Inkrementelle Transformation einer monolithischen Geschäftssoftware

Sascha Hunold*, Matthias Korch*, Björn Krellner‡,
Thomas Rauber*, Thomas Reichel‡, Gudula Rünger‡

* *Fakultät für Mathematik, Physik und Informatik, Universität Bayreuth*
{hunold,matthias.korch,rauber}@uni-bayreuth.de

‡ *Fakultät für Informatik, Technische Universität Chemnitz*
{bjoern.krellner,thomas.reichel,ruenger}@informatik.tu-chemnitz.de

Abstract: Pflege und Wartung von Altsoftware (*legacy software*) sowie die Erweiterung und Anpassung dieser Applikationen an neue Märkte machen einen immensen Kostenanteil von vielen Softwareunternehmen aus. Die manuelle Migration von Software ist durch zunehmend kürzer werdende Abstände der Entwicklungsstadien (release cycles) nur durch immensen personellen Aufwand zu leisten. Die Entwicklung von Werkzeugen und Verfahren zur Unterstützung dieses Migrationsprozesses ist deshalb für die Unternehmen von entscheidender Bedeutung. In diesem Artikel erarbeiten wir einen inkrementellen Transformationsprozess für Geschäftssoftware, der unter Zuhilfenahme von Verfahren des Compilerbaus und des Software-Engineerings eine Kette von Transformationsschritten definiert, mit deren Hilfe eine monolithisch strukturierte Altsoftware in eine modulare Client-Server-Applikation überführt werden kann.

1 Einleitung

Softwaresysteme zur Realisierung von Geschäftsprozessen werden in vielen Unternehmen eingesetzt und tragen oft entscheidend zum Erfolg des Unternehmens bei. Die schnelle Entwicklung der Hardware- und Software-Technologien, aber auch Veränderungen innerhalb von Unternehmen, z. B. die Anpassung an neue Geschäftsfelder, erfordern jedoch eine stetige Wartung, Anpassung und Erweiterung der Softwaresysteme. Ist die Pflege einer Software mit vertretbarem Aufwand nicht mehr möglich, spricht man von *Altsoftware* [Ulr02].

An eine moderne Geschäftssoftware werden Anforderungen gestellt, die eine solche monolithische Architektur nicht erfüllen kann. Diese sollte auf einer Vielzahl von Hardware- und Software-Plattformen einsetzbar sein. Sie sollte modular aufgebaut und flexibel konfigurierbar sein, um an spezifische Kundenanforderungen und neue Geschäftsstrategien angepasst werden zu können. Eine schichtenbasierte, in Objekte und Module gegliederte Softwarearchitektur ermöglicht darüber hinaus den Austausch von Programmteilen und erleichtert somit eine spätere Migration zu anderen Plattformen und Technologien. Die Anwender sollten unterschiedliche Thin- und Rich-Clients nutzen können; insbesondere

sollte die Nutzung einer Web-Schnittstelle möglich sein. Die Geschäftslogik sollte von den Clients abgekoppelt und in Form von verteilten Diensten über ein Netzwerk nutzbar sein.

Der Herausforderung der Umstrukturierung ihrer Software stellen sich viele Firmen, die ihre Softwareprodukte Ende der 80er oder Anfang der 90er Jahre im Hinblick auf den damaligen Standard der Modularität und Flexibilität entworfen haben, aber die den heutigen technologischen Anforderungen nicht mehr genügen [CMER+06]. Zu einem großen Teil wurden diese Anwendungen in einer objektorientierten Programmiersprache implementiert. Zwar besitzen sie eine interne Aufteilung in Unterprogramme und Klassen sowie Gruppierungen dieser Programmelementen zu Quelltextpaketen, es fehlt jedoch eine klare Aufteilung in Schichten und eigenständige Module. Vielmehr sind die Programmelemente so eng miteinander verwoben, dass man von einer monolithischen Struktur spricht. Oftmals sind diese Business-Applikationen dadurch gekennzeichnet, dass mehrere Benutzer über ihre grafische Benutzerschnittstelle (engl.: *graphical user interface*, GUI) Daten auf einem gemeinsamen Datenbankserver lesen oder modifizieren.

Das Ziel des Projekts TransBS[1] ist es, erprobte monolithische Unternehmenssoftware zu modularisieren und diese Module in einen verteilten Programmkontext einzubetten. Im Rahmen des Projekts betrachten wir vor allem Geschäftssoftware, die in der Programmiersprache Delphi implementiert wurde. Das hat die Konsequenz, dass sich der logische Aufbau der Software stark am Rapid-Application-Development-(RAD)-Ansatz von Borland Delphi orientiert. Da eine Softwaretransformation facettenreich auf verschiedenen Abstraktionsschichten arbeitet (Codeebene, Modellebene) [KWC98] wird innerhalb des Projekts versucht, verwandte Verfahren und Werkzeuge in den einzelnen Teilschritten wiederzuverwenden. Der inkrementelle Transformationsprozess gliedert sich in drei Schritte: (1) Generierung des Softwaremodells aus der Ursprungssoftware, (2) Modifizierung des Modells mittels Model-Driven-Architecture-(MDA)-Ansätzen [VS06] und (3) Erzeugen der Zielsoftware.

2 Inkrementeller Transformationsprozess

Wie bereits beschrieben, ist die Transformation von Geschäftssoftware eine vielschichtige Problemstellung, für die sich folgende wichtige Aspekte ergeben:

- Trennung von Nutzerschnittstelle, Geschäftslogik und Daten (Model-View-Controller-(MVC)- oder Presentation-Abstraction-Control-(PAC)-Architektur),
- Identifikation von Geschäftsobjekten und Geschäftsprozessen der Geschäftslogik,
- Modularisierung und Umsetzung in Dienste und Workflows (explizite Modellierung der Geschäftsprozesse auf der Basis von Workflows),
- Wiederverwendung des Codes der Altsoftware und Integration in das Zielsystem (z. B. zur Anbindung unterschiedlicher Thin- und Rich-Clients),
- Validierung der Zielsoftware und
- Nutzung von Metainformationen über das Altsystem (Laufzeitverhalten, fehleranfällige Funktionen).

[1] `http://www.transbs.de`

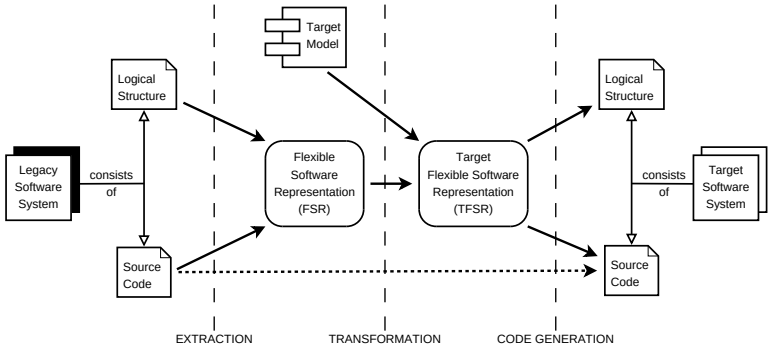

Abbildung 1: Ablauf der Transformation der Altsoftware mit `TransFormr`.

Zur Lösung der genannten Problemstellungen schlagen wir einen inkrementellen Transformationsprozess für die Transformation eines Altsystems in ein verteiltes, workflowbasiertes System vor. Unser Transformationssystem bietet ein Toolset zur interaktiven Unterstützung des Reengineeringprozesses. Wie in Abbildung 1 dargestellt, werden im ersten Schritt (Extraktion) des Prozesses Informationen über Quellcode und Struktur des Altsystems extrahiert und analysiert. Das Ergebnis dieses Schritts ist die Erstellung der flexiblen Sofwarerepräsentation (FSR) [RR07]. Die FSR ist eine mit Meta-Informationen angereicherte, sprachunabhängige Darstellung des Sourcecodes der Altsoftware. Sprachunabhängige Transformationen der Applikation können im nächsten Schritt (Transformation) auf die FSR angewendet werden. Im abschließenden Schritt (Generierung) wird aus dem transformierten Softwaremodell der Zielcode des Systems erzeugt.

Im Analyseschritt (Extraktion) können verschiedene Verfahren zum Reverse-Engineering zum Einsatz kommen [MJS+00], z. B. das Identifizieren von Geschäftsobjekten durch das Verfolgen des Datenwegs im Programm bis hin zu Formularfeldern. Ein wichtiges Ziel ist die Extraktion des Workflows, wobei auch die Identifikation von Business-Logik und die Gewinnung der Softwarearchitektur eine wesentliche Rolle spielt [ZH06]. Die Korrektheit der Transformation ist für eine erfolgreiche Migration von großer Bedeutung. Da die FSR eine Baumdarstellung der Software enthält, müssen u. a. Verfahren zur Validierung der Graphtransformationen berücksichtigt werden [GGZ+05].

Im Rahmen unseres Forschungsprojektes wird das Werkzeug `TransFormr` entwickelt, mit dem die Transformation einer monolithischen Applikation in den drei definierten Schritten möglich ist. Die folgenden Absätze beschreiben die Implementierungsdetails der einzelnen Schritte.

Extraktion Zur Analyse der Altsoftware nutzen wir eine Grammatik der Ausgangssprache und verschiedene Extraktionsregeln, um mit Hilfe von TXL [CCH05] die FSR zu erzeugen. Die FSR enthält die Struktur des Systems (package, module, etc.) und Beziehungen (Generalisierungen, Realisierungen, Methodenaufrufe, etc.) innerhalb des Systems mit ihren jeweiligen Eigenschaften und Referenzen zum Quellcode. Aus der FSR können UML-Klassendiagramme, Paketabhängigkeitsdiagramme und Call-Dependency-Dia-

gramme für die gesamte Software, für Packages oder einzelne Klassen erzeugt werden, die bei der Einschätzung und Bewertung von Transformationen hilfreich sein können.

Transformation Für eine Systemtransformation ist es nötig, ein Zielmodell (target model) zu definieren, welches Metainformationen über die Anforderungen an das Zielsystem und dessen Eigenschaften enthält. Diese Informationen über das Zielsystem müssen vom Nutzer in einzelne Transformationsschritte zerlegt werden, die mit `TransFormr` angewendet werden können. Notwendige Transformationsschritte sind das Verschieben von Funktionalität in separate Subsysteme (oder auch Dienste) und das Erzeugen von Client- und Serverklassen und verteilter Objekte, welche die Funktionalitäten der Kommunikationspartner kapseln. Zur Durchführung solch komplexer Transformationen definieren wir eine Reihe von Basistransformationen (Refactorings, z. B. `rename`, `move`, `split`) aus denen sich ein vollständiger Reengineeringprozess zusammensetzt. Die Transformationen der FSR und ihre Abhängigkeiten können mit `TransFormr` visualisiert werden. Eine Auswahl der aktuell verfügbaren Refactorings innerhalb der Transformationskomponente von `TransFormr` ist in Abbildung 2 dargestellt.

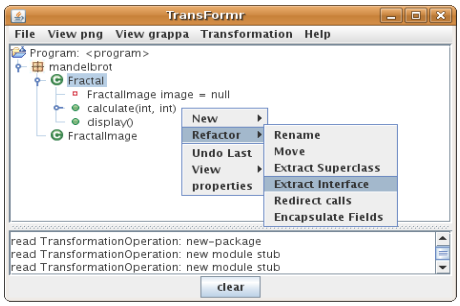

Abbildung 2: Refactoring mit `TransFormr`

Generierung Die Ziel-FSR ist durch eine Reihe von Transformationen der FSR definiert. Der Zielcode kann durch die Anwendung der Transformationen auf Codeebene erzeugt werden, wobei Teile des Originalsourcecodes wiederverwendet werden können. Die aktuelle Version von `TransFormr` unterstützt die Erzeugung von Quellcode der Zielsoftware in derselben Programmiersprache wie die Altsoftware. Für die definierten Basistransformationen sind sprachabhängige Regeln definiert, die diese umsetzen. Der erzeugte Quellcode kann entsprechend in Client und Server oder andere definierte Subsysteme/Dienste aufgeteilt sein.

3 Konfigurationsvarianten zur Generierung des Zielsystems

Die umfangreichen Steuerungsmöglichkeiten der Schritte des inkrementellen Transformationsprozesses ermöglichen eine flexible Konfiguration des Zielsystems. Das Zielsystem kann sich aus Subsystemen der Altsoftware, neu erzeugtem Programmcode, abstrakten, modellbasierten Beschreibungen von Teilen der Software (z. B. explizite Workflows oder Teile der Nutzerschnittstelle) sowie einem statischen Framework, das die Ausführung der übrigen Komponenten durch die Bereitstellung einer Laufzeitumgebung unterstützt, zusammensetzen. Eine mögliche Zielarchitektur wäre die Verwendung eines Java-Enterprise-

Servers[2] (Java EE). Datenbankobjekte müssten dafür von `TransFormr` in Entity-Beans transformiert und erforderliche Schnittstellen können dann als Web-Services für andere Systeme zugänglich gemacht werden.

Mit entsprechenden Transformationsregeln ist es auch möglich, Code zur Anbindung verschiedener Datenbanksysteme sowie die Bereitstellung und Anbindung von Diensten über unterschiedliche Middleware-Lösungen und Protokolle (z. B. CORBA, RMI, SOAP) zu generieren, wobei auch spezifische Erweiterungen des Applikationsservers berücksichtigt werden können. Für die Integration von Subsystemen der Altsoftware sind Wrapping-Ansätze geeignet, die z. B. das Subsystem in einem verteilten Objekt kapseln und auf diese Weise dessen Funktionalität im Netzwerk zur Verfügung stellen. Außerdem lassen sich damit Codeblöcke aus der Legacy-Anwendung in die neue Software-Plattform integrieren, auch ohne eine komplette Sprachmigration. Muss neuer Programmcode (z. B. aus modellbasierten Zwischendarstellungen) erzeugt werden, können mit spezifischen Codegeneratoren verschiedene Programmiersprachen unterstützt werden. Ein Ändern der Programmiersprache kann eine besseren Plattformunabhängigkeit oder Wartbarkeit ermöglichen.

Oft ist es notwendig, dass verschiedene Teile der Alt- und Neusoftware koexistieren müssen. Als Kommunikationsprotokolle zur Vermittlung zwischen Subsystemen der Altsoftware und neu erzeugten Teilen des Zielsystems (auch in anderer Programmiersprache) kommen u. a. CORBA und SOAP in Frage. Der CORBA-Standard beinhaltet jedoch für einige relevante Programmiersprachen, u. a. auch Delphi, keine standardisierte Abbildung der Schnittstellenbeschreibungssprache (IDL). Die aktuellen Delphi-Entwicklungsumgebungen bieten aber eine umfangreiche Unterstützung für SOAP. Aufgrund des einfachen, XML-basierten Protokolls können SOAP-Implementierungen bei Bedarf auch für bisher nicht unterstützte Programmiersprachen mit überschaubarem Aufwand erstellt werden.

Für die Realisierung von Client-Anwendungen können durch das Transformationssystem unterschiedliche GUI-Frameworks unterstützt werden (z. B. Delphi VCL, Java Swing). Rich-Clients vereinen Präsentationsschicht und Steuerungslogik innerhalb eines einzelnen Anwendungsprogramms; Netzwerkkommunikation führen sie nur zum Zugriff auf Dienste der Geschäftsschicht durch. Thin-Clients, wie z. B. Web-Anwendungen, erfordern Netzwerkkommunikation auch zwischen Präsentationsschicht und Controller. Speziell für die Realisierung von Web-Anwendungen eignen sich hierfür die unter dem Begriff „AJAX“ (*Asynchronous JavaScript and XML*) zusammengefassten Technologien und Protokolle.

Die Verwendung expliziter Workflows im Transformationsergebnis kann oftmals viele Vorteile mit sich bringen. Zum einen müssen Geschäftsprozesse explizit definiert werden, und zum anderen wird durch den Einsatz einer Workflow-Engine (z. B. jBPM, Enhydra Shark, WfmOpen) die Veränderung und das Hinzufügen einer Prozesskette deutlich erleichtert. Auch die Reduzierung der möglichen Fehlerquellen im Programmcode ist ein nicht zu unterschätzender Vorteil. Als Beschreibungssprache für Geschäftsprozesse bietet sich die von der Workflow Management Coalition (WfMC) standardisierte Sprache XPDL an. Zusammengesetzte Dienste, die sich mit Hilfe eines Workflows modellieren lassen, können mit einer Workflow-Beschreibungssprache (im Fall von Web-Diensten z. B. BPEL) beschrieben und in einer Workflow-Engine (z. B. jBPM) ausgeführt werden.

[2] `http://java.sun.com/javaee/`

4 Zusammenfassung

In dieser Arbeit wurde eine Methode zur Transformation monolithischer Geschäftssoftwaresysteme in eine modulare Softwarearchitektur entwickelt. Es ist eines der Ziele, dass die entstandenen Module in einen verteilten Kontext eingebettet werden können. Der zugehörige Transformationsprozess arbeitet in drei Schritten: Extraktion, Transformation und Generierung. Eine Anpassung der Schnittstellen unseres Transformationsvorschlags an die ADM-Standardspezifikation [ADM07] der OMG wird zurzeit diskutiert. Die Anforderungen des Unternehmens und dessen Kunden sowie die Eigenschaften der Altsoftware legen die Zielarchitektur fest. Aus diesem Grund unterstützt das Transformationssystem verschiedene Konfigurationsmöglichkeiten zur Auswahl der Zielarchitektur (Workflow-Extraktion, Abbildung in verteilte Komponenten, Code-Restrukturierung in MVC-Architektur). Exemplarisch werden wir im Verlauf des Projekts verschiedene Konfigurationen für eine gegebene monolithische Delphi-Anwendung hinsichtlich ihrer Anwendbarkeit evaluieren und implementieren.

Literatur

[ADM07] ADM task force. Architecture-Driven Modernization: Transforming the Enterprise. http://adm.omg.org, 2007.

[CCH05] James R. Cordy, Ian H. Carmichael und Russell Halliday. The TXL Programming Language, Version 10.4, January 2005. `http://www.txl.ca/docs/TXL104ProgLang.pdf`.

[CMER+06] R. Correia, C. Matos, M. El-Ramly, R. Heckel, G. Koutsoukos und L. Andrade. Software Reengineering at the Architectural Level: Transformation of Legacy Systems. Bericht, Department of Computer Science, University of Leicester, U.K., 2006.

[GGZ+05] Lars Grunske, Leif Geiger, Albert Zündorf, Niels Van Eetvelde, Pieter Van Gorp und Daniel Varro. Using Graph Transformation for Practical Model Driven Software Engineering. In Sami Beydeda, Matthias Book und Volker Gruhn, Hrsg., *Model-driven Software Development*, Seiten 91–118. Springer, 2005.

[KWC98] R. Kazman, S. G. Woods und S. J. Carrière. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. In *Proceedings of WCRE 98, (Honolulu, HI)*, Oktober 1998.

[MJS+00] Hausi A. Müller, Jens H. Jahnke, Dennis B. Smith, Margaret-Anne Storey, Scott R. Tilley und Kenny Wong. Reverse Engineering: A Roadmap. In *Proc. of the Conf. on The Future of Software Engineering (ICSE '00)*, Seiten 47–60. ACM Press, 2000.

[RR07] Thomas Rauber und Gudula Rünger. Transformation of Legacy Business Software into Client-Server Architectures. In *Proc. of the 9th ICEIS*, 2007.

[Ulr02] William M. Ulrich. *Legacy Systems: Transformation Strategies*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[VS06] Markus Völter und Thomas Stahl. *Model-Driven Software Development*. Wiley, 2006.

[ZH06] Ying Zou und Maokeng Hung. An Approach for Extracting Workflows from E-Commerce Applications. In *Proc. of the 14th IEEE Int. Conf. on Program Comprehension (ICPC'06)*, Seiten 127–136. IEEE Computer Society, 2006.