# Modellbasierte Migration eines komplexen Versicherungssystems von RPG nach Java

Matthias Gutheil
gutheil@itemis.de
itemis AG

Bernhard Stadler
stadler@itemis.de
itemis AG

## Zusammenfassung

Große Legacy-Systeme auf aktuelle Technologien zu migrieren wird immer häufiger notwendig und ist eine spannende und vielschichtige Aufgabe. In diesem Paper beschreiben wir die Migration eines komplexen Versicherungssystems von RPG nach Java, mit Schwerpunkt auf UI-Replatforming und Testing.

## 1 Einleitung

Um im wirtschaftlichen Wettbewerb bestehen zu können, müssen sowohl Softwareprodukte als auch In-House-Entwicklungen mit den sich ändernden Anforderungen des Marktes (z.B. Kosten, Entwicklungsgeschwindigkeit, Technologien) mithalten. Meist reicht es aus, die Softwaresysteme auf der bestehenden Plattform weiterzuentwickeln, aber was wenn der Markt für Plattformen, Programmiersprachen oder Hersteller, an die man sich durch Investition zehn- oder hunderttausender Personentage gebunden hat, immer weiter schrumpft und die Wettbewerbsfähigkeit leidet?

In einem Kundenprojekt für einen großen Anbieter von Versicherungssoftware migrierten wir den Lebensversicherungsteil des Systems von RPG auf Java (JSF, Spring, JPA) mit einem modellbasierten Ansatz. Damit transformierten wir 7 Millionen Zeilen Legacy-Code in 3 Millionen Zeilen Java-Code und etwa 700.000 Zeilen Modellcode. Der Modellcode ist die Basis für die Erzeugung beispielsweise von JSF Facelets, SQL-Anweisungen, JPA-Mapping-Dateien und Integrationstests.

Wir beschreiben den angewendeten modellbasierten Ansatz, die wichtigsten Teile des Projektes und unsere Erfahrungen.

## 2 Projektphasen

Das Projekt bestand aus einer Proof-of-Concept-Phase, in der die Machbarkeit des modellbasierten Ansatzes verifiziert wurde, und einer Realisierungsphase, in der die Migration umgesetzt wurde.

### 2.1 Proof of Concept

Während der Proof-of-Concept-Phase erhoben wir in enger Zusammenarbeit mit dem Kunden Anforderungen für die Zielarchitektur, die Migration und den Entwicklungsprozess. Darauf aufbauend entwarfen wir Prototypen des Zielsystems und domänenspezifischer Sprachen (DSL). Mit diesen lassen sich die technischen und fachlichen Aspekte nicht nur des bestehenden Systems, sondern auch künftiger neu geschriebener Anwendungen abbilden.

### 2.2 Realisierungsphase

Nachdem sichergestellt war, dass der gewählte Ansatz erfolgversprechend ist, setzten wir die Migration durch folgende Punkte um:

- Implementierung der Codegenerierung von RPG nach Java
- Finalisierung der Zielarchitektur
- Entwicklung von DSLs und Generatoren für verschiedene Aspekte des Systems
- Entwicklung eines Testverfahrens

## 3 Modellbasierte Softwaremigration

Abb.1 zeigt eine Übersicht der modellbasierten Migration. Teile des Systems werden in Modelle überführt, aus denen Code für das neue System generiert wird. Hierzu zählen beispielsweise DSLs für Datenmodellierung, UI und ORM. Andere Systemteile wie die Businesslogik werden direkt in die Zielsprache konvertiert.

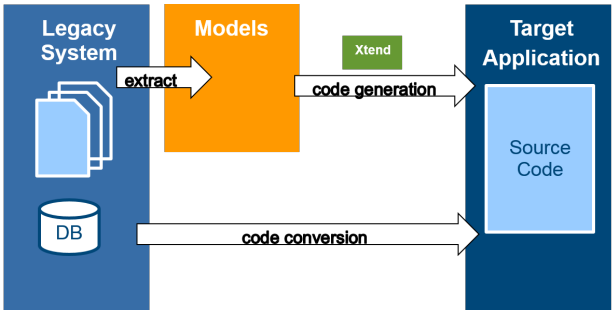


Abbildung 1: Modellbasierte Migration

Durch die unterschiedlichen Arten der Sourcecode-Erzeugung (generiert/konvertiert) war es besonders wichtig, eine funktionierende Continous-Integration-Umgebung aufzubauen.

## 4 UI-Replatforming

Als Zieltechnologie für die Benutzerschnittstelle wurde JSF gewählt. In der Original-Anwendung wurde sie über Display Files (DSPF) beschrieben, die selbst eine urtümliche DSL für IBM 5250-Textterminals darstellen [2]. Aufgrund Anzahl und Größe der Eingabedateien (über 1 Mio. LoC) bestand die Herausforderung darin, die 5250-Dialoge strukturtreu automatisiert in JSF Facelets zu überführen, dabei aber Spielraum für Verbesserungen und Anpassungen an moderne Webtechnologien zu lassen. Hierzu wurden DSLs für eine MVVM-Architektur entwickelt:

- Business Object DSL: Geschäftsobjekte
- Viewmodel DSL: UI-Zustand
- Screen DSL: UI-Struktur

Mit diesen DSLs lassen sich neu entwickelte JSF-Anwendungen abbilden, aber durch Legacy-Support-Features vor allem in der Viewmodel DSL wird auch eine struktur- und verhaltenstreue Darstellung von Legacy-Dialogen ermöglicht. Gleichzeitig kann man durch die Separierung von Dialogstruktur, Dialogzustand und Geschäftsobjekten ebendiese unabhängig voneinander anpassen und wiederverwenden.

Um das Verhalten der Dialoganwendungen mit Java-Enterprise-Mitteln nachzubilden, war die Entwicklung einer umfangreichen Legacy-Laufzeitumgebung notwendig, die in Kombination mit den Sprachmitteln der DSLs unter anderem folgende Unterschiede überbrücken musste:

- Synchrone vs. asynchrone Benutzerinteraktion
- Zustand vs. statische Webseiten
- Absolute vs. relative Positionierung
- Dicktengleiche vs. proportionale Schrift
- Funktionstasten und Schlüsselzahlen vs. Hyperlinks zur Navigation

Der UI-Zustand wurde in den generierten Viewmodels gehalten, die sofern nötig mit der Legacy-Laufzeitumgebung interagierten. Die Positionsinformationen für die Dialogelemente ließen sich gut in ein Grid-Layout überführen.

## 5 Testen

Mit der wichtigste Teil eines Migrationsprojektes ist das Testen. Durch eine ausreichende Testabdeckung muss gewährleistet werden, dass das migrierte System sich so verhält wie das Altsystem. Da für das Legacy-System keine für eine Migration ausreichende Testabdeckung vorhanden war, musste eine Möglichkeit gefunden werden, Tests im Altsystem zu erstellen und diese in das neue System zu überführen.

Die Altanwendung konnte mittels IBM WebFacing mit einem normalen Browser aufgerufen werden. Da die migrierte Anwendung auf JSF basiert, entschlossen wir uns für folgenden Testansatz:

1. Manuelles Erstellen von Testfällen im Altsystem durch Aufzeichnen
2. Transformieren der aufgezeichneten Testfälle in die Zielplatform

Das verwendete Testwerkzeug zum Aufzeichnen war Unified Functional Testing (UFT) [3] von HP. Das Format der Testfälle ermöglichte es uns, eine DSL dafür zu erstellen, so dass wir die aufgezeichneten Testfälle in die Test DSL transformieren konnten. Die Test DSL (siehe Abb.2) ist eine werkzeugunabhängige Sprache zum Beschreiben von Tests im Web, mit besonderer Unterstützung für mittels Screen-DSL geschriebener JSF-Anwendungen.

Im nächsten Schritt transformierten wir die Tests in ausführbare Selenium [1] Tests.

Dieses Verfahren funktioniert nur, wenn die IDs der HTML-Komponenten im Altsystem auf IDs im neuen System gemappt werden können. Dies konnten wir sicherstellen und eine ausreichende Testabdeckung gewährleisten.

```
package extract_PAR00001

scenario extract_PAR00001 description "extract par00001"

step extract_PAR00001 description "extract par00001" {
  inputField(id = "userID").setText("testuser")
  inputField(id = "password").setSecureText("...")
  button(value = "Log in").click()
  browser.navigateTo("http://test.i2s.pt/myportal/...")
  button(tag = "INPUT").click()
  ...
  assertEquals("CHECK", "1", outputField(id="11...").getText())
}
```

Abbildung 2: Test DSL

## 6 Zusammenfassung

Das Projekt hat bestätigt, dass der modellbasierte Migrationsansatz geeignet ist um ein Legacy-System in ein neues System mit moderner Architektur, aktueller Programmiersprache und state-of-the-art UI-Technologien zu überführen. Die Zukunftsfähigkeit des Ergebnisses ist sichergestellt, da auch die Entwicklung neuer, mit den migrierten Programmen integrierbarer Anwendungen nach modernen Ansätzen wie Continuous Testing und Continuous Integration ermöglicht wurde.

## Literatur


[1] *Selenium - Web Browser Automation.* `http://www.seleniumhq.org/`. Accessed: 2016-04-24.

[2] *System i: Application Display Programming.* Version 6 Release 1. IBM. 2008.

[3] *Testautomatisierung, Unified Functional Testing, UFT — Hewlett Packard Enterprise Deutschland.* `http://www8.hp.com/de/de/software-solutions/unified-functional-automated-testing/`. Accessed: 2016-04-24.