Sichern der Zukunftsfähigkeit bei der Migration von Legacy-Systemen durch modellgetriebene Softwareentwicklung

Marvin Grieger, Baris Güldali, Stefan Sauer
Universität Paderborn, s-lab – Software Quality Lab, Zukunftsmeile 1, 33102 Paderborn
{mgrieger, bguldali, sauer}@s-lab.upb.de

1 Modernisierung von Legacy-Systemen

Softwareunternehmen stehen zunehmend vor der Herausforderung, große betriebliche Informationssysteme, die mit Softwaretechnologien wie den 4. Generationssprachen (4GL) und -werkzeugen aus den 90er Jahren entwickelt wurden, durch moderne Systeme mit mehrschichtigen Architekturen abzulösen, die auf fortgeschrittenen Softwaretechnologien aufsetzen. Für sie gibt es leistungsfähige Entwicklungsplattformen, und die neuen Technologien und Architekturen erlauben es, die Systeme einfacher zu warten und sie leichter an sich ändernde Anforderungen anzupassen. Das erhöht die Zukunftsfähigkeit der neuen Systeme. Gleichzeitig sind die Softwareunternehmen bestrebt, die mit viel Aufwand entwickelte Funktionalität der Legacy-Systeme wieder zu verwenden.

Benötigt werden deshalb Migrationsverfahren für Legacy-Systeme, die das Wissen der Entwicklerteams wiederverwendbar festhalten und die Zukunftsfähigkeit der neuen Systeme über die Migration hinaus sicherstellen. Dabei setzen wir auf Techniken der modellgetriebenen Softwareentwicklung (engl. Model-driven Software Development, MDSD). Modelle werden als formale Beschreibungsmittel verwendet, und aus den Modellen werden automatisiert Software-Artefakte (weitere Modelle oder Programmcode) generiert. Dadurch dokumentieren wir erstens das Wissen über die Systeme mittels der Modelle explizit und stellen es in dieser Form für die weiteren Migrationsaktivitäten zur Verfügung. Zweitens liefern die Modelle die notwendige Abstraktion um die Auswahl der Zieltechnologie flexibel zu halten.

Die Idee der modellgetriebenen Migration wurde bereits von anderen Autoren adressiert [1-3]. Viele dieser Arbeiten beschäftigen sich mit der kurzfristigen strukturellen und architektonischen Migration der Legacy-Systeme und vernachlässigen dabei das während der Migration entstehende Wissen zu erhalten. Mit unserer Forschung möchten wir Verfahren entwickeln, dass dieses Migrationswissen bewahrt und langfristig nutzbar macht.

In einer früheren Arbeit [4] haben wir die Abgrenzung von anderen Ansätzen beschrieben. Hier erläutern wir unseren Migrationsprozess und die behandelten Abstraktionsebenen, um das Migrationswissen zu konservieren.

2 Verwendung von MDSD-Methoden

Ein zentrales Prinzip der MDSD ist die Verwendung formaler Modelle zur automatischen Generierung von lauffähigen Softwaresystemen. Formale Modelle beschreiben dabei einen Aspekt des Softwaresystems vollständig. Wir beabsichtigen, diese formalen Modelle soweit wie möglich automatisch aus dem vorhandenen Legacy-System abzuleiten und sie für die Migration in das neue System zu nutzen. Die Modelle sollen technologieunabhängig sein, wodurch die Wiederverwendbarkeit erhöht wird. Wir erläutern im Folgenden, wie die verschiedenen Modellarten in den Migrationsaktivitäten erstellt bzw. verwendet werden.

2.1 Abstraktionsebenen

Modelle stellen immer eine Abstraktion dar und können somit einer Abstraktionsebene zugeordnet werden. Dies macht die Komplexität, die einem Softwaresystem zu Grunde liegt, beherrschbar. Gleichzeitig werden verschiedene Sichten auf das System ermöglicht.

Die Abstraktionsebenen sind in der Abb. 1 dargestellt, entsprechend dem MDA-Ansatz der OMG. Auf der untersten Ebene befindet sich die Systemebene, welche die Implementierung einer Applikation bzgl. einer Plattform sowie die zugehörige Laufzeitumgebung beinhaltet. Ausgehend davon leiten wir unter Verwendung von Referenzmodellen plattformspezifische Modelle (Platform Specific Model, PSM) ab. Sie ermöglichen eine Dekomposition des Systems in Komponenten, aus denen wir wiederum plattformunabhängige Modelle (Platform Independent Model, PIM) ableiten. Diese Modelle beschreiben die einzelnen Komponenten formal und technologieunabhängig.

Neben den formalen Modellen werden zusätzlich fachliche Modelle benötigt, um die Refaktorierung bestehender Nutzungsprozesse zu ermöglichen. Diese können wir jedoch nicht aus den Softwaresystemen ableiten, da sie zwar die Fachlichkeit umsetzen, aber in den betrachteten Systemen die Nutzungsprozesse nicht explizit zu erkennen sind. Aus diesem Grund erstellen wir sie manuell, in Form von fachlichen Modellen bzw. Geschäftsmodellen (Computation Independent Model, CIM).

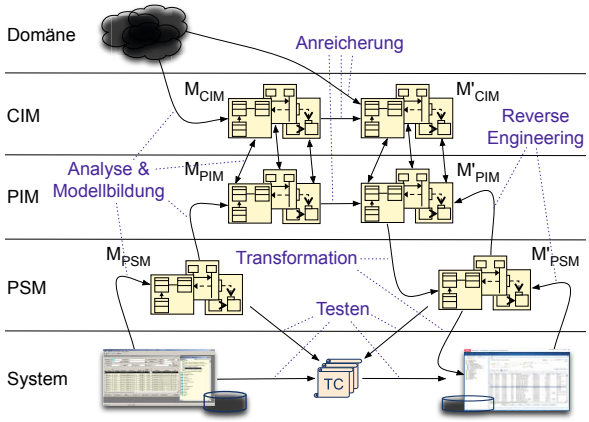

Abb. 1. Migrationsebenen und -aktivitäten

2.2 Migrationsaktivitäten

Unser Ansatz enthält die folgenden Aktivitäten, um die Abstraktionsebenen des Legacy-Systems zu erfassen und sie schrittweise in das Zielsystem zu überführen: Analyse & Modellbildung, Anreicherung, Transformation, Roundtrip-Engineering und Test.

2.2.1 Analyse und Modellbildung

In der Analysephase werden das Legacy-System sowie die zu Grunde liegende Fachlichkeit in Form von Modellen erfasst. Die Modellierung der Fachlichkeit erfolgt manuell, während wir die formalen Modelle des Legacy-Systems automatisch ableiten. Dazu greifen wir auf statische und dynamische Analysemethoden zurück [1]. Statische Methoden ermöglichen uns, Strukturen und Abhängigkeiten im Programmcode und den Architekturebenen zu erkennen. Dabei setzen wir Code-Parser zur Erstellung von Syntaxbäumen und Data-Mining-Werkzeuge zur Klassifikation von Programmcode-Teilen ein. Dynamische Methoden dienen der Erfassung von Abläufen. Ein Ansatz ist die Verwendung der Aspektorientierten Programmierung zur Erstellung von auswertbaren Logdateien.

Der Schwerpunkt in der Analysephase ist der Aufbau einer Werkzeugkette für eine automatische Extraktion der Modelle. Sofern vorhanden sollen dazu existierende Werkzeuge verwendet werden. Beispielsweise existieren Lösungen um strukturelle Informationen automatisch plattformunabhängig zu extrahieren [5]. Die allgemeine Extraktion von Informationen über Zusammenhänge oder Abläufe wird insbesondere dadurch erschwert, dass die Plattform durch die zugehörige Laufzeitumgebung Funktionalität bereitstellt, deren Semantik nicht formal beschrieben ist. Dieses Problem möchten wir lösen, indem wir Analyseverfahren definieren, die Informationen über die Plattform sowie das Entwicklungsprofil, das der Implementierung zu Grunde liegt, einbeziehen. Dieses Profil umfasst u.a. Programmierrichtlinien oder anderweitige Konventionen.

Alle extrahierten Informationen werden in einem zentralen Repository gespeichert und miteinander verknüpft. Dieser Ansatz resultiert in angereicherten formalen Modellen und sichert somit eine hohe Wiederverwendbarkeit.

Eine weitere Herausforderung besteht in der Verknüpfung der extrahierten formalen und manuell erstellten fachlichen Modelle. Erst dadurch wird eine fachliche Anreicherung möglich. Wir beabsichtigen den Einsatz eines Werkzeuges, das eine Abbildung der fachlichen auf die formalen Modelle erzeugt, sobald der fachliche Prozess durchlaufen wird. Solch ein Werkzeug wurde z.B. im Projekt SOAMIG entwickelt [3].

2.2.2 Weitere Aktivitäten

Um die Ergebnismodelle der Analyse in das Zielsystem zu überführen, verwenden wir Modelltransformation und Code-Generierung. Dabei wird das M'_{PIM}, das durch die (interaktive) Anreicherung des M_{PIM} entsteht, zu M'_{PSM} transformiert, das die technologischen Eigenschaften der Zieltechnologie enthält. Aus dem M'_{PSM} werden mittels Code-Generatoren syntaktisch vollständige Codeteile erstellt. Der generierte Code wird manuell vervollständigt. Um die Konsistenz zwischen Code und Modellen zu erhalten, verwenden wir Roundtrip-Engineering-Ansätze, die die Änderungen im Code durch statische Analyse und Mustererkennung finden und die Modelle synchronisieren (vgl. www.fujaba.de).

Um die funktionale Korrektheit nach der Migration zu prüfen, testen wir das neue System mittels Testfällen (*TC*), die auf Basis des Legacy-Systems erstellt wurden. Aus den plattformspezifischen Modellen erstellen wir Testfälle durch modellbasierte Testverfahren. Die Laufzeit-Informationen in den Logdateien dienen als Quelle für konkrete Testdaten. Damit agiert das Legacy-System als Testorakel im Testprozess.

Literatur

[1] A. van Hoorn et al.: DynaMod project: Dynamic analysis for model-driven software modernization. In Proc. MDSM 2011, vol. 708 of CEUR Workshop Proc., pp. 12-13, 2011

[2] S. Efftinge et al.: Einsatz domänenspezifischer Sprachen zur Migration von Datenbankanwendungen. In Proc. BTW 2011, vol. 180 of LNI, GI 2011

[3] Andreas Fuhr, Tassilo Horn, Volker Riediger: Dynamic Analysis for Model Integration, In Softwaretechnik Trends, Band 30, Heft 2, Mai 2010

[4] B. Güldali, S. Sauer, P. Löhr: Entwicklung eines Softwarewerkzeugs für die modellgetriebene Migration betrieblicher Informationssysteme. In Proc. MMSM 2012 (to be published in Softwaretechnik-Trends)

[5] O. S. Ramón et al.: Model-Driven Reverse Engineering of Legacy Graphical User Interfaces. In Proc. ASE 2010, pp. 147-150