Investitionsschutz durch Java Hooks, die Forms Trigger ersetzen

Jens Hüttemann
Triestram & Partner GmbH
Bochum

Schlüsselworte
rapid.java, Oracle, Forms, Trigger, Java Hooks, Investitionsschutz

Einleitung

In Business-Anwendungen, die über einen langen Zeitraum entwickelt, gewartet und betrieben werden, fließen mit der Zeit viele Investitionen.
Dabei handelt es sich um *messbare* Investitionen, wie Lines of Code, Masken oder Datenmodelle, bei denen man die Entwicklungszeit addieren kann und um *nicht messbare* Investitionen, die in Form von Erfahrungen mit der Anwendung vorliegen. Dies sind Erfahrungen von Entwicklern, Projektleitern und Anwendern.
Business-Anwendungen auf Basis von Oracle Forms enthalten mitunter einen beachtlichen Anteil von PL/SQL in der Oracle Datenbank. Eine solche Architektur, in der Business-Logik in der Datenbank implementiert ist, wird auch als datenbank-zentriert bezeichnet.
Im Unterschied dazu ist die Architektur von Java-Enterprise-Anwendungen meist mittelschicht-zentriert, da hier die Business-Logik im Application Server implementiert wird.
Durch den mächtigen Triggermechanismus haben Oracle Forms Anwendungen ein typisches Grundverhalten, das allen damit erstellten Modulen zueigen ist. Langjährige Anwender sind die Bedienung von Forms Modulen gewohnt. Java-Enterprise-Anwendungen haben ein solches typisches Verhalten nicht.
Folgende Fragen beschäftigen viele IT-Verantwortliche, die eine Forms-Migration planen.
Was bedeuten die Unterschiede zwischen den unterschiedlichen Architekturen bei der Migration einer datenbank-zentrierten Oracle Forms Anwendung in eine Java-Enterprise-Anwendung?
Muss die vorhandene PL/SQL-Logik in der Datenbank komplett verworfen und im Application Server neu implementiert werden?
Sind die Erfahrungen bei Entwicklern, Projektleitern und Anwendern mit der Einführung einer Java-Enterprise-Anwendung veraltet und somit nicht mehr zu gebrauchen?
Müssen die Anwender neu geschult werden, um die Bedienung der migrierten Anwendung zu erlernen?

Ausgangslage

Die vorangegangenen Fragen stammen aus einem Migrationsprojekt, bei dem die Triestram & Partner GmbH (**t&p**) ihr Forms basiertes Standard-Software-Produkt lisa.lims auf eine Java-Enterprise-Architektur umgestellt hat. Bei lisa.lims handelt es sich um ein Labor-Informations- und Managementsystem, das bis zur Version 9 auf den „klassischen“ Oracle-Technologien basierte, nämlich der Datenbank sowie Forms und Reports. Das System ist nach den Architektur-Empfehlungen von Oracle gestaltet worden - das heißt: Die Domänen- bzw. Business-Logik steckt in Form von PL/SQL-Logik in der Datenbank.
Dies zeigt sich in folgendem Mengengerüst:

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

Für die Version 10 wurde lisa.lims nach Java migriert. Dabei wurde die Oracle Datenbank beibehalten und Oracle Forms und Oracle Reports wurden durch Java Open-Source Komponenten ersetzt:

- die Eclipse Rich Client Platform (RCP) bzw. die Eclipse Rich Ajax Platform (RAP) für die Benutzer-Oberfläche und
- die Eclipse Business Intelligence and Reporting Tools (BIRT) für die Berichte

Investitionsschutz

Bei der Auswahl der Migrations-Strategie spielte der große Umfang der vorhandenen PL/SQL-Logik in der Datenbank eine entscheidende Rolle, denn sie stellte eine beachtliche Investition von ungefähr 80 – 100 Personen-Jahren dar. Um diese Investition zu schützen war es nicht sinnvoll, die Business-Logik einer Big-Bang-Migration zu unterziehen und sie „auf der grünen Wiese“ in Java auf der Mittel-Schicht neu zu implementieren. Das wäre wirtschaftlich und zeitlich nicht sinnvoll gewesen. Daher hatte die Wiederverwendung des vorhandenen PL/SQL eine zentrale Bedeutung in der Liste der Anforderungen an die migrierte Anwendung.

Die Entscheidung für die Wiederverwendung der PL/SQL Logik hatte Konsequenzen darauf, welche Grundanforderungen sich an die Verbindung von Java zur Oracle-Datenbank ergaben.

Die migrierte Anwendung musste:

1. in der Lage sein, in der Datenbank gespeichertes PL/SQL aufzurufen
2. jedem Applikationsbenutzer eine exklusive Datenbank-Verbindung zuteilen können, die er über die Dauer der Anwendungssitzung beibehält
3. den Benutzer unter seinem spezifischen Account in der Datenbank anmelden
4. pessimistisches, exklusives Sperren von Datensätzen ermöglichen, wenn schreibende Zugriffe auf die Daten erfolgen

Neben der zu migrierenden Oracle Forms Anwendung nutzten weitere Systeme die PL/SQL-Logik für schreibende Zugriffe auf die Datenbank. Aus diesen verschiedenen Zugriffsarten resultierten die genannten Anforderungen.

Erfahrung bei Anwendern, Entwicklern und Projektleitern

Neben den rein technischen Betrachtungen zum Investitionsschutz, die sich in der Wiederverwendung der PL/SQL-Logik widerspiegelten, spielten bei der Migration weitere Rahmenbedingungen eine wichtige Rolle.

Die Erfahrungen im Umgang mit der Software waren eine nicht unerhebliche Investition, die bei der Migration nicht verloren gehen durfte.

lisa.lims ist eine Anwendung, die seit über 20 Jahren entwickelt wird, die in vielen Kundenprojekten installiert und in zahlreichen Workshops an Kundenwünsche angepasst wurde.

Bei Entwicklern und Projektleitern hat sich in dieser Zeit ein immense Fachkenntnis in Bezug auf das Datenmodell, die PL/SQL-Logik und die Prozesse und Verhaltensweisen der Anwendungen aufgebaut.

Auch in vorhanden Spezifikationen, Datenmodellbeschreibungen und Testprotokollen steckten erhebliche Investitionen.

Die Seite der Anwender wurde ebenfalls betrachtet. Hier war natürlich auch eine Menge Know-How im Umgang mit lisa.lims vorhanden. Die Anwender wurden für den Umgang mit lisa.lims geschult und konnten das System sicher bedienen.

Es war uns wichtig, dass Anwender, die lisa.lims kannten, nach der Migration, weiterhin problemlos mit der umgestellten Anwendung konnten.

Daraus wurde die Vorgabe abgeleitet, Prozesse in der migrierten Anwendung ähnlich zu implementieren, wie in der Forms Anwendung, so dass die bereits vorhandene Dokumentation weiter gültig war und die Anwender, Projektleiter und Entwickler ihre bereits errungenen Erfahrungen weiter nutzen konnten.

Ziel der Migration
Unser Ziel bei der Migration war es, in der migrierten Software das Beste aus zwei Welten miteinander zu vereinen.
Die Oracle-Datenbank war gesetzt. Die Datenbank wurde bei der Migration nicht als reiner Datenspeicher gesehen, wie es bei vielen Java-Frameworks der Fall ist. Wir wollten die Stärken der Oracle-Datenbank weiter nutzen.
Auch die Prozesse und Verhaltensweisen innerhalb der Anwendung, die zum Teil durch Forms vorgegeben waren, wollten wir beibehalten.
Auf der anderen Seite sollte durch die Migration eine Open-Source basierte Java-Applikation entstehen. Benutzeroberflächen in Java Anwendungen können wesentlich moderner und ergonomischer gestaltet werden, als es mit Forms möglich ist.
Java Programme sind flexibel erweiterbar und können an Sonderwünsche angepasst werden.
Es gibt viele Bibliotheken, auf die im Bedarfsfall zugegriffen werden kann, z.B. Office-Integration oder Grafiken. Es ist wesentlich einfacher Drittsoftware einzubinden als es bei Forms Anwendungen möglich ist.

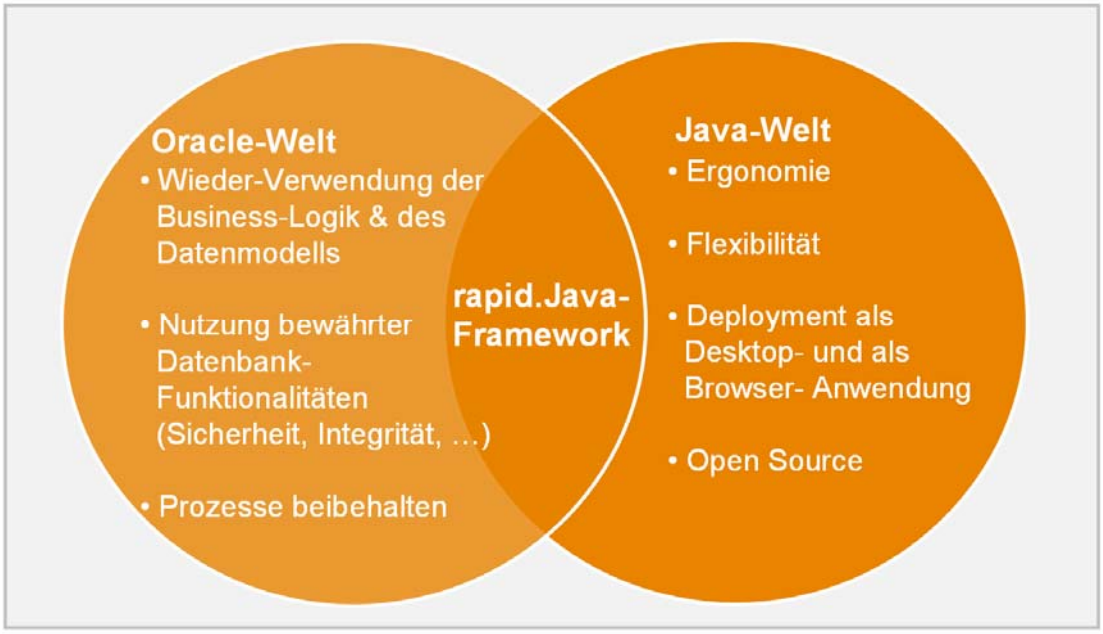

Abb. 1: Das rapid.java Framework verbindet die Oracle-Welt mit der Java-Welt

Die Schnittmenge, die wir aus den Anforderungen gebildet haben, ist das rapid.Java Framework. Dies ist zum einen eng mit der Oracle-Datenbank verknüpft, zum anderen ein Java-Framework, dass auf Java Open-Source-Technologien beruht. Dadurch dass es moderne Oberflächenelemente enthält ist es wesentlich ergonomischer als man es von Forms gewohnt ist.

Forms Trigger und Java
Das rapid.java Framework bildet die Grundlage unserer migrierten Anwendung.
In ihm sind die Mechanismen für die Zugriffe auf die Datenbank enthalten. Das Framework stellt sicher, dass PL/SQL-Logik in der Datenbank aufgerufen und das dabei auf bewährte Datenbankfunktionalitäten (Sessionmanagement, Integrität, pessimistisches Locking, Verwaltung von Master-Detail-Beziehungen) zurückgegriffen werden kann.
Es reicht aber nicht aus, im Framework nur die Möglichkeit zu schaffen, auf die Datenbank zugreifen und die PL/SQL-Logik aufrufen zu können. Im Framework haben wir die Abläufe so verankert, dass

die richtigen Stellen im Programmablauf entstanden, an denen die vorhandene PL/SQL-Logik aufgerufen werden konnte.
In Forms Anwendungen sind diese Stellen durch die Trigger vorgegeben. Trigger geben einer Forms Anwendung ein Grundverhalten.
Auch im rapid.java Framework ist ein Grundverhalten implementiert, das es ermöglicht, Anwendungen zu erstellen, deren Abläufe denen von Oracle-Forms Anwendungen ähneln. Unterschiedliche Methoden im Framework ermöglichen das gezielte Eingreifen in den Programmablauf und den Einbau spezieller Anwendungslogik. Diese Methoden imitieren die Forms Trigger.
Durch die Imitation der Forms Trigger haben wir erreicht, dass unser vorhandener PL/SQL-Code wiederverwendet werden kann, da Datenbankroutinen zu gleichen Zeitpunkten im Anwendungsablauf aufgerufen werden und somit die selbe Aufgabe haben wie vor der Migration. Für den Anwender ergibt sich daraus ein Wiedererkennungswert, da die Anwendung bei gleichen Benutzeraktionen genauso reagiert, wie er es aus der Forms Anwendung gewohnt ist.

Entwurfsmuster für die Imitation von Triggern in Java
Für die Imitation der Trigger im rapid.java Framework stützen wir uns auf gängige Entwurfsmuster.
Die Herausforderung liegt nicht darin, die Entwurfsmuster zu verstehen und zu implementieren, sondern den Programmcode so zu gestalten, dass das gezielte Eingreifen in den Programmablauf an den richtigen Stellen möglich ist.
Das am häufigsten eingesetzte Entwurfsmuster ist die Vererbung bzw. die Vererbung mit Hook-Methoden.

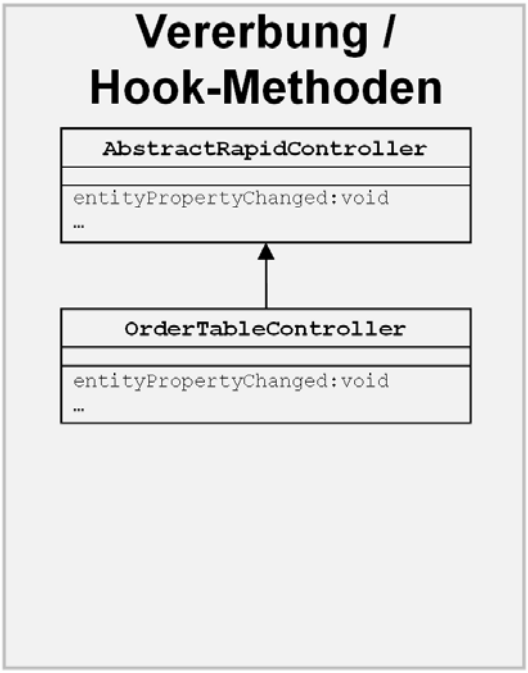

Abb. 2: Trigger durch Vererbung und Hook-Methoden

Das Objektmodell im rapid.java Framework implementiert das Standardverhalten, das zunächst mal alle wichtigen Voraussetzungen für eine funktionsfähige Anwendung schafft.
Zum Beispiel registriert das Framework, wenn Benutzer mit der Eingabe von Daten beginnen, wenn ein Feld verlassen, ein Datensatz gewechselt oder angelegt wird.
Die genannten und noch weitere Funktionen werden in Methoden im Objektmodell des Frameworks behandelt. Hier können nun durch Vererbung von Frameworkklassen, allgemeine Methoden weiter

spezialisiert und zusätzliche anwendungs- oder maskenspezifische Funktionalitäten hinzugefügt werden.
Eigentlich ist die Vererbung im objektorientierten Umfeld nichts Besonderes; es ist eher eine Grundeigenschaft von objektorientierten Programmiersprachen und die Konzepte sind bekannt.
Die Herausforderung bei der Entwicklung bzw. beim Design besteht darin, genau zu planen, was für die Vererbung freigegeben werden soll, d.h. welche Methode welche Sichtbarkeit bekommt. Nicht bei allen Methoden macht es Sinn, dass sie vererbt werden können.
Man muss die Methoden in den Klassen so strukturieren, dass dadurch Einheiten entstehen, die man durch Vererbung sinnvoll überschrieben kann.
Außerdem muss beim Design sichergestellt werden, dass durch die Vererbung nicht Standardverhalten, das zwingend notwendig ist, überschrieben wird.
Beim Entwurfsmuster der Hook-Methode (oder auch Einschubmethode) wird durch Vererbung spezialisiert.
Eine Hook-Methode ist eine Methode, die in einer Klasse definiert ist, die aber eine leere oder eine Default-Implementierung enthält. Dadurch verschafft man Unterklassen die Möglichkeit, sich an definierten Stellen gezielt in einen Algorithmus einzuhängen. Sinnvoll sind Hook-Methoden dann, wenn mitten in einer Methode die Möglichkeit geschaffen werden soll, zu spezialisieren.
Bei der normalen Vererbung kann das Standardverhalten einer Methode aus der Superklasse nur vor oder nach der Spezialisierung ausgeführt werden.
Mit Hook-Methoden können auch während des Ablaufs in der Methode der Superklasse Spezialisierungen eingefügt werden.
Auf diese Weise ist es auch möglich, das Standardverhalten zu schützen, wenn nur die Hook-Methoden die notwendige Sichtbarkeit für die Vererbung bekommen.

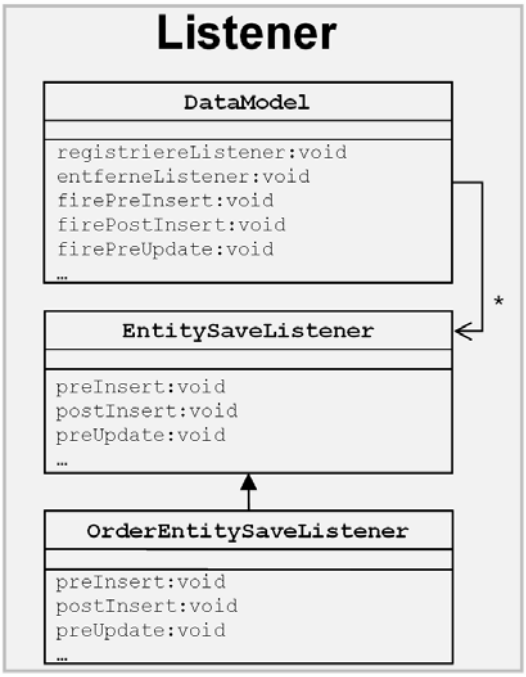

Abb. 3: Trigger durch Listener bzw. das Observer-Pattern

Ein weiteres Entwurfsmuster, das für die Imitation von Triggern verwendet wird, sind Listener bzw. das Observer-Pattern.
Hier ist es ebenfalls so, dass die Klassen aus dem rapid.java Framework ein Standardverhalten implementieren, das ausreicht, um eine Anwendung zu erstellen.

Um mit diesem Entwurfsmuster Spezialisierungen zuzulassen, löst eine Frameworkklasse an definierten Stellen im Programmablauf Ereignisse aus, auf die andere Klassen reagieren können - also der klassische Triggeransatz.
Eine Klasse, die auf die Ereignisse reagieren möchte, kann sich bei der ereignisauslösenden Klasse registrieren oder auch einhaken, um im Bild zu bleiben.
Wenn ein solches Ereignis eintritt, werden alle registrierten Klassen informiert, können darauf reagieren und spezialisierten Programmcode ausführen.

Fazit
Wir haben mit dem rapid.java Framework ein Framework geschaffen, dass es ermöglicht, Anwendungen zu erstellen, die in ihrem Verhalten dem von Oracle Forms Anwendungen ähneln. Erreicht haben wir das dadurch, dass wir in das Framework Java Hooks eingebaut haben, die die Forms Trigger ersetzen.

Dadurch haben wir sichergestellt, dass wir bei der Migration unserer Oracle Forms Anwendung lisa.lims, die vorhandene Business-Logik in Form von PL/SQL-Logik in der Datenbank wiederverwenden konnten, da im rapid.java Framework die gleichen Zeitpunkte im Programmablauf wie in Oracle Forms vorhanden sind. Die in die Entwicklung der PL/SQL-Logik geflossenen Investitionen wurden geschützt.

Die in Form von Erfahrungen mit der Oracle Forms Variante von lisa.lims bei Anwendern, Projektleitern und Entwicklern vorhandenen Investitionen wurden durch unseren Ansatz bei der Migration ebenfalls geschützt. Die migrierte Anwendung verhält sich bei gleichen Benutzeraktionen genauso wie vor der Migration, wodurch nur eine kurze Umgewöhnungsphase notwendig ist.

Kontaktadresse:
Jens Hüttemann
Triestram & Partner GmbH
Kohlenstrasse 55
D-44795 Bochum

Telefon: +49 (0) 234-94375 0
Fax: +49 (0) 234-45 22 06
E-Mail j.huettemann@t-p.com oder info@t-p.com
Internet: www.t-p.com