Wiederverwendbarkeit von Migrationswissen durch Techniken der modellgetriebenen Softwareentwicklung

Marvin Grieger, Stefan Sauer

Universität Paderborn
s-lab – Software Quality Lab
Zukunftsmeile 1
33102 Paderborn
mgrieger@s-lab.upb.de
sauer@s-lab.upb.de

Abstract: Softwaresysteme müssen auf Grund sich ändernder Anforderungen fortwährend gepflegt und weiterentwickelt werden. Eine Migration in eine neue Umgebung, die auf modernen Softwaretechnologien basiert, ist oftmals der einzige Ausweg, die neuen Anforderungen vollständig zu erfüllen. Viele Unternehmen versuchen dennoch, eine Migration zu vermeiden, da sie mit einem hohen wirtschaftlichen Investment und Risiko verbunden ist. Dies ist insbesondere bedingt durch die fehlende Werkzeugunterstützung in der Durchführung der Migration. In diesem Beitrag beschreiben wir diesbezüglich ein konkretes Fallbeispiel aus der Praxis. Darauf aufbauend entwickeln wir einen Migrationsprozess, der durch die Nutzung von Semi-Automatismen basierend auf der Formalisierung und Wiederverwendung des in der Migration entstehenden Wissens in Form von Modellen und Transformationen gekennzeichnet ist.

1 Einleitung und Motivation

Softwaresysteme altern über die Zeit. Dieser Alterungsprozess ist dadurch gekennzeichnet, dass sich die Kluft zwischen den Anforderungen an die Systeme und deren tatsächliche Leistungsfähigkeit immer weiter vergrößert [SWH10].Oftmals erfolgt eine Erhaltung und Weiterentwicklung der Systeme, um dem Alterungsprozess entgegenzuwirken. Über die Zeit geht dies mit einer verringerten Qualität der Software und einer steigenden Wartungskomplexität einher. Der Umfang der Änderungsmöglichkeiten durch eine Weiterentwicklung ist jedoch begrenzt, gleichzeitig können Einschränkungen der zu Grunde liegenden Technologie dazu führen, dass es nicht möglich ist, alle Anforderungen umzusetzen. Ein pragmatischer Ausweg ist laut Sneed die Migration des Systems in eine neue Umgebung [SWH10].
Die Motivation, eine Migration anstelle einer Neuentwicklung durchzuführen, ist die Erhaltung des in die Software eingeflossenen Wissens und so auch der Schutz der in das System getätigten Investitionen. Eine manuelle Überführung dieses Wissens birgt jedoch erhebliche Risiken bedingt durch die Größe und Komplexität der Softwaresysteme. Durch die Entwicklung von Werkzeugen und Techniken, die diesen Schritt unterstützen,

können Kosten gespart und Fehler minimiert werden. Die Entwicklung solcher Werkzeuge und Techniken untersuchen wir innerhalb eines Forschungsprojekts.
In diesem Beitrag gehen wir zunächst im Detail auf ein Fallbeispiel aus der Praxis ein. Darin beschreiben wir die Herausforderungen in der Migration und leiten Anforderungen an eine Unterstützung ab. Das Fallbeispiel nutzen wir im Rahmen des Forschungsprojekts als Ausgangspunkt für die Entwicklung des Migrationsprozesses und zur Evaluierung unseres Ansatzes. Anschließend stellen wir den von uns verfolgten Ansatz vor und beschreiben verwandte Arbeiten.

2 Fallbeispiel

Im Rahmen eines Forschungsprojekts kooperieren wir mit einem Industriepartner, der sich auf die Entwicklung und den Vertrieb von datenbankbasierten Anwendungen in der Logistik-Domäne spezialisiert hat. Dabei untersuchen wir die Migration von Applikationen, die für die Plattform Oracle Forms 6i entwickelt wurden. Lange Zeit herrschte Unsicherheit darüber, inwiefern die Plattform auch in Zukunft weiterentwickelt werden würde, da der Hersteller verstärkt die Entwicklung alternativer Produkte in den Vordergrund stellte. Anfang des Jahres wurde diesbezüglich eine Stellungnahme der Firma Oracle veröffentlicht, in der die langfristigen Strategien für die vorhandenen Plattformen offengelegt wurden [Or12]. In dieser wird bestätigt, dass auch in Zukunft eine Weiterentwicklung der Forms-Plattform auf Grund der großen Installationsbasis stattfinden wird. Gleichzeitig werden Empfehlungen abgegeben, welche Plattformen sich als potenzielle Zielplattform für eine Migration eignen. Dies umfasst unter anderem die Plattform Oracle ADF, welche wir als Zielplattform betrachten. Dabei wird auch klargestellt, dass seitens der Firma Oracle keine Werkzeugunterstützung für eine solche Migration bereitgestellt wird. Begründet wird dies mit den Unterschieden zwischen Quell- und Zielplattform. Die wichtigsten Unterschiede haben wir nachfolgend zusammengefasst:

1. *Programmiersprache*
   Die Programmiersprache der Quellplattform Oracle Forms ist PL/SQL, welche proprietär und prozedural aufgebaut ist. Die Programmiersprache der Zielplattform Oracle ADF hingegen ist die objektorientierte Programmiersprache Java.

2. *Architektur*
   Während Applikationen der Quellplattform monolithisch aufgebaut sind, werden Anwendungen in der Zielplattform nach dem Model-View-Controller-Architekturmuster strukturiert. Folglich muss eine Dekomposition der Quellapplikation durchgeführt werden.

3. *Paradigma*
   Dem Design der Quell- und Zielplattform liegen unterschiedliche Prinzipien zu Grunde. Beispielsweise wird in der Zielplattform eine Prozessorientierung verfolgt, während Applikationen der Quellplattform datenorientiert entwickelt werden. Ein weiteres Beispiel ist die Funktionalität, die durch eine Datenbank als Komponente innerhalb der Applikation bereitgestellt wird. Während die Datenbank in der Zielplattform lediglich als Datenspeicher genutzt werden soll, stellt sie in der Quellap-

plikation eine zentrale serverseitige Komponente dar, auf der beispielsweise Validierungen durch Prozeduren (sog. Stored Procedures) ausgeführt werden. Solche Paradigmenwechsel erschweren die Überführung auf der technischen Ebene.

Der Bedarf, bestehende Forms-Applikation nach Oracle ADF zu migrieren ist dennoch vorhanden. Dies wird besonders durch die Präsenz des Themas Migration innerhalb der Oracle Community deutlich[1]. Die Gründe für eine Migration sind dabei vielseitig. Viele Unternehmen sind weiterhin bzgl. der langfristigen Entwicklung der Forms-Plattform verunsichert. Zudem besteht der Wunsch, leistungsfähige Entwicklungsumgebungen sowie aktuelle Softwaretechnologien und -architekturen einzusetzen. Dadurch soll die Wartbarkeit der Systeme erhöht und eine schnellere Anpassungsfähigkeit an neue Anforderungen ermöglicht werden.
In diesem Kontext untersuchen wir als Fallbeispiel die Migration einer größeren Forms-Applikation. Sie wurde über mehrere Jahre entwickelt und umfasst ca. vier Millionen Zeilen Code. Um diese umfangreiche Migration zu unterstützen, erarbeiten wir im Rahmen unseres Forschungsprojekts einen werkzeuggestützten Migrationsprozess basierend auf Techniken der modellgetriebenen Softwareentwicklung. Dazu definieren wir zunächst die Anforderungen an diesen Prozess, in den unsere Erfahrungen mit auf dem Markt befindlichen Lösungen eingeflossen sind:

a) *Wiederverwendbarkeit*
Über unseren Industriepartner erhalten wir Zugriff auf mehrere Applikationen der Quellplattform, die von unterschiedlichen Entwicklergruppen entwickelt wurden. Das Wissen, welches während der Migration einer Applikation aufgebaut wird, soll ebenfalls in nachfolgenden Migrationsprojekten genutzt werden können. Dazu muss dieses Wissen explizit erfasst werden, beispielsweise in Form eines Katalogs von Templates bzw. Regeln, der dann kundenspezifisch genutzt, angepasst und erweitert werden kann. Dies ist ein zentraler Aspekt unseres Migrationsprozesses.

b) *Automatismen*
Die Nutzung von Automatismen in der Migration bietet erhebliche Vorteile. So können insbesondere für wiederkehrende Tätigkeiten Kosten gespart und gleichzeitig manuelle Implementierungsfehler ausgeschlossen werden. Dabei müssen jedoch auch die Grenzen der Automatisierbarkeit bekannt sein. Oftmals erzwingen Werkzeuge einen hohen Grad an Automatisierung, der die Qualität des Resultats verringert, da applikationsspezifische Eigenheiten nicht berücksichtigt werden. Dies ist insbesondere im Rahmen einer 4GL-Migration der Fall, d.h. einer Migration ausgehend von einer Umgebung einer Vierten Generationssprache (4GL). Werkzeuge stützen sich in diesem Fall auf die proprietären, vorgegebenen Strukturen einer solchen Applikation und ignorieren applikationsspezifische Erweiterungen. In unserem Migrationsprozess möchten wir aus diesen Gründen eine semi-automatisierte Werkzeugunterstützung bereitstellen, in der insbesondere Flexibilitätspunkte für applikationsspezifische Anpassungen bereitstehen.

[1] http://www.doag.org/

c) *Abstraktion*
Eine Transformation der Applikation auf Basis der reinen syntaktischen Informationen ist an vielen Stellen nicht zielführend. Durch Einschränkungen der Quellplattform mussten zum einen Funktionalitäten der Applikation explizit ausprogrammiert werden, wobei die Freiheitsgrade der Quellplattform ausgenutzt und vorgegebene Strukturen zweckentfremdet wurden. Zum anderen sollen in der Migration die Möglichkeiten der Zielplattform ausgenutzt werden, zum Beispiel in der Überführung der Benutzungsoberfläche: In der Quellplattform sind die Elemente auf der Oberflächen absolut positioniert, während in der Zielplattform eine relative Positionierung angewendet wird. Zusätzlich existieren in der Zielplattform neue Visualisierungsmöglichkeiten der Datensätze, z.B. in Form von Diagrammen. Auf Grund dieser Unterschiede wird eine direkte 1:1-Überführung nicht verfolgt, vielmehr sollen die relevanten Informationen einer Benutzungsoberfläche extrahiert, in einer Zwischenrepräsentation abgebildet und in der Zielplattform umgesetzt werden.

d) *Individuelle Zielarchitektur*
Im Rahmen der Migration muss die Architektur für die Applikation in der Zielplattform (Zielarchitektur) entworfen werden. Diese unterliegt einem Spannungsfeld zwischen der optimalen Zielarchitektur und der vorhandenen Altanwendung [ZGW10], weshalb die Erarbeitung der Zielarchitektur ein manueller Prozess ist, der ein tiefes Verständnis sowohl der Altanwendung (einschließlich der Quellplattform) als auch der Zielplattform voraussetzt. Sobald ein Werkzeug eine vollautomatische Migration anstrebt, wird die Zielarchitektur meistens komplett vorgegeben. Dies ist darin begründet, dass Umsetzungsentscheidungen Seiteneffekte haben können, die andere Umsetzungsmöglichkeiten beeinflussen oder ausschließen. Durch unseren Migrationsprozess möchten wir die Definition einer individuellen Zielarchitektur ermöglichen. Gleichzeitig sollen bewährte Architekturen bzw. Umsetzungsmöglichkeiten wiederverwendet werden können.

e) *Prozessorientierung*
Applikationen der Zielplattform werden ausgehend von den zu Grunde liegenden Geschäftsprozessen entworfen, wodurch auf technischer Ebene eine Strukturierung in wiederverwendbare Arbeitsabläufe und -schritte erfolgt. Das Konzept eines prozessorientierten Ansatzes ist in der Quellplattform nicht vorhanden, die konkreten Abläufe, die auf Grund eines Geschäftsprozesses durchgeführt werden, sind auf technischer Ebene nicht sichtbar. In der Migration muss die Prozessorientierung berücksichtigt werden, um die Zielapplikation entsprechend den Konzepten der Zielplattform sinnvoll zu strukturieren. Dies kann beinhalten, Arbeitsabläufe explizit zu erfassen, um zusätzliche Strukturierungsinformationen bereitzustellen.

Diese Anforderungen werden durch bestehende Werkzeuge und die zugehörigen Migrationsprozesse nicht erfüllt. Deshalb entscheiden sich viele Unternehmen, die den Bedarf haben, ihre Altanwendung zu modernisieren, für eine Weiterentwicklung. Im Rahmen des Forschungsprojekts untersuchen wir, inwiefern Techniken der modellgetriebenen Softwareentwicklung dazu genutzt werden können, die benötigte Flexibilität zur Entwicklung eines individuellen Migrationspfades und gleichzeitig die Werkzeugunterstützung in der Überführung von Altanwendungen bereitzustellen.

3 Ansatz

Techniken der modellgetriebenen Softwareentwicklung wurden bereits in verschiedenen Projekten im Kontext der Softwaremigration eingesetzt und untersucht (vgl. Abschnitt 4). Die meisten dieser Arbeiten adressieren jedoch nur die einmalige Migration eines spezifischen Softwaresystems. Im Rahmen unserer Forschung möchten wir ein Verfahren entwickeln, welches das während der Migration entstandene Wissen explizit festhält. Dies umfasst sowohl das Wissen über den Aufbau der resultierenden Applikation, welches durch Modelle auf verschiedenen Abstraktionsebenen repräsentiert wird, als auch das Wissen über die Zusammenhänge in der Migration, welches für spätere Migrationen von Applikationen derselben Plattform in Form einer Werkzeugunterstützung genutzt werden kann. In einer früheren Arbeit haben wir unseren Ansatz in Form einer artefaktbasierten Sicht auf den Migrationsprozess beschrieben [GGS12]. In diesem Beitrag möchten wir auf Teilaspekte dieses Prozesses im Detail eingehen.

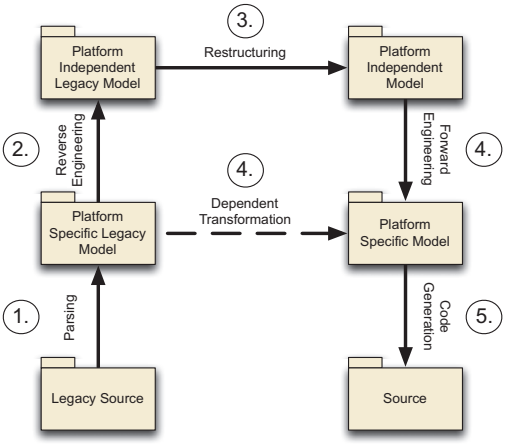

Abbildung 1: Modelle und Transformationen im Migrationsprozess

In Abbildung 1 ist ein Ausschnitt unseres Migrationsprozesses dargestellt, angelehnt an das Reengineering-Hufeisen [KWC98]. Die grundlegende Idee des Ansatzes ist die Nutzung von Modellen unterschiedlicher Abstraktionsebenen, die durch Modelltransformationen ineinander überführt werden. Die Abstraktionsebenen orientieren sich dabei an dem MDA-Ansatz der Object Management Group (OMG). Dabei werden alle Modelle in einem zentralen Repository gespeichert und über Traceability-Informationen der Transformationen miteinander verknüpft. Nachfolgend wird auf die Rolle der verschiedenen Modelle und der Transformationen separat eingegangen:

1. *Modellbildung*
   Zunächst wird die Applikation in Form eines plattformspezifischen Modells (*Platform Specific Legacy Model*) erfasst. Dies umfasst eine verlustfreie Abbildung der Quellartefakte, welche strukturell in kleinste Einheiten zerlegt werden, indem zugehörige Parser die Syntaxbäume der enthaltenen Code-Blöcke berechnen. Schnittstel-

len externer Komponenten, mit denen die Applikation kommuniziert, werden in diesem Schritt ebenfalls erfasst. Zusätzlich können Ergebnisse automatisch durchgeführter dynamischer Analysen einfließen, um beispielsweise Aufrufreihenfolgen oder Ausführungszeiten zu ermitteln. Nachdem die initiale Abbildung der Applikation erfolgt ist, müssen implizit vorhandene Beziehungen zwischen Modellelementen explizit sichtbar gemacht werden. Im Rahmen des Fallbeispiels konnten wir feststellen, dass viele Beziehungen auf Quellcode-Ebene nicht direkt sichtbar waren, sondern erst durch das Wissen über das Laufzeitverhalten der Quellplattform deutlich wurden. Das Modell wird um diese Informationen angereichert. Auf Grund des erhöhten Informationsgehalts des Modells gegenüber den Quellartefakten werden Letztere nicht mehr als Grundlage für weitere Migrationsschritte verwendet.
Die Modellbildung läuft für unterschiedliche Applikationen derselben Plattform identisch ab, so dass eine Routine entwickelt werden kann, die diesen Schritt vollautomatisch durchführt.

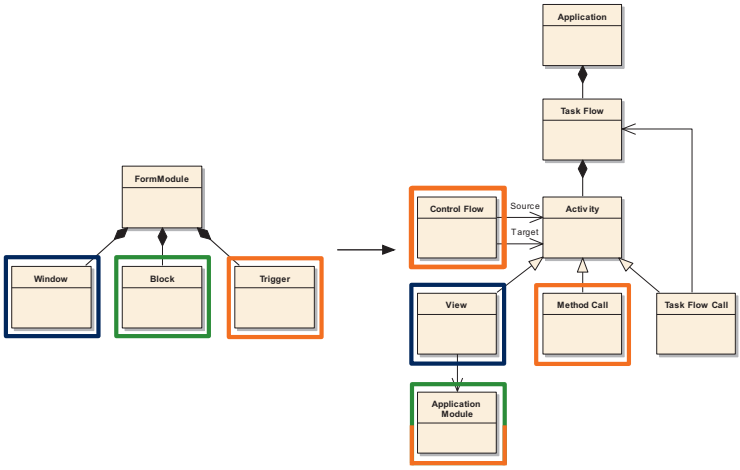

Abbildung 2: Ausschnitte des plattformspezifischen Metamodells der Quell- (links) und Zielplattform (rechts) sowie die Abbildung zwischen einzelnen Elementen (Farbe)

2. *Abstraktion*
Nachfolgend wird ein Abstraktionsschritt durchgeführt, in welchem eine Zwischenrepräsentation der Quellapplikation erzeugt wird. Um die Notwendigkeit für diesen Schritt zu verdeutlichen, wird an dieser Stelle ein konkretes Beispiel angeführt.
In Abbildung 2 ist auf der linken Seite ein Ausschnitt des plattformspezifischen Metamodells der Quellplattform dargestellt. Durch den Modellbildungsschritt wurde die Applikation konform zu diesem Metamodell erfasst. Die Klasse *FormModule* repräsentiert ein abgeschlossenes Modul der Oracle Forms Plattform, alle Strukturen innerhalb eines Quellartefakts sind dieser Klasse untergeordnet. Eine Anwendung besteht aus einer Menge von Quellartefakten. Die *Window*-Klasse beschreibt die enthaltenen Oberflächen, während die *Block*-Klasse die angebundene Datenhaltung

abbildet. Des Weiteren existiert eine Menge an *Trigger*-Klassen, welche Code-Blöcke in der Programmiersprache PL/SQL enthalten.
Auf der rechten Seite ist das plattformspezifische Metamodell der Zielplattform dargestellt. Eine Anwendung ist in mehrere Applikationen unterteilt, dargestellt durch die Klasse *Application*. Eine Applikation enthält einen oder mehrere Arbeitsabläufe in Form von *Task Flows*, die sich aus einer Menge an Aktivitäten zusammensetzen. Eine Aktivität umfasst z.B. die Anzeige einer Benutzungsoberfläche (*View*), in der die angebundenen Datensätze (*Application Module*) angezeigt werden. Weitere Aktivitäten repräsentieren Methodenaufrufe (*Method Call)* oder die Ausführung eines Arbeitsablaufes (*Task Flow Call*). Der Kontrollfluss zwischen einzelnen Aktivitäten wird explizit dargestellt (*Control Flow*).
Im Kontext der Migration werden Instanzen des Metamodells der Quellplattform in Instanzen des Metamodells der Zielplattform überführt, so dass die Funktionalität der Anwendung erhalten bleibt. Dies wird durch die Definition von Modelltransformationen in Form von Abbildungen zwischen den Metamodellen realisiert. Eine Voraussetzung für die Definition einer solchen Abbildung ist, dass beide Metamodelle ein Konzept in einer ähnlichen Granularität beschreiben. Dies ist in dem Beispiel sowohl bzgl. der Benutzungsoberflächen als auch der Datenanbindung der Fall, beide Konzepte werden sowohl in der Quell- als auch in der Zielplattform explizit repräsentiert. Dadurch ergibt sich eine Abbildung von *Window* auf *View* (blau) sowie von *Block* auf *Application Module* (grün).
Der Wechsel von einer Benutzungsoberfläche auf eine andere wird in der Zielplattform durch einen zugehörigen Kontrollfluss ebenfalls explizit beschrieben. In der Quellplattform hingegen wird diese Navigation programmatisch durch einen Code-Block innerhalb eines *Triggers* durchgeführt. Gleiches gilt für andere Konzepte, wodurch Teile der Code-Blöcke auf unterschiedliche Klassen abgebildet werden müssen (orange). Die syntaktische Zerlegung eines Code-Blocks resultiert jedoch lediglich in einer Menge an Anweisungen, welche nicht näher klassifiziert sind. Folglich muss eine semantische Interpretation durchgeführt werden, um die relevanten Informationen zu extrahieren und in der benötigten Granularität bezüglich der Zielplattform zu erfassen.
In unserem Ansatz wird solch eine Zwischenrepräsentation der Applikation explizit erstellt (*Platform Independent Legacy Model*). Dazu wird eine Beschreibungssprache in Form eines Metamodells bereitgestellt, durch welche unterschiedliche Applikationen in einer einheitlichen Sprache repräsentiert werden können. Dadurch können Analyse- oder Transformationsverfahren, welche im Rahmen der Migration einer spezifischen Applikation entwickelt wurden, auch für die Migration anderer Applikationen wiederverwendet werden. Das Metamodell der Sprachdefinition ermöglicht die Beschreibung verschiedener Aspekte der Applikation auf unterschiedlichen Abstraktionsebenen; ein plattformunabhängiges Modell untergliedert sich intern folglich entsprechend dieser Ebenen. Dies umfasst unter anderem den strukturellen Aufbau der Benutzungsoberflächen und die Möglichkeit, architektonische Sichten der Applikation zu erstellen. Dabei werden explizit plattform- und domänenspezifische Aspekte berücksichtigt. Gleichzeitig kann das Metamodell um applikationsspezifische Besonderheiten in einer Migration situativ oder kundenspezifisch erweitert werden. Ein Beispiel ist die Erfassung der Benutzungsoberflächen. Wie bereits in Abschnitt 1 angeführt, wird eine 1:1-Überführung nicht verfolgt. Vielmehr sollen

die wesentlichen Informationen erfasst und in der Zielplattform umgesetzt werden. In unserem Fallbeispiel konnten wir feststellen, dass in der Applikation drei Typen von Benutzungsoberflächen existieren. Diese stellten entweder die Funktionalität zum Anzeigen, Selektieren oder Editieren von Datensätzen bereit. Für jeden Typ wurde daraufhin eine Umsetzung in der Zielplattform festgelegt. Solche applikationsspezifischen Aspekte zu berücksichtigen soll in Form einer leichtgewichtigen Erweiterung, durch die Nutzung von Profilen, ermöglicht werden. Ein Aspekt kann dadurch in dem Abstraktionsschritt erfasst und in der Überführung ausgewertet werden.

Auf Grund der Freiheitsgrade der Quellplattform wird eine Vollautomatisierung des Abstraktionsschritts nicht angestrebt. Vielmehr soll ein Semi-Automatismus genutzt werden, indem Expertenwissen über die Applikation einbezogen und mit leistungsfähigen Automatismen kombiniert wird. Dies umfasst die Annotation der plattformspezifischen Modelle, indem z.B. die Klassifikation einzelner Code-Abschnitte in Bezug auf deren architektonische Zugehörigkeit angegeben wird. Anschließend können Transformationsregeln basierend auf dem plattformspezifischen Metamodell ausgedrückt werden, welche die angefügten Annotationen auswerten und die eine Transformation-Engine automatisch ausführt. Wir verfolgen dabei den Ansatz, die einzelnen Transformationsregeln modular in Form eines Katalogs zu verwalten, so dass diese in anderen Migrationsprojekten wiederverwendet werden können. Inwiefern eine Wiederverwendung gewährleistet werden kann, soll im Rahmen des Fallbeispiels evaluiert werden. Insbesondere im Kontext einer 4GL-Migration existieren jedoch seitens der Quellplattform vorgegebene Standard-Lösungen bzgl. der Umsetzung von Funktionalitäten, für welche ein Katalog bereitgestellt werden kann.

Die Modellstrukturen, die bis zu diesem Schritt in dem Repository gebildet wurden, müssen genutzt werden, um die Zielarchitektur der Applikation zu definieren. Dabei handelt es sich um eine manuelle Tätigkeit, bei der die extrahierten Informationen zur Entscheidungsfindung ausgewertet werden. Während und nach dem Entscheidungsprozess kann es notwendig sein, zusätzliche Informationen zu extrahieren. Aus diesem Grund wird der Abstraktionsschritt iterativ ausgeführt, und das Repository wird inkrementell angereichert. Die Definition einer Zielarchitektur wird unterstützt, indem das Wissen aus anderen Migrationsprojekten wiederverwendet wird. Beispielsweise soll auf einen Katalog bewährter Zielarchitekturen und Umsetzungsmöglichkeiten zurückgegriffen werden können. Die Beschreibung solcher bewährten Lösungen kann mit Automatismen einhergehen, welche die bestehende Applikation auf die Konformität überprüfen und Probleme aufdecken.

Aus der Zielarchitektur geht letztendlich der Umfang hervor, in welchem eine Applikation auf dieser Abstraktionsebene beschrieben wird. Die Zielplattform stellt z.B. mehrere Möglichkeiten bereit, Geschäftsregeln abzubilden. Diese können programmatisch umgesetzt oder explizit durch eine entsprechende Engine ausgewertet werden. Wenn Letzteres genutzt werden soll, müssen die Geschäftsregeln jedoch aus dem Quellcode der Quellapplikation extrahiert und in einer zugehörigen Beschreibungssprache ausgedrückt werden, während bei einer programmatischen Umsetzung Code-Wrapping eingesetzt werden könnte. Die Extraktion erzeugt in diesem Beispiel zusätzlichen Aufwand, bietet jedoch auch viele Vorteile wie eine einheitliche Verwaltung oder verständliche Repräsentation der Regeln. Allgemein stellen Modelle auf dieser Abstraktionsebene einen

Mehrwert über die Migration hinaus dar, da sie Teilaspekte der Applikation verständlich dokumentieren.

3. *Restrukturierung*
   Nachdem die aktuelle Ausprägung der Applikation auf verschiedenen Abstraktionsebenen erfasst wurde, ist der Mehrwert einer Restrukturierung der Applikation zu bestimmen, durch welche diese an die Konzepte der Zielplattform angepasst werden könnte (*Platform Independent Model*). Dies kann Anhand des plattformspezifischen Metamodells der Quell- und der Zielplattform erläutert werden, dargestellt in Abbildung 2. In Oracle Forms werden Quellartefakte zu Modulen (*FormModule*) zusammengefasst, die eine wiederverwendbare Einheit bilden. Oftmals wurden durch ein Modul Benutzungsoberflächen (*Window*) zusammengefasst, die auf ähnlichen Datensätzen arbeiten (*Block*), da die Anbindung an die Datenschicht nicht modularisiert werden konnte. In Oracle ADF hingegen ist die Modularisierung der Anbindung an die Datenschicht explizit vorgesehen (*Application Module*). Hierdurch wird die Unterteilung einer Applikation in wiederverwendbare Arbeitsschritte (*Activity*) und Arbeitsabläufe (*Task Flow*) ermöglicht. In der Migration muss evaluiert werden, inwiefern die Struktur der zu migrierenden Applikation auf Konzepte der Zielplattform abgebildet werden kann. Falls die resultierende Zielapplikation nicht ausreichend gut strukturiert ist, so dass ein Mehraufwand durch eine anschließende Restrukturierung entstünde, sollte eine systematische Restrukturierung in der Migration durchgeführt werden. Dazu ist semi-automatisch zu erfassen, welche Benutzungsoberflächen in einem Arbeitsablauf beteiligt sind. Basierend auf den resultierenden Modellen muss eine Entscheidung getroffen werden, an welchen Stellen wiederverwendbare Komponenten, entsprechend den Konzepten der Zielplattform, gebildet werden können. Der Umfang dieses Schritts variiert folglich in Abhängigkeit von der Applikation, im optimalen Fall wird er nicht benötigt.

4. *Konkretisierung*
   Aus der abstrakten Beschreibung der Applikation muss ein plattformspezifisches Modell abgeleitet werden (*Platform Specific Model*). Dazu wird für die Zielplattform ein Metamodell bereitgestellt, durch welches Applikationen plattformspezifisch beschrieben werden können. Neben den Sprachelementen der Zielplattform enthält das Metamodell zusätzliche Elemente, um bewährte Lösungsmöglichkeiten auf einer höheren Abstraktionsebene zu beschreiben. Die Ableitung des plattformspezifischen Modells wird äquivalent zum Abstraktionsschritt durch Modelltransformationen durchgeführt, wobei die einzelnen Transformationen modular in Form eines Katalogs zu verwalten sind. Dadurch soll die Wiederverwendung von Transformationen ermöglicht werden.
   Wie in Abbildung 1 angedeutet, gehen wir davon aus, dass eine Ableitung des plattformspezifischen Modells der Zielapplikation (*Platform Specific Model*) nicht nur durch Transformationen des abstrakten, plattformunabhängigen Modells (*Platform Independent Model*) erfolgt. Denn in diesem Modell wurde von technischen Implementierungsdetails abstrahiert. Solche Details sind jedoch essenziell, um vollständig lauffähigen Code zu erzeugen. Aus diesem Grund sollen in der Generierung des plattformspezifischen Modells (*Platform Specific Model*) zusätzlich Transformationen ermöglicht werden, welche Informationen des plattformspezifischen Modells

der Altanwendung (*Platform Specific Legacy Model*) aufgreifen, um einzelne Teilbereiche der Applikation automatisch und möglichst vollständig zu generieren.

5. *Code-Generierung*
   Im letzten Schritt wird aus dem plattformspezifischen Modell der Zielapplikation (*Platform Specific Model*) Quellcode (*Source*) in der Zielplattform generiert. Dazu wird ein Code-Generator verwendet, der auf dem plattformspezifischen Metamodell der Zielplattform aufsetzt.

Wir haben bereits begonnen, Teilaspekte des Ansatzes zu implementieren. Dies umfasst insbesondere die Erstellung der plattformspezifischen Metamodelle sowie eine Routine, welche die initiale Modellbildung konform zu dem definierten Metamodell durchführen kann. Dazu verwenden wir das Eclipse Modeling Framework[2] (EMF). Die Erstellung des Metamodells ist insbesondere im Kontext einer 4GL-Migration eine Herausforderung, da die meisten Strukturen proprietär sind. In unserem Fallbeispiel haben wir die Strukturen der Quellartefakte automatisch aus einem Hersteller-Werkzeug extrahiert. Zudem muss, wie in Abschnitt 3 beschrieben, das nicht explizit sichtbare Verhalten der Quellplattform beschrieben werden. Dies umfasst beispielsweise Aufrufreihenfolgen oder die Klassifizierung von plattformspezifischen Aufrufen in Code-Abschnitten.
Ein Aspekt, der von Anfang an beachtet werden muss, ist die Skalierbarkeit der entstehenden Modelle. Da die von uns betrachteten Applikationen mehrere Millionen Zeilen Code haben, müssen die Modelle effizient verwaltet werden. Dieses Problem haben wir durch die Verwendung des Eclipse CDO-Projekts[3] (Connected Data Objects) gelöst, wodurch die Modelle in einer Datenbank persistent gespeichert werden. Der zugehörige CDO-Server ermöglicht eine effiziente Abfragemöglichkeit der Modellstrukturen. Gleichzeitig wird ein verteilter Zugriff gewährleistet.
Des Weiteren sollte von Anfang eine Anbindung der Entwicklungsumgebung der Zielplattform an das Repository eingeplant werden. Manuelle Implementierungsschritte im Anschluss an die Migration können dadurch unterstützt werden, indem die Informationen der Modelle in der Entwicklungsumgebung angezeigt bzw. verwendet werden können. In unserem Fallbeispiel haben wir eine Bibliothek entwickelt, die durch die Entwicklungsumgebung der Zielplattform eingebunden werden kann. Dadurch ist ein Zugriff auf die Modellstrukturen möglich.

4 Verwandte Arbeiten

Die modellgetriebene Softwaremigration ist mittlerweile ein etabliertes Themenfeld in der Softwaretechnik. Die OMG hat aus diesem Anlass die Architecture-Driven Modernization (ADM) Task Force[4] geschaffen, um die Modellstrukturen zu standardisieren, die während des Modernisierungsprozesses entstehen und um somit die Interoperabilität von Werkzeugen zu gewährleisten. Ein Ergebnis ist das Knowledge Discovery Metamodel (KDM) [Om11], welches eine Ontologie zur Beschreibung von Aspekten eines Soft-

[2] http://www.eclipse.org/modeling/emf/
[3] http://wiki.eclipse.org/CDO
[4] http://adm.omg.org/

waresystems auf unterschiedlichen Abstraktionsebenen darstellt.
Im Projekt SOAMIG [Wi11a] wurde die Migration von Altsystemen in serviceorientierte Architekturen (SOA) untersucht, und es wurde ein Referenzprozess vorgestellt, welcher die Phasen und Disziplinen einer SOA-Migration beschreibt [Wi11b]. In diesem Kontext wurden Techniken eingeführt, um separat erstellte Modelle durch dynamische Analysen automatisch mit Code-Abschnitten der Altanwendung zu verknüpfen [FHR10].
Im Projekt DynaMod [Ho11]. werden statische und dynamische Analysen eingesetzt, um die Architektur der bestehenden Applikation zu rekonstruieren. Basierend auf diesen Informationen wird eine Modernisierung durchgeführt.
In [RGD06] wird ein Reverse-Engineering-Ansatz beschrieben, durch den aus einer bestehenden Applikation Modelle im Sinne des MDA-Ansatzes der OMG extrahiert werden. Der Quellcode der Quellapplikation wird durch Parser zerlegt und anschließend in eine plattformunabhängige Repräsentation überführt. Aus dieser Repräsentation werden UML-Modelle abgeleitet, die zur Dokumentation oder Code-Generierung genutzt werden. Die Wiederverwendung des entstandenen Migrationswissens wird jedoch nicht diskutiert.
Der Ansatz in [Fl07] ähnelt dem, den wir in unserem Projekt verfolgen. Die Quellartefakte werden ebenfalls zerlegt, in einer plattformunabhängigen Beschreibungssprache repräsentiert und letztendlich zur Code-Generierung genutzt. Dabei wird explizit auf die Wiederverwendbarkeit und Automatisierung einzelner Teilschritte eingegangen. Allerdings wird keine Restrukturierung der bestehenden Applikation berücksichtigt.
In [GC12] werden abhängige Transformationen im Kontext der Modernisierung nach dem Hufeisenmodell diskutiert. Diesbezüglich wird eine implizite und explizite Parametrisierung unterschieden.

5 Zusammenfassung und Ausblick

Im Rahmen unseres Forschungsprojekts untersuchen wir die Anwendung von Techniken der modellgetriebenen Softwareentwicklung im Kontext der Softwaremigration. Dabei betrachten wir als konkretes Fallbeispiel die Migration von Anwendungen der Plattform Oracle Forms nach Oracle ADF. In diesem Beitrag sind wir auf die Charakteristiken dieser Migration eingegangen und haben darauf aufbauend die Anforderungen an eine Werkzeugunterstützung aufgestellt, wobei die Erfahrungen mit bereits verfügbaren Lösungen eingeflossen sind.
Aufbauend auf den beschriebenen Anforderungen wurde in diesem Beitrag ein Migrationsprozess vorgestellt. Die zu Grunde liegende Idee ist die Nutzung von Modellen auf verschiedenen Abstraktionsebenen, welche durch Modelltransformationen ineinander überführt werden. Ein zentraler Aspekt ist die Nutzung einer einheitlichen Beschreibungssprache, um das in einer Migration entstehende Wissen formal festzuhalten. Die Wiederverwendung dieses Wissens unterstützt durch Automatismen soll letztendlich das wirtschaftliche Risiko in der Migration von Applikationen verringern, indem Implementierungsfehler minimiert und Kosten gespart werden.
Die technische Umsetzung des beschriebenen Ansatzes wurde bereits begonnen, insbesondere auf plattformspezifischer Ebene konnten bereits Metamodelle, eine Modellbildungs-Routine und ein Code-Generator umgesetzt werden. Im nächsten Schritt fokussieren wir die Erstellung einer plattformunabhängigen Beschreibungssprache, um eine

Abstraktion von der technischen Umsetzung erreichen zu können. Dabei berücksichtigen wir bestehende Ansätze wie KDM, ziehen aber auch explizit Erfahrungen mit ein, die im Rahmen von Migrationen im Kontext des Fallbeispiels gemacht wurden. Die Herausforderung ist, die Aspekte zu identifizieren, welche für eine Migration im Kontext des Fallbeispiels essenziell erfasst werden müssen. Des Weiteren evaluieren wir verfügbare Modelltransformationssprachen und -engines. Transformationen sollen effizient sein und inkrementell durchgeführt werden können. Zusätzlich ist eine Modularisierung von Transformationsregeln erforderlich.

Literaturverzeichnis

[SWH10] Sneed, H. M.; Wolf, E.; Heilmann, H.;: Software-Migration in der Praxis: Übertragung alter Softwaresysteme in eine moderne Umgebung, dpunkt Verlag 2010.

[Or12] Oracle: Oracle Application Development Tools Statement of Direction: Oracle Forms, Oracle Reports and Oracle Designer. http://www.oracle.com/technetwork/issue-archive/2010/toolssod-3-129969.pdf, Accessed: 21-Dec-2012

[GGS12] Grieger, M.; Güldali, B.; Sauer, S.: Sichern der Zukunftsfähigkeit bei der Migration von Legacy-Systemen durch modellgetriebene Softwareentwicklung. Proceedings of the 14th Workshop Software-Reengineering (WSR 2012). Softwaretechnik-Trends, vol. 32, no. 2, pp. 37–38, 2012.

[KWC98] Kazman, R.; Woods, S. G.; Carrière, S. J.: Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. Proceedings of WCRE 98, pp. 154–163, 1998.

[ZGW10] Zillmann, C.; Gringel, P.; Winter, A.: Iterative Zielarchitekturdefinition in SOAMIG. Softwaretechnik-Trends, vol. 30, no.2, pp. 39–40, 2010.

[Om11] OMG, The OMG Knowledge Discovery Metamodel (KDM), Sep. 2011.

[Wi11a] Winter, A. et al.: SOAMIG Project: Model-Driven Software Migration Towards Service-Oriented Architectures. Proceedings of MDSM 2011, vol. 708 of CEUR Workshop Proceedings, p. 15–16, 2011.

[Wi11b] Winter, A. et al.: The SOAMIG Process Model in Industrial Applications. Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR), pp. 339–342, 2011.

[FHR10] Fuhr, A.; Horn, T.; Riediger, V.: Dynamic Analysis for Model Integration. Softwaretechnik-Trends, vol. 30, no. 2, p. 70–71, 2010.

[Ho11] van Hoorn, A. et al.: DynaMod Project: Dynamic Analysis for Model-driven Software Modernization. Proceedings of MDSM 2011, vol. 708 of CEUR Workshop Proceedings, pp. 12–13, 2011.

[RGD06] Reus, T.; Geers, H.; Van Deursen, A.: Harvesting Software Systems for MDA-based Reengineering. In: Model Driven Architecture-Foundations and Applications, pp. 213–225, 2006.

[Fl07] Fleurey, F. et al.: Model-driven Engineering for Software Migration in a Large Industrial Context. In: Model Driven Engineering Languages and Systems, pp. 482–497, 2007.

[GC12] Cuadrado, J. S.; García, O. Á.; Canovas, J.; Herrera, A. S. B.: Parametrización de las transformaciones horizontales en el modelo de herradura. Jornadas de Ingeniería del Software y Bases de Datos, 2012.