In diesem Artikel wird die Forms2Java-Migration von Komponenten zur Pflege von Systemtechnik- und Sektor-Daten in einer Anwendung für die Verwaltung des Auf- und Umbaus eines Mobilfunknetzes bei einem unser Kunden beschrieben. Hierbei werden sowohl die technische Umsetzung als auch dabei aufgetretene Probleme erläutert. Im Anschluss wird auf besondere Herausforderungen explizit eingegangen und ein Fazit für künftige Forms2Java-Migrationen gezogen.

Forms2Java-Migration komplexer Anwendungs-Komponenten für die Verwaltung eines Mobilfunknetzes

Alexander Joedt, OPITZ CONSULTING GmbH

Eine Forms-10*g*-Anwendung verwaltet sowohl den Netzaufbau als auch den Netzumbau eines Mobilfunkanbieters. Dafür dokumentiert die Anwendung sämtliche Infrastruktur-Komponenten (Ausrüstung, Standorte und Kandidaten für neue Standorte) sowie zusätzliche Informationen (Kontakte, Vertragsdaten, Dokumente etc.). Sie dient

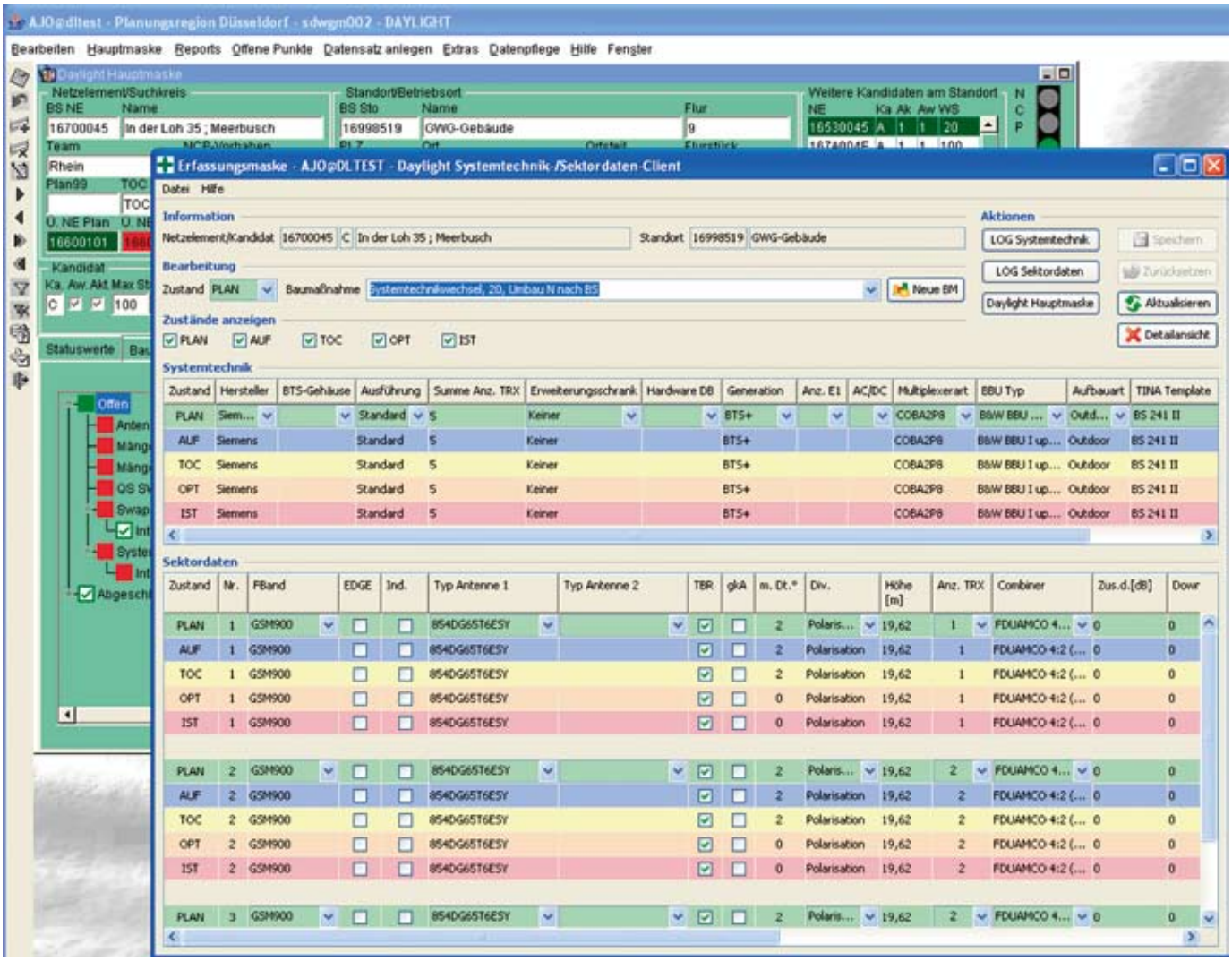

Abbildung 1: Forms-Anwendung mit Java-Rich-Client im Vordergrund

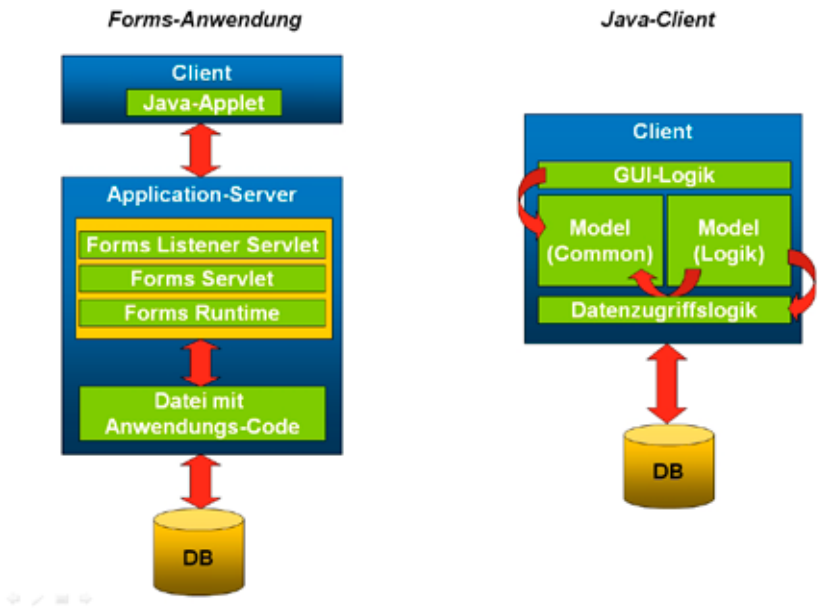

Abbildung 2: Architekturvergleich: Forms-Anwendung vs. Java-Rich-Client

weiterhin der Verwaltung der ablauforganisatorischen Regelungen und des Projekt-Controllings. Die Anwendung wird auf Basis einer Oracle-11*g*-Datenbank betrieben. Wartung und Weiterentwicklung erfolgen im Rahmen eines Outtasking-Projekts, an dem auch der Autor dieses Beitrags mitwirkt. Insgesamt arbeiten etwa 1200 Benutzer mit der Anwendung, davon bis zu 600 parallel.

Ziel der Migration war eine Erhöhung der Anwender-Ergonomie durch das Zusammenfassen von zwei thematisch zusammengehörigen Registerkarten einer Forms-Maske in einer übersichtlichen Java-Anwendung. Außer dem wollte man durch die Einführung von zusätzlichen Plausibilitätsprüfungen und einer geführten Benutzereingabe eine Verbesserung der Datenqualität erreichen. Neue, flexible Konfigurationsmöglichkeiten der Daten sollten zudem zukünftige Entwicklungsaufwände reduzieren, und die Modularisierung der Java-Anwendung sollte die Wartbarkeit deutlich erhöhen.

Projektumfang und -team

Die Realisierungsphase hatte einen Umfang von ca. 350 Manntagen. Das Projektteam bestand in der Realisierungsphase aus drei Java-Entwicklern, einem für die Forms- und PL/SQL-Anpassungen zuständigen Entwickler und einem fachlichen Projektleiter. Vorab wurde mit Key-Usern (Funk-, Festnetz- und Parameterplaner) und System-Analytikern des Auftraggebers ein Workshop durchgeführt und ein detailliertes Fachkonzept erstellt. Aufgrund von verschiedenen Anwenderwünschen, die im Workshop geäußert worden waren (etwa eine Darstellung in Tabellenform mit der Möglichkeit von Spaltenverschiebungen durch den Anwender), fiel die Entscheidung gegen eine Umsetzung in Forms und für den Einsatz von Java.

Aufbauend auf dem Fachkonzept wurde daraufhin zunächst ein Java-Rich-Client-Prototyp implementiert, um die Akzeptanz hinsichtlich der Anwenderergonomie frühzeitig sicherzustellen. In der Realisierungsphase wurden neben den beiden Forms-Registerkarten zur Pflege der Systemtechnik- beziehungsweise Sektordaten auch noch sechzehn weitere Forms-Masken und -Dialoge in eine Java-Rich-Client-Anwendung migriert beziehungsweise neu erstellt.

Technische Umsetzung

Die Umsetzung erfolgte als Java-Rich-Client-Anwendung mit zwei physikalischen (Java-Client und Datenbank) und vier logischen Schichten (Präsentationsschicht, Anwendungslogik, Datenzugriffsschicht und Datenhaltungsschicht). Die Präsentationsschicht wurde unter Einsatz von Java Swing und dem JUNE-Framework von OPITZ CONSULTING realisiert. JUNE ist ein Rich-Client-Framework auf Basis des OSGi-Standards mit dem Ziel, die Entwicklung von erweiterbaren Rich-Client-Anwendungen auf Swing-Basis zu unterstützen. Dabei steht im Vordergrund, die Anwendung auf eine gut strukturierte Basis zu stellen und zu gewährleisten, dass sie auch bei steigendem Projektfortschritt sauber aufgeteilt bleibt. Methodenaufrufe wurden zur Laufzeit mittels JGoodies-Binding und JUNE-Binding umgesetzt.

Die bestehende Anwendungslogik, die in PL/SQL-Prozeduren und -Funktionen auf der Datenbank lag, wurde 1:1 durch den Java-Client genutzt, um das Risiko zu minimieren, dass in die komplexe bestehende Funktionalität Fehler programmiert werden. Die in den Forms-Masken enthaltene Anwendungslogik wurde in die Datenbank ausgelagert oder in Java reimplementiert. Hierzu kam ein Java-Objektmodell auf Bean-Basis zum Einsatz. Neue Logik wurde direkt im Java-Client implementiert. Als Persistenz-Framework diente Hibernate. Die Datenhaltung wurde über eine Oracle-9*i*-Datenbank abgewickelt, die mittlerweile auf 11*g* migriert worden ist.

Die Kommunikation zwischen der Forms-Anwendung und dem Java-Rich-Client erfolgte über eine bidirektionale Software-Schnittstelle zur Interprozess-Kommunikation (Socket). Die Synchronisierung wurde über TCP/IP-Connects erreicht, wobei man Strings als Messages über Reader und Writer ausgetauscht hat.

Als Netzwerkprotokoll kam User Datagram Protocol (UDP) zum Einsatz. Damit konnte zu jeder Zeit genau ein Port der Forms-Anwendung mit genau einem Port des Java-Clients kommunizieren.

Herausforderungen

Die größte Herausforderung stellte die unterschiedliche Transaktionsabwicklung zwischen Forms und Java dar: Aus

den alten Forms-Masken heraus wurden viele Commits abgesetzt. Da das Transaktionsmanagement der Java-Rich-Client-Anwendung im Normalfall auf der Service-Ebene durch das Spring-Framework übernommen wurde, konnte dieses Verhalten aus zwei Gründen nicht problemlos übertragen werden:

- Es gab die Anforderung, die zu bearbeitenden Daten direkt in der Datenbank zu sperren („Pessimistic Locking“). Dies setzte voraus, dass während der gesamten Bearbeitungszeit eine Transaktion offen gehalten wurde.
- Es wurden viele PL/SQL-Prozeduren und -Funktionen aufgerufen, die erst nach dem Absetzen der SQL-Statements die Änderungen mitbekamen, die der Benutzer in der Oberfläche gemacht hat.

Die Lösung

Gelöst wurden diese Probleme mit einer logischen, Client-seitigen Transaktion innerhalb des Frameworks, einer Synchronisierung der Client- mit der Datenbank-Transaktion und den Einstellungen der unterschiedlichen Masken und Dialoge für die Transaktionsgrenzen (es wird für jede Maske, die innerhalb des Java-Clients geöffnet wird, eine neue Transaktion geöffnet, die mit dem Schließen der Maske wieder beendet wird). Ein weiteres Problem war das Mapping auf vorhandene Datenbank-Strukturen: In einigen Fällen erwies sich das Mapping der Tabellen auf Java-Klassen nicht als optimal.

Eine besondere Herausforderung war das generische Mappen der „1:n“-Assoziation einer Spalte auf unterschiedliche Tabellen. Gelöst werden konnte dies nur über eine Datenbank-View mit künstlich berechneten IDs, was anschließend aber zu Problemen mit anderen Prozeduren führte, die die ursprüngliche ID benötigten. Hier musste die ID an einigen Stellen umgerechnet werden. Des Weiteren wurde in Teilbereichen keine optimale Performance erreicht: Durch die große Anzahl der anzuzeigenden Daten musste sehr viel Wert auf das optimierte Laden dieser Daten gelegt werden.

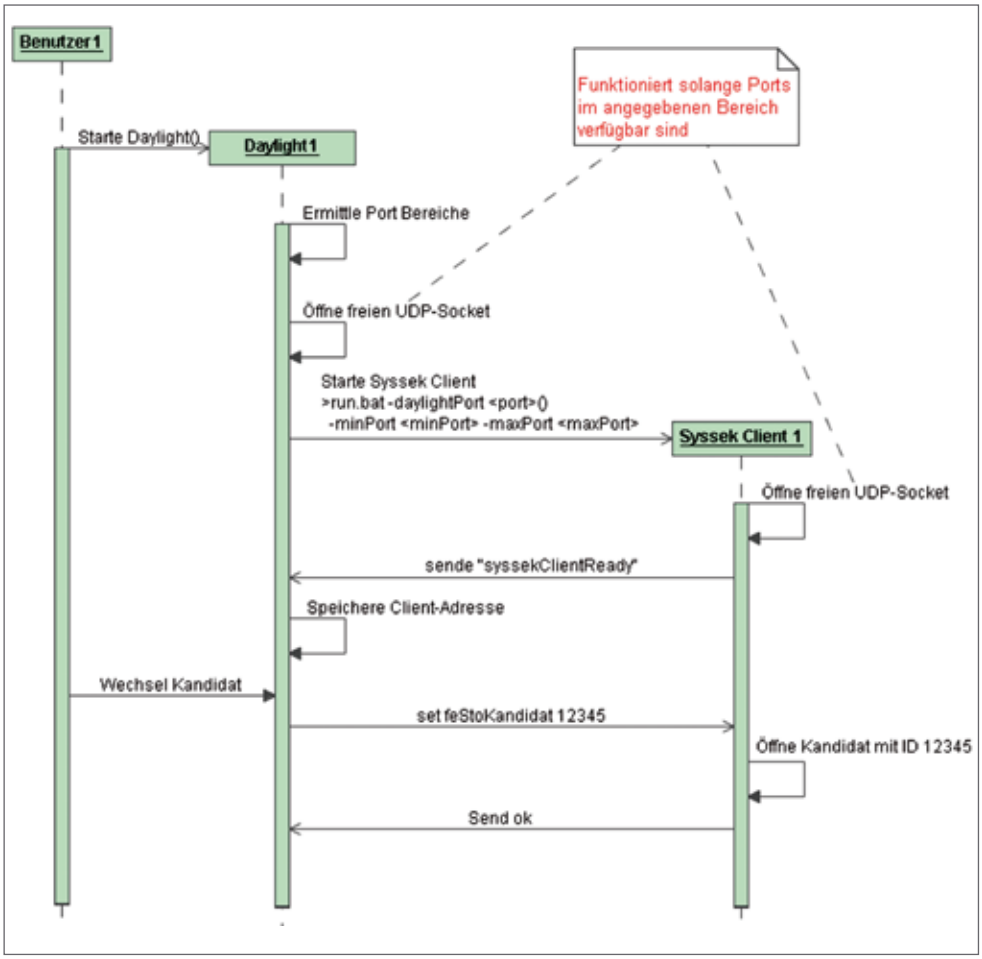

Abbildung 3: Schematischer Ablauf des Kommunikationsaufbaus zwischen der Forms-Anwendung („Daylight“) und dem Java-Client („SysSek-Client“)

Fazit

Die neue Java-Rich-Client-Anwendung fand auf Kundenseite hohe Akzeptanz. Der neue Client lehnt sich an das „Look and Feel“ an, wie es der Anwender durch Windows-Programme gewohnt ist. Die Bearbeitung der System- und Sektordaten wurde durch die Zusammenfassung in einer einzigen Maske deutlich vereinfacht und war durch die Einführung zusätzlicher Plausibilitätsprüfungen zwischen den beiden Bereichen auch weniger fehleranfällig. Aufgrund der hohen Komplexität der Anwendungslogik war eine ausführliche Testphase erforderlich. Es hat sich gezeigt, dass vor Beginn der Migration eine genaue Analyse des Datenmodells, das der Forms-Anwendung zugrunde liegt, notwendig ist und dass sich Komplexität in Forms sehr gut „verstecken“ lässt. Deshalb ist eine wichtige Voraussetzung für das Gelingen eines solchen Migrationsprojekts, dass die beteiligten Java-Entwickler auch über gute PL/SQL-Kenntnisse verfügen. Außerdem stellt das Transaktionshandling zwischen Forms und Java eine nicht zu unterschätzende Herausforderung dar.

Alexander Joedt
OPITZ CONSULTING GmbH
alexander.joedt@
opitz-consulting.com