geschätzter Standard sind: die Mehrbenutzer-Fähigkeit inklusive des pessimistischen DML-Sperrverhaltens und der individualisierten Anmeldung an der Datenbank sowie die Integration von vorhandener PL/SQL-Logik.
2. Die Kombination von EclipseLink mit der Remote-Lösung des Spring Frameworks ermöglicht es, auf Basis von OSGi einen Applikations-Server „zusammenzustecken“, der die zuvor genannten Datenbank-Anforderungen erfüllt und dabei ohne JEE-Overhead auskommt. Diese Lösung zeichnet sich dadurch aus, dass sie leichtgewichtig und äußerst flexibel ist: Der Server kann entweder stand-

alone laufen oder in jeden Servlet-fähigen Web- beziehungsweise Applikations-Server als Standard-Webanwendung integriert werden. Somit ist die gewählte Architektur unabhängig von den Spezifika einzelner Applikations-Server.

3. Die Integration von vorhandener PL/SQL-Datenbank-Logik bedeutet einen Schutz der bereits getätigten Investition und einen geringeren zeitlichen und finanziellen Aufwand bei der Migration.
4. Der Ansatz ermöglicht eine „smart migration“ – und zwar im Sinne eines schrittweisen, fließenden Übergangs von der alten auf die neue Technologie-Plattform. Die Grundlage hierfür ist die Wiederverwendung der vorhandenen PL/SQL-Business-Logik: Einzelne besonders zentrale Dialoge können bereits auf die neue Technologie für die Benutzer-Oberfläche umgestellt werden, während andere Dialoge zunächst weiter auf Oracle Forms basieren. So können alt und neu koexistieren, bis die gesamte Migration abgeschlossen ist. Anschließend kann die Neu-Implementierung der Business-Logik in Java ins Auge gefasst werden.

Kontakt:

Thomas Haskes, t.haskes@t-p.com
Oliver Zandner, o.zandner@t-p.com

EIERLEGENDEWOLLMILCHSAUDASBLAUEVOMHIMMEL
NULLACHTFÜNFZEHNBLABLABLABLABLABLABLA
BLABLABLAGOLDESELTRALALA *

* machen wir nicht.

Wir bringen Sie in **FORMs 11^g**
DEVELOPMENT • MIGRATION • SUPPORT

rely on your profIT-partner | www.nexlet.de



nexlet