TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

MAGISTERARBEIT

Reengineering zu Web-Anwendungen

Ein architektureller Ansatz

ausgeführt am

Institut für Computersprachen
der Technischen Universität Wien

unter der Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Franz Puntigam

durch

Robert David Bakk.techn.

Erlaaer Platz 3a/2/20
1230 Wien

Wien, im August 2007

Zusammenfassung

Diese Magisterarbeit befasst sich mit dem Reengineering von Legacy Systemen zu modernen Web-Anwendungen. Aufgrund der wachsenden Verbreitung der Web-Technologien und der damit verbundenen Vorteile werden neue Systeme zunehmend als Web-Anwendungen entwickelt. Ihnen gegenüber stehen veraltete Systeme, Legacy Systeme genannt, die erfolgreich Wartungs- und Erweiterungsversuchen widerstehen. In dieser Arbeit werden die Möglichkeiten untersucht, solche Altsysteme zu modernen Web-Anwendungen zu transformieren. Dazu wird zuerst ein Web-Reengineering Prozess definiert, der auf den bekannten Reengineering-Aktivitäten basiert und diese für das Zielsystem Web-Anwendung konkretisiert. Im Vordergrund steht dabei die architekturelle Transformation. Das Ausgangssystem soll so verändert werden, dass die Architektur der Web-Anwendung bereits dort berücksichtigt wird. Für jeden Teilschritt des Prozesses werden bekannte Reengineering-Methoden vorgestellt, die sich für die praktische Durchführung eignen. Das Ergebnis ist ein durchgehender Prozess, der für das Reengineering zu Web-Anwendungen als Basis verwendet werden kann. Danach wird der Prozess im Rahmen einer Fallstudie angewendet, bei der ein Altsystem zu einer funktional äquivalenten Web-Anwendung transformiert wird. Dabei zeigt sich, dass eine Unterscheidung von physischer und logischer Architektur und die Repräsentation beider im Ausgangssystem strukturelle Vorteile für das Reengineering bringt. Mit den Ergebnissen der Durchführung soll der architekturelle Ansatz des Prozesses analysiert werden, um weitere Erkenntnisse über das Reengineering zu Web-Anwendungen zu erlangen.

Abstract

This master thesis covers the reengineering of legacy systems towards modern web applications. With the widespread use of web technologies and the associated advantages we increasingly develop new systems as web applications. There are still out-dated systems, termed legacy systems, that significantly resist maintenance and evolution. This thesis examines the options for transforming legacy systems into modern web applications. For this purpose we define a web-reenigneering process. It is based on known reengineering activities that we concretize for web applications as the target system. In the process the most importance is given to the architectural transformation. We change the examined system such that the architecure of the web application is represented in its structure. For each step in the process we present known reengineering methods for the practical implementation. We get a continuing process for use as a basis in the reengineering towards web applications. After its definition we use the process in a case study, where a legacy system is transformed into a functionally equivalent web application. We show that if we differentiate between physical and logical architecture and represent both in the legacy system we can improve the structure of the target system. We use the result of the implementation to evaluate the architectural approach of the process to gain further knowledge for the reengineering towards web applications.

Danksagung

An dieser Stelle möchte ich mich ganz herzlich bei allen bedanken, die durch ihre Unterstützung zum Gelingen dieser Magisterarbeit beigetragen haben. Zuerst möchte ich mich bei meinem Betreuer Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Franz Puntigam für die hervorragende Betreuung bedanken. Weiters danke ich all meinen Freunden und Studienkollegen, die durch anregenden Diskussionen, konstruktive Kritik und Korrekturlesen zum Erfolg dieser Arbeit beigetragen haben. Zu guter Letzt gilt der größte Dank meiner Familie und ganz speziell meiner Freundin, die mich mit Geduld und Unterstützung durch meine Studienzeit begleitet hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Ziel - Web-Anwendungen

Aufgrund der wachsenden Verbreitung von Internet und Web-Technologien geht der Trend bei der Entwicklung neuer Software-Systeme zunehmend in Richtung Web-basierter Anwendungen. Darunter versteht man Anwendungen, die auf einem Webserver ausgeführt werden und einen Web-Browser für die Interaktion mit dem Anwender verwenden. Diese bieten durch die Verwendung von Web-Standards für Benutzerschnittstelle und Kommunikation sowie durch ihre zentralisierte Architektur wesentliche Vorteile.

- Der universelle Zugriff auf die Anwendung über das Internet unter Verwendung von Web-Standards wird ermöglicht.
- Die Anwendung steht allen Benutzern ohne Konfiguration und Installation zur Verfügung.
- Die Anwendung ist plattformunabhängig. Die Client-Seite ist nicht an ein bestimmtes Betriebssystem gebunden. Der Zugriff erfolgt über einen Web-Browser.
- Web-Anwendungen sind Thin-Client Lösungen. Dadurch sind die Anforderungen an die Performance der Client-Rechner geringer als bei anderen Lösungen.
- Web-Anwendungen verwenden ohne zusätzlichen Aufwand bestehende Netzwerkstrukturen für die verteilte Kommunikation. Dabei müssen keine Änderungen an der Konfiguration von Netzwerkkomponenten wie Firewalls und Proxys durchgeführt werden.

- Änderungen und Erweiterungen der Anwendung können aufgrund der zentralisierten Architektur einfach eingebracht werden, da sie nur serverseitig durchgeführt werden müssen.

Weiters bieten Web-Anwendungen auch den Entwicklern Vorteile. Aufgrund der mehrschichtigen Architektur und der Verwendung von objektorientierten Technologien, wie es moderne Web-Anwendungsserver ermöglichen, werden bessere Wartbarkeit und Erweiterbarkeit solcher Systeme ermöglicht.

Problem - Legacy Systeme

Diesen modernen Web-Anwendungen stehen Altanwendungen gegenüber, die Legacy Systeme genannt werden. Es handelt sich dabei um monolithische Systeme, die erfolgreich Wartungs- und Erweiterungsversuchen widerstehen. Allgemein werden darunter alte, umfangreiche Systeme verstanden, die sich aufgrund ihrer Wichtigkeit immer noch im Einsatz befinden, obwohl sie kritische Nachteile aufweisen.

- Sie wurden mit Technologien entwickelt, die heute nicht mehr verwendet werden.
- Es sind keine Informationen über die internen Abläufe vorhanden.
- Sie bilden kritische Geschäftsprozesse ab.

Die Kosten, die ein Legacy System verursacht, steigen mit der Lebensdauer. Irgendwann ist dieser Zustand nicht mehr tragbar und es wird die Entscheidung getroffen, es abzulösen und durch eine moderne Anwendung zu ersetzen. Um eine langfristige Lösung zu erhalten, sollen der Datenbestand und möglichst viel vom Legacy System selbst in eine moderne Umgebung migriert werden, um diese Teile dort wiederverwenden zu können. Nicht migrierbare Teile werden neu entwickelt und zusätzlich gewünschte Funktionalität ebenfalls im Zuge dessen eingebracht. Das Ergebnis ist eine funktional äquivalente Neuanwendung, die moderne Technologien verwendet und die Schwächen des Legacy Systems vermeidet. Sie kann zukünftig nach Bedarf geändert und erweitert werden.

Ansatz - Software Reengineering

Oft steht man bei der Behandlung eines Legacy Systems vor dem Problem, dass Wissen über die Anwendung und die verwendeten Technologien nicht mehr verfügbar ist. Dokumentation über Anforderungen und Design-Dokumente sind nicht vorhanden. Oder sie wurden über Jahre nicht mehr gewartet und sind dadurch nicht mehr zutreffend. Um die Wiederverwendung

und Neuentwicklung eines Legacy Systems zu ermöglichen, muss dieses Wissen wiedererlangt werden. Dieses kann dann als Basis für die Transformation zum Zielsystem verwendet werden. Dabei kann die bestehende Funktionalität übernommen und Erweiterungen direkt mit eingebracht werden. Weiters werden Wartungsaktivitäten, wie die Verbesserung der internen Struktur, bereits vor der Transformation unterstützt. Der Prozess, bei dem verlorene Informationen wiedererlangt werden, um diese für die Transformation des alten Systems in ein neues zu verwenden, wird Reengineering genannt [CI90].

Dieser Prozess und seine Aktivitäten befinden sich allerdings auf einem hohen abstrakten Niveau und sind sehr allgemein definiert. Sie stellen zwar eine gute Basis für eine Durchführung dar, gehen aber nicht auf Systemeigenschaften, wie Sprachparadigma und Architektur, ein, da sie sich diesbezüglich nicht festlegen. Will man aber zu einem konkreten Zielsystem, wie einer Web-Anwendung, gelangen, kann man mehr ins Detail gehen und die Aktivitäten konkretisieren, um dessen Eigenschaften zu erreichen. Nachdem diese festgelegt sind, können Methoden ausgewählt werden, mit denen sie sich umsetzen lassen.

Betrachtet man die Eigenschaften von Web-Anwendungen, steht dabei die mehrschichtige Architektur im Vordergrund. Weiters bietet die Verwendung von objektorientierten Technologien eine gute Möglichkeit, Wartbarkeit und Erweiterbarkeit des Zielsystems zu unterstützen. Durch eine architekturelle Sichtweise des Reengineering Prozesses kann man sich auf diese wichtigen Eigenschaften konzentrieren. Eine möglichst frühe Berücksichtigung der architekturellen Transformation im Ablauf des Prozesses bietet sich dabei an.

1.2 Ziel der Arbeit

Diese Arbeit behandelt die Verwendung von Software Reengineering mit dem Ziel, Altsysteme zu mehrschichtigen objektorientierten Web-Anwendung zu transformieren. Dafür wird ein Web-Reengineering Prozess definiert, dessen Ausgangspunkt die bekannten Reengineering-Aktivitäten sind. Diese werden als Teilschritte des Prozesses in eine festgelegte Reihenfolge gebracht und für das Zielsystem Web-Anwendung konkretisiert. Es wird dabei untersucht, welche Eigenschaften umzusetzen sind und wie diese Umsetzung im Zuge des Prozesses durchgeführt werden kann. Für die Durchführung jedes Teilschrittes werden dabei bekannte Methoden zur Bewältigung vorgestellt, die sich gut für dieses Zielsystem eignen.

Weiters soll der Prozess unter einer architekturellen Sichtweise betrachtet werden. Da die mehrschichtige Architektur einer Web-Anwendung im Vordergund steht, sollen die Möglichkeiten untersucht werden, diese bereits im Ausgangssystem umzusetzen. Dadurch soll die Transformation zum Zielsy-

stem erleichtert werden. Die Auswirkungen einer solchen Vorgangsweise auf die Durchführung des gesamten Web-Reengineering Prozesses werden im Zuge dieser Arbeit untersucht.

Nach der Definition des Web-Reenginering Prozesses wird der im Rahmen einer Fallstudie untersucht, um dessen praktische Aspekte analysieren zu können. Dafür wird ein für Legacy Systeme repräsentatives Ausgangssystem gewählt. Dieses wird unter Verwendung des Prozesses zu einer funktional äquivalenten Web-Anwendung transformiert. Nach Abschluss der Durchführung wird das Ergebnis bezüglich der Ziele dieser Arbeit analysiert. Die Eigenschaften des Prozesses und die Umsetzung der Zieleigenschaften unter Einhaltung der funktionalen Äquivalenz werden beurteilt. Weiters wird untersucht, welche Vorteile und Nachteile die architekturelle Umsetzung, die bereits im Ausgangssystem erfolgt, für den gesamten Prozess mit sich bringt.

Diese Informationen sollen weiterführendes Verständnis für das Reengineering zu Web-Anwendungen und die architekturelle Sichtweise dafür schaffen. Sie zeigen Schwierigkeiten auf, die dabei zu erwarten sind und bieten nützliche Hinweise, die eine zukünftige Durchführung erleichtern.

1.3 Aufbau

Kapitel 2 - Vom Legacy System zur Web-Anwendung

In diesem Kapitel werden die Grundlagen und das Umfeld für diese Arbeit erklärt. Es führt in die Thematik des Reengineering von Legacy Systemen zu Web-Anwendungen ein. Dabei werden, ausgehend von der Erklärung der Legacy Systeme, alle für das Reengineering relevanten Bereiche beschrieben, die hin zu einer Web-Anwendung führen. Für diese Bereiche werden wesentliche Begriffe erklärt und unterschiedliche Sichtweisen darauf einander gegenüber gestellt. Die Kenntnis dieser Grundlagen ist wesentlich für das Verständis der folgenden Kapitel.

Kapitel 3 - Der Web-Reengineering Prozess

In diesem Kapitel wird für das Reengineering mit dem Zielsystem Web-Anwendung ein dafür geeigneter Prozess definiert. Angelehnt an die Reengineering-Aktivitäten Reverse Engineering, Restructuring und Forward Engineering, werden dafür aufeinander folgende Teilschritte definiert. Für diese Schritte werden die zu lösenden Teilaufgaben betrachtet und unter Berücksichtigung der Zielsystemeigenschaften Lösungsansätze vorgestellt. Für diese werden konkrete Methoden angegeben, die für eine praktische Umsetzung verwendet werden können. Bei der Festlegung des Prozesses steht die Architektur der Web-Anwendung im Vordergund. Diese soll während des gesamten

Prozessverlaufs berücksichtigt werden. Damit soll untersucht werden, wie eine Umsetzung der Architektur bereits im Legacy System erfolgen kann und wie sich diese auf die Transformation zur Web-Anwendung auswirkt.

Kapitel 4 - Fallstudie

In diesem Kapitel wird die Durchführung einer Fallstudie beschrieben, die zur vertiefenden und praktischen Untersuchung des Themas in Kombination mit dem zuvor definierten Prozess verwendet wird. Als Untersuchungsobjekt wird ein möglichst repräsentatives Legacy System ausgewählt. Dieses wird dann unter Verwendung des Prozesses zu einer funktional äquivalenten Web-Anwendung transformiert. Für jeden Teilschritt wird die Durchführung und die Ergebnisse analysiert. Das für diese Fallstudie gewählte Ausgangssystem ist ein in der Sprache C implementiertes Karteisystem für Unix-basierte Systeme, dass eine zeichenorientierte Benutzerschnittstelle aufweist und Textdateien für die persistente Datenhaltung verwendet. Dieses wird in eine Java EE Web-Anwendung transformiert.

Kapitel 5 - Erkenntnisse

In diesem Kapitel wird der zuvor definierte Web-Reengineering Prozess unter Berücksichtigung der Ergebnisse der Fallstudie neu betrachtet. Die neuen Erkenntnisse werden verwendet, um unter einer praktischen Sichtweise die Lösungsansätze des Prozesses zu diskutieren und zu verbessern. Damit sollen weitere Informationen für das Reengineering zu Web-Anwendung gewonnen werden. Der architekturelle Ansatz des Prozesses wird bewertet und dessen Vorteile und Nachteile aufgezeigt.

Kapitel 2

Vom Legacy System zur Web-Anwendung

2.1 Legacy Systeme

Unter Legacy Systemen versteht man große, komplexe Systeme, die sich bis zu einem Zustand entwickelt haben, in dem sie weiteren Änderungen und Evolution signifikant widerstehen [BS95]. [Ben95] beschreibt sie als große Software-Systeme, mit denen man entwicklungstechnisch nicht umzugehen weiss, die aber wichtig für die verwendende Organisation sind. Legacy Software wurde vor Jahren unter Verwendung veralteter Methoden geschrieben. Eine genaue Definition, wann genau ein System als Legacy System bezeichnet wird, gibt es nicht. In der Literatur sind viele Eigenschaften zu finden.

- Legacy Systeme bilden häufig den Backbone des Informationsflusses innerhalb einer Organisation und sind für deren Geschäftsprozesse unverzichtbar. Falls eines dieser Systeme ausfallen sollte, kommen diese Prozesse zum Stillstand.
- Sie können nur sehr schwer erweitert werden, um neu auftretende Anforderungen zu erfüllen. Dies ist aber für die Organisation notwendig, um konkurrenzfähig zu bleiben.
- Die Wartung ist schwierig und teuer. Fehlersuche ist sehr zeitaufwendig und schwierig aufgrund mangelnder Dokumentation und fehlenden Wissens über die internen Abläufe des Systems.
- Eine Integration mit anderen Systemen ist schwierig, weil keine klaren Schnittstellen vorhanden sind.
- Sie laufen auf veralterter Hardware, die kostspielig zu warten ist und redizieren damit die Produktivität aufgrund der schlechten Performanz.

Allgemein sind Legacy Systeme groß, wichtig und nur schwer änderbar. Oft sind sie auch sehr alt und wurden mit Entwicklungswerkzeugen und Software-Engineering Methoden entwickelt, die damals zwar aktuell waren, heute jedoch veraltet sind. Das Alter ist allerdings nicht maßgeblich, es gibt auch relativ neue Systeme, die Wartung und Erweiterungen widerstehen, und damit als Legacy Systeme behandelt werden müssen.

Dennoch werden Legacy Systeme oft weit über ein angemessenes Lebensende hinaus weiterbetrieben. Einerseits verursacht die Ablösung hohe Kosten, wobei auch eine mögliche Ausfall- bzw. Stillstandszeit einkalkuliert werden muss. Andererseits decken selbst Legacy Systeme oft lange Zeit die Bedürfnisse der Organisation und eine Umstellung würde in diesem Fall keinen direkten Gewinn bringen.

Die Kosten, die ein Legacy System verursacht, steigen allerdings mit der Lebensdauer. Irgendwann ist dieser Zustand dann nicht mehr tragbar. Es wird die Entscheidung getroffen, das Legacy System abzulösen und durch eine moderne äquivalente Anwendung zu ersetzen. Dieses soll die weiter bestehenden funktionalen Anforderungen des alten Systems als auch neu hinzugekommene aktuelle Anforderungen erfüllen.

Ein Beispiel wäre ein bereits lange bestehendes Kundenservice, das nun als Web-Anwendung verfügbar sein soll. Der Betrieb darf bei einer Umstellung nicht unterbrochen werden, die Anwendung muss für die Kunden immer verfügbar sein. Die Funktionalität der originalen Anwendung muss dieselbe bleiben, wobei Erweiterungen im Zuge der Umstellung ebenfalls gleich eingebracht werden sollen. Und eine Umstellung auf Web-Technologien ist für die betreibende Organisation notwendig, um konkurrenzfähig zu bleiben.

Um dieses Legacy Problem zu bewältigen, gibt es verschiedene Strategien. Diese werden im Folgenden betrachtet.

2.2 Bewältigungsstrategien

Die verschiedenen Methoden zur Bewältigung des Legacy Problems können grob in 4 strategische Kategorien eingeteilt werden [BLWG99].

- **Wartung**
  Die Wartung von Software ist nach [IEE98] die Modifizierung eines Softwareprodukts nach seiner Auslieferung zur Korrektur von Fehlern, Verbesserung der Systemeigenschaften oder zur Anpassung des Produktes an eine veränderte Umgebung. Wartung lässt sich in 4 Kategorien einteilen. Alle haben gemeinsam, dass die Aktivitäten erst nach der Auslieferung des Systems durchgeführt werden.

  - **Korrektive Wartung**
    Die Behebung von Fehlern.
  - **Adaptive Wartung**
    Die Anpassung des Systems an eine veränderte oder sich ändernde Umgebung.
  - **Perfektionierende Wartung**
    Die Änderung des Systems zur Verbesserung von Eigenschaften wie der Leistung oder der Wartbarkeit.
  - **Notfallwartung**
    Eine ungeplante korrektive Wartung, die durchgeführt wird, um das System einsatzfähig zu halten.

  In Bezug auf Legacy Systeme bedeutet diese Strategie, dass das System weiterhin gewartet und erweitert wird und die steigenden Kosten dafür akzeptiert werden. Diese Strategie sei hier aber nur der Vollständigkeit halber erwähnt. Wenn ein System mit akzeptablem Budget gewartet werden kann, wird es für gewöhnlich nicht als Legacy System angesehen. Wenn die Kosten nicht mehr tragbar sind, wird diese Strategie nicht mehr weiter betrieben und eine der folgenden wird gewählt.

- **Wrapping**
  Wrapping ist das Umhüllen von Datenbeständen, Programmteilen oder ganzen Systemen mit neuen Schnittstellen. Es ermöglicht alten Komponenten die Erweiterung mit neuer Funktionalität und bietet die Möglichkeit neuer Benutzerschnittstellen. Der umhüllte Teil stellt dabei seine Dienste dem Wrapper zur Verfügung, dieser wird dann von einem externen Client aus angesprochen. Da der Client nur mit dem Wrapper kommuniziert, muss er kein Wissen über die umhüllte Komponente haben und kann trotzdem deren Dienste verwenden. Das Wrapping stellt eine kostengünstige Möglichkeit zur Wiederverwendung dar.

Ein weit verbreitetes Beispiel für die Anwendung von Wrapping ist das Screen Scraping. Dabei wird eine zeichenbasierte Benutzerschnittstelle durch eine graphische client-basierte Benutzerschnittstelle ersetzt. Der Wrapper übersetzt zwischen beiden Schnittstellen, eine Änderung des Legacy System ist nicht notwendig. Screen Scraping bietet eine schnelle, kostengünstige Möglichkeit, eine moderne Benutzerschnittstelle für alte Systeme zu implementieren. Es ist allerdings nur eine kurzfristige Lösung. Alle Nachteile des Legacy Systems bleiben weiterhin bestehen. Oft erhöhen sich die Kosten sogar, da zusätzlich zur Wartung des Legacy Systems die des Wrappers hinzukommt.

- **Neuentwicklung**
  Bei dieser Strategie wird das System von Grund auf neu entwickelt. Dabei werden moderne Architekturen, Technologien und Entwicklungswerkzeuge verwendet. Es findet keine Wiederverwendung des ursprünglichen Systems statt. Nach der Fertigstellung wird das alte System abgeschalten und das neue System geht in Betrieb. Aufgrund der Wichtigkeit von Legacy Systemen ist das hohe vorhandene Risiko eines Fehlschlags dabei oft nicht akzeptabel. Ebenfalls müssen die im Vergleich zu den anderen Methoden hohen Kosten berücksichtigt werden.

- **Migration**
  Die Migration beschäftigt sich mit der Entwicklung eines Zielsystems, dass die Funktionalität und vor allem den Datenbestand des Legacy Systems beibehält, aber im Gegensatz zu diesem gut wartbar und erweiterbar ist. Dadurch wird es möglich, zukünfige Anforderungen zu erfüllen [WLB+97c].

  Es wird versucht, möglichst viel an Teilen und Informationen des Legacy Systems wiederzuverwenden. Im Gegensatz zur Neuentwicklung wird das Ausgangssystem in das Zielsystem übergeführt und geeignete Teile dabei beibehalten. Die Umstellung wird meist kontinuierlich und für einzelne Teilsysteme durchgeführt. Damit sollen Betriebsunterbrechungen vermieden werden. Dabei stellt sich das Problem, dass sich während dieser Umstellung der Datenbestand laufend ändert, da das Ausgangssystem ja in Betrieb bleibt. Die Daten werden aber ebenfalls im Zielsystem benötigt. Deshalb wird eine Methode benötigt, um diese Umstellung durchführen zu können.

  Migration beschränkt sich aber nicht nur auf den Datenbestand, sondern beinhaltet auch den Umstieg auf neue Plattformen, den Wechsel von Programmiersprache und Paradigma, und die Umstellung auf neue Architekturen. Oft wird die Software modulweise migriert und die Komponenten der beiden Systeme aneinander angebunden.

  Die Migration eines Legacy System ist komplizierter und aufwendi-

ger als das Wrapping, bietet aber im Vergleich die grösseren Langzeitvorteile. Im Vergleich zu einer vollständigen Neuentwicklung hat die Migration ein geringeres Risiko. Sie ist eine Langzeitlösung, die gute Wartbarkeit und Erweiterbarkeit bietet. Allerdings ist sie auch ein komplexes Unterfangen mit hohem Zeitaufwand.

Diese Strategien bewirken unterschiedlich starke Änderungen des Ausgangssystem. Abbildung 2.1 [BLWG99] zeigt die verschiedenen Strategien und die Stärke der Änderungen, die sie an einem System bewirken.

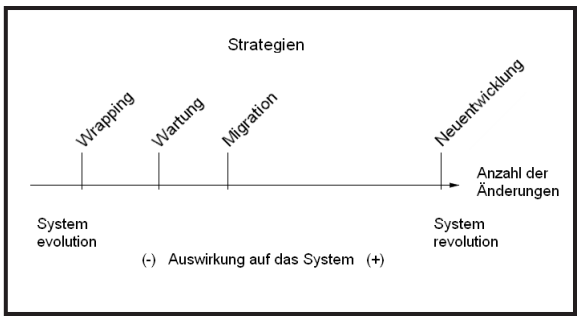


Abbildung 2.1: Vergleich der Änderungen eines Systems durch die Strategien

Hier ist gut zu sehen, dass eine Neuentwicklung im Gegensatz zu den anderen Strategien sehr starke Änderungen bewirkt und dadurch entsprechend aufwendig durchzuführen ist. Im Gegensatz dazu bewirkt Wrapping kaum Änderungen. Es behebt allerdings auch nicht die ursprünglichen Probleme, sondern umgeht sie nur. Die Migration liegt in der Mitte. Sie ändert das System soweit, dass es in der Zielumgebung effektiv eingesetzt werden kann.

Die Einteilung eines Anwendungsfalls in diese Kategorien gestaltet sich schwierig, da er oft mehreren Strategien gleichzeitig zugeordnet werden kann. Dies trifft vor allem auf Neuentwicklung und Migration zu. Der Übergang ist hier fließend. Für eine klare Anwendung der Begriffe werden im Folgenden die Strategien aus praktischer Sicht betrachtet.

2.3 Strategien in der Anwendung

Migrationsmethoden

Ist der Aspekt der Migration bei einem Anwendungsfall vorhanden, wird eine Methode für den eigentlichen Übergang vom Ausgangssystem zum Zielsystem benötigt. Solche Methoden orientieren sich am Datenbestand, da bei einer unterbrechungsfreien Umstellung dieser weiterverwendet werden muss. Dies hat gerade für Legacy Systeme hohe Wichtigkeit.

4 bekannte Migrationmethoden werden hier beschrieben [BLW+97]. Die Neuentwicklung als fünfte Methode stellt eine weitere Möglichkeit dar und kann ebenfalls dazu gezählt werden.

- **Forward Migration Methode**
  Die Forward Migration Methode ist auch unter dem Namen Database First bekannt. Wie der Name schon sagt, wird dabei beim Datenbestand begonnen. Dieser wird zuerst in ein modernes, möglicherweise relationales, Datenbanksystem migriert. Danach werden inkrementell der Anwendungsteil und die Schnittstellen migriert. Das Legacy System und das Zielsystem arbeiten während dieses Vorganges parallel. Forward Gateways ermöglichen dabei den Zugriff des Legacy Systems auf die neue Datenbank.

- **Reverse Migration Methode**
  Die Reverse Migration Methode ist auch unter dem Namen Database Last bekannt. Dabei wird der Anwendungsteil des Legacy Systems migriert, während der Datenbestand in der ursprünglichen Datenbank bleibt. Die Migration dieses wird erst am Ende des Vorgangs durchgeführt. Reverse Gateways ermöglichen den Zugriff des Zielsystems auf die Datenhaltung des Legacy Systems.

- **Composite Database Migration Methode**
  Bei der Composite Database Migration Methode wird der Anwendungsteil des Legacy Systems stufenweise in die Zielumgebung unter Verwendung von modernen Technologien und Werkzeugen entwickelt. Das Legacy System und dass Zielsystem bilden während des Vorgangs ein aus beiden Teilen zusammengesetztes System. Ein Forward Gateway ermöglicht dem Legacy System den Zugriff auf die neue Datenbank. Ein Reverse Gateway ermöglicht dem Zielsystem den Zugriff auf die ursprüngliche Datenbank. Diese Methode ist also eine Kombination der beiden zuvor genannten, wobei es hier keine 2 getrennten Anwendungsteile gibt. Daten können dabei über beide Datenbanken dupliziert werden. Um die Integrität zu bewahren, wird ein Koordinator verwendet. Dieser überprüft alle eingehenden Updateanfragen und überprüft,

ob sie auf replizierte Daten zutreffen. Ist das der Fall, wird das Update in beiden Datenbanken durchgeführt. Eine ähnliche Methode ist die Chicken Little Strategie [BS93]. Sie stellt eine Verfeinerung der Composite Database Migration Methode dar.

- **Butterfly Methode**
  Die Butterfly Methode konzentriert sich auf die Datenmigration und entwickelt das Zielsystem vollständig getrennt vom Legacy System. Im Unterschied zur Composite Database Migration Method gibt es bei der Butterfly Methode keine Notwendigkeit der Zusammenarbeit der beiden Systeme. Das Legacy System bleibt fast während des gesamten Migrationsvorgangs in Betrieb. Eine Duplizierung von Daten in beiden Datenbanken kommt nicht vor. Zu jedem Zeizpunkt kann die Migration rückgängig gemacht werden. Allerdings wird das Zielsystem bei dieser Methode erst nach Abschluss des Vorgangs in Betrieb genommen. Eine genaue Beschreibung dieser Methode ist in [WLB+97a] und [WLB+97b] zu finden.

- **Big Bang Methode**
  Auch unter dem Namen Cold Turkey bekannt, wird die Neuentwicklung in der Literatur teilweise auch als Migrationsmethode gesehen [BLW+97]. Dies unterstützt die Sichtweise, dass ein fließender Übergang zwischen Neuentwicklung und Migration besteht.

Diese Methoden legen Zeitpunkt und Reihenfolge der Übernahme der alten Datenbestände und Systemteile fest. Sie stellen jedoch keine Methoden für die Transformation des Ausgangssystems zum Zielsystem dar.

Migration und Neuentwicklung

Wie schon zuvor erwähnt, ist es nicht immer möglich, eine auf ein Legacy System angewendete Lösung eindeutig zu kategorisieren [BLWG99]. Dies gilt besonders für Neuentwicklung und Migration. Wartungsaktivitäten und Wrapping sind zwar besser abgrenzbar, jedoch kann auch beispielsweise das Wrapping als Wartungsaktivität gesehen werden. Wann jedoch ein Zielsystem als Neuentwicklung oder als migriert gilt, ist schwer zu definieren. Ein Beispiel wäre der Umstieg auf ein modernes Datenbanksystem. Falls eine Anbindung mit der Ausgangstechnologie nicht möglich ist, könnte man diesen Teil neu entwickeln. Der Rest des Systems wird migriert und an den neuentwickelten Teil angebunden.

In der Anwendung ist der Übergang von Migration zur Neuentwicklung fließend. Je mehr vom Ausgangssystem direkt wiederverwendet werden kann, desto eher ist es als Migration zu sehen. Können hingegen keine Teile wiederverwendet werden, ist es eine Neuentwicklung. Da aber auch in diesem

Fall das neue System das Legacy System ersetzen soll, findet selbst bei einer Neuentwicklung oft Wiederverwendung statt. Diese bezieht sich nicht auf Teile der Implementierung, sondern auf die ursprünglichen Anforderungen an das Legacy System. Eine weitere Wiederverwendung wäre die Verwendung des ursprünglichen Designs als Basis für ein neu zu implementierendes Zielsystem.

Soll eine funktionale Äquivalenz zum Ausgangssystem erreicht werden, muss eine Wiederverwendung stattfinden. Selbst wenn keine Teilsysteme wiederverwendet werden, kann hier von Migration gesprochen werden. Im Gegensatz dazu werden auch bei der direkten Wiederverwendung von Teilsystemen Änderungen notwendig und neu entwickelte Teile werden hinzugefügt. Die Begriffe Migration und Neuentwicklung schliessen sich bezüglich einer konkreten Anwendung also nicht gegenseitig aus. Sie treten dadurch in einem Anwendungsfall oft vermischt auf. Es ist sinnvoll, die Begriffe nicht auf gesamte Systeme, sondern nur auf Teilaspekte dieser anzuwenden.

Durchführung

Um eine Transformation auf Basis des Ausgangssystems zum Zielsystem durchzuführen, wird Software Reengineering verwendet. Reengineering wird oft als Synonym für Migration verwendet. [SPL03] sieht es als Methode dafür an. Dagegen spricht aber, dass viele Durchführungen eine komplette Neuimplementierung des Ausgangssystems ergeben. Deshalb ordnet [BLWG99] es eher diesem Bereich zu. Nachdem eine klare Trennung der Strategien nicht besteht, kann Reengineering sowohl der Migration als auch der Neuentwicklung zugeordnet werden. Dabei stellt es selbst keine Strategie dar, sondern eine Möglichkeit , diese in die Praxis umzusetzen. Vor allem Neuentwicklung und Migration, aber auch Wartung, werden dadurch ermöglicht.

In einem Anwendungsfall stellen sich einige Aufgaben. Diese erfordern jeweils unterschiedliche Bewältigungsstrategien. Das Reengineering ermöglicht dabei die Durchführung.

- Das Zielsystem soll funktional äquivalent zum Legacy System sein. Eine Migration oder funktional äquivalente Neuentwicklung ist notwendig.
- Das Zielsystem soll zusätzlich alle aktuellen Anforderungen implementieren. Eine Neuentwicklung von zusätzlichen Teilen, die diese Anforderungen erfüllen, muss durchgeführt werden.
- Der Datenbestand des Legacy Systems soll übernommen werden. Die Ausfallszeit bei der Umstellung soll minimal sein. Eine Migrationsmethode muss in die Durchführung eingebunden werden.

- Das Zielsystem soll moderne Technologien verwenden (Relationale Datenbanken, 3-Schichten-Architektur, Objektorientierung, ...). Das Legacy System soll in ein Zielsystem transformiert werden, dass diese Eigenschaften aufweist.

Zusammenfassend stellen die Bewältigungsstrategien Möglichkeiten dar, wie mit einem Legacy System umgegangen werden kann. Ist bei einem Anwendungfall der Migrationsaspekt vorhanden, werden Migrationsmethoden verwendet, um den Übergang vom Altsystem zum Zielsystem durchzuführen. Das Reengineering ist für diese Strategien Mittel zum Zweck, um das gewünschte Zielsystem zu realisieren.

2.4 Software Reengineering

Definition

> Reengineering ist die Untersuchung und Änderung eines bestehenden Systems mit dem Ziel, es in eine neue Form zu bringen und diese dann zu implementieren [CI90].

Diese Definition stammt von Chikofsky und Cross, die in [CI90] auch weitere Begriffe wie Forward und Reverse Engineering sowie Restructuring definieren. Sie orientieren sich dabei an 3 Phasen der Software-Entwicklung, den Anforderungen, dem Design und der Implementierung. Während das Forward Engineering eine übliche Software-Entwicklung darstellt, bei der diese Phasen in genannter Reihenfolge durchlaufen werden, geht das Reverse Engineering den umgekehrten Weg. Aus der Implementierung werden Design und Anforderungen rekonstruiert. Das Restructuring ändert die Struktur von Design und Code, bleibt dabei aber in der jeweiligen Phase.

Das Reengineering selbst besteht allgemein zuerst aus einem Reverse Engineering, danach folgt ein Forward Engineering oder ein Restructuring, bei dem das System zur gewünschten Zielform transformiert wird. Hier können auch gewünschte Änderungen eingebracht werden, falls neue Anforderungen zu erfüllen sind. Reengineering setzt sich zwar aus Forward und Reverse Engineering zusammen und verwendet auch deren Technologien, diese sind aber als eigenständig und unabhängig vom Reengineering zu sehen.

Reverse Engineering

Reverse Engineering ist der Prozess der Analyse eines bestehenden Systems mit dem Ziel

- der Identifikation von Systemkomponenten und deren Beziehungen untereinander, sowie
- der Erzeugung von Darstellungen des Systems in anderen Formen oder höheren Abstraktionsstufen.

Reverse Engineering ist ein Beobachtungsprozess. Es dient der Wiedererlangung von verloren gegangenen Informationen. Das betrachtete System wird dabei nicht geändert. Generell liefert es Designinformationen und abstraktere Systemdarstellungen, die unabhängig von einer Implementierung sind. Ergebnisse sind beispielsweise die Wiedergewinnung der ursprünglichen Anforderungen, die das System implementiert, oder die Rekonstruktion des Designs. Reverse Engineering kann in jeder der 3 Phasen und zu jeder Zeit im

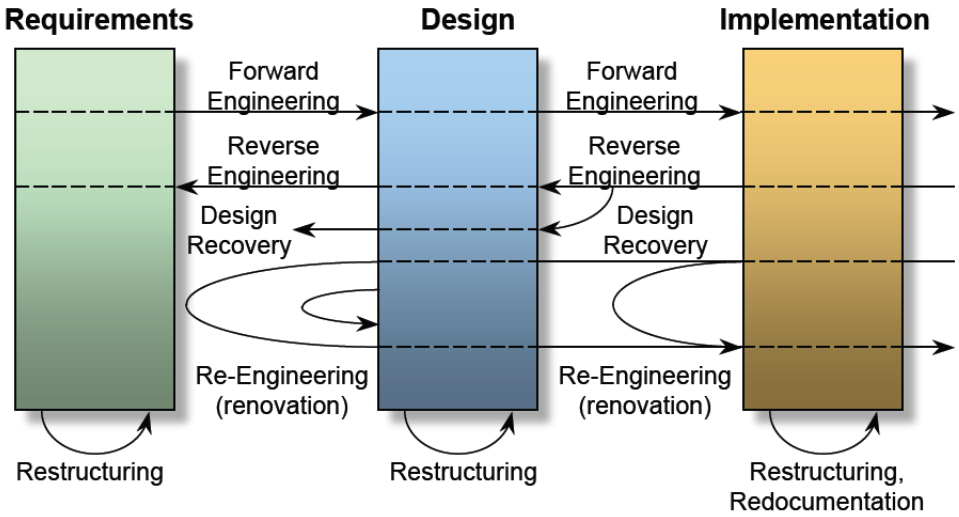


Abbildung 2.2: Aktivitäten des Software Reengineering

Entwicklungsprozess stattfinden. Es muss dabei nicht unbedingt auf Quellcode arbeiten.

2 bekannte Subprozesse des Reverse Engineering sind:

- **Redocumentation**
  Redocumentation ist die Erzeugung oder Überarbeitung von semantisch äquivalenten Repräsentationen innerhalb desselben Abstraktionsniveaus. Die Ergebnisse sind alternative Darstellungen des Systems, die für Menschen besser lesbar sind. Sie sollen ein besseres Verständnis für den Zusammenhang von Programmkomponenten ermöglichen. Redocumentation wird verwendet, um Dokumentation zu erhalten, die eigentlich bereits existieren sollte. Beispiele sind Datenfluss- und Kontrollflussdiagramme. Diese Darstellungen können automatisch generiert werden, beinhalten aber keine zusätzlichen Informationen.

- **Design Recovery**
  Design Recovery ist ein Prozess, in dem zusätzliche Information verwendet wird, um Abstraktionen des Systems auf höheren Abstraktionsstufen zu generieren. Wissen über Problemstellung und Anwendungsdomäne, bestehende Dokumentation und Code und Erfahrungen von Anwendungsexperten werden dafür verwendet. Es liefert die Informationen, um das System in seiner Gesamtheit verstehen zu können.

Forward Engineering

Das Forward Engineering ist der traditionelle Prozess der Software-Entwicklung, bei dem Anforderungen, Design und Implementierung durchlaufen werden. Man bewegt sich von höheren Abstraktionsstufen und logischen, implementierungsunabhängigen Designs zu der physischen Implementierung eines Systems. Obwohl es nicht notwendig ist, diesem Prozess mit Forward Engineering einen neuen Namen zu geben, sei er in diesem Zusammenhang so genannt, um ihn vom Reverse Engineering zu unterscheiden und die umgekehrte Richtung hervorzuheben.

Restructuring

Restructuring ist die Transformation von einer Repräsentationsform in eine andere auf dem selben Abstraktionsniveau. Dabei wird das äußere Verhalten des Systems funktional und semantisch beibehalten. Es wird also nur die innere Struktur verändert. Ein Beispiel ist eine Code-to-Code Transformation, die goto-Anweisungen durch strukturierte Äquivalente ersetzt. Das Restructuring wird allgemein verwendet, um bessere Struktureigenschaften und Modularisierung zu erhalten. Dadurch verbessern sich Wartbarkeit und Erweiterbarkeit eines Systems.

Refactoring

Refactoring ist diszipliniertes Restructuring im objektorientierten Paradigma. Refactorings sind semantikerhaltende, restrukturierende Code-Transformationen. Jede für sich bewirkt nur eine geringe Änderung an der Codestruktur, aber durch die kombinierte Anwendung wird eine wesentliche Gesamtverbesserung erreicht. Durch die kleinen Einzelschritte wird das Fehlerrisiko minimiert. Beschreibungen sind in [Fow99] und [Fow07] zu finden.

Anwendung

Das Ziel des Reverse Engineering ist das allgemeine Verständnis über das System zu erhöhen, um Wartung und Erweiterung zu unterstützen. Mittels Forward Engineering und Restructuring gelangt man durch Transformationen zum gewünschten Zielsystem. Zuvor wurden einige Aufgaben genannt, die mittels Reengineering gelöst werden können. Hier werden nun konkrete Systemeigenschaften beschrieben, die während dieser Aufgaben transformiert werden.

- **Datenmigration**
  z.B. Migration einer Flat File Datenbankstruktur zu einer relationalen Datenbankstruktur.
- **Migration der Benutzerschnittstelle**
  z.B. Transformation einer Kommandozeileneingabe zu einer graphischen Benutzeroberfläche.
- **Wechsel der Programmiersprache**
  z.B. Cobol zu Java.
- **Wechsel des Sprachparadigmas**
  z.B. prozedural zu objektorientiert. Dies ist meist mit einem Wechsel der Programmiersprache verbunden. Der Wechsel des Paradigmas im Reengineering ist unter dem Begriff Re-Architecting bekannt.

Dabei können im Zuge des Forward Engineering eine oder mehrere dieser Eigenschaften geändert werden. Ein Beispiel wäre ein System, dessen prozedurale Programmiersprache auf einer neuen Zielplattform nicht verfügbar ist. Deshalb soll das System zu einer neuen Sprache transformiert werden. Die Transformation beinhaltet den Wechsel der Programmiersprache und des Paradigmas. Ein weiteres Beispiel wäre der Wechsel des Datenbanksystems, um z.B. Performancesteigerungen zu erreichen. Die bestehende Datenbank wird zu einer neuen relationalen Datenbank migriert. Dabei werden die Daten mit übernommen. Die Eigenschaften der Anwendung selbst werden nicht verändert.

Architekturelle Sichtweise

Man kann eine Reengineering-Aufgabe unter verschiedenen Sichtweisen betrachten. Ein Beispiel wäre die Betrachtung auf Code-Ebene. Hier finden Aktivitäten wie Restrukturierungen und Programmiersprachenwechsel statt. Eine andere Sichtweise ist die architekturelle Sicht, bei der Reengineering als architekturelle Transformation durchgeführt wird. Als Beispiel dafür sei hier das Hufeisenmodell [KWC98] beschrieben, das verschiedene Sichtweisen in Zusammenhang stellt, sich dabei aber auf die Architektur konzentriert. Das Hufeisenmodell teilt den Reengineering-Prozess in 3 Subprozesse auf.

- **Rekonstruktion**
  Reverse Engineering, wobei die bestehende Architektur anhand der aus dem Code generierten Artefakte rekonstruiert wird.
- **Transformation**
  Die bestehende Architektur wird zur Zielarchitektur transformiert. Dies wird in diesem Schritt auf den abstrakten Architekturbeschreibungen durchgeführt, die zuvor rekonstruiert wurden.

- **Entwicklung**
  Forward Engineering anhand der Transformationsergebnisse zur Implementierung des gewünschten Zielsystems.

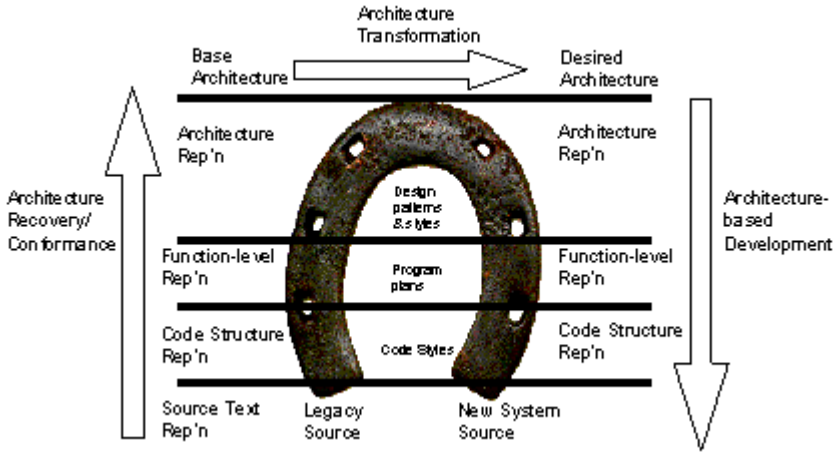


Abbildung 2.3: Das Hufeisenmodell

Ähnlich wie bei der Reengineering-Definition von [CI90] werden hier 3 Abstraktionsstufen in entgegengesetze Richtungen durchlaufen. Unterschiedliche Transformationen können in jeder dieser Stufen durchgeführt werden. Die architekturelle Transformation findet dabei auf der obersten Stufe statt. Sie führt beim Herabsteigen der Abstraktionsstufen zu niedrigerstufigen Transformationen.

- **Code-Struktur**
  Auf dieser Stufe befinden sich der Code und andere Darstellungen für diesen (Abstrakte Syntaxbäume, Flussgraphen, ...), die aus dem Code erzeugt wurden.
- **Funktionsstufe**
  Auf dieser Stufe werden die Zusammenhänge von Funktionen, Daten und Kombinationen dieser dargestellt.
- **Architekturelle Repräsentation**
  Auf dieser Stufe werden die Artefakte der vorherigen Stufen gruppiert und zu Mustern aus architekturellen Komponenten zusammengefügt.

Das Hufeisenmodell kann als eine konkretere Version des Modells von [CI90] gesehen werden, dass sich auf den architekturellen Aspekt des Reengineering-Vorganges konzentriert. Diese ist besonders wichtig, da die Architektur eines Systems einen wesentlichen Faktor für dessen Weiterentwicklung darstellt [SPL03].

2.5 Web-Anwendungen

Als gewünschte moderne Anwendung, die ein Legacy System ablösen soll, wurden bereits Web-Anwendungen genannt. Wie in der Einleitung beschrieben, bietet diese Anwendungskategorie eine Reihe von Vorteilen. Deshalb soll sie hier das Zielsystem für das Reengineering sein. Das Legacy System soll zu einer Web-Anwendung transformiert werden.

Unter Web-Anwendungen versteht man Anwendungen, die auf einem Web-Server ausgeführt werden und einen Web-Browser für die Interaktion mit dem Anwender verwenden. Die dabei verwendete Client-Server Kommunikation kann ortsunabhängig über bestehende Netzwerke erfolgen. Der Web-Browser kommuniziert mittels HTTP-Protokoll mit dem Web-Server. Die Benutzerschnittstelle für den Anwender besteht aus den Webseiten, die dabei übertragen werden.

Dynamik

Webseiten sind Dokumente, die in der textbasierten Auszeichnungssprache HTML (oder der XML-basierten Sprache XHTML) geschrieben sind. Der Web-Browser setzt diese in entsprechende graphische Darstellungen um. Ursprünglich wurden diese Dokumente statisch verwendet, da nur am Web-Server bereits bestehender HTML-Quellcode übertragen wurde. Mittels Technologien für Web-Anwendungen können Webseiten dynamisch erzeugt werden. Eine Web-Anwendung kann also auf gesendete Benutzereingaben in den Seiten reagieren und den Quellcode für die Antwortseite situationsabhängig erzeugen.

Adressierung

Zur Adressierung von Ressourcen wie Webseiten werden URLs (Uniform Resource Locator) verwendet. Dabei ist es für den Web-Browser transparent, ob es sich um eine statische Seite oder dynamisch generierten Inhalt handelt.

Kommunikation

Die Client-Server Kommunikation zwischen dem Web-Browser und dem Web-Server wird mit dem Kommunikationsprotokoll HTTP (Hypertext Transfer Protocol) abgewickelt. Es handelt sich um ein einfaches, Text-basiertes Protokoll zur Datenübertragung im World Wide Web. HTTP ist zustandslos. Nach erfolgreichem Senden wird die Verbindung beendet. Sollen weitere Daten übertragen werden, muss zuerst eine neue Verbindung aufgebaut werden.

Der Benutzer startet eine Web-Anwendung, indem er in einem Web-Browser den URL einer Web-Anwendung am Web-Server eingibt und damit die erste Anfrage (HTTP Request) sendet. Der Web-Server nimmt diese Anfrage entgegen und übergibt sie an die eigentliche Web-Anwendung. Diese verarbeitet die Daten der Anfrage und generiert daraufhin den HTML-Quellcode einer Webseite, welche vom Web-Server als Antwort zurück an den Browser des Benutzers geschickt wird (HTTP Response). Diese Seite wird dann vom Web-Browser dargestellt. Die graphische Benutzeroberfläche besteht also aus dynamisch generierten Webseiten, die als Reaktion auf Anfragen erzeugt werden.

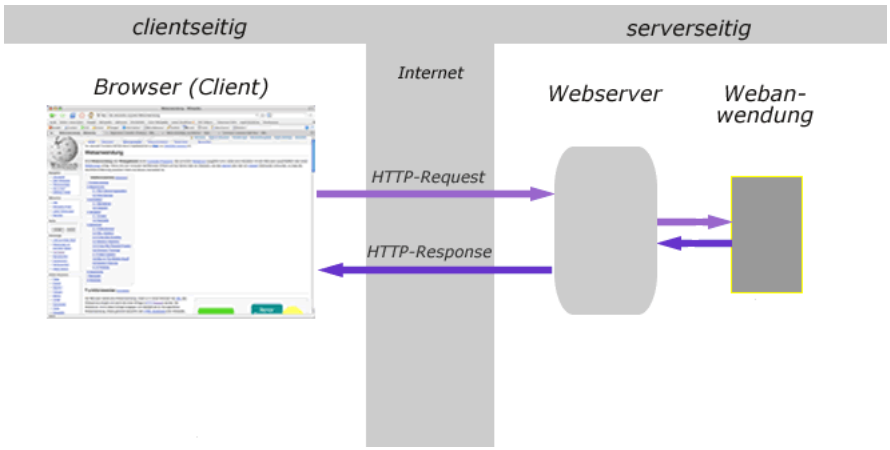


Abbildung 2.4: Client-Server Kommunikation einer Web-Anwendung

Es werden nun wichtige Eigenschaften von Web-Anwendungen beschrieben, die bei der Entwicklung und daher auch beim Reengineering beachtet werden sollten. Die wichtigste dabei ist die Anwendungsarchitektur.

Zustandsloses Kommunikationsprotokoll

Wie zuvor schon erwähnt, ist das Kommunikationsprotokoll HTTP zustandslos. Das bedeutet, dass verschiedene Anfragen an den Web-Server nicht in Verbindung zueinander stehen. Eine Anfrage weiß nichts von den vorherigen Anfragen des selben Clients. Viele Anwendungen erfordern jedoch, dass der Zustand, der sich durch eine Folge von Benutzeraktionen ergibt, berücksichtigt wird. So ist es auch bei Web-Anwendungen. Beispielsweise soll ein Benutzer einer Webmail-Anwendung erst dann Zugriff auf seine Mails erhalten, wenn er sich erfolgreich angemeldet hat. Weiters soll er nur seine eigenen Mails einsehen können. Auf die Mails anderer Benutzer des Systems hat er keinen Zugriff. Der Zustand muss also für jeden Client berücksichtigt

werden, da mehrere Benutzer gleichzeitig die Web-Anwendung verwenden können. Diese Sitzungen können jedoch nicht mittels HTTP-Protokoll implementiert werden, da dieses zustandslos ist. Um also eine zustandslose Anfrage des Clients korrekt seiner Sitzung zuordnen zu können, müssen zusätzliche Techniken verwendet werden. Dabei gibt es mehrere Möglichkeiten [Zdu02].

- **URL Encoding**
  Bei einer Anfrage können bei einem URL nach der eigentlichen Adresse zusätzliche Informationen hinzugefügt werden. Diese haben die Form Parameter=Wert und werden von der Adresse durch ein Fragezeichen getrennt. Sie stehen als Teil des URLs im Adressfeld des Web-Browsers und werden dadurch vom Benutzer gesehen. Wird die Anfrage dann zum Server gesendet, können diese Parameter dort ausgewertet werden. So kann zum Beispiel ein Sitzungsschlüssel zur Identifikation bei jeder Anfrage mit übertragen werden. Die Erzeugung dieser Schlüssel darf natürlich nicht nachvollziehbar sein. Da Benutzer die Möglichkeit haben, Parameter selbst im Adressfeld anzugeben, könnten sie damit unerlaubt die Sitzung eines anderen Benutzers verwenden.

- **Hidden Form Fields**
  Bei der Verwendung einer Webseite mit einem Formular können zusätzliche Informationen darin versteckt werden. Diese versteckten Formularfelder werden im Web-Broser nicht angezeigt. Sie sind jedoch im Quellcode ersichtlich.

- **Cookies**
  Cookies bieten eine Möglichkeit, Informationen auf der Client-Seite zu speichern und wieder auszulesen. Cookies bestehen aus Name und Wert, weiters aus Attributen wie zum Beispiel einem Ablaufdatum. Die Speicherung kann auch vom Server vorgenommen werden. Cookies können verwendet werden, um einen Sitzungsschlüssel beim Client abzulegen. Bei weiteren Anfragen werden die Informationen dann transparent an den Server mitgesendet. Der Benutzer hat jedoch die Möglichkeit, Cookies zu deaktivieren. Der Browser speichert dann keine Cookies mehr ab. Im Falle einer Web-Anwendung, die ihre Sitzung mittels Cookies realisiert, würde diese dann nicht mehr funktionieren.

Ereignisgesteuert

Der Programmfluss einer Web-Anwendung ist ereignisgesteuert. Die Ereignisse werden vom Benutzer durch Anfragen ausgelöst. Die Anfragen können Parameter im URL haben oder es werden Formulare zum Server geschickt. Die Interaktion geht also vom Client aus und der Server muss darauf rea-

gieren. Generell sind Anwendungen mit graphischen Benutzeroberflächen Benutzer-preemtiv. Der Benutzer erzeugt Ereignisse und die Anwendung muss darauf reagieren. Dabei entscheidet der Benutzer, wann die Ereignisse stattfinden. Dagegen sind die meisten Anwendungen mit zeichenorientieren Benutzerschnittstellen System-preemtiv. Die Interaktion geht dabei von der Anwendung aus. An bestimmten Stellen im Ablauf erwartet sie eine Benutzereingabe. Nach der Eingabe wird die Verarbeitung fortgesetzt. Der Ablauf besteht also aus einer sequentiellen Folge von Eingaben und Weiterverarbeitung. Er steht im Gegensatz zur ereignisgesteuerten Verarbeitung. Viele Legacy Systeme weisen zeichenotientierte Benutzerschnittstellen auf und sind System-preemptiv [MM00].

Anwendungsumgebung

Web-Anwendungen sind Anwendungen, die auf einem Web-Server ausgeführt werden. Auf der Clientseite findet nur Präsentation in Form von HTML-Dokumenten statt. Die Anwendung selbst läuft auf dem Web-Server. Sie verwendet die Benutzereingaben, die der Browser bei einer Anfrage mitsendet. Dafür müssen diese der Anwendungsumgebung zur Verfügung gestellt werden. Um solche Anwendungen zu ermöglichen, wurden verschiedene Technologien erfunden [Tur99], die nachfolgend beschrieben werden.

Common Gateway Interface

Ein Gateway realisiert den Zugriff auf eine Informationsquelle, so dass die Information für einen Web-Client wie eine Datei auf einem Web-Server erscheint. Der Mechanismus zur Kommunikation zwischen Web-Server und Anwendungsteil heißt Common Gateway Interface (CGI). Er ermöglicht Clients, Programme auf dem Web-Server zu starten und deren Ausgabe als Antwort zu erhalten. Diese Programme erzeugen oft HTML-Dokumente oder Bilder. Sie bekommen gesendete Eingaben und Informationen über die Anfrage als Umgebungsvariablen oder über die Standardeingabe zur Verfügung gestellt. Der Vorteil bei dieser Methode liegt in der Gateway-Anbindung. Dadurch können Programme gestartet werden, die mit beliebigen Programmiersprachen entwickelt werden. Es besteht eine strikte Trennung zwischen Web-Server und Programm. Der Nachteil dabei ist, das für jede Client-Anfrage ein eigener Prozess für die Verarbeitung gestartet wird. Die Folge sind erhebliche Einschränkungen in der Verfügbarkeit aufgrund des hohen Ressourcenverbrauchs.

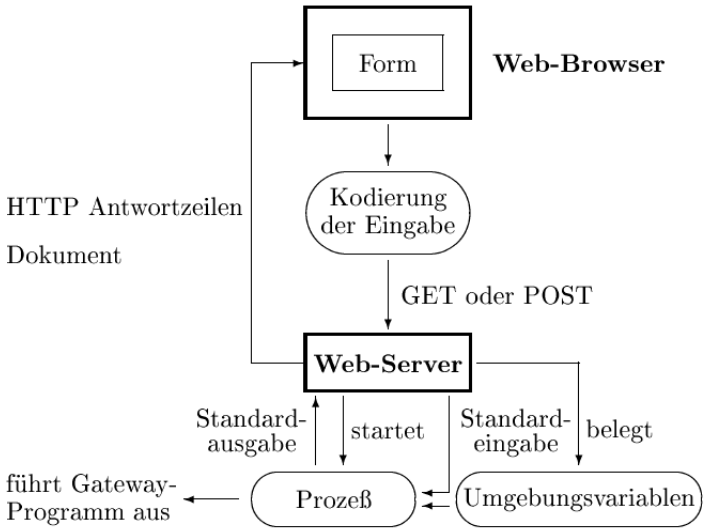


Abbildung 2.5: Common Gateway Interface

Server APIs

Server APIs sind modulare Erweiterungen für Web-Server. Sie verbleiben nach dem Laden im Speicher und können für weitere Anfragen wiederverwendet werden. Es werden keine einzelnen Prozesse gestartet. Die Kommunikation ist mittels Funktionsaufrufen realisiert. Es werden hier keine Umgebungsvariablen oder die Standardeingabe verwendet. Durch diese Eigenschaften bieten Server APIs einen wesentlichen Geschwindigkeitsvorteil gegenüber CGI. Sie sind allerdings produktspezifisch. Da die Anwendungen von Server APIs im gleichen Prozessraum wie der Web-Server ausgeführt werden, können Fehler in den Anwendungen den Server zum Absturz bringen. Das ist bei CGI nicht der Fall.

Servlets, JSP und JSF

JavaServer Pages (JSP) und Servlets von Sun microsystems [SM07] sind Beispiele solcher Server APIs. Sie verwenden die Programmiersprache Java und erweitern die Funktionalität des Web-Servers.

Servlets sind Java Klassen, die direkt über URLs angesprochen werden. Sie werden bei Bedarf instanziert und Laufen in einem Container. Mittels Servlet wird der Anwendungsteil realisiert wobei der Quellcode für die Antwort direkt erzeugt wird.

Bei JSPs werden Java-Anweisungen in HTML Dokumente eingebettet. Sie kombinieren damit statischen und dynamischen Inhalt. Die Anweisungen werden serverseitig ausgewertet, um den Quellcode der Seite zu erhalten.

JavaServer Faces (JSF) erweitern diesen Ansatz um ein komponentenorientiertes Rahmenwerk für Benutzerschnittstellen. Dabei wird nicht mehr direkt HTML verwendet, sondern Dokumente werden aus höher abstrahierten Komponenten zusammengesetzt. Diese werden dann für die Antwort vom Server zu HTML-Seiten gerendert. Weiters kann die Navigation zwischen den Dokumenten regelbasiert definiert werden.

Mehrschichtige Anwendungsarchitektur

Web-Anwendungen weisen mehrschichtige Architekturen auf. Diese Software-Architektur teilt die Struktur einer Anwendung in mehrere Schichten auf, die linear verbunden sind. Die Grundlage ist die 3-Schichten-Architektur. Sie sieht die Trennung von Benutzerschnittstelle, Anwendungslogik und persistenter Datenhaltung vor. Die Anwendungslogik selbst kann in weitere Schichten aufgeteilt sein. Dann spricht man von mehrschichtigen oder n-schichtigen Anwendungen.

Die 3-Schichten-Architektur entwickelte sich aus der 2-Schichten-Architektur. Diese sieht eine Trennung der Anwendung selbst und der verwendeten Daten vor. Bezogen auf die Client-Server Aufteilung übernimmt hier der Client die Präsentation und Anwendungslogik, während die Datenbank auf dem Server läuft. Um die Nachteile dieser Aufteilung zu vermeiden, wurden Präsentation und Logik getrennt. Die Logik wurde auf die Serverseite verschoben, während die Präsentation auf der Clientseite verblieb. Dabei können Anwendungsteil und Datenbank auch auf verschiedenen Rechnern betrieben werden und kommunizieren wie zuvor über das verbindende Netzwerk miteinander. Das Ergebnis war die 3-Schichten-Architektur.

Aufteilung in 3 Schichten:

- **Präsentationsschicht**
  Diese Schicht stellt die Benutzerschnittstelle dar. Sie ist für die Präsentation von Daten zuständig und nimmt Benutzereingaben entgegen. Bei Web-Anwendungen handelt es sich um den Web-Browser, der die HTML-Dokumente darstellt, die er vom Web-Server erhalten hat.
- **Logikschicht**
  Die Logikschicht stellt den eigentlichen Anwendungsteil dar. Hier findet die Programmverarbeitung statt. Die Logikschicht wird auch mittlere Schicht genannt, da sie zwischen Präsentation und Datenschicht steht.

- **Datenschicht**
  Die Datenschicht ist für die persistente Datenhaltung zuständig. Sie wird durch Datenbanken realisiert.

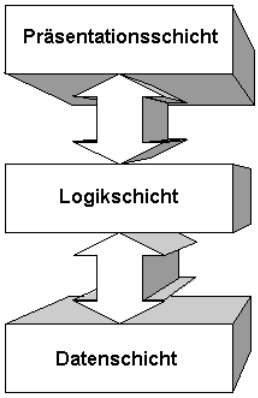


Abbildung 2.6: 3-Schichten-Architektur einer Web-Anwendung

Diese 3 Schichten sind linear miteinander verbunden. Das bedeutet, dass eine Schicht immer nur mit der nächstgelegenen Schicht kommunizieren darf. Zuvor wurde schon erklärt, wie die Client-Server Kommunikation von Web-Anwendungen funktioniert. Dabei kommunizieren Präsentationsschicht und Logikschicht miteinander. Benutzereingaben werden dem Anwendungsteil übergeben. Dieser greift dann bei Bedarf auf die Datenschicht zu, verarbeitet die Daten und erzeugt entsprechende Antworten. Die Logikschicht steht also zwischen der Präsentationsschicht und der Datenschicht. Es gibt keine direkte Kommunikation zwischen der Präsentationsschicht und der Datenschicht.

Eine Verwendung dieser Architektur bringt wesentliche Vorteile mit sich. Die Trennung der Präsentation von Logik und Daten in Zusammenhang mit der Verwendung von Web-Standards ist die Grundlage für die anfangs erwähnten Vorteile von Web-Anwendungen. Die Aufteilung in 3 Schichten bietet weitere Vorteile auf der Serverseite.

- **Skalierbarkeit**
  Mehrschichtige Anwendungen sind gut skalierbar, da die Schichten physisch getrennt betrieben werden können. Während die Präsentation beim Client stattfindet, können Anwendungslogik und Datenbank auf verschiedenen Servern betrieben werden.

- **Wartbarkeit**
  Durch die lineare Schichtenaufteilung können einzelne Schichten einfach ausgetauscht werden. Es ist beispielsweise möglich, den Datenbankserver der Datenschicht gegen einen anderen auszutauschen, ohne

die anderen Schichten ändern zu müssen.

Zusammenfassung

Die nachfolgende Graphik soll die zuvor gezeigten Eigenschaften von Web-Anwendungen in einem Gesamtbeispiel verdeutlichen. Dieses Beispiel verwendet für den Anwendungsteil einen vom Web-Server eigenständigen Anwendungsserver. Praktisch können Web-Server und Anwendungsserver auch integriert vorkommen. Wesentlich ist die HTTP Kommunikation des Servers mit dem Client und die Realisierung des Anwendungsteils auf der Serverseite.

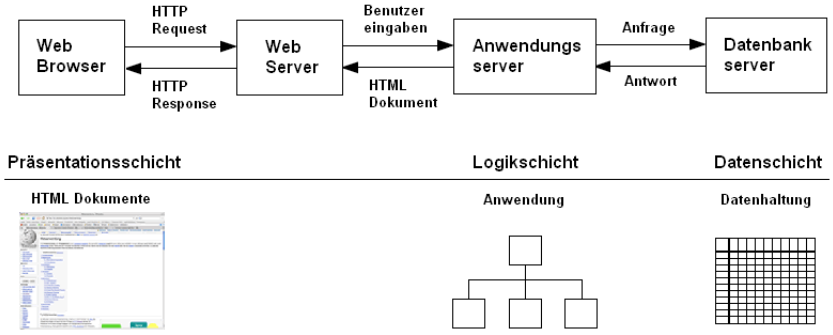


Abbildung 2.7: Kontrollfluss in der Anwendungsarchitektur

Die Präsentationsschicht wird durch den Web-Browser ermöglicht, der die vom Web-Server empfangenen HTML-Dokumente darstellt. Der Benutzer kann Eingaben machen und diese zur Anwendungsschicht senden. Die Kommunikation geschieht hier über den Web-Server mittels dem zustandslosen Kommunikationsprotokoll HTTP. Der Web-Server übergibt dann weiter an die Logikschicht. Der Anwendungsteil wird hier durch einen eigenen Anwendungsserver implementiert. Die Logik verarbeitet die Eingaben und greift dabei bei Bedarf auf den getrennt laufenden Datenbankserver der Datenschicht zu. Abschließend generiert sie eine Antwort als HTML-Dokument, welches vom Web-Server zurück an den Web-Browser gesendet wird.

Kapitel 3

Der Web-Reengineering Prozess

3.1 Allgemeine Beschreibung

3.1.1 Aufgabenstellung

Aufgrund der Vielzahl an möglichen Legacy Systemen und auch Zielsystemen wurden viele spezielle Vorgangsweisen entwickelt, um von einem konkreten Ausgangssystem zu einem konkreten Zielsystem zu gelangen. Oft wurden sie erst im Zuge eines Migrationsprojektes entwickelt [SPL03]. Sie spezialisieren sich oft auf einen bestimmten Aspekt der Migration, wie die COREM Methode [KG95] auf das Sprachparadigma. Sie kann zur Transformation von prozeduralem Code in eine objektorientierte Umgebung verwendet werden. Andere Methoden, wie beispielsweise das Hufeisenmodell, befinden sich auf einem hohen Abstraktionsniveau. Ein praktischer Einsatz setzt zuerst eine Konkretisierung voraus.

Für die Transformation eines Legacy Systems zu einem Zielsystem Web-Anwendung, wird Reengineering als Methode zur Umsetzung verwendet. Es ermöglicht dabei:

- **Wartung**
  Die Verbesserung der Struktur bereits am Ausgangssystem. Dadurch ergibt sich schon vor der Transformation eine verbesserte Wartbarkeit und Erweiterbarkeit.
- **Migration**
  Es soll möglichst viel vom Legacy System wiederverwendet werden, um den hohen Aufwand einer vollständigen Neuentwicklung zu vermeiden.

- **Neuentwicklung**
  Nicht migrierbare Teile müssen abgeändert und neu entwickelt werden.
- **Weiterentwicklung**
  Zusätzliche Funktionalität soll bereits während der Transformation eingebracht werden. Eine nachträgliche Weiterentwicklung soll durch gute Wartbarkeit und Erweiterbarkeit ermöglicht werden.

Um das Thema Reengineering zu Web-Anwendungen zu untersuchen, wird in dieser Arbeit für diesen Zweck ein eigener Prozess definiert. Dieser geht zunächst von [CI90] und [KWC98] aus. Die Neuerung gegenüber diesen besteht in der Konkretisierung für das Zielsystem Web-Anwendung, durch die eine Berücksichtigung der Zielsytemeigenschaften bereits zu Beginn des Prozesses möglich wird und die Reengineering-Aktivitäten daran angepasst werden können.

Der grundlegende Ansatz dabei ist die architekturelle Transformation. Da als Zielsystem Web-Anwendungen festgelegt sind, ist die Umstellung auf eine 3-schichtige Architektur ein wesentlicher Faktor. Deshalb soll diese an einem möglichst frühen Zeitpunkt im Verlauf des Prozesses berücksichtigt werden. Diese Arbeit verwendet gezielte Restrukturierungen, um dies zu erreichen. Dadurch kann danach auch das Forward Engineering leichter an das Zielsystem angepasst werden. Die Möglichkeiten dafür und die Auswirkungen auf die Durchführung des Reengineerings, die der in dieser Arbeit definierte architekturelle Ansatz hat, sollen untersucht werden.

Weiters soll dabei das objektorientierte Paradigma am Zielsystem unterstützt werden. Einerseits bietet es eine gute Möglichkeit, dauerhaft Wartbarkeit und Erweiterbarkeit am Zielsystem zu gewährleisten. Andererseits verwenden moderne Web-Umgebungen auch häufig Sprachen dieses Paradigmas. Obwohl der Prozess nicht für eine konkrete Technologie definiert wird, soll dies dennoch damit berücksichtigt werden.

Eigenschaften, die der Prozess haben soll:

- **Abstraktionsniveau**
  Er soll sich nicht auf ein bestimmtes Ausgangssystem festlegen. Dies gilt vor allem für die verwendeten Technologien. Dadurch wird ein höheres Abstraktionsniveau des Prozesses notwendig. Es wird nicht auf Implementierungsebene operiert, sondern auf der Ebene der Struktur und Architektur.

  Das Abstraktionsniveau soll allerdings nicht höher als notwendig angesetzt werden. Dadurch soll sich eine spezifische Vorgehensweise auf struktureller und architektureller Ebene ergeben, die leichter umgesetzt werden kann. Die jeweiligen Systemeigenschaften des Ausgangssystems bleiben jedoch offen.

- **Prozess-Struktur**
  - **Festgelegte Schritte**
    Der Prozess führt geradlinig vom Ausgangssystem zu Zielsystem. Dabei ist er in Teilschritte unterteilt. Jeder Teilschritt verlangt festgelegte Ausgangsinformationen und liefert festgelegte Ergebnisse. Die Ergebnisse jeden Teilschrittes sind die Ausgangsinformationen für den darauf folgenden Teilschritt. Diese Aufteilung soll die Durchführung vereinfachen und aufgrund der Teilergebnisse für eine bessere Übersicht sorgen.
  - **Flexibilität innerhalb der Schritte**
    Wie innerhalb der Teilschritte zu den Ergebnissen gelangt wird, bleibt offen. Dadurch soll Flexibilität in der Anwendung auf verschiedene Ausgangssysteme erreicht werden. Wie zuvor schon erwähnt, soll der Prozess gerade ein solches Abstraktionsniveau haben, dass konkrete Eingeschaften des Ausgangssystems, wie beispielsweise die Programmiersprache, nicht festgelegt sind.
- **Festlegung auf das Zielsystem**
  Da die Eigenschaften des Zielsystems bezüglich Architektur und Sprachparadigma bekannt sind und die Web-Anwendung das Legacy Ausgangsystem ablösen soll, wird der Prozess für dieses Zielsystem konkretisiert.
  - Unterstützung der Transformation zu einer 3-schichtigen Architektur.
  - Unterstützung der Transformation zu einer Sprache des objektorientierten Paradigmas.
  - Unterstützung der Migrationsmethoden

3.1.2 Prozessdefinition

Der Prozess basiert auf den allgemeinen Reengineering-Aktivitäten Reverse Engineering, Restrukturierung und Forward Engineering. Diese werden als 3 Teilschritte des Prozesses in Reihenfolge gesetzt. Nach Durchlaufen dieser Schritte erhält man als Ergebnis eine zum Ausgangssystem funktional äquivalente Web-Anwendung in der gewählten Zielumgebung.

Im Laufe der Prozessbeschreibung wird immer von unterschiedlichen Technologien für Aspekte des Ausgangssystems und des Zielsystems ausgegangen. Zum Beispiel wird die Transformation in eine objektorientierte Umgebung besprochen. Verwendet das Ausgangssystem bereits eine Sprache dieses Paradigmas, sind Aktionen zur Umsetzung selbstverständlich nicht notwendig.

Dieser Trivialfall wird im Zuge der Beschreibung nicht mehr erwähnt.

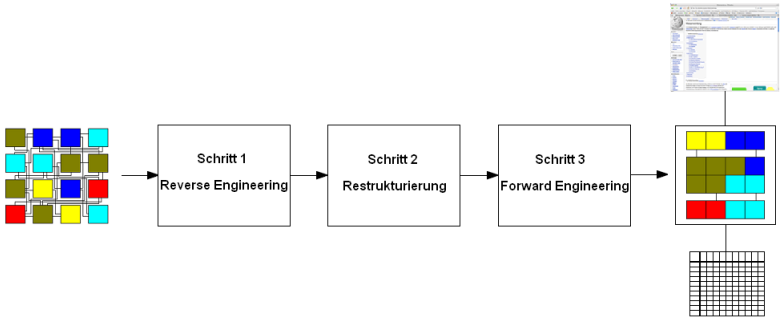


Abbildung 3.1: Der Web-Reengineering Prozess

Schritt 1 - Reverse Engineering

Reverse Engineering wird zunächst benötigt, um verlorenes Wissen über das Ausgangssystem wiederzuerlangen. Es wird verwendet, um Informationen über Eigenschaften, Funktionalität und Struktur des Systems zu rekonstruieren. Mit der Durchführung wird ein Verständnis über das System entwickelt, das wichtig für das weitere Vorgehen ist. Dokumentation über die Anforderungen an das System sowie von Struktur und Architektur sind die Ergebnisse dieses Schrittes.

Die Rekonstruktion der Anforderungen ist wichtig, da diese die Funktionalität festlegen, die auch das Zielsystem umsetzen soll. Dokumentation über Struktur und Architektur des Systems soll den inneren Aufbau verständlich machen. Diese kann teilweise automatisch aus dem Quellcode erzeugt werden. Als Ergebnis erhält man abstrakte Darstellungen in Form von Diagrammen, die bestimmte Sichtweisen auf den Code bieten. Sie stellen die Zusammenhänge zwischen einzelnen Programmeinheiten dar. Dadurch wird die innere Struktur des Systems erfasst. Diese wiedererlangten Informationen sollen Verständnis für das Ausgangssystem schaffen. Sie stellen die Basis für weitere Reengineering-Aktivitäten dar.

Die Anforderungen des Ausgangssystems werden im dritten Schritt verwendet und dort an das Zielsystem angepasst. Die erzeugten Diagramme werden als Grundlage für den nächsten Schritt, die Restrukturierung, verwendet.

Schritt 2 - Restrukturierung

Die Ergebnisse des ersten Schrittes werden benötigt, um diesen Schritt durchführen zu können. Die Restrukturierung verändert die interne Struktur des Ausgangssystems. Die Funktionalität bleibt dabei gleich. Das Verhalten des Systems nach außen hin ändert sich nicht. Dieser Schritt soll die Struktur verbessern und das Ausgangssystem für die Transformation zum Zielsystem Web-Anwendung vorbereiten. Das wird erreicht, indem der Übergang zu Zielarchitektur und Zielparadigma bereits hier unterstützt wird. Das Ergebnis wird dokumentiert, indem die Ergebnisse des ersten Schrittes entsprechend modifiziert werden.

Zuerst wird die allgemeine Struktur des Ausgangssystems verbessert. Durch Remodularisierung ergibt sich eine bessere Aufteilung der strukturellen Programmeinheiten. Das vereinfacht die Transformation im dritten Schritt und begünstigt das objektorientierte Paradigma. Es ermöglicht eine bessere Übersicht über die Gesamtstruktur und damit tieferes Programmverständnis. Es vereinfacht auch die zweite Restrukturierung dieses Schrittes. Das Zwischenergebnis ist eine strukturell verbesserte Variante des Ausgangssystems.

Durch die Einteilung nach bestimmten Eigenschaften können Einheiten mit Benutzerschnittstellen und solche mit Datenbankzugriffen zusammengefasst werden. Die strukturell verbesserte Variante wird nun nach diesen Kriterien restrukturiert. Es ergeben sich daraus 3 Teile. Ein Teil enthält Programmeinheiten mit Benutzerschnittstellenfunktionalität, ein weiterer enthält Datenbankzugriffe. Der verbleibende Teil enthält nur mehr Anwendungslogik. Diese Aufteilung vereinfacht die Transformation im dritten Schritt. Sie berücksichtigt die 3-Schichten-Architektur des Zielsystems bereits hier. Die Abhängigkeiten der Programmeinheiten untereinander werden verringert, da die 3 Teile wie bei der 3-Schichten-Architektur linear angeordnet sind.

Das Ergebnis ist eine strukturell verbesserte Variante des Ausgangssystems, die in 3 Teile untergliedert ist. Diese ist weiterhin funktional äquivalent zum Ausgangssystem. Die erhaltene Programmstruktur ist die Basis für die Transformation zur 3-Schichten-Architektur im dritten Schritt.

Schritt 3 - Forward Engineering

In diesem Schritt werden wie beim traditionellen Prozess der Softwareentwicklung Anforderungen, Design und Implementierung durchlaufen. Der Unterschied ist, dass die Ergebnisse der vorherigen Schritte als Grundlage verwendet werden. Das Ziel ist die Transformation zum Zielsystem Web-Anwendung.

Bei der Anforderungsanalyse werden zunächst die Anforderungen des Ausgangssystems als Grundlage verwendet. Dann wird entschieden, welche Anforderungen abgeändert werden müssen. Ursprüngliche Anforderungen sind unter Umständen am Zielsystem nicht mehr angebracht oder nicht mehr notwendig. Neue Anforderungen an das Zielsystem können hinzukommen. Ein Beispiel ist der Zugriff des Ausgangssystems auf lokale Systemresourcen, die mittels Web-Browser nicht umsetzbar sind. Das Ergebnis ist hier die Dokumentation der Anforderungen an das Zielsystem.

Das Design wird ebenfalls basierend auf den Ergebnissen der vorherigen Schritte erstellt. Es stellt die Umsetzung der zuvor festgelegten Anforderungen in einem Zielsystem auf hohem Abstraktionsniveau dar. Die Transformation entsprechend den Zielsystemeigenschaften, wie objektorientiertes Paradigma, 3-Schichten-Architektur, ereignisbasierte Verarbeitung und Datenhaltung in einem modernen Datenbanksystem, wird hier entworfen. Die Zieltechnologie wird hier ebenfalls festgelegt. Durch die zuvor gemachten Restrukturierungen kann das Ergebnis aus dem zweiten Schritt strukturell als Basis für die Erstellung des objektorientierten Designs verwendet werden. Weiters können aufgrund der bereits im Ausgangssystem durchgeführten Schichtentrennung die Schichten der Zielarchitektur getrennt behandelt werden. Ein objektorientiertes Design des Zielsystems wird erstellt. Die Anwendung wird auf eine ereignisbasierte Verarbeitung umgestellt. Dafür wird die Benutzerschnittstelle in äquivalente HTML-Seiten umgesetzt. Die Anwendungslogik wird daran angepasst, indem ein Zustand für jede Sitzung eines Clients gespeichert wird, der eine zustandbasierte Anwendung ermöglicht. Für die Datenhaltung wird ein Datenmodell für das neue Datenbanksystem unter Berücksichtigung des ursprünglichen Modells erstellt. Die Anpassung kann lokal in der Schicht mit den Datenzugriffen erfolgen. Die Zugriffe auf das Datenbanksystem sind damit für die anderen Schichten nicht ersichtlich. Weiters wird im Zuge des Designs die Durchführung der Migration geplant.

Anschließend wird die Transformation zum Zielsystem durchgeführt. Dabei wird das Design in eine Implementierung der Web-Anwendung umgesetzt. Diese erfüllt alle zuvor festgelegten Systemeigenschaften. Die Web-Anwendung ist durch den durchgeführten Prozess funktional äquivalent zum Legacy System und kann es daher ablösen. Durch die modernen Technologien vermeidet sie dessen Nachteile und kann in Zukunft einfach gewartet und erweitert werden. Durch die Transformation zu einer Web-Anwendung erhält man alle damit verbundenen Vorteile solcher Systeme. Testen der Implementierung gewährleistet die Erfüllung der funktionalen Äquivalenz zum Legacy System. Durch die Durchführung der Migration im Zuge der Transformation kann der alte Datenbestand übernommen werden. Dadurch wird die Dauer der Umstellung minimiert und die Web-Anwendung kann äquivalent weiterverwendet werden. Damit ist der Prozess abgeschlossen.

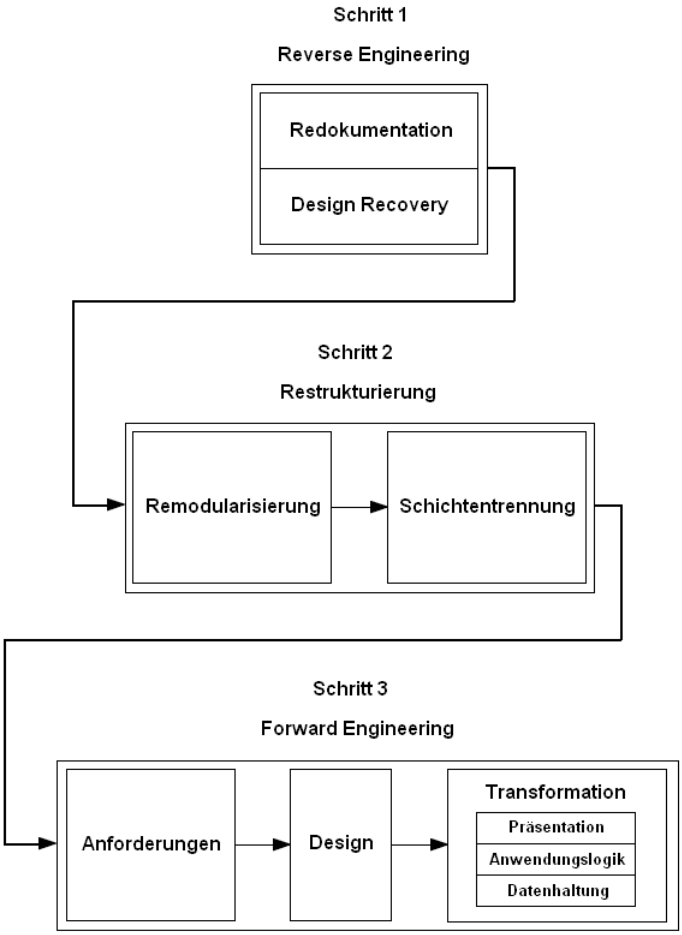


Abbildung 3.2: Ablauf des Web-Reengineerings

Der Rest dieses Kapitels beschreibt die 3 Schritte im Detail. Dabei wird bei jedem Schritt folgende Struktur eingehalten, um systematisch vorgehen zu können.

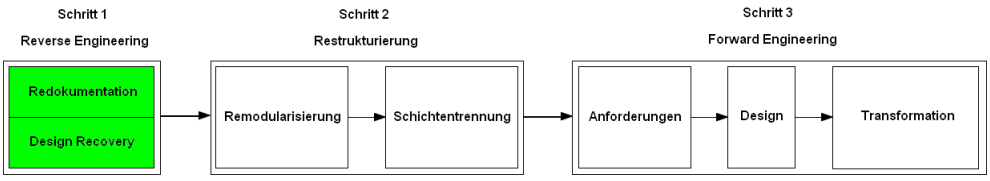
- **Ziele**
  Die Zieldefinition beschreibt, was in diesem Schritt erreicht werden soll.
- **Ausgangsbasis**
  Die Ausgangsbasis beschreibt die Vorraussetzungen und Eingaben des Schrittes. Diese müssen erfüllt und vorhanden sein, um eine Durchführung zu ermöglichen.
- **Aktivitäten**
  Die Aktivitätenbeschreibung schildert, wie die zuvor festgelegten Ziele

aufgrund der Ausgangsbasis erreicht werden. Sie beschreibt allgemein die Problematik und die Handlungen, die notwendig sind, um zu den Ergebnissen zu gelangen.

- **Methoden**
  Hier werden bereits existierende Methoden für die Aktivitäten beschrieben. Diese sind so konkret, dass sie direkt für eine Durchführung verwendet werden können. Weiters werden hier auch Werkzeuge für die Methoden beschrieben, die als Hilfestellung dienen können oder automatisch Ergebnisse für den Schritt liefern.

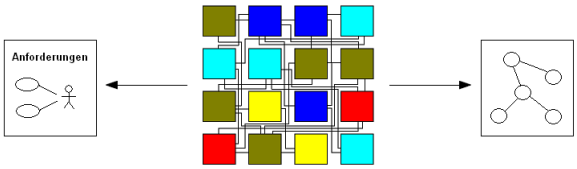
- **Ergebnisse**
  Die Ergebnisse beschreiben das Resultat der Durchführung der Aktivitäten. Eine Vorschau auf die folgenden Schritte zeigt, wo und wofür sie im Zuge des Prozesses weiter verwendet werden.

3.2 Schritt 1 - Reverse Engineering



Ziele

In diesem Schritt wird das System analysiert und Dokumentation darüber erstellt. Das dabei erlangte Wissen ermöglicht die Entwicklung eines Verständnisses über das zu analysierende System. Dies spielt beim Reengineering eine zentrale Rolle. Es beinhaltet zum wesentlichen Teil, dass Entwickler ohne Kenntnisse über das Ausgangssystem die Bestandteile und Funktionen kennenlernen [BR04]. Erst dann ist es möglich, umfassende Restrukturierungen vorzunehmen, ohne die Funktionalität zu beeinträchtigen, und die Transformation zum Zielsystem durchzuführen.



Die Erzeugung von abstrakten Darstellungen des Ausgangssystems ist dabei ein wesentlicher Faktor. Sie unterstützen das Programmverständnis der Entwickler, indem sie die Zusammenhänge verdeutlichen. Weiters werden diese Darstellungen als Basis für die folgenden Restrukturierungen verwendet.

Ausgangsbasis

Die Ausgangsbasis dieses Schrittes ist der Quellcode und eine lauffähige Version des Ausgangssystems. Bereits vorhandene Dokumentation kann verwendet werden, um schon vor dem Reverse Engineering Verständnis für das System zu schaffen.

Aktivitäten

Dokumentation der Anforderungen

Die Anforderungen des Ausgangssystems sollen rekonstruiert werden. Dabei werden funktionale und nicht-funktionale Anforderungen erfasst und dokumentiert. Diese stellen eine grundlegende Systembeschreibung des äußeren Verhaltens dar. Funktionale Anforderungen beschreiben, welche Leistungen das System erbringt. Sie definieren die Funktionalität des Systems. Nichtfunktionale Anforderungen beschreiben die Rahmenbedingungen, unter denen diese Funktionalität erbracht wird. Die Festlegung der Anforderungen ist der Ausgangspunkt in der traditionellen Softwareentwicklung. Durch sie wird festgehalten, zu welchem Zweck das System entwickelt wird. Durch die hier erfolgende Rekonstruktion können sie im dritten Schritt als Basis für die Anforderungsdefinition des Zielsystems verwendet werden.

Darstellungen des Designs

Abstrakte Darstellungen von Designinformationen sollen aus dem Quellcode des Ausgangssystems erzeugt werden. Sie stellen unterschiedliche Abhängigkeiten von Einheiten des Programms zueinander dar und bieten verschiedene Sichtweisen darauf. Sie unterstützen einerseits das Programmverstehen und stellen andererseits die Basis für Restrukturierungsaktivitäten dar.

Als Repräsentationen eignen sich gut graphenbasierte Darstellungsformen. Dabei werden Programmeinheiten als Knoten und deren Beziehungen untereinander als Kanten zwischen diesen Knoten dargestellt. Graphen bieten gute Möglichkeit, Abhängigkeiten für die Entwickler auf „natürliche“ Art zu visualisieren [FKKQ05].

Dabei sollte die hierarchische Ebene so gewählt werden, dass eine sinnvolle Darstellung erhalten wird. Dies ist vor allem bei großen, komplexen Systemen wichtig, da die Visualisierung von Graphen aufgrund der Vielzahl von dargestellten Abhängigkeiten oft für Menschen nicht zu gebrauchen ist. Dann ist es sinnvoll, die Struktur auf einer höheren Ebene darzustellen. Ein Beispiel ist die Darstellung auf Modulebene, falls ein Graph auf Funktionsebene den Entwicklern keine Erkenntnisse bringt.

Die Erzeugung von Graphendarstellungen kann teilweise automatisch mittels Werkzeugen durchgeführt werden. Diese Werkzeuge sind allerdings quellsprachspezifisch und müssen deshalb abhängig von der Sprache des Ausgangssystems gewählt werden.

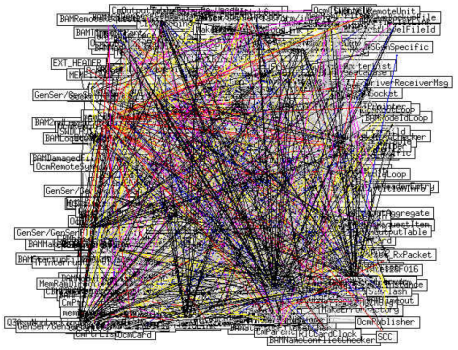


Abbildung 3.3: Beispiel für eine Graphendarstellung auf zu niedriger Abstraktionsebene

Methoden

Dokumentation der Anforderungen

Eine Möglichkeit zur Dokumentation von funktionalen Anforderungen sind die Anwendungsfälle der Unified Modeling Language UML. Damit stehen Elemente für die Modellierung von Anforderungen zur Verfügung. Die einzelnen Anwendungsfälle beschreiben das Verhalten eines Systems nach außen hin. Sie spezifizieren eine Menge von Aktionen, die ein System ausführen muss, um ein beobachtbares Resultat zu erzeugen. Mehrere Anwendungsfälle können gruppiert und zueinander in Relation gesetzt werden. Ein weiteres Element neben den Anwendungsfällen stellen die Akteure dar. Diese interagieren mit dem System. Sie lösen Anwendungsfälle aus und erhalten deren Ergebnisse. Akteure können Menschen oder auch andere Systeme sein. Neben dem Modell wird jeder Anwendungsfall mit beschreibendem Text dokumentiert. Der Ablauf, Vorbedingungen, Nachbedingungen und weitere Eigenschaften, werden damit festgehalten. Sie dienen als genaue Beschreibung des Anwendungsfallverhaltens.

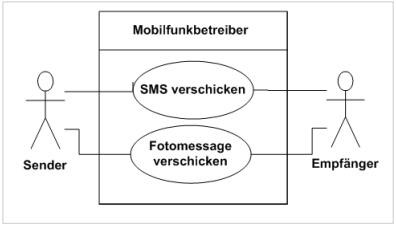


Abbildung 3.4: Beispiel für ein Anwendungsfalldiagramm

Die Anwendungsfälle ermöglichen es, die Funktionalität eines Systems systematisch textuell zu beschreiben und graphisch als Diagramm darzustellen. Dadurch wird das gesamte Verhalten des Systems nach außen hin dokumentiert und stellt eine Spezifikation der funktionalen Anforderungen dar. Eine genaue Beschreibung der UML Anwendungsfälle ist in [Mac01] zu finden.

Neben der Funktionalität sollen auch die nicht-funktionalen Anforderungen dokumentiert werden. Diese werden meist informal als aufzählender und beschreibender Text festgehalten. Ein Beispiel wäre „Die Antwortzeit muss <= 1 Sekunde sein.“.

Darstellungen des Designs

Designinformationen können mittels verschiedener Abhängigkeitsgraphen dargestellt werden. Die Knoten dieser Graphen stellen verschiedene Einheiten des Programmcodes dar. Die Beziehungen dieser Einheiten zueinander werden durch die Kanten repräsentiert. Graphendarstellungen können teilweise automatisch aus dem Quellcode erzeugt werden.

Folgend werden verschiedene Arten von Abhängigkeitsgraphen beschrieben. Diese vermitteln unterschiedliche Sichtweisen auf die Struktur eines Systems. Der Kontrollflussgraph stellt das Zusammenwirken der funktionalen Bestandteile eines Systems dar. Man kann zwei Ebenen von Kontrollflüssen mit zwei spezifischen Darstellungsformen unterscheiden [BR04]. Datenfluss und Modulabhängigkeit bieten weitere Sichtweisen.

- **Intraprozeduraler Kontrollflussgraph**

  Beim intraprozeduralen Kontrollflussgraphen sind Anweisungen bzw. Anweisungsblöcke die kleinsten funktionalen Einheiten. Jede imperative Programmiersprache stellt Anweisungen zur Steuerung des Kontrollflusses als auch Anweisungen für die Speichermanipulation. die Zuweisung, zur Verfügung. Ein Kontrollflussgraph gibt die mögliche zeitliche Ausführungsreihenfolge der Anweisungen wieder.

- **Interprozeduraler Kontrollflussgraph (Aufrufgraph)**

  Beim interprozeduralen Kontrollfluss sind Subroutinen (Funktionen, Prozeduren und Methoden) die kleinsten funktionalen Einheiten. Jeder Knoten des Graphen stellt eine Subroutine dar. Diese können sich gegenseitig aufrufen. Im Gegensatz zum Kontrollfluss auf Anweisungsebene folgt einem Aufruf dabei immer der Rücksprung zum Aufrufort. Solche Beziehungen können durch einen Aufrufgraphen repräsentiert werden.

  Die Erzeugung eines solchen Graphen gelingt nur bei Programmiersprachen ohne späte Bindung durch statische Analyse auf eindeutige

Weise. Eine konkrete Eigenschaft objektorientierter Sprachen sind jedoch polymorphe Aufrufstrukturen, bei denen erst zur Laufzeit entschieden wird, welche Methode durch eine Aufrufanweisung aktiviert wird. Erst der Übergang zu dynamischer Analyse kann dann die exakten Aufrufstrukturen hervorbringen.

- **Datenflussgraph**

  Ein Datenflussgraph stellt die Beziehungen zwischen Zuweisungen an Variablen und deren Benutzung dar. Eine Abhängigkeit zwischen 2 Einheiten besteht dann, wenn eine Einheit eine Variable verändert und die andere diese Variable verwendet. Wie zuvor beim Kontrollflussgraphen kann dieser Graph intraprozedural oder interprozedural dargestellt werden.

- **Modulabhängigkeitsgraph**

  Ähnlich wie der Aufrufgraph stellt der Modulabhängigkeitsgraph die Abhängigkeiten der einzelnen Module untereinander dar. Die Struktur wird auf einer noch höheren Ebene als der prozeduralen betrachtet. Im Vergleich nimmt hier die Detailgenauigkeit ab, da Einheiten auf Modulebene die Knoten des Graphen darstellen.

Die hier beschriebenen Graphen sind nicht die einzig möglichen Darstellungsformen. Es existiert noch eine Reihe von weiteren Graphen. Weiterführende Informationen zu System Dependence Graph, Resource Flow Graph und Ansätze für die Darstellung von objektorientierten Systemen sind in [BR04] und [ABB+02] zu finden.

Werkzeugunterstützung

Ein Vorteil der Repräsentation von Strukturinformationen mittels Graphendarstellungen ist die semi-automatische Erzeugung aus dem Quellcode. Als Beispiel dafür sei hier das Werkzeug Rigi [oCS06] genannt. Rigi ist ein interaktives, visuelles Werkzeug für die strukturelle Redokumentation und die Entwicklung von Programmverständnis. Es wurde an der University of Victoria entwickelt. Rigi erzeugt automatisch Abstraktionen aus dem Quellcode und stellt diese in visueller Form als Graphen dar. Diese können dann von den Entwicklern verwendet werden, um die Struktur zu analysieren und darin zu navigieren. Sie dienen dem Programmverständnis und Design Recovery. Das Parsen des Quellcodes für die automatische Erzeugung ist sprachspezifisch. Bisher werden die Sprachen C, C++, Cobol und Java unterstützt.

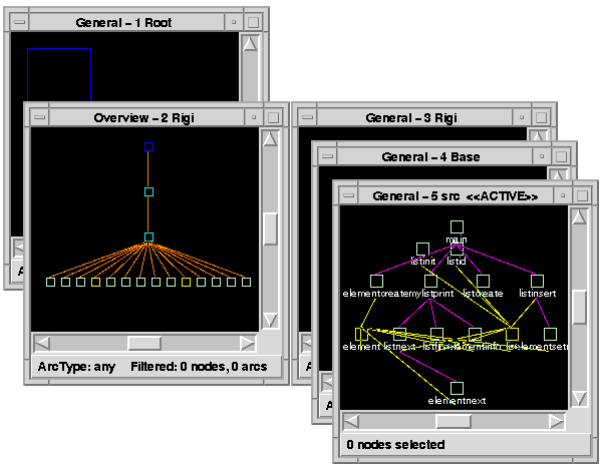


Abbildung 3.5: Rigi

Ergebnisse

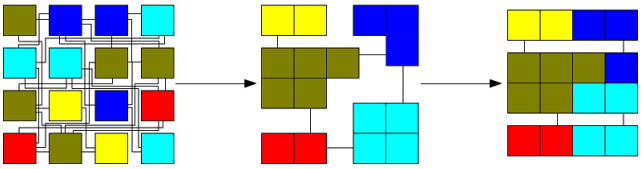
In diesem Schritt wurde verlorenes Wissen über das Ausgangssystem wiedererlangt und Verständnis der Entwickler für das System geschaffen. Das wurde durch Analyse, Erzeugung und Dokumentation von abstrakten Darstellungen des Systems ermöglicht. Als Ergebnis erhält man die ursprünglichen Systemanforderungen und Graphendarstellungen der internen Struktur des Systems.

Die Anforderungen werden im dritten Schritt weiterverwendet und dort an das Zielsystem angepasst, um die neuen Anforderungen für die Web-Anwendung zu erhalten.

Weiters wurden verschiedene Graphendarstellungen des Systems erzeugt. Diese verdeutlichen die Zusammenhänge verschiedener Systemeinheiten. Sie stellen die Grundlage für den nächsten Schritt, die Restrukturierung, dar. Auf Basis des wiedererlangten Wissens kann nun das System gezielt verändert werden.

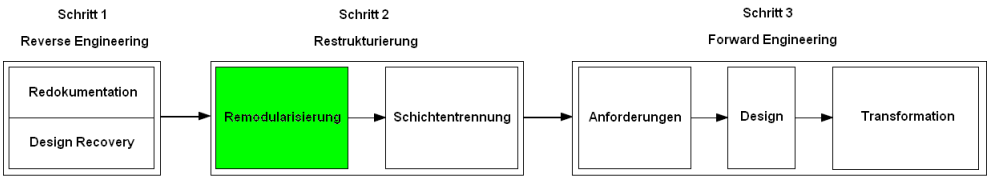
3.3 Schritt 2 - Restrukturierung

In diesem Schritt wird die interne Struktur des Systems verändert. Die Funktionalität bleibt dabei gleich. Die Restrukturierung besteht aus 2 Teilschritten, der Remodularisierung und der Schichtentrennung. Diese werden aufeinander folgend durchgeführt.



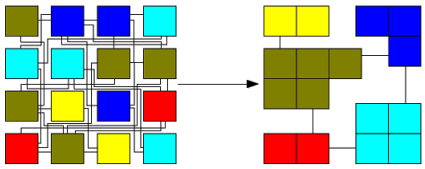
Während das Reverse Engineering noch unabhängig vom Zielsystem war, wird in diesem Schritt gezielt in Richtung Zielarchitektur und Zielparadigma restrukturiert. Dadurch soll bereits hier die Transformation unterstützt werden.

3.3.1 Remodularisierung



Ziele

Bei der Remodularisierung werden Programmeinheiten des Quellcodes so zu Gruppen zusammengefasst, das semantisch zusammengehörende Einheiten in ein und dieselbe Gruppe platziert werden. Dadurch werden ähnliche Einheiten zusammengefasst und verschiedene Teile getrennt. Die Abhängigkeiten der Gruppen untereinander werden dadurch verringert und der Zusammenhalt innerhalb der Gruppen erhöht. Das Ergebnis ist ein zum Ausgangssystem funktionales Äquivalent, dessen interne Struktur jedoch besser aufgeteilt und dadurch leichter verständlich ist. Dadurch können weitere Änderungen einfacher und problemloser durchgeführt werden.



Die Remodularisierung ermöglicht:

- Verbesserung der internen Struktur des Systems.
- Bessere Wiederverwendung in anderen Systemen.
- Erleichterung der Transformation zu einem anderen Sprachparadigma.
- Erweiterung des bereits erlangten Programmverständnisses.

Abhängig von der verwendeten Methode zur Gruppenfindung erhält man hier direkt Objektkandidaten.

Ausgangsbasis

Um eine Einteilung in Gruppen zu erreichen, werden die Ergebnisse aus dem vorherigen Schritt verwendet. Das entwickelte Programmverständnis wird benötigt, um eine sinnvolle Einteilung zu finden, da diese nicht gänzlich automatisch getroffen werden kann. Wissen um die Struktur des Systems ist hier also wichtig. Weiters können die erzeugten Graphen als Basis für algorithmische Einteilungen verwendet werden. Das Ergebnis kann von den Entwicklern verwendet werden, um zu einer guten Aufteilung zu gelangen.

Aktivitäten – Clustering

Unter dem Begriff Clusterverfahren fasst man eine Reihe von Ansätzen, die dazu dienen, Elemente einer Menge anhand von Ähnlichkeiten hinsichtlich bestimmter Eigenschaften zu klassifizieren, zusammen [ABB+02]. Sie können verwendet werden, um strukturgebende Einheiten des Quellcodes, wie Module, Funktionen oder Klassen, aber auch intraprozedurale Codeteile, zu gruppieren. Die Einteilung kann auf verschiedenen hierarchischen Ebenen des Quellcodes durchgeführt werden.

Als Eigenschaft für ein Ähnlichkeitsmaß wird hier die Abhängigkeit der Programmeinheiten untereinander verwendet. Im ersten Schritt wurden bereits automatisch Graphen erzeugt, die verschiedene Abhängigkeiten im Quellcode darstellen. Diese können nun verwendet werden, um Clusteralgorithmen

darauf anzuwenden. Das Ergebnis sind Einteilungen von semantisch zusammengehörigen Programmeinheiten in Gruppen.

Diese Einteilungen können automatisch erzeugt werden. Das Ergebnis muss dann hinsichtlich der Semantik und Sinnhaftigkeit evaluiert werden. Unterschiedliche Algorithmen liefern hier unter Umständen unterschiedliche Ergebnisse. Es gibt kein absolut richtiges Ergebnis, deshalb kann auch kein Optimum berechnet werden. Wichtig ist hier eine sinnvoll getroffene Aufteilung, die zweckmäßig ist und für das weitere Vorgehen Vorteile bietet. Die endgültige Entscheidung muss von den Entwicklern auf Basis des erlangten Programmverständnisses getroffen werden. Anschließend wird das Ausgangssystem der Gruppeneinteilung entsprechend restrukturiert.

Methoden

Im Bereich der Clustering-Algorithmen gibt es bereits eine Vielzahl von Methoden. Zuvor wurden sie als graphenbasiert beschrieben. Es werden die Graphen des ersten Schrittes als Input für den Algorithmus verwendet. Es gibt jedoch weitere unterschiedlich funktionierende Verfahren, die nicht auf Graphen arbeiten. Eine Übersicht ist in [KE00] zu finden. Hier sollen 3 Ansätze vorgestellt werden, die Dominanzanalyse, das Program Slicing und die Objektkandidatenfindung der COREM Methode. Die Dominanzanalyse liefert eine Einteilung auf Grundlage der funktionalen Abhängigkeiten. Sie soll hier genauer beschrieben werden. Das Program Slicing orientiert sich an der Variablenverwendung. COREM liefert ein Objektmodell, das aus einem prozeduralen Programm anhand der Datenhaltung erstellt wird. Slicing und COREM werden hier nur kurz vorgestellt, da eine genaue Beschreibung zu umfangreich wäre.

Dominanzanalyse

Die Dominanzanalyse [CV95] ist ein auf einem gerichteten Graphen arbeitender Algorithmus, um Knoten nach bestimmten Kriterien zusammenzufassen. Die Analyse basiert auf dem graphentheoretischen Begriff der Dominanz. Sie operiert auf einem gerichteten Graphen mit einem Wurzelknoten. Sowohl interprozeduraler Kontrollflussgraph als auch intraprozeduraler Kontrollflussgraph erfüllen diese Bedingung, wenn als Wurzelknoten der Eintrittspunkt in das Programm verwendet wird. Ausgehend von diesem stellt der Graph die Aufrufbeziehungen der einzelnen Programmteile untereinander dar. Durch die Anwendung des Algorithmus können diese in Gruppen zusammengefasst werden. Wesentlich für die Analyse ist, dass es nur einen einzigen Wurzelknoten gibt [FKKQ05]. Dieser muss festgelegt werden, da die Analyse sonst nicht angewendet werden kann.

Die Dominanz für gerichtete Graphen mit Wurzelknoten wird nun definiert. Sei W die Wurzel eines Graphen. Ein Knoten K dominiert einen Knoten N genau dann, wenn jeder Pfad von W nach N den Knoten K enthält. Ein Dominanzbaum stellt alle Dominanzrelationen zwischen Knoten des Graphen dar. Es werden starke und normale Dominanzrelationen unterschieden. Eine starke Dominanzrelation besteht, wenn dabei ein Knoten genau einen Vorgänger hat.

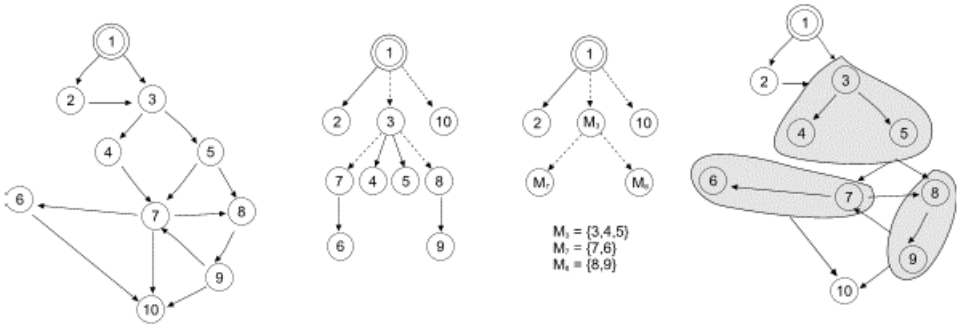


Abbildung 3.6: Graph, zugehöriger Dominanzbaum, Dominanzbaum mit Gruppen, Gruppen im ursprünglichen Graphen

Mittels Dominanzanalyse kann auf Grundlage folgender Argumente eine bessere strukturelle Aufteilung erreicht werden [BR04]:

- Falls zwei Knoten durch starke Dominanzrelationen verbunden sind, bedeutet dies, dass die aufgerufene Funktion eine spezielle Teilfunktion der aufrufenden Funktion realisiert. Folglich liegt es nahe, alle Teilbäume eines Dominanzbaumes zu Modulen zusammenzufassen, welche ausgehend von der Wurzel des Teilbaums lediglich starke Dominanzrelationen enthalten.
- Falls ein Knoten eine normale eingehende Dominanzrelationen besitzt, wird die damit assoziierte Funktion von mehreren Funktionen aufgerufen. Diese Funktion stellt also einen allgemeinen Dienst zur Verfügung. Solche Funktionen können nicht unmittelbar zu größeren Modulen zusammengefasst werden.

Die Dominanzanalyse zur Gruppierung wurde ursprünglich auf Aufrufgraphen imperativer Systeme durchgeführt. Eine Verwendung zur Komponentenfindung in objektorientierten Systemen ist in [LL03] zu finden.

Durch die Anwendung der Dominanzanalyse ergibt sich eine bessere Aufteilung der Programmeinheiten mit geringeren Abhängigkeiten der Gruppen untereinander. Die endgültige Entscheidung für eine sinnvolle Aufteilung ist

nur durch das Programmverständnis der Entwickler durchführbar.

Program Slicing

Das Program Slicing war ursprünglich eine Methode, um Programmverständnis zu entwickeln. Es dient im Zuge eines Vorhabens der Fokussierung auf bestimmte Teile eines Programms. Heute wird es in vielen Bereichen verwendet, in denen es nützlich ist, ein System in kleinere Teilsysteme zu zerlegen. Beispiele sind Programmverständnis, Testen, Debuggen und auch Restrukturierung.

Ein Slice ist ein ausführbares Subset von Programmteilen, das die ursprüngliche Funktionalität beibehält. In diesem werden nur jene Variablen betrachtet, die aktuell von Interesse sind. Alle Codeteile, die damit nicht in Verbindung stehen, werden ausgeblendet. Dadurch ergibt sich eine fokussierte Sichtweise auf den Quellcode.

Es wurden bereits viele verschiedene Arten des Program Slicing entwickelt. Diese unterscheiden sich in den an sie gestellten Anforderungen und eignen sich für unterschiedliche Anwendungen. Für diese sind sie dann auch sehr spezifisch. Man unterscheidet statische und dynamische Methoden. Manche liefern ausführbare Teile, andere wiederum nicht. Das Transform Slicing ist eine Methode, die explizit für die Remodularisierung gedacht ist. Eine Übersicht ist in [Luc01] zu finden.

Durch das Slicing erhält man eine Aufteilung des Quellcodes in Teilsysteme, die für die Restrukturierung zu sinnvollen Gruppen verwendet werden können. Eine solche Einteilung lässt sich nur semi-automatisch treffen, da Entscheidungen nur von den Entwicklern getroffen werden können.

COREM - Objektorientiertes Anwendungsmodell

Die Capsule Oriented Reverse Engineering Method (COREM) [KG95] ist eine Methode zur Transformation von prozeduralen zu objektorientierten Systemen. Im Zuge dessen wird ein objektorientiertes Modell des prozeduralen Ausgangssystems erstellt. Der Ausgangspunkt dafür ist die persistente Datenspeicherung. Für diese wird ein Entity-Relationship-Diagramm (ERD) erstellt. Dieses wird unter Zuhilfenahme von Strukturdiagrammen und Datenflussdiagrammen erstellt, die aus dem Quellcode gewonnen werden. Das ERD ist wiederum die Basis für die Erzeugung eines objektorientierten Modells der Anwendung. Aus der Datenhaltung der Anwendung werden geeignete Objektkandidaten ermittelt. Die Prozeduren des Ausgangssystems werden als Dienste den Objekten zugeordnet. Die ursprüngliche Modulstruktur wird dabei aufgebrochen. Die prozedural orientierte Aufteilung der strukturellen

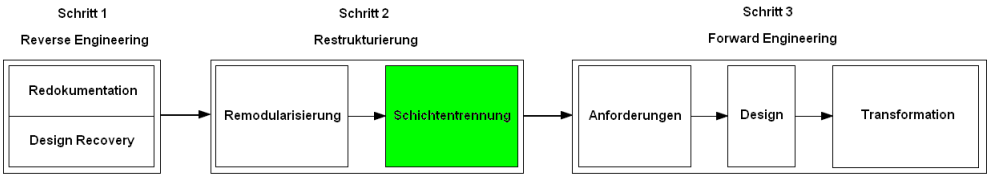
Einheiten wird zugunsten einer Aufteilung in brauchbare Objektdienste zerlegt.

Durch diese Modellierung erhält man eine objektorientierte Aufteilung der Programmteile des Ausgangssystems. Die Methode liefert also nicht nur eine Gruppeneinteilung wie die zuvor beschriebenen, sondern direkt ein objektorientiertes Modell als Basis für einen Paradigmenwechsel. Die Durchführung ist nur semi-automatisch möglich. Die Entscheidungen für eine sinnvolle objektorientierte Struktur auf Basis des prozeduralen Systems sind dabei durch die Entwickler zu treffen.

Ergebnisse

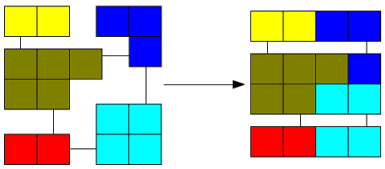
In diesem Teilschritt wurden Programmeinheiten des Quellcodes so zu Gruppen zusammengefasst, dass eine semantisch sinnvolle Aufteilung erfolgt. Dadurch wurde die interne Struktur verbessert und leichter verständlich gemacht. Die Funktionalität blieb dabei gleich. Das Ergebnis ist eine restrukturierte Variante des Ausgangssystems, die Programmverständnis, Erweiterung, Wiederverwendung und die Transformation zum Zielsystem besser unterstützt. Weitere Änderungen können nun einfacher und problemloser durchgeführt werden. Die verbesserte Struktur begünstigt die Transformation in ein objektorientiertes Zielsystem. Sie erleichtert ebenfalls die modulare Migration und weitere Restrukturierungen. Das hier restrukturierte System wird im nächsten Teilschritt als Ausgangsbasis für die Trennung der Schichten verwendet.

3.3.2 Schichtentrennung



Ziele

Nachdem im vorherigen Teilschritt die interne Struktur der Anwendung verbessert und eine Aufteilung der Einheiten in Gruppen aufgrund ihrer semantischen Zusammengehörigkeit durchgeführt wurde, wird nun in diesem Schritt die Vorbereitung für die Transformation zur Zielarchitektur durchgeführt. Durch einen weiteren Restrukturierungsschritt werden die 3 Schichten bereits hier getrennt und linear angeordnet. Dadurch soll die folgende Transformation zum Zielsystem im dritten Schritt erleichtert werden.



Dabei werden die strukturellen Programmeinheiten wie zuvor eingeteilt. Das Kriterium für die Einteilung ist nun aber die Zugehörigkeit zu einer der 3 Schichten. Dadurch werden Benutzerschnittstelle, Datenbankzugriffe und Anwendungslogik strukturell im Quellcode getrennt. Das Ergebnis ist eine restrukturierte, funktional äquivalente, Variante des Ausgangssystems. Diese weist eine sinnvolle interne Struktur auf, die zusätzlich auch die 3 Schichten der Zielarchitektur berücksichtigt.

Ausgangsbasis

Als Ausgangsbasis für die Trennung der Schichten wird das restrukturierte System und dessen Dokumentation aus dem vorherigen Teilschritt verwendet. Dieses weist bereits eine verbesserte interne Struktur auf. Das begünstigt eine weitere Einteilung nach zusätzlichen Kriterien.

Aktivitäten

Für eine genauere Betrachtung des Vorhabens wird zuerst auf die Zerlegbarkeit von Systemen eingegangen.

Zerlegbarkeit vom Systemen

Nach der Zerlegbarkeit eines Legacy Systems [BS95] kann die Architektur in eine der 3 Kategorien „Zerlegbar“, „Teilweise zerlegbar“ und „Unzerlegbar“, eingeteilt werden. In einem zerlegbaren System sind Benutzerschnittstelle, Anwendungslogik und Datenbank eigenständige Komponenten mit wohldefinierten Schnittstellen. Solche Systeme eignen sich am besten, da die gewünschte Struktur bereits vorhanden ist.

In einem teilweise zerlegbaren System sind nur Benutzerschnittstelle und Anwendungslogik eigenständige Einheiten. Anwendungslogik und Datenbank sind aufgrund der komplexen Struktur nicht trennbar. Solche Systeme sind schwieriger zu migrieren, da das Zusammenspiel von Anwendungslogik und Datenbank schwerer zu verstehen ist [ZK04].

Ein nicht zerlegbares System ist eine einzige unstrukturierte monolithische Einheit. Solche Systeme sind am schwierigsten zu behandeln. Eine Auftrennung nach Schichten ist unter Umständen nicht möglich.

Generell ist eine Trennung der Schichten genau dann schwierig, wenn Benutzerschnittstelle und Datenbankzugriffe strukturell im Ausgangssystem nicht als eigenständige Einheiten berücksichtigt wurden. In diesem Teilschritt wird versucht, eine solche Struktur zu erreichen.

Die Trennung erfolgt in 2 Schritten. Zuerst erfolgt eine Identifikation von Teilen mit Benutzerschnittstellen und solchen mit Datenbankzugriffen. Danach werden diese Teile vom Rest des Systems getrennt und in eigene strukturelle Einheiten platziert. Die Schichten selbst können ebenfalls durch Strukturierungsmöglichkeiten der Quellsprache realisiert werden, mittels Verzeichnisstrukturen strukturiert werden oder nur in der Dokumentation angegeben werden. Wichtig ist nur, dass eine strukturelle Abtrennung von Benutzerschnittstelle und Datenbankzugriffen von den Programmeinheiten erfolgt. Je sinnvoller die Struktur ist, die aus dem vorherigen Teilschritt erhalten wurde, desto einfacher ist diese Abtrennung möglich. Je geringer die Zerlegbarkeit des ursprünglichen Systems ist, desto schwieriger lässt sie sich durchführen. Weiters muss bei geringer Zerlegbarkeit zunehmend intraprozedural gearbeitet werden, da unter Umständen mehrere Schichten innerhalb einer einzelnen Subroutine vorkommen und diese deshalb aufgetrennt werden muss. Die daraus resultierenden Subroutinen können dann einzeln den 3 Schichten

zugeordnet werden.

Die Identifikation und Einteilung der Teile kann ebenfalls mittels Remodularisierungsmethoden erfolgen. Es können solche verwendet werden, die die Angabe eines selbst definierten Ähnlichkeitsmaßes erlauben. Eines wird für die Benutzerschnittstellenteile benötigt, ein weiteres für die Datenbankzugriffe. Eine automatische Identifikation von Teilen liefert genaue Ergebnisse. Die Ungenauigkeit liegt hier in der Definition eines geeigneten Kriteriums für die Suche. Der Entwickler muss entscheiden, was als Benutzerschnittstelle gilt und welche Anweisungen Datenbankzugriffe darstellen. Je weniger zerlegbar das Ausgangssystem ist, desto schwieriger ist auch diese Definition. Das Ergebnis der Identifikation ist eine Einteilung der Programmeinheiten in die 3 Schichten. Dabei kann eine Einheit durchaus auch in mehreren Schichten vertreten sein. Nur bei gut zerlegbaren Systemen, bei denen Benutzerschnittstelle, Anwendungslogik und Datenbank bereits eigenständige Einheiten sind, ist das nicht der Fall. Die Durchführung sollte auf der selben hierarchischen Ebene wie im vorherigen Teilschritt begonnen werden. Ist die resultierende Einteilung zu grob, kann die Hierarchiestufe verringert werden. Allgemein sollte die Stufe so niedrig wie möglich sein, sodass eine feine Zerlegung erhalten wird. Dies liegt wiederum im Ermessen der Entwickler. Eine Einteilung möglichst kleiner struktureller Einheiten vereinfacht das folgende auftrennende Restrukturieren.

Während die durchzuführende Restrukturierung nach der Gruppenfindung im vorherigen Schritt durch diese genau festgelegt war, ist bei der Schichtentrennung die Auftrennung selbst schwierig. Je nach Zerlegbarkeit sind die Schichten stark ineinander verwoben und nicht strukturell getrennt. Das zeigt sich auch bei einer Einteilung, in der Einheiten zu mehreren Schichten zugewiesen werden. Diese Einheiten müssen entsprechend zerlegt werden. Benutzerschnittstelle und Datenbankzugriffe werden daraus herausgelöst und aus diesen Teilen neue strukturelle Einheiten erstellt. Ein Beispiel wäre eine Funktion, die Anwendungslogik und Datenbankzugriffe enthält. Die Datenbankzugriffe werden herausgetrennt und daraus neue Funktionen erstellt. In der ursprünglichen Funktion werden diese mittels Aufrufen verwendet. Damit wurden die Schichten mittels Aufteilung auf mehrere Funktionen strukturell getrennt. Die herausgetrennten Einheiten können nach Abhängigkeiten wiederum innerhalb der Schichten strukturiert werden. Eine semantisch sinnvolle Zusammenfassung zu größeren Strukturen ist hier aufgrund der hierarchisch niedrigen Stufe angebracht. Wie zuvor auch müssen Entscheidungen hier von den Entwicklern getroffen werden. Dies ist nach der Trennung nun einfacher, da die Schichten getrennt betrachtet werden können.

Methoden

Methoden zur Schichtentrennung sollen Programmteile mit Benutzerschnittstellen und solche mit Datenbankzugriffen identifizieren können. Danach ermöglichen sie ein Heraustrennen dieser Teile, um das System entsprechend restrukturieren zu können. Diese Vorgänge können je nach verwendeter Methode automatisch durchgeführt werden. Wie im vorherigen Teilschritt liefern verschiedene Methoden unterschiedliche Ergebnisse. Das ist hier allerdings stark vom Ausgangssystem abhängig. Die Methoden verwenden unterschiedliche Definitionen von Benutzerschnittstellen und Datenbankzugriffen. Bei einer automatischen Durchführung können deshalb die Ergebnisse variieren. Weiters gibt es hier ebenfalls kein optimales Ergebnis, die Resultate werden sich aber stark ähneln. Eine Einteilung aufgrund dieser 2 Kriterien kann bei festgelegten Eigenschaften viel einfacher getroffen werden, als die semantisch sinnvolle Einteilung der Remodularisierung. Wie zuvor liegt die endgültige Entscheidung bei den Entwicklern, die die Ergebnisse mit Programmverständnis bewerten müssen.

Konzeptanalyse

Die Konzeptanalyse [SR99] ist eine Methode zur Identifikation von Ähnlichkeiten zwischen Objekten einer Menge anhand ihrer Attribute. Sie kann verwendet werden, um automatisch zu einer Einteilung eines Systems in Module zu gelangen. Sie wird hier als Methode zur Schichtentrennung vorgestellt, kann aber natürlich auch für den vorherigen Teilschritt zur Gruppeneinteilung verwendet werden. Die Konzeptanalyse ist sehr allgemein gehalten und kann für eine Vielzahl von Aufgabenstellungen herangezogen werden, bei denen eine Einteilung aufgrund von Eigenschaften getroffen werden soll. Eine Definition dieser Eigenschaften ist jedoch die Vorraussetzung für die Anwendung. Für die Schichtentrennung kann eine Einteilung in Benutzerschnittstelle, Datenbankzugriff und Anwendungslogik damit realisiert werden.

Ein Konzept ist eine maximale Ansammlung von Objekten mit gemeinsamen Eigenschaften. Ein Modul mit starkem Zusammenhalt ist eine Ansammlung von Funktionen mit gemeinsamen Eigenschaften. Verwendet man die Konzeptanalyse zur Modularisierung eines Systems, stimmen Funktionen mit Objekten überein. Als Attribute können für Benutzerschnittstelle und Datenbankzugriffe jeweils dafür repräsentative Quellcode-Elemente verwendet werden. Eine Festlegung muss von Fall zu Fall entschieden werden. Sobald dieses Entschiedung getroffen wurde, können Konzepte automatisch aus dem Quellcode ermittelt werden. Eine gute Aufteilung im Sinne der Schichteneinteilung muss wie bei den zuvor beschriebenen Methoden vom Entwickler selbst anhand der ermittelten Konzepte getroffen werden.

[SR99] beschreibt Konzept und Implementierung eines Werkzeugs für die automatische Modularisierung von C Programmen. Es liefert als Ergebnis mögliche Konzepte, die der Anwender weiterverwenden kann.

MORPH - Erkennung von Benutzerschnittstellen

Eine konkrete Methode für Benutzerschnittstellen mit vorhandener Werkzeugunterstützung stellt der Model Oriented Reengineering Process for Human-Computer Interface (MORPH) [MM00] dar. Er wurde für das automatische Reengineering von zeichenorientierten Benutzerschnittstellen von Legacy Systemen zu graphischen Benutzerschnittstellen entwickelt. Mit ihm ist es möglich, sequentiell strukturierte Systeme so zu restrukturieren, dass eine ereignisgesteuerte Web-basierte Verwendung möglich wird. Der Prozess umfasst dabei den gesamten Reengineering-Vorgang. Dieser ist in die 3 Teilschritte Erkennung, Transformation und Generierung unterteilt. Hier sei vorerst nur der erste Teil des Prozesses, die Erkennung, beschrieben, in dem automatisch die Benutzerschnittstellen aus dem Ausgangssystem extrahiert und entsprechende graphische Benutzerschnittstellen generiert werden. Damit kann eine Trennung der Präsentationsschicht vom Rest des Systems erreicht werden.

Die Erkennung des MORPH Prozesses verwendet 2 Formen von statischer Analyse, die Flusskontrolle und die Mustererkennung (Pattern Matching). Bei der Flusskontrolle werden ebenfalls 2 Methoden verwendet, die Kontrollflussanalyse und eine modifizierte Datenflussanalyse. Diese ist darauf ausgelegt, Ein- und Ausgabepunkte im Quellcode zu finden. Dadurch werden Implementierungen von Benutzerschnittstellen in Legacy Systemen erkannt. Die Methode folgt dem Fluss der Ein- und Ausgangsvariablen und analysiert die Auswirkungen auf den Rest des Programms. Nachdem die Analyse abgeschlossen ist, werden die Ergebnisse mittels Mustererkennung untersucht. Eine Reihe von Regeln wird angewendet, um Vorkommen von Benutzerschnittstellen zu identifizieren. Nachdem die Mustererkennung auf die Ergebnisse der Flussanalyse, die ja abstrakte Darstellungen sind, angewendet wurde, ist dieser Schritt unabhängig von der Implementierungssprache. Nach Abschluss der Erkennung werden die Ergebnisse in einer Wissensbasis gespeichert. Diese können dann weiterführend für die Transformation verwendet werden.

Ergebnisse

In diesem Schritt wurde das Ausgangssystem grundlegend restrukturiert. Die Funktionalität ist dabei gleich geblieben. Die Struktur des Quellcodes wurde derart verändert, dass eine Transformation zum Zielsystem Web-Anwendung

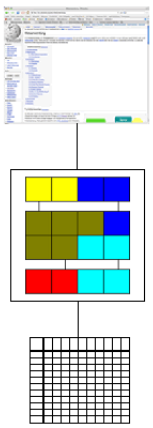
begünstigt wird und leichter durchgeführt werden kann. Das wurde durch vorbereitende Restrukturierungen bezüglich Zielarchitektur, Zielparadigma und eine semantisch sinnvolle Struktur erreicht.

Das Ergebnis ist eine strukturell verbesserte Variante des Ausgangssystems, die zusätzlich in 3 Teile strukturiert ist. Diese repräsentieren die 3 Schichten der Architektur des Zielsystems und sind wie bei dieser linear angeordnet. Die Vorbereitungen für die Transformation sind damit abgeschlossen. Die erhaltene Programmstruktur ist die Basis für das Forward Engineering zum 3-schichtigen objektorientierten Zielsystem im folgenden dritten Schritt.

Wie sich später herausstellen wird, kann diese Einteilung in 3 Schichten noch verbessert werden. Eine Trennung von Präsentation, Anwendungslogik und Datenhaltung ist im Zielsystem ohnehin notwendig. Im Zuge der Fallstudie wird sich herausstellen, dass es vorteilhaft ist, durch eine weitere Restrukturierung die Anwendungslogik selbst in 3 Schichten einzuteilen, um die Architektur des Zielsystems darin zu repräsentieren. Dadurch wird das Forward Engineering der Anwendungslogik vereinfacht und man erreicht für diese eine vorteilhaftere Struktur im Zielsystem.

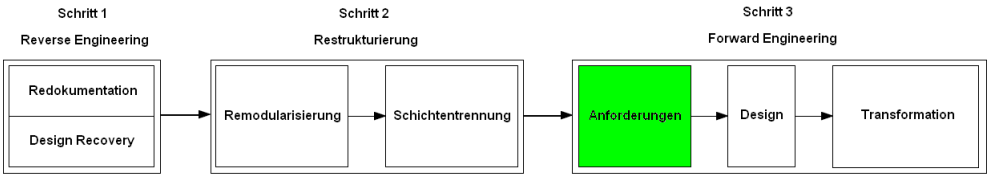
3.4 Schritt 3 - Forward Engineering

In diesem Schritt wird die Transformation zum Zielsystem Web-Anwendung durchgeführt. Wie beim traditionellen Prozess der Software-Entwicklung werden Anforderungsanalyse, Design und Implementierung durchlaufen. Alle Aktivitäten in diesem Schritt werden allerdings auf Grundlage der Ergebnisse der vorherigen Schritte gemacht. Die Anforderungen aus dem ersten Schritt werden übernommen und für das Zielsystem modifiziert. Das rekonstruierte Design wurde im zweiten Schritt für die Restrukturierung verwendet. Das erhaltene Ergebnis dient nun als Grundlage für das Design der Zielanwendung. Dabei wurde die ursprüngliche Struktur bereits so weit wie möglich in Richtung objektorientierter Struktur und 3-Schichten-Architektur verändert, ohne die Ausgangstechnologien zu verlassen.



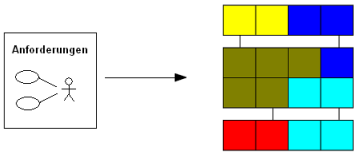
Nun wird sie als Grundlage für die Umsetzung des neuen Designs und die Transformation zur Web-Anwendung verwendet. Unter Transformation wird dabei die Durchführung von Migration und Neuentwicklung zum Zielsystem verstanden. Das Ergebnis verwendet objektorientierte Sprachen und ist in einer 3-schichtigen Architektur umgesetzt. Im Zuge des Forward Engineering kann der Migrationsvorgang integriert werden. Je nach Anforderungen der verwendeten Methode ist eine kontinuierliche modulare Migration in die Zielumgebung möglich. Das Ergebnis ist eine Web-Anwendung, die die ursprüngliche Funktionalität implementiert, dabei moderne Technologien verwendet und nicht mehr die Nachteile des Legacy Systems in sich trägt.

3.4.1 Anforderungen



Ziele

Das Ziel dieses Teilschrittes ist der Definition und Dokumentation der Anforderungen für das Zielsystem. Funktionale und nicht-funktionale Anforderungen werden festgelegt. Da die Web-Anwendung funktional äquivalent zum Legacy System sein soll, soll sie auch das gleiche äußere Systemverhalten implementieren. Dabei müssen allerdings Unterschiede aufgrund der Zieltechnologie berücksichtigt werden. Die Umsetzung neuer Anforderungen, die das Ausgangssystem nicht implementiert, können hier ebenfalls eingebracht werden. Die Festlegung der Anforderungen ist der Ausgangspunkt für das Forward Engineering zur Web-Anwendung.



Ausgangsbasis

Die Ausgangsbasis für die Anforderungen an das Zielsystem sind die Anforderungen des Ausgangssystems, die im ersten Schritt rekonstruiert wurden. Diese legen die Funktionalität fest, die das Ausgangssystem implementieren soll. Da eine funktionale Äquivalenz zum Ausgangssystem erreicht werden soll, werden sie als Anforderungen für das Zielsystem übernommen. Sie müssen jedoch aufgrund der technologischen Unterschiede in Hinsicht auf Umsetzungsmöglichkeiten analysiert und gegebenenfalls entspechend modifiziert werden.

Aktivitäten

Die Anforderungen an das Zielsystem werden festgelegt. Diese beschreiben das äußere Systemverhalten der Web-Anwendung. Funktionale und nichtfunktionale Anforderungen werden erfasst und dokumentiert. Der Unterschied zur traditionellen Software-Entwicklung ist die Verwendung der Anforderungen des Ausgangssystems aus Schritt 1 als Grundlage für die Anforderungen an das Zielsystem. Der optimale Fall dabei ist eine direkte unveränderte Wiederverwendung der ursprünglichen Anforderungen. Da das Ausgangssystem und das Zielsystem aber unterschiedliche Technologien verwenden, müssen die ursprünglichen Anforderungen des Legacy Systems in Hinsicht auf die Umsetzbarkeit an der Web-Anwendung analysiert werden. Einige Funktionalitäten werden sich unter Umständen nicht umsetzen lassen oder sind im Kontext der Web-Anwendung einfach nicht mehr notwendig. Es können auch neue Anforderungen hinzukommen, die wesentlich für die Web-Anwendung sind, aber im Kontext des Legacy Systems nicht angebracht sind. Weiters können von den Entwicklern gewünschte zusätzliche Anforderungen im Zuge der Transformation mitimplementiert werden. Bei der Analyse der Anforderungen können 3 Fälle unterschieden werden.

Anpassung der Anforderungen an das Zielsystem

- **Migrierbare Anforderungen**
  Diese sind in der Web-Anwendung umsetzbar und können vom Legacy System direkt übernommen werden.
- **Nicht zu migrierende Anforderungen**
  Diese werden in der Web-Anwendung nicht umgesetzt. Dabei sind 2 Fälle zu unterscheiden. Nicht umsetzbare Anforderungen können nicht migriert werden, nicht angebrachte Anforderungen sollen nicht migriert werden.
  - **Nicht umsetzbar**
    Diese Anforderungen, die das Legacy System implementiert, können aufgrund der Technologie der Web-Anwendung nicht am Zielsystem umgesetzt werden. Eine Wiederverwendung ist nicht möglich. Ein Beispiel ist der Zugriff auf lokale Systemressourcen, da die Anwendung ja am Web-Server ausgeführt wird. Ein weiters Beispiel ist die Verwendung von Tastaturkürzeln für die Benutzerschnittstelle des Legacy Systems. Diese können mit HTML nicht umgesetzt werden.
  - **Nicht angebracht**
    Diese Anforderungen sollen nicht umgesetzt werden, da sie im Kontext der Web-Anwendung keinen Sinn ergeben oder hier nicht

mehr angebracht sind. Diese Entscheidung ist unabhängig davon, ob sie umsetzbar wären oder nicht. Eine Wiederverwendung ist nicht notwendig. Ein Beispiel ist die Möglichkeit, eine Serveradresse in einem Client-Server System anzugeben, um eine Verbindung mit dem Server herzustellen. Dies ist bei einer Web-Anwendung nicht mehr notwendig, da diese Funktionalität in der Web-Umgebung bereits vom Web-Browser umgesetzt wird.

- **Neue Anforderungen**
  - **Neu hinzugekommen**
    Aufgrund der Eigenschaften des Zielsystems können sich neue Anforderungen ergeben, die im Ausgangssystem nicht vorhanden waren. Besonders aufgrund der verteilten Verwendung der Web-Anwendung können sich solche Anforderungen ergeben. Wird ein Einzelplatzsystem transformiert, muss die verteilte gleichzeitige Verwendung der Web-Anwendung durch mehrere Benutzer berücksichtigt werden. Dadurch kann eine Benutzerverwaltung notwendig werden, die im Ausgangssystem nicht vorhanden war. Weitere funktionale und nicht-funktionale Anforderungen sind möglich. Dies ergibt sich zum Beispiel aus der Möglichkeit, Konfigurationen zu speichern. Diese sollen in der Web-Anwendung dann benutzerspezifisch sein. Ist das beim Legacy System nicht der Fall, hat sich dadurch eine neue Anforderung ergeben.
  - **Neu gewünscht**
    Es sollen gänzlich neue Anforderungen eingebracht werden, die sich nicht notwendigerweise aus den Zieleigenschaften ergeben und nicht bereits im Legacy System vorhanden waren. Diese können dann gleich im Zuge der Transformation mit umgesetzt werden.

Methoden

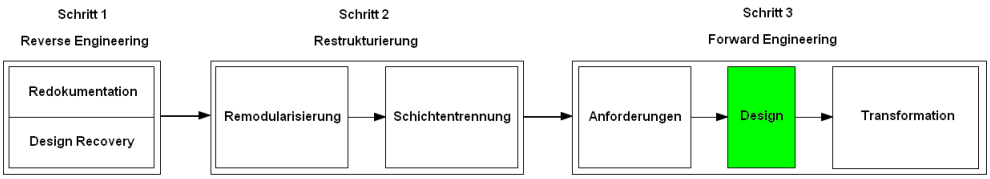
Für diesen Schritt können dieselben Methoden verwendet werden, mit denen bereits im ersten Schritt die Anforderungen dokumentiert wurden. Die Anwendungsfälle der UML eignen sich gut für die Definition der funktionalen Anforderungen. Nicht-funktionale Anforderungen können als aufzählender Text beschrieben werden.

Ergebnisse

In diesem Teilschritt wurden die Anforderungen an das Zielsystem festgelegt und dokumentiert. Dabei wurden die ursprünglichen Anforderungen an das

Ausgangssystem aus Schritt 1 als Grundlage genommen und für das Zielsystem entsprechend angepasst. Als Ergebnis erhält man Anforderungen, die die ursprüngliche Funktionalität, für das Zielsystem angepasst, repräsentieren. Diese werden nun im nächsten Teilschritt als Ausgangsbasis verwendet, um das Design für die Web-Anwendung zu erstellen, die diese Anforderungen dann implementiert.

3.4.2 Design



Ziele

Das Ziel dieses Teilschrittes ist die Erstellung des Designs für die Zielanwendung, mittels dessen die Transformation durchgeführt werden kann und das die Anforderungen an das Zielsystem umsetzt. Die Schwierigkeit dabei ist, dass es auf Grundlage des Ausgangssystems gemacht wird. Das Legacy System soll zu einer objektorientierten Web-Anwendung transformiert werden. Dabei sind mehrere Zieleigenschaften gleichzeitig zu berücksichtigen. Die Notwendigkeit dafür ist von den Eigenschaften des Ausgangssystems abhängig. Verwendet es beispielsweise bereits das objektorientierte Paradigma, entfällt eine solche Transformation. Bei der Definition des Prozesses muss aber der Worst-Case angenommen werden. Bei diesem unterscheiden sich alle Eigenschaften, die das Zielsystem haben soll, von denen des Ausgangssystems. Folgende Zieleigenschaften sollen umgesetzt werden.

- **Sprachwechsel**
  Die Implementierungssprache soll gewechselt werden. Da die Zieltechnologie eine andere Sprache als das Ausgangssystem verwendet, wird der Quellcode in eine Implementierung in der Zielsprache transformiert. Bei reinen Migrationen oder der Verwendung von Wrapping für Subsysteme wird die Sprache beibehalten. Ansonsten erfordern unterschiedliche Sprachen eine Transformation.
- **Paradigmenwechsel**
  Das Sprachparadigma der Implementierungssprache soll gewechselt werden. Aus der Struktur des Quellcodes wird ein objektorientiertes De-

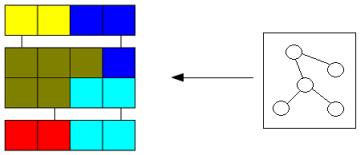
sign für die Zielanwendung ermittelt. Diese Transformation beinhaltet meist eine Umsetzung in einer neuen Sprache. Das muss aber nicht unbedingt notwendig sein. Ein Wechsel zu einer objektorientierten Erweiterung der Ausgangssprache ist ebenfalls denkbar. Ein Beispiel wäre die Transformation zu C++, wobei das Ausgangssystem C verwendet.

- **Architektur**
  Die bereits 3-schichtige Einteilung der Programmeinheiten aus dem zweiten Schritt wird verwendet, um eine 3-schichtige Architektur am Zielsystem umzusetzen. Die Möglichkeit zu einer verteilten Verwendung bietet die Web-Serverumgebung. Die unterschiedlichen Anforderungen zu einem Einzelplatzsystem wurden im vorherigen Schritt in den Anforderungen festgelegt und werden im Design berücksichtigt.

- **Ereignisbasierte Verarbeitung**
  Der Programmfluss wird auf eine ereignisbasierte Verarbeitung umgestellt. Da Web-Anwendungen das zustandslose HTTP-Protokoll verwenden, muss dabei die Verwendung von Benutzersitzungen und jeweilige Zustandshaltung geplant werden, um benutzerspezifischen Programmablauf implementieren zu können. Entsprechende Techniken sind dabei abhängig von der Zieltechnologie zu verwenden.

- **Benutzerschnittstelle**
  Die ursprüngliche Benutzerschnittstelle wird mittels HTML-Seiten umgesetzt.

- **Datenhaltung**
  Anstatt der ursprünglichen Datenbank soll ein modernes relationales Datenbanksystem verwendet werden. Eine dafür verwendbare Datenhaltungsstruktur wird entworfen. Dies kann zu Änderungen in der Verarbeitung führen und Datenzugriffsfunktionen müssen dafür entsprechend angepasst werden.



Weiters soll das Design in Hinsicht auf die verwendete Zieltechnologie entworfen werden. Diese schränkt ebenfalls die Möglichkeiten zur Umsetzung ein. Verschiedene Technologien stellen dabei verschiedene Features zur Verfügung. Da die Zieltechnologie offen gehalten ist und nur die Eigenschaften der Zielanwendung festgelegt sind, werden hier auch keine Annahmen gemacht.

Ausgangsbasis

Die Ausgangsbasis für diesen Teilschritt sind zunächst einmal die Anforderungen aus dem vorherigen Teilschritt. Diese stellen das Systemverhalten dar, dass das Zielsystem implementieren soll. Deshalb muss das Design aufgrund dieser Anforderungen gemacht werden.

Weiters wird das Design auf Grundlage des restrukturierten Systems aus dem zweiten Schritt gemacht. Dafür wird die Dokumentation der Programmstruktur herangezogen. Das bisher entwickelte Programmverständnis ist besonders beim Design des Zielsystems wesentlich, da die Umsetzung mehrerer Zieleigenschaften gleichzeitig geplant werden muss.

Zusätzlich sind gute Kenntnisse über die Möglichkeiten, die die Zieltechnologie bietet, wesentlich, da die Umsetzungsmöglichkeiten von dieser eingeschränkt werden und dies beim Design berücksichtigt werden muss.

Aktivitäten

Aktivitäten in diesem Teilschritt sind die Erstellung des Designs, die Entscheidung bezüglich Bewältigungsstrategien für Teilsysteme und die Planung der Migration.

Umsetzung der Zieleigenschaften

Das Design berücksichtigt alle Eigenschaften des Zielsystems. Da sich diese von den Eigenschaften des Ausgangssystems unterscheiden und verschiedene Aspekte des Zielsystems Web-Anwendung, wie Architektur und Sprachparadigma, darstellen, muss eine Möglichkeit für die strukturierte kombinierte Umsetzung gefunden werden. Dabei können hier die Vorteile der vorbereitenden Restrukturierung aus dem zweiten Schritt bezüglich Struktur und Architektur verwendet werden. Dadurch steht bereits eine gute Ausgangsbasis für das objektorientierte Design zur Verfügung. Weiters besteht dadurch bereits eine 3-schichtige lineare Struktur im restrukturierten Ausgangssystem. Diese kann nun als 3-Schichten-Architektur am Zielsystem umgesetzt werden. Eine getrennte Behandlung der 3 Schichten bei der Erstellung des Designs und der darauf folgenden Transformation ist dadurch möglich.

1. **Sprache und Paradigma**

   Ausgehend von der strukturellen Aufteilung des restrukturierten Systems wird das objektorientierte Design für die Zielanwendung erstellt. Die Architektur, die sich hierarchisch über der Programmstruktur befindet, gibt dabei bereits eine übergeordnete Aufteilung vor. Diese kann für die Erstellung des objektorientierten Designs verwendet werden.

Durch die Trennung der Zuständigkeiten, die bereits bei der Restrukturierung mittels Remodularisierung und Schichtentrennung erfolgte, kann die Umsetzung der objektorientierten Struktur anhand der vorhandenen Aufteilung geplant werden. Da bei dieser die Abhängigkeiten verringert und der Zusammenhalt erhöht wurde, bietet sie eine gute Ausgangsbasis dafür.

Wichtig dabei ist der Entwurf der Dienste von Objekten, die für die Kommunikation von Schichten und Teilsystemen untereinander verwendet werden. Eine nachträgliche Änderung dieser wirkt sich auf die anderen Teile aus, die dann ebenfalls modifiziert werden müssen. Dieses Design wird durch die lineare Anordnung der Schichten erleichtert, da nur angrenzende Schichten betrachtet werden müssen. Innerhalb der Schichten kann ausgehend von den Gruppen des restrukturierten Systems das objektorientierte Design erstellt werden. Dabei sollen auch die Möglichkeiten der konkreten Zieltechnologie zur Programmorganisation genutzt werden.

Die Umstellung auf eine andere Implementierungssprache bezieht sich hier auf die Möglichkeiten zur strukturierten Programmierung, die diese Sprache zur Verfügung stellt. Verschiedene Sprachen bieten unterschiedliche Möglichkeiten zur Programmorganisation. Hier besteht auch ein enger Zusammenhang mit dem Paradigma. Beispielsweise bieten objektorientierte Sprache die Möglichkeit, Klassen und Objekte zu verwenden. Weiters wird hier auch abhängig von den Möglichkeiten der Zielsprache der Datenaustausch zwischen den strukturellen Einheiten festgelegt. Der eigentliche Wechsel der Sprache erfolgt bei der Transformation.

Ein gutes objektorientiertes Design ermöglicht eine gute Aufteilung der Verarbeitung mit geringen Abhängigkeiten innerhalb der gesamten Web-Anwendung. Vor allem das Design der Kommunikation von Teilsystemen untereinander ist dabei wesentlich. Dadurch wird eine modulare Transformation von Präsentation, Anwendungslogik und Datenhaltung unterstützt.

Nicht migrierbare Anforderungen können beim objektorientierten Design berücksichtigt werden, indem ausgehend von der Benutzerschnittstelle in der Präsentationsschicht vertikal über die 3 Schichten alle damit in Zusammenhang stehenden Programmeinheiten entfernt werden. Ist die physische Datenhaltung an der Verarbeitung beteiligt, kann sie ebenfalls entsprechend modifiziert werden. Die Design-Dokumente des restrukturierten Systems stellen bereits die Abhängigkeiten dar, deren Kenntnis für diesen Vorgang benötigt wird. Nur Programmeinheiten mit Funktionalität, die exklusiv für diese Anforderungen verwendet wird, darf entfernt werden. Das mag als Widerspruch zur zuvor erwähn-

ten getrennten Behandlung der Schichten erscheinen. Allerdings stellt das vertikale Entfernen nur eine Möglichkeit dar. Es muss nicht unbedingt durchgeführt werden. Wird die Funktionalität nur in der Präsentationsschicht entfernt, ergeben sich nicht verwendete Programmteile im Zielsystem.

2. **Präsentationsschicht - Ereignisbasierte Verarbeitung**

   Bei der Planung der ereignisbasierten Verarbeitung kann die zuvor durch Restrukturierung erhaltene Präsentationsschicht aus dem restrukturierten System als Grundlage verwendet werden. Dabei muss hier nur mehr diese Schicht betrachtet werden. Die beiden anderen Schichten müssen sich der Web-basierten Kommunikation, die über den Web-Server erfolgt, nicht bewusst sein. Deshalb beschränkt sich dieser Design-Schritt nur auf die Präsentationsschicht. Dies ermöglicht eine einfachere Umstellung der Benutzerschnittstelle auf HTML-Seiten.

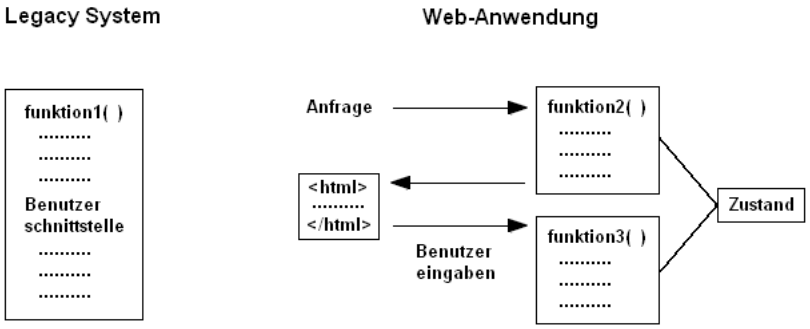


Abbildung 3.7: Umstellung auf ereignisbasierte Verarbeitung

Um eine ereignisbasierte Verarbeitung aus einem System-preemtiven Ausgangssystem zu erhalten, muss der Programmfluss an der Interaktionsstellen, die die Benutzerschnittstelle darstellen, aufgetrennt werden. Die Schnittstelle selbst wird nun nicht mehr als Teil des Quellcodes der Anwendung umgesetzt sondern mittels HTML-Seiten, die einzelne unabhängige Aufrufe an die Web-Anwendung darstellen. Dadurch ist eine Auftrennung der strukturellen Einheiten notwendig, die Benutzerschnittstellenelemente beinhalten. Die Struktur muss an die Web-basierte Verarbeitung angepasst werden. Die Auftrennung ergibt 2 neuen Einheiten, die die ursprüngliche Einheit jeweils vor und nach der Benutzerschnittstelleninteraktion darstellen. Weiters muss der Programmzustand dabei mittels Methoden für die Zustandshaltung um-

gesetzt werden, damit er zwischen einzelnen Aufrufen zur Verfügung steht. Dies muss für jede Benutzersitzung getrennt erfolgen.

Ein Beispiel wäre eine Funktion, die mitten in ihrem Ablauf eine Benutzereingabe erwartet. Nachdem sie gemacht wurde, wird der Ablauf der Funktion fortgesetzt. Nach der Auftrennung wird die Benutzereingabemöglichkeit mittels HTML-Seite umgesetzt, die ein Formular enthält. Die Funktion wird in 2 Teile aufgetrennt. Der erste Teil läuft bei der Anfrage der HTML-Seite ab. Der zweite Teil wird weiterhin als Reaktion auf eine Benutzereingabe ausgeführt. Diese wird mittels Formular zum Web-Server gesendet, wo dann die Verarbeitung ausgeführt wird. Da diese 2 Aktionen unabhängig voneinander sind und 2 getrennte Ereignisse darstellen, muss zwischen diesen der Zustand der Anwendung am Server gehalten werden. Erst dadurch ergibt sich aus getrennten Ereignissen eine zustandsorientierte Web-Anwendung.

3. **Datenschicht - Persistente relationale Datenhaltung**

   Die Datenschicht stellt die Funktionalität für die persistente Datenhaltung zur Verfügung. Als Grundlage wird die Datenschicht des restrukturierten Systems genommen. Wie bereits bei der Präsentationsschicht kann die Funktionalität innerhalb dieser Schicht unabhängig von den anderen Schichten betrachtet werden. Die Präsentationsschicht greift nicht direkt darauf zu, da die Schichten linear angeordnet sind. Die Anwendungslogik muss sich der persistenten Datenspeicherung nicht bewusst sein, da über diese durch die Aufteilung abstrahiert wurde. Die Datenschicht stellt ihr nur entsprechende Dienste zur Verfügung. Die Technologie der Datenspeicherung ist für sie nicht relevant. Deshalb beschränkt sich die Planung der persistenten Datenhaltung nur auf die Datenschicht. Dies ermöglicht einen einfacheren Umstieg von der Technologie für die Datenspeicherung des Legacy Systems.

   Wichtig für die Anwendungslogikschicht sind die Dienste, die über die Datenschicht verfügbar sind. Innerhalb der Datenschicht kann die persistente Datenspeicherung auf ein modernes relationales Datenbanksystem umgestellt werden.

   Ein neues Datenmodell für die physische Datenspeicherung innerhalb des Datenbanksystems muss dafür entworfen werden. Dies wird ebenfalls basierend auf der Datenspeicherung des Legacy Systems durchgeführt, da sich der zu speichernde Datenbestand aufgrund der Bedingung der funktionalen Äquivalenz nicht ändern soll. Die Umstellung der physischen Datenspeicherung ist nicht Web-spezifisch, sondern generell ein Thema bei der Migration von Legacy Systemen. Deshalb wird sie in dieser Arbeit nicht weiter behandelt. Weitere Informationen zu

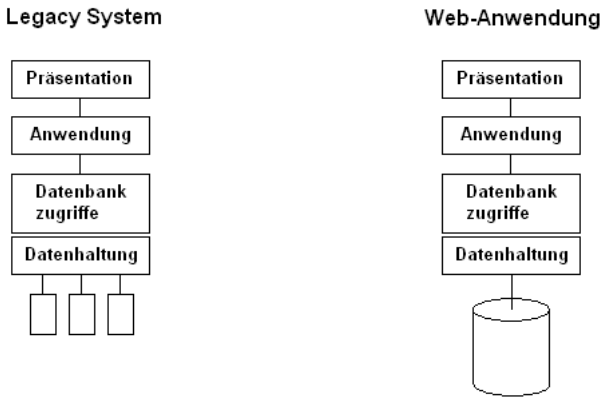


Abbildung 3.8: Umstellung auf ein relationales Datenbanksystem

diesem Thema sind in [DA00] und [JW99] zu finden.

4. **Anwendungslogikschicht - Programmverarbeitung**

   Die Anwendungslogikschicht stellt den Mittelpunkt in der linearen Organisation der Schichten dar. Hier findet die eigentliche Programmverarbeitung statt. Die Anwendungslogik kommuniziert mit der Präsentationsschicht, um Benutzereingaben entgegenzunehmen und Ergebnisse zurückzuliefern, und mit der Datenschicht, um persistente Daten zu beziehen und zu sichern.

   Obwohl die Anwendungslogikschicht sich der Web-basierten Kommunikation nicht bewusst sein muss, müssen in der Anwendung Benutzersitzungen und ein jeweiliger Programmzustand berücksichtigt werden. Die Auswahl einer Sitzung für die Verarbeitung eines Ereignisses erfolgt durch die Informationen, die über die Präsentationsschicht erhalten werden. Das Ereignis wird dann basierend auf dem Zustand dieser Sitzung verarbeitet, ein geänderter Zustand gespeichert und das Ergebnis der Verarbeitung an die Präsentationsschicht zurückgegeben. Dabei greift die Anwendungslogik auf die Datenschicht zu, falls persistente Daten benötigt werden. Die ereignisbasierte Verarbeitung muss in der Anwendungslogik berücksichtigt und entsprechend entworfen werden.

Neuentwicklung und Migration

Im Zuge des Designs der Zielanwendung wird entschieden, welche Teilsysteme des Ausgangssystems migriert werden sollen und welche neu entwickelt

werden. Die Auswahl von Teilen erfolgt dabei anhand des restrukturierten Systems. Innerhalb der verbesserten Struktur lassen sich Teilsysteme für die Wiederverwendung besser auswählen, da aufgrund von Zusammengehörigkeiten restrukturiert wurde und die Gruppen daher einen höheren Wiederverwendungswert haben. Diese Teilsysteme können nun direkt am Zielsystem wiederverwendet, mittels Wrapping in das Gesamtsystem eingebunden oder in die Zielumgebung übersetzt werden. Eine Neuentwicklung basierend auf der Struktur des restrukturierten Systems lässt sich ebenfalls durchführen. Der Unterschied zur Migration besteht darin, dass hier keine direkte Wiederverwendung stattfindet. Eine funktional äquivalente Neuentwicklung, bei der Sprache und Paradigma gewechselt werden und die auf der bereits vorhandenen Struktur basiert, wäre ein Beispiel dafür. Eine vollständige Neuentwicklung von Teilsystemen ist ebenfalls möglich. Dabei muss beachtet werden, welche Dienste das Zielsystem von diesen Teilsystemen erwartet. Wie bei der getrennten Behandlung der Schichten kann dann innerhalb des Teilsystems die Entwicklung unabhängig erfolgen. Diese Vorgangsweisen werden durch die verbesserte Struktur aus dem zweiten Schritt unterstützt.

Es existieren einige Arbeiten, die sich mit der Identifikation und Migration von wiederverwendbaren Teilsystemen der Anwendungslogik beschäftigen. Dabei werden solche Teile identifiziert und in moderne Umgebungen migriert, wodurch eine Wiederverwendung des Legacy Systems ermöglicht wird. Beispiele sind in [ZK04] und [ABB+02] zu finden. Diese sind nicht direkt spezifisch für Web-Anwendungen, sondern beschreiben allgemein die Wiederverwendung von Legacy Systemen mittels moderner Technologien. In [ZK04] wird die Wiederverwendung von Legacy-Teilsystemen als Komponenten in Web-Umgebungen beschrieben. [ABB+02] beschreibt Möglichkeiten zur Komponentenfindung und zur Migration zu komponenten-orientierten Systemen.

Planung der Migration

Im Zuge der Erstellung des Designs wird auch die Durchführung der Migration geplant. Dadurch können das Design und die gewählte Migrationsmethode aufeinander abgestimmt werden. Die Planung und Durchführung der Migration wird durch die Ergebnisse des zweiten Schrittes unterstützt. Die strukturell bessere Aufteilung der restrukturierten Variante begünstigt eine modulare Migration von einzelnen Teilsystemen in die Zielumgebung. Dadurch wird die Composite Database Migration Methode unterstützt. Weiters werden durch die lineare 3-schichtige Struktur Forward und Reverse Migrationsmethoden unterstützt. Innerhalb der Schichten kann ebenfalls modular migriert werden.

Durch die verbesserten Struktureigenschaften kann die restrukturierte Vari-

ante flexibel als Basis für verschiedene Migrationsmethoden verwendet werden. Durch die lineare Anordnung der 3 Schichten werden die datenbankbezogenen Ansätze der Migrationsmethoden unterstützt.

Komponentenorientierte Software

Als Unterstützung für gute Wartbarkeit und Erweiterbarkeit, die im Gegensatz zum Legacy System Eigenschaften des Zielsystems sein sollen, wurde der Wechsel zum objektorientierten Paradigma genannt.

Ein weiterer Ansatz, der auf diesem Paradigma basiert, stellt die komponentenorientierten Softwareentwicklung dar. Eine Software-Komponente ist ein wiederverwendbares Element einer komponentenbasierten Anwendung mit vertraglich definierten Schnittstellen zur Verbindung mit anderen Komponenten. Diese können ohne Änderungen zu Anwendungen zusammengefügt werden. Dadurch soll ein hohes Maß an Wiederverwendung erreicht werden. Objektorientierte Programmierung stellt dabei die Grundlage der Komponentenbasierten Programmierung dar.

Die genaue Form einer Komponente ist abhängig vom jeweiligen Komponentenmodell. Es existieren verschiedene Komponentenmodelle, die sich in Leistungsmerkmalen, unterstützten Betriebssystemen und der Möglichkeiten zur Komponenten-Erzeugung und Verbindung unterscheiden.

Im vorherigen Kapitel wurde die architekturelle Sichtweise beim Reengineering beschrieben. Komponentenorientierte Entwicklung kann ebenfalls als Sichtweise bei der Behandlung eines Legacy Systems gesehen werden [SPL03]. Diese schließen sich gegenseitig aber nicht aus.

Diese Arbeit verbleibt bei der Transformation zum objektorientierten Paradigma und der architekturellen Sichtweise, da eine Berücksichtigung von Komponenten den Umfang dieser Arbeit sprengen würde. Eine Erweiterung des Prozesses, um Komponenten zu berücksichtigen, kann durch Verwendung entsprechender Methoden in den einzelnen Schritten erreicht werden. Grundlegendes am Prozess muss nicht verändert werden, da Komponenten im Vergleich zu Objekten nur zusätzliche Einschränkungen haben und zusätzliche Anforderungen erfüllen müssen.

Eine Arbeit zum Thema Komponenten in Web-Anwendungen ist in [Ost04] zu finden.

Methoden

Die Verwendung von Methoden ist in diesem Teilschritt schwieriger als in den vorherigen Schritten. Das Design der Zielanwendung soll erstellt werden. Da-

bei sind alle gewünschten Eigenschaften des Zielsystems zu berücksichtigen. Da konkrete Methoden auf bestimmte Zieleigenschaften festgelegt sind und damit unter Umständen nicht alle Zielsystemeigenschaften berücksichtigen, müssen hier die Ergebnisse einzelner Methoden von den Entwicklern mittels Programmverständnis zusammengeführt werden. Eine Kombination von Methoden ist notwendig. Die Umsetzung verschiedener Zielsystemeigenschaften muss aufeinander abgestimmt werden, um ein Gesamtdesign des Zielsystems zu erhalten. Ein Beispiel ist die COREM Methode, die ein objektorientiertes Design für das Zielsystem liefert, aber Datenhaltung und Programmverarbeitung des Ausgangssystems beibehält. Dadurch wird es bei der Verwendung dieser Methode notwendig, die Ergebnisse in Hinsicht auf diese Zielsystemeigenschaften zusätzlich zu modifizieren. Dabei können Ergebnisse weiterer Methoden dafür verwendet werden.

Folgend werden die bereits vorgestellten Methoden COREM und MORPH weiterführend in Bezug auf das Design beschrieben. Weiters wird das Design für die Datenhaltung in der Zieldatenbank angesprochen.

COREM - Objektorientiertes Design

Weiterführend zur vorherigen Beschreibung dieser Methode als Möglichkeit zur Remodularisierung im zweiten Schritt wird hier die Erstellung des objektorientierten Designs beschrieben. Unter Verwendung des Wissens, das im Zuge der Anforderungsanalyse erworben wurde, wird zuerst ein objektorientiertes Modell des Zielsystems erstellt. Dieses basiert nicht auf der Struktur des Ausgangssystems, sondern stellt ein neues objektorientiertes Design des gewünschten Zielsystems aufgrund der Anforderungen dar. An diesem Punkt im COREM Prozess existieren nun 2 verschiedene Modelle. Das objektorientierte Modell des prozeduralen Ausgangssystems (reverse ooAM) stellt das Ist-System dar, das neu erstellte Modell des Zielsystems (forward ooAM) stellt das Soll-System dar. Um das objektorientierte Design für das Zielsystems zu erhalten, werden diese beiden Modelle kombiniert. Das Ist-System wird an das Soll-System angepasst, um ein möglichst optimales Modell zu erhalten. Basierend darauf wird danach das objektorientierte Design für das Zielsystem erstellt. Dieses wird unter Berücksichtigung von Eigenschaften der Zielsprache gemacht. Danach kann die Transformation des Ausgangssystems zum objektorientierten Zielsystem durchgeführt werden.

MORPH - Function Splitting

Im zweiten Teilschritt wurde der MORPH Prozess bereits zur Erkennung von Benutzerschnittstellen im Quellcode vorgestellt. Hier wird nun das Function Splitting [MM00] beschrieben, das ein Teil des dritten Schrittes von MORPH, der Generierung und Restrukturierung, ist. Es wird verwendet,

um für MORPH eine ereignisbasierte Programmverarbeitung für ein sequentielles Ausgangssystem zu ermöglichen. Dabei werden Funktionen, die Benutzerschnittstellen enthalten, an diesen Punkten aufgetrennt und in 2 Teilfunktionen zerlegt. Benutzereingaben, die im ursprünglichen System gemacht wurden, werden nun durch Aufrufparameter für die zweite Teilfunktion dargestellt. Diese werden über die neue Benutzerschnittstelle mitgesendet, um ein entsprechendes Ereignis auszulösen. Dadurch wird eine ereignisbasierte Verarbeitung ermöglicht, da diese Aufrufe unabhängig voneinander erfolgen können. Um einen zustandsbasierten Programmablauf zu ermöglichen, wird der aktuelle Zustand zwischen einzelnen Ereignissen gespeichert. Somit wurde das sequentiell verarbeitende Ausgangssystem auf eine ereignisbasierte Programmverarbeitung umgestellt.

Design der Zieldatenbank

Eine Übersicht über das Thema Datenbank-Reverse Engineering ist in [DA00] zu finden. [JW99] beschreibt eine nichtlineare Methode zur Erstellung von Datenbankschemata basierend auf der Datenhaltung des Ausgangssystems.

Aufgrund des Umfangs des Themas Datenbank-Reengineering und der Tatsache, dass es zwar Teil des gesamten Reengineerings ist, aber einen eigenen getrennt zu behandelnden Bereich mit eigenen Methoden darstellt, wird es in dieser Arbeit nicht ausführlicher beschrieben.

Dokumentation des Designs

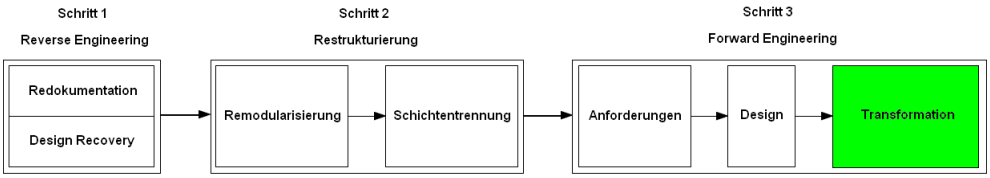
Für die Dokumentation des Designs können wieder Diagramme der Unified Modeling Language UML verwendet werden. Die Anwendungsfälle wurden bereits zur Dokumentation der Anforderungen beschrieben. Das objektorientierte Design kann mit den Objektdiagrammen und Klassendiagrammen dokumentiert werden. Die Paketdiagramme bieten die Möglichkeit, die Programmorganisation zu beschreiben. Diese 3 Diagrammarten beschreiben die interne Struktur eines Systems. Der Programmablauf, der dynamische Aspekte der Anwendung darstellt, kann mittels Sequenzdiagrammen und Aktivitätsdiagrammen dargestellt werden. Damit kann die ereignisbasierte Verarbeitung dokumentiert werden. Diese 2 Diagrammarten beschreiben das Verhalten des Systems, wobei verschiedene dynamische Aspekte betrachtet werden.

Mittels dieser Diagramme können die verschiedenen Aspekte, die durch die festgelegten Zielsystemeigenschaften repräsentiert werden, dargestellt werden. Jedes Diagramm beschreibt dabei einen Teilaspekt des Gesamtsystems. Eine genaue Beschreibung der UML Diagramme ist in [Mac01] zu finden.

Ergebnisse

In diesem Teilschritt wurde das Design für das Zielsystem festgelegt und dokumentiert. Die Ausgangsbasis dafür waren die zuvor festgelegten Anforderungen an das Zielsystem und die Ergebnisse der Restrukturierung aus dem zweiten Schritt. Bei der Erstellung des Designs werden alle Zieleigenschaften der Web-Anwendung berücksichtigt. Weiters wird die Zieltechnologie festgelegt und in das Design mit einbezogen. Die Transformation zum objektorientierten Paradigma, zur 3-schichtigen Architektur und zur ereignisbasierten Verarbeitung werden im Design festgelegt. Weiters wird hier entschieden, welche Teilsysteme migriert werden sollen und welche neu entwickelt werden. Aufgrund der Aufteilung in 3 Schichten können Benutzerschnittstelle und Datenbank lokal in der jeweilig zugehörigen Schicht umgesetzt werden. Weiters werden dadurch, und aufgrund der verbesserten Struktur, Migrationsmethoden unterstützt. Das Ergebnis ist das dokumentierte Design für das Zielsystem, das verschiedene Aspekte der Web-Anwendung auf hohem Abstraktionsniveau darstellt. Dieses wird nun im nächsten Teilschritt als Ausgangsbasis verwendet, um die Transformation zum hier entworfenen Zielsystem durchzuführen und dessen Implementierung zu erhalten.

3.4.3 Transformation

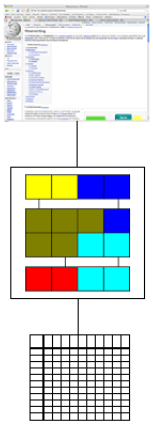


Ziele

Die Transformation ist der letzte Teilschritt im Web-Reengineering Prozess. Dabei wird das Zielsystem auf Implementierungsebene unter Berücksichtigung aller geplanten Zielsystemeigenschaften umgesetzt. Das Ergebnis ist eine Web-Anwendung, die funktional äquivalent zum Ausgangssystem ist und moderne Technologien verwendet. Diese trägt die Nachteile des Legacy Systems nun nicht mehr in sich. Das Zielsystem ist durch die zuvor durchgeführten Schritte gut dokumentiert und lässt sich bei Bedarf leicht ändern und erweitern. Weiters bietet es alle Vorteile einer modernen Web-Anwendung.

Ausgehend von der 3-schichtigen Struktur und der objektorientierten Aufteilung wird die Transformation zur Zielsprache und dem objektorientierten

Paradigma durchgeführt. Diese kann durch das zuvor festgelegte Design getrennt für jede Schicht umgesetzt werden. Dabei erhält man für alle Schichten eine objektorientierte Implementierung der Funktionalität des Ausgangssystems in der Zielsprache. Durch die voran gegangenen Schritte ist das Ergebnis dem Legacy System gegenüber strukturell verbessert. Dadurch soll dauerhaft Wartbarkeit und Erweiterbarkeit für die Web-Anwendung gewährleistet werden.



Durch die Auftrennung der Objektdienste, die mit der Benutzerschnittstelle in Zusammenhang stehen, wird die ereignisbasierte Verarbeitung ermöglicht. Diese geänderten Dienste kommunizieren mit der Benutzerschnittstelle, um eine Web-basierte Verwendung zu realisieren. Dabei wird der Programmzustand, der sich durch eine Folge von Ereignissen ergibt, am Server gespeichert.

Weiters wird die Benutzerschnittstelle selbst transformiert, wobei man eine Implementierung der ursprünglichen Schnittstelle als HTML-Seiten erhält. Diese werden in die geänderte ereignisbasierte Programmverarbeitung integriert, um ein Benutzer-preemtives System zu erhalten, wie es Web-Anwendungen erfordern.

Die Datenbankstruktur des Ausgangssystems wird in einer modernen relationalen Datenbank umgesetzt. Durch die Verwendung von Migrationsmethoden kann im Zuge der Transformation der originale Datenbestand in die neue Datenhaltung übernommen werden und steht damit der Web-Anwendung zur Verfügung.

Weiters ist die Web-Anwendung gut dokumentiert. Im Laufe des Prozesses wurden die Anforderungen und das Design sowohl des Legacy Systems als auch der Web-Anwendung dokumentiert. Diese beschreiben den aktuellen

Stand der Systeme und können für weitere Änderungen und Erweiterungen herangezogen werden.

Ausgangsbasis

Die Ausgangsbasis für diesen Teilschritt sind die Anforderungen und das Design, das in den vorherigen Teilschritten erstellt wurde. Wie bei der traditionellen Software-Entwicklung wurden zuerst die Anforderungen festgelegt und danach das Design erstellt. Die Anforderungen beschreiben Funktionalität und Verhalten, das von der Web-Anwendung umgesetzt werden soll. Das Design stellt die Planung des eigentlichen Systems dar, wobei alle Zielsystemeigenschaften berücksichtigt werden. Die darauf folgende Implementierung setzt die Web-Anwendung in der Zielumgebung um. Sie wird hier als Transformation bezeichnet, da sie weiterhin basierend auf dem restrukturierten Ausgangssystem, das im Design berücksichtigt wurde, gemacht wird. Dabei stellt die Transformation die Umsetzung auf Implementierungsebene dar, deren Ergebnis eine funktionierende Web-Anwendung ist.

Aktivitäten

Aktivitäten in diesem Teilschritt setzen das Design in eine Implementierung der Web-Anwendung um. Hier findet die Wiederverwendung von Informationen des Ausgangssystems für die Umsetzung statt. Wie bereits im Design geplant, werden Teilsysteme migriert oder aber neu implementiert. Der Grad der Wiederverwendung reicht hier von der direkten Übernahme von Quellcode, über strukturelle Migration bis hin zur komplett neu geschriebenen Teilsystemen. Ebenfalls kann die Implementierung voll-automatisch, semiautomatisch oder manuell durchgeführt werden.

Anwendung

Bei der Transformation zur objektorientierten Zielsprache wird das objektorientierte Design umgesetzt. Dieses unterscheidet sich vom Ausgangssystem sowohl in Sprache als auch Paradigma. Der Unterschied besteht hier aber nur in der Struktur. Die Funktionalität des Ausgangssystems wird für das Zielsystem übernommen. Das bedeutet, dass innerhalb der strukturellen Einheiten nur ein reiner Sprachwechsel stattfindet. Man erhält eine objektorientierte Implementierung der Funktionalität des Ausgangssystems. Ausnahmen stellen die Verbindungen zur Benutzerschnittstelle und der Datenbank dar, da die Anwendung für die Verwendung dieser angepasst werden muss. Wird die Ausgangssprache auch am Zielsystem unterstützt und erfüllt die Anforderungen an die Zielsystemeigenschaften, können Teile des Ausgangssystems direkt

wiederverwendet werden. Andernfalls kann die Übernahme mittels automatischer Übersetzung durchgeführt werden oder es kann in der Zielsprache neu implementiert werden.

Durch die Auftrennung der Objektdienste, die mit der Benutzerschnittstelle in Zusammenhang stehen, wird die ereignisbasierte Verarbeitung ermöglicht. Diese geänderten Dienste kommunizieren mit der Benutzerschnittstelle, um eine Web-basierte Verwendung zu realisieren. Dabei wird der Programmzustand, der sich durch eine Folge von Ereignissen ergibt, am Server gespeichert.

Benutzerschnittstelle

Die Benutzerschnittstelle des Ausgangssystems wird mittels HTML-Seiten umgesetzt. Dabei müssen alle Möglichkeiten zur Eingabe, die auch im Ausgangssystem vorhanden waren, realisiert werden. Benutzereingaben werden als Formulare umgesetzt, Möglichkeiten zur Steuerung können als Hyperlinks implementiert werden. Die Interaktionsmöglichkeiten müssen äquivalent gewährleistet sein. Optisch kann die originale Schnittstelle übernommen oder auch vollkommen neu gestaltet werden. Eine automatische Erzeugung von HTML-Seiten aus dem Ausgangssystem ist dabei möglich.

Um die neue Benutzerschnittstelle in die Anwendung zu integrieren, werden Dienste der Objekte am Zielsystem, die Benutzereingaben erfordern, in 2 Teile aufgetrennt. Der erste Teil wird vor dem Senden der HTML-Seite an den Client ausgeführt, der zweite Teil verarbeitet die Eingaben auf der Client-Seite und wird als Reaktion auf das Senden an den Server ausgeführt. Dadurch werden einzelne Ereignisse realisiert. Um diese für eine zustandsbasierte Anwendung verwenden zu können, wird der Zustand, der sich durch ein Ereignis ergibt, für jede Benutzersitzung getrennt am Server gespeichert. Für folgende Ereignisse wird er dann als Ausgangszustand weiterverwendet. Dadurch ergibt sich eine zusammenhängende Abfolge von Ereignissen, die eine zustandbasierte Anwendung benötigt. Die Realisierung von Benutzersitzungen und Zustandshaltung ist dabei von der Zieltechnologie abhängig.

Datenbank

Die Umstellung auf das neue Datenbanksystem kann lokal innerhalb der Datenschicht erfolgen. Die Dienste, die sie der Anwendungslogikschicht zur Verfügung stellt, werden dabei nicht geändert. Für die Realisierung von Persistenz mit einer geänderten Datenhaltungsstruktur im neuen Datenbanksystem werden auch Änderungen im Quellcode notwendig. Vor allem bei einem Wechsel des Datenbankmodells ist dies erforderlich, da eine unterschiedliche

Strukturierung der Daten und ihrer Beziehungen zueinander besteht.

Migrationsmethoden

Im Zuge der Implementierung wird auch der Übergang vom Legacy System zur Web-Anwendung unter Verwendung der bereits festgelegten Migrationsmethode durchgeführt. Dabei kann modular oder schichtenweise migriert werden, wobei Gateways für die Anbindung verwendet werden. Oder das Zielsystem wird vollständig getrennt entwickelt, wobei eine reine Datenmigration stattfindet.

Testen

Begleitend zur Transformation wird ein ausführliches Testen des Zielsystems durchgeführt. Dies wird im Zuge der Umstellung je nach gewählter Migrationsmethode für einzelne Teilsysteme, die 3 Schichten und das komplette System durchgeführt. Das Testen schließt in der traditionellen Software-Entwicklung den Prozess als letzter Schritt ab. Hier wird es der Transformation zugeordnet. Es stellt dafür eine wichtige qualitätssichernde Maßnahme dar, ist jedoch kein eigener Schritt des Reengineerings. Das Testen ist bei der Transformation besonders wichtig, um die gewünschte funktionale Äquivalenz zum Ausgangssystem zu gewährleisten.

Methoden

Methoden für die Transformation setzen die Implementierung des Zielsystems basierend auf dem Design und damit auf der Struktur des restrukturierten Ausgangssystems um. Dessen Quellcode kann hier verwendet werden, um mittels semi-automatischer und automatischer Methoden Ergebnisse zu generieren. Eine manuelle Umsetzung durch die Entwickler ist natürlich ebenfalls möglich. Bei der automatischen Generierung müssen die Ergebnisse wieder durch die Entwickler geprüft werden, um eine korrekte Umsetzung der gesamten Anwendung entsprechend dem Design zu gewährleisten.

Automatische Quellcode-Übersetzung

Die automatische Quellcode-Übersetzung transformiert von der Ausgangssprache direkt in die Zielsprache. Der Vorgang kann voll-automatisch durchgeführt werden. Er bietet sich für Fälle an, in denen die Ausgangssprache am Zielsystem nicht verfügbar ist oder generell die Sprache gewechselt werden soll. Die automatische Übersetzung bietet einerseits den Vorteil, dass der Zeitaufwand für den Sprachwechsel dadurch sehr gering ist. Weiters wird damit die funktionale Äquivalenz gewährleistet. Andererseits können auch vie-

le Probleme entstehen. Die Übersetzung kann die Struktur des Codes nicht wesentlich ändern. Diese Eigenschaft steht in Konflikt zu dem zuvor festgelegten Design und erfordert daher weitere Schritte durch die Entwickler, um die Vorgaben zu erfüllen. Restrukturierungen und manuelle Änderungen werden dadurch notwendig. Weiters erhält man zwar eine Implementierung in der Zielsprache, diese erhält aber die Struktur des Quellcodes [SPL03]. Ein Beispiel wäre die Übersetzung von COBOL zu Java. Man erhält ein Java-Programm, das wie ein COBOL-Programm aussieht. Dadurch werden Wartung und Erweiterung sowohl für COBOL-Programmierer als auch Java-Programmierer erschwert.

Allgemein lässt sich sagen, dass bei Sprachen mit geringen Unterschieden eine automatische Übersetzung die besseren Lösungen liefert, da dabei nur geringe Änderungen erforderlich sind. Zum Beispiel ist ein Wechsel der Version innerhalb derselben Sprache ein guter Kandidat für eine automatische Übersetzung. Ein Wechsel des Paradigmas bringt jedoch Probleme mit sich. Beispielsweise erhält man bei der Übersetzung von Quellcode einer prozeduralen Sprache zu einer objektorientierten Sprache ein Ergebnis, das sich nicht an objektorientierte Design-Prinzipien hält. Es stellt eine objektorientierte Implementierung eines prozedural entworfenen Systems dar. Weitere Beispiele sind in [SPL03] zu finden.

Automatische Übersetzung sollte vorsichtig verwendet werden. Für ausgewählte Teilsysteme oder intraprozeduralen Code kann sie aber durchaus eingesetzt werden.

COREM - Objektorientierte Transformation

Wie bei der automatischen Quellcode-Übersetzung beschrieben, ist eine sinnvolle Transformation eines prozeduralen Systems in ein objektorientiertes System nur unter Mithilfe der Entwickler möglich, da die Übersetzung keine wesentlichen strukturellen Änderungen durchführt. So auch bei der COREM Systemtransformation. Diese lässt sich nicht voll-automatisch durchführen [KG95], da Wissen und Programmverständnis der Entwickler in den Prozess einfließen und Entscheidungen durch sie getroffen werden müssen. Dieser letzte Schritt der COREM Methode liefert als Ergebnis eine objektorientierte Implementierung des prozeduralen Ausgangssystems.

MORPH - Transformation von Benutzerschnittstellen

Zuvor wurden bereits die Erkennung und das Function Splitting von MORPH beschrieben. Als zweiter Schritt in diesem Prozess wird die Transformation durchgeführt. Das Ergebnis ist eine funktional äquivalente Umsetzung der Benutzerschnittstelle des Ausgangssystems als HTML Formulare. Dies wird

auf Grundlage der Ergebnisse des ersten Schrittes, der Erkennung, gemacht. Die Transformation liefert dabei automatisch eine Beschreibung der neuen Schnittstelle. Die Entwickler können danach zusätzliche Entscheidungen einfließen lassen. Darauf folgt die Generierung, ein Teil des dritten Schrittes von MORPH, bei der die HMTL-Seiten erzeugt werden. Diese können dann als neue Benutzerschnittstelle für das Zielsystem verwendet werden. Obwohl die Generierung bei MORPH eine manuelle Durchführung verlangt, ist es möglich, für diese Aufgabe eine semi-automatische Durchführung zu erreichen [MM00].

Transformation des Datenbestandes

Die Migrationsmethoden führen die Transformation des Datenbestandes im Zuge der Migration durch. Das Ergebnis ist ein äquivalenter Datenbestand in der Datenbank des Zielsystems. Diese Daten-Migration hat für Legacy Systeme hohe Wichtigkeit, da der Betrieb bei einer Umstellung nicht oder nur sehr kurz unterbrochen werden darf. Indem man den Datenbestand für ein funktional äquivalentes Zielsystem übernimmt, kann man sowohl Stillstandzeit als auch Probleme bei der Inbetriebnahme des Zielsystems minimieren.

Ergebnisse

In diesem Teilschritt wurde die Transformation zur Zielanwendung durchgeführt. Basierend auf dem Design aus dem vorherigen Teilschritt wurden alle definierten Zieleigenschaften umgesetzt. Die Transformation zum objektorientierten Paradigma, zur 3-schichtigen Architektur und zur ereignisbasierten Verarbeitung mittels HTML-basierter Benutzerschnittstelle wurden durchgeführt, um das Zielsystem zu realisieren. Das Ergebnis ist eine Implementierung einer zum Legacy System funktional äquivalenten Web-Anwendung. Diese verwendet moderne Technologien und trägt nun nicht mehr die Nachteile des Legacy Systems in sich.

Bei der Durchführung wurde auch eine gewählte Migrationsmethode integriert. Dadurch kann der Datenbestand des Legacy Systems übernommen und die Dauer der Umstellung vom Legacy System auf die Web-Anwendung minimiert werden. Diese kann nun mit den aktuellen Daten verwendet werden.

Das Legacy System geht außer Betrieb, es wurde durch die entwickelte funktional äquivalente Web-Anwendung ersetzt. Damit ist der Web-Reengineering Prozess abgeschlossen.

Kapitel 4

Fallstudie

Im vorherigen Kapitel wurde ein Web-Reengineering Prozess definiert. Dieser basiert auf den allgemeinen Reengineering-Aktivitäten, setzt sie als Schritte des Prozesses in Reihenfolge und konkretisiert sie für das Zielsystem Web-Anwendung. Dadurch werden die Systemeigenschaften festgelegt, die schließlich mit dem Zielsystem umgesetzt werden. Trotzdem befindet er sich nach wie vor auf einem hohen Abstraktionsniveau, das dadurch notwendig wird, dass die Eigenschaften des Ausgangssystems nicht festgelegt sind. Für jeden Teilschritt wurden deshalb Methoden für die Umsetzung beschrieben, die als Beispiele dienen sollen.

Um weitere Informationen über das Reengineering zu Web-Anwendungen zu erlangen und um den Prozess zu veranschaulichen, wird er nun im Zuge einer Fallstudie angewendet.

Verallgemeinerung der Erkenntnisse aus der Fallstudie

Da Ausgangssystem und Zieltechnologie in der Prozessdefinition nicht festgelegt wurden, arbeitet der Prozess hauptsächlich auf Abstraktionen und der architekturellen Ebene. Bei der Fallstudie müssen aber implementierungsabhängige Aktionen durchgeführt werden, um zu einem konkreten Zielsystem zu gelangen. Dafür ist eine Berücksichtigung der Technologien von Ausgangssystem und Zielsystem notwendig. Dies schließt die Implementierungssprachen beider Systeme mit ein. Erkenntnisse, die darauf beruhen, können im Zuge dieser Arbeit nicht verallgemeinert werden. Wohl aber können auf struktureller und architektureller Ebene allgemein verwendbare Erkenntnisse gewonnen werden.

4.1 Auswahlkriterien für das Ausgangssystem

Das gewählte Ausgangssystem soll repräsentativ für Legacy Systeme sein. Weiters sollen alle in der Prozessdefinition behandelten Aspekte abgedeckt werden. Das wird durch die Auswahl eines Systems erreicht, dessen Eigenschaften mit denen des Zielsystems für jede Eigenschaft nicht übereinstimmen. Dadurch muss eine entsprechende Transformation durchgeführt werden.

Eigenschaften des Ausgangssystems:

- Die Implementierung wurde in einer prozeduralen Sprache durchgeführt. Gibt es für diese eine objektorientierte Erweiterung, soll diese von der Zieltechnologie nicht unterstützt werden. Dadurch müssen sowohl Sprache als auch Paradigma transformiert werden.
- Die Architektur des Ausgangssystems ist nicht in 3 oder mehr Schichten umgesetzt.
- Das Ausgangssystem ist ein nicht verteilt funktionierendes Einzelplatzsystem.
- Die Benutzerschnittstelle ist nicht mittels HTML-Seiten implementiert. Andere graphische Benutzerschnittstellen eignen sich zwar für das Ausgangssystem, der technologische Unterschied sollte allerdings so groß wie möglich sein. Die Auswahl eines Systems mit zeichenorientierter Benutzerschnittstelle bietet sich an.
- Die Programmverarbeitung des Ausgangssystems soll nicht ereignisbasiert sein.
- Das Datenbankmodell ist von dem des Zielsystems verschieden. Wie zuvor bei der Benutzerschnittstelle sollte der Unterschied so groß wie möglich sein. Eine flache Dateistruktur, im Gegensatz zur relationalen Datenbank des Zielsystems, bietet sich an.

Durch die Erfüllung dieser Kriterien soll die Fallstudie eine größtmögliche Aussagekraft erhalten. Die zuvor beschriebenen Eigenschaften von Legacy Systemen können jedoch im Rahmen dieser Fallstudie nicht alle berücksichtigt werden.

Repräsentation der Eigenschaften von Legacy Systemen:

- Die Wichtigkeit und das Umfeld des Einsatzes des Legacy Systems sind nicht vorhanden. Dadurch können Fehler oder Unzulänglichkeiten im Zielsystem nach der Fertigstellung unwichtiger erscheinen, als sie es bei

einem Fall in der Praxis tatsächlich wären. Im Rahmen dieser Fallstudie werden solche Auswirkungen auf Geschäftsprozesse nicht berücksichtigt. Die Arbeit verbleibt bei der Behandlung des Reengineerings selbst.

- Die Eigenschaft von Legacy Systemen, dass sie nur schwer erweiterbar sind, wird durch die zuvor festgelegten Eigenschaften des Ausgangssystems erfüllt. Es ist nicht möglich, direkt zu einer objektorientierten Web-Anwendung innerhalb der technologischen Möglichkeiten des Ausgangssystems zu gelangen.
- Die Schwierigkeit der Wartung wird insofern erfüllt, dass es für das Ausgangssystem keine Dokumentation abseits des Benutzerhandbuchs gibt. Es sind keine Design-Dokumente vorhanden und es besteht kein Wissen über die internen Abläufe des Systems. Dadurch wird ein Reverse Engineering notwendig.
- Die Integrationsmöglichkeiten des Legacy Systems zur Verknüpfung mit anderen Systemen werden im Rahmen dieser Fallstudie nicht untersucht. Es findet keine Behandlung der Integration mit anderen Systemen sowohl des Legacy Systems als auch der Web-Anwendung statt. Die Web-Anwendung bietet Integrationsmöglichkeiten aufgrund der dafür verwendeten modernen Technologien. Die Auswahl der Zieltechnologie erfolgt aber ebenso nicht aufgrund dieses Kriteriums.
- Der Betrieb auf veralteter Hardware wird nur insofern berücksichtigt, dass die Plattform generell auf eine Web-basierte Umgebung umgestellt wird. Da, wie bei den Ausgangssystemeigenschaften angegeben, die Sprache des Legacy Systems am Zielsystem nicht unterstützt wird, wird ein Plattformwechsel auf jeden Fall notwendig. Weiter wird dieser Aspekt allerdings nicht behandelt.
- Die Migrationsmethoden selbst werden nicht untersucht. Deshalb wird auch kein Migrationsszenario erstellt. Die Fallstudie verbleibt bei der Behandlung des Reengineerings. Es werden aber die Integrationsmöglichkeiten der Migrationsmethoden, wie im Prozess angegeben, untersucht.

4.2 Auswahl der Zieltechnologie

Im Zuge der Beschreibung von Web-Anwendungen im zweiten Kapitel wurden bereits die Technologien Servlets, JavaServer Pages und JavaServer Faces beschrieben. Als Zieltechnologie für die Fallstudie soll die Java Enterprise Edition 5 von Sun Microsystems verwendet werden. Hier folgt eine kurze Beschreibung der Technologien, die dabei verwendet werden. Eine ausführliche

Beschreibung ist unter [SM07] zu finden. Für den Betrieb der Zielanwendung wird der Java System Application Server Platform Edition 9.0 Update 1 ausgewählt. Als Entwicklungsumgebung wird die NetBeans IDE 5.5 verwendet. Die Zieltechnologie für die Fallstudie wird hier bereits festgelegt, aber, wie im Web-Reengineering Prozess beschrieben, erst bei der Erstellung des Designs berücksichtigt.

Die Java Enterprise Edition (Java EE) ist ein Rahmenwerk für die Erstellung von modernen verteilten Systemen. Sie verwendet als Basis die Programmiersprache Java. Die Java EE stellt APIs und Laufzeitumgebung für die Entwicklung und den Betrieb von umfangreichen, mehrschichtigen, skalierbaren, verlässlichen und sicheren verteilten Systemen, wie Web-Anwendungen, zur Verfügung.

Mehrschichtige Anwendungen

Für die Umsetzung mehrerer Schichten innerhalb des Zielsystems werden verschiedene Technologien der Java EE, wie die JSF und die Enterprise Beans, verwendet. Die Java EE teilt diese Technologien dabei den 3 Schichten Web Tier, Business Tier und Enterprise Information Systems Tier zu, die den in dieser Arbeit verwendeten Schichtendefinitionen Präsentationschicht, Anwendungslogikschicht und Datenschicht entsprechen.

Der Web Tier stellt die Verbindung des Clients zur Anwendung her. Hier werden dynamische Inhalte für den Client erzeugt. Eingaben von Benutzern auf der Client-Seite werden entgegengenommen und Ergebnisse des Business Tiers werden aufbereitet und an den Client gesendet. Der Fluss zwischen den Seiten wird hier kontrolliert und Benutzersitzungen werden hier behandelt. Technologien für diese Schicht sind Servlets, JavaServer Pages und JavaServer Faces, die bereits kurz erklärt wurden. Weiters werden JavaBeans zur Kontrolle von Struktur und Verhalten der Seiten verwendet.

Der Business Tier besteht aus Komponenten, die die Anwendungslogik implementieren. Die grundlegende Funktionalität einer Anwendung ist hier umgesetzt. Technologien für diese Schicht sind die Enterprise Bean Komponenten.

Der Enterprise Information Systems Tier steht allgemein für an die Anwendung über den Business Tier angebundene Back-End Systeme. Dabei sind Datenbanksysteme für die Datenhaltung enthalten. Es können aber auch andere Systeme angebunden werden. Technologien für diese Schicht stellen die Java Database Connectivity (JDBC) und die Java Persistence API dar. Diese können für die Verwendung von Datenbanksystemen für die persistente Datenhaltung in Zuammenhang mit dem Business Tier verwendet werden.

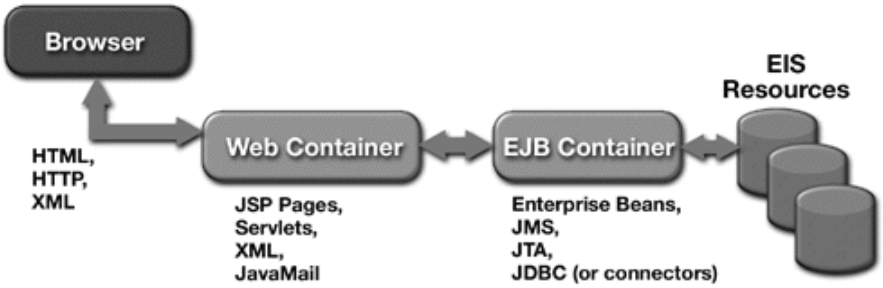


Abbildung 4.1: Java Enterprise Web-Anwendungsarchitektur

JavaBeans

JavaBeans Komponenten sind Java Klassen, die sich an bestimmte Richtlinien halten. Das Lesen und Schreiben der Attribute solcher Klassen wird über jeweils eine Methode durchgeführt. Diese müssen sich dabei an eine Namensrichtlinie halten. JavaBeans sind für die einfache Integration und Wiederverwendung in Anwendungen gedacht. Sie können für die Verbindung der Benutzerschnittstelle mit der Anwendungslogik verwendet werden.

Enterprise Beans

Enterprise Beans sind serverseitige Komponenten, die die grundlegende Funktionalität einer Anwendung enthalten. Der Business Tier einer Anwendung wird aus solchen Komponenten zusammengesetzt. Enterprise Beans eignen sich gut für die Entwicklung von umfangreichen verteilten Anwendungen. Sie werden dabei in einem Container ausgeführt, der ihnen Dienste auf Systemebene zur Verfügung stellt. Der Container kümmert sich dabei um das Transaktionsmanagement und die Sicherheit der ausgeführten Beans. Dadurch können sich Entwickler auf die Funktionalität der Anwendung konzentrieren. Weiters werden diese Komponenten am Server ausgeführt, was die Umsetzung von Anwendungen für Clients ohne Funktionalität, wie Web-Anwendungen, ermöglicht. Die gesamte Anwendung wird am Server ausgeführt und der Client dient nur der Präsentation und Interaktion. Enterprise Beans sind ebenfalls Java Klassen. Diese erfüllen die Richtlinien des Enterprise JavaBeans (EJB) Standards.

Es gibt verschiedene Arten von Enterprise Beans für unterschiedliche Verwendungszwecke. Diese werden im Folgenden beschrieben.

- **Session Beans**
  Eine Session Bean führt eine Aufgabe für einen Client innerhalb der Anwendung aus. Der Client ruft dabei Methoden der Bean auf, die die-

se ausführt und ihm das Ergebnis zurückliefert. Dabei schirmt sie den Client vor ihren internen Vorgängen, wie Zugriffe auf eine Datenbank über den EIS Tier, ab. Eine Session Bean existiert nur für jeweils einen Client. Mehrere Zugriffe haben zur Folge, dass die Bean innerhalb des Containers mehrfach instanziert wird. Dadurch wird eine sitzungsorientierte Verarbeitung möglich. Die Bean existiert dabei nur für die Dauer einer Sitzung.

- **Message-Driven Beans**
  Message-Driven Beans werden für die asynchrone Verarbeitung verwendet. Sie können Nachrichten empfangen und darauf reagieren.

Persistenz

Persistenz in der Java EE wird über Entities ermöglicht. Diese werden als Java Klassen angegeben, die jeweils eine Tabelle der darunter liegenden Datenbank darstellen. Durch Instanzierung erhält man Objekte, die persistente Daten repräsentieren und einer Reihe in der Tabelle entsprechen. Die Attribute solcher Objekte enthalten dabei die Daten. Beziehungen der Entities untereinander können ebenfalls repräsentiert werden.

JavaServer Faces

JavaServer Faces stellen ein komponentenorientiertes Rahmenwerk für die Definition von Benutzerschnittstellen für Web-Anwendungen dar. Die Erstellung von JSF-Seiten erfolgt nicht direkt mittels HTML, sondern es werden dabei höher abstrahierte Komponenten verwendet. Diese beschreiben Struktur und Verhalten der Seite und ermöglichen die Behandlung von Ereignissen und die Validierung von Eingaben. Weiters bieten sie Unterstützung für Internationalisierung. Die Navigation zwischen den Seiten kann mittels JSF einfach und zentral definiert werden. Bei der Ausführung einer Web-Anwendung können diese Seiten dann automatisch in entsprechende HTML-Seiten umgesetzt werden, die als Benutzerschnittstelle zum Client gesendet werden.

Die Verbindung der HTML-Benutzerschnittstelle mit der Anwendung geschieht mittels Backing Beans. Dabei handelt es sich um JavaBeans Komponenten, die mit einer JSF-Seite verknüpft sind und dieser ermöglichen, auf die Objektattribute und Methoden der Backing Bean zuzugreifen. Dadurch kann die Seite dynamisch erzeugt, Eingaben als Objektattribute der Bean gespeichert und Methoden als Reaktion auf Benutzeraktionen ausgeführt werden. Beispiele sind die Validierung von Formulareingaben und das Abschicken eines Formulars.

PostgreSQL Datenbankserver

Zusätzlich zur Java EE wird bei dieser Fallstudie ein Server für relationale Datenbanken für die Persistenz der Anwendungsdaten der Zielanwendung verwendet. Dafür wurde der PostgreSQL Datenbankserver Version 8.0 ausgewählt. Dieser wird über die Java EE in die Entwicklung eingebunden und das Schema für die Anwendung über deren Persistenzmechanismen mittels Entities verwendet.

4.3 Beschreibung des Ausgangssystems

Als repräsentatives Ausgangssystem wurde die Anwendung Cardfile [Lam06] ausgewählt. Es handelt sich um einen einfachen Datenbankmanager mit zeichenorientierter Benutzerschnittstelle, der ein Karteisystem darstellt. Man kann damit Karteien definieren und darin Einträge erstellen, ändern, löschen, suchen und drucken. Weiters stellt er Wartungsfunktionen für die zugrunde liegende Datenbank zur Verfügung. Ein Karteieintrag besteht aus mehreren Feldern, in denen jeweils auch mehrere Einträge vorkommen können. Die Suche kann über reguläre Ausdrücke erfolgen. Cardfile ist in der Programmiersprache C geschrieben und kann unter den Betriebssystemen Linux, System V, Sun OS/Solaris und BSD verwendet werden.

Cardfile erfüllt alle Auswahlkriterien für ein repräsentatives Ausgangssystem:

- Die Implementierung wurde in C, einer prozeduralen Sprache, durchgeführt. Es gibt mit C++ für diese zwar eine objektorientierte Erweiterung, diese wird aber nicht von der Zieltechnologie direkt unterstützt. Dadurch müssen sowohl Sprache als auch Paradigma transformiert werden.
- Die Architektur des Ausgangssystems ist nicht 3-schichtig. Die Anwendung unterteilt sich in den Anwendungsteil selbst und die Datenbank. In der Aufteilung auf Module und Funktionen innerhalb des Quellcodes sind Präsentation, Anwendungslogik und Datenbankzugriffe vermischt. Eine Trennung wahr offensichtlich kein Design-Kriterium. Deshalb eignet sich Cardfile als Untersuchungsobjekt für die Schichtentrennung.
- Cardfile ist ein nicht verteilt funktionierendes Einzelplatzsystem.
- Die Benutzerschnittstelle ist zeichenorientiert.
- Die Programmverarbeitung ist System-preemptiv.
- Das Datenbankmodell ist von dem des Zielsystems verschieden. Daten werden in einer flachen Dateistruktur gespeichert.

Die Anwendung des Web-Reengineering Prozesses soll das Ausgangssystem Cardfile in ein Zielsystem Web-Cardfile transformieren.

4.4 Schritt 1 - Reverse Engineering

4.4.1 Allgemeine Betrachtung

Ausgehende Artefakte für das Reverse Engineering sind der Quellcode, eine Beispieldatenbank und ein Benutzerhandbuch in Form einer kurzen Beschreibung in einer Textdatei. Weiters sind Informationen auf der Webseite der Anwendung zu finden.

Verwenden der Anwendung:

Zuerst werden das Benutzerhandbuch und die Informationen auf der Webseite gelesen. Danach wird eine lauffähige Version von Cardfile erzeugt. Daraus erhält man eine ausführbare Datei. Die Anwendung wird damit über die Kommandozeile einer Shell gestartet und läuft innerhalb dieser ab, wobei die Benutzerschnittstelle zeichenorientiert ausgegeben wird. Diese lauffähige Version wird verwendet, um sich mit der Anwendung vertraut zu machen. Es werden alle Funktionen ausprobiert, um ein erstes Programmverständnis aus Anwendersicht zu erhalten. Die Anwendung wird damit also Black-Box untersucht.

Untersuchen des Quellcodes:

Nach dieser Untersuchung wird der Quellcode durchgesehen. Dabei sind mehrere Artefakte zu untersuchen. Die Anwendung wurde mit der Programmiersprache C entwickelt. Der Quellcode ist auf mehrere Module und Header-Dateien aufgeteilt. Die Datenbank besteht aus Textdateien, wobei jede einzelne erzeugte Kartei durch mehrere Dateien repräsentiert wird.

Der Quellcode ist in 24 c-Module aufgeteilt. Für die Funktionen sind Prototypen angegeben. Die main-Funktion findet sich im Modul cardfile.c. Die Funktionen der Anwendung, die bei der Verwendung angewendet werden können, scheinen Entsprechungen bei den Modulen zu haben. Add, change, delete, find und printdb sind als Module vorhanden. Weiters sind ebenfalls die Wartungsfunktionen compress, dumpdb, extract und rbuildak als Module vorhanden. Für die Definition einer Datenbank ist das Modul define vorhanden. Diese und alle weiteren Module werden betrachtet, um einen ersten Eindruck des Verwendungszwecks zu erhalten. Dabei ist die Beschreibung, die sich in den Modulen am Anfang findet, hilfreich. Diese ist teilweise sehr ausführlich. Weitere Kommentare im Quellcode sind allerdings kaum vorhanden.

Bei der Betrachtung der Modulaufteilung bemerkt man, dass innerhalb der Module nur sehr wenige Funktionen implementiert sind. Bei Modulen mit mehreren Funktionen findet sich meist eine modulextern verwendete Funkti-

on. Weitere Funktionen innerhalb des Moduls werden nur von dieser verwendet. Die Strukturierung des Quellcodes zeichnet sich durch die Aufteilung auf eine große Anzahl von Modulen aus.

Weiters werden 3 Header-Dateien verwendet. In der Header-Datei des cardfile.c Moduls werden Strukturen definiert und es finden sich externe Funktionsdeklarationen. In der ascii.h sind Definitionen für ascii-Zeichen zu finden, in der patchlevel.h ist die Versionsnummer angegeben.

Die persistente Datenhaltung wird mittels Textdateien realisiert. Eine Kartei wird durch folgende Textdateien repräsentiert:

- Dateien mit der Endung .db enthalten die Datensätze einer Kartei. Dabei wird je Zeile ein Datensatzeintrag gemacht. Die Feldwerte eines Datensatzes werden mit : getrennt.
- In Dateien mit der Endung .def wird die Struktur einer Kartei deklariert. Datenbankname und Felddefinitionen sind hier zu finden. Weiters sind Anzahl und Namen der Indexdateien angegeben.
- Dateien mit der Endung .akX, wobei X eine Zahl ist, sind Indexdateien, die für die Suche innerhalb dieser Kartei verwendet werden. Die Indexzahl jedes Eintrags steht für die Zeilennummer des korrespondierenden Datensatzes in der .db Datei.

Bezüglich der Architektur ist eine 2-schichtige Aufteilung vorhanden. Diese besteht aus der Anwendung selbst, die Benutzerschnittstelle und Anwendungslogik implementiert, und weiters aus der Datenbank, die mittels Textdateien umgesetzt ist.

4.4.2 Redokumentation der Anforderungen

Nachdem die erste Betrachtung von Cardfile abgeschlossen ist, werden nun die Anforderungen rekonstruiert. Dabei wird die vorhandene Dokumentation sowie die lauffähige Anwendung herangezogen. Die funktionalen Anforderungen werden als UML Anwendungsfälle dokumentiert und diese in einem Anwendungsfalldiagramm für die gesamte Anwendung dargestellt.

Es werden folgende Anwendungsfälle identifiziert. Diese stellen die funktionalen Anforderungen dar.

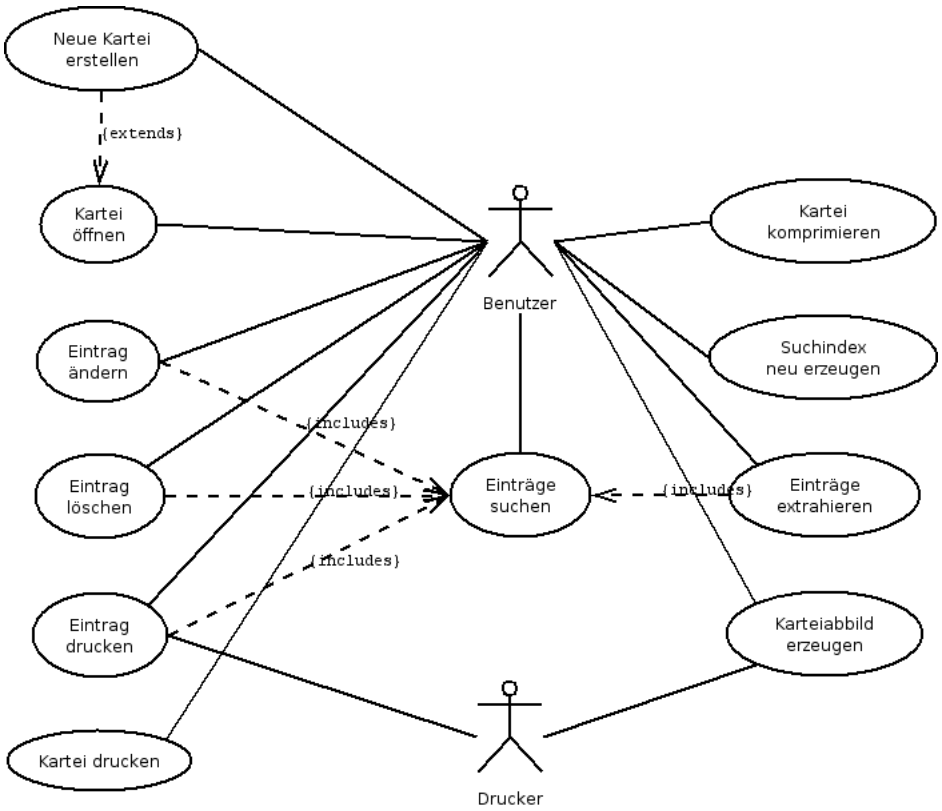


Abbildung 4.2: Anwendungsfalldiagramm

Beschreibung der Anwendungsfälle:

Name: Kartei öffnen
Beschreibung: Der Benutzer öffnet eine bestehende Kartei.
Akteure: Benutzer
Vorbedingungen: Keine
Ablauf: Der Benutzer öffnet eine bestehende Kartei. Er gibt dafür den Namen der Kartei an. Weiters gibt er an, ob die Kartei nur lesbar geöffnet werden soll.
Alternativer Ablauf: Die angegebene Kartei existiert nicht. Der Anwendungsfall „Neue Kartei erstellen“ wird ausgelöst.
Nachbedingungen: Die angegebene Kartei wurde geöffnet.

Name: Neue Kartei erstellen
Beschreibung: Der Benutzer erstellt eine neue Kartei. Er definiert die einzelnen Felder und legt ihre Eigenschaften fest.
Akteure: Benutzer
Vorbedingungen: Eine Kartei mit dem angegebenen Namen existiert nicht.
Ablauf: Der Benutzer definiert nacheinander alle Felder für die Einträge der Kartei. Für jedes Feld können dabei folgende Eigenschaften festgelegt werden.

- Name des Feldes
- Maximale Länge eines Wertes in diesem Feld
- Eine Angabe, ob ein Wert eingegeben werden muss
- Eine Angabe, ob nach diesem Feld gesucht werden kann
- Trennzeichen für die Trennung mehrerer Wertangaben
- Die Seite, auf der das Feld angezeigt wird
- Die Position als Zeile und Spalte, an der der Feldname angezeigt wird
- Die Position als Zeile und Spalte, an der der Wert des Feldes eingegeben wird
- Ein Format als regulärer Ausdruck, dem eine Wertangabe entsprechen muss

Alternativer Ablauf: Keiner
Nachbedingungen: Die definierte Kartei wurde erstellt.

Name: Einträge suchen
Beschreibung: Der Benutzer sucht nach Einträgen in der Kartei. Alle gefundenen Einträge werden angezeigt.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer sucht nach Einträgen in der Kartei. Er gibt dafür Suchkriterien für Felder an, nach denen gesucht werden soll. Mehrere Suchkriterien können logisch mittels „UND“ und „ODER“ verknüpft werden. Die Suchkriterien selbst können als reguläre Ausdrücke angegeben werden. Danach liefert die Suche die Ergebniseinträge. Diese können nacheinander betrachtet werden.
Alternativer Ablauf: Es wurden keine mit den Suchkriterien übereinstimmenden Einträge gefunden.
Nachbedingungen: Keine

Name: Eintrag hinzufügen
Beschreibung: Der Benutzer fügt der Kartei neue Einträge hinzu.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer kann für alle Felder eines Eintrags Werte angeben. Er kann mehrere Einträge hintereinander erzeugen. Danach werden die Einträge zur Kartei hinzugefügt.
Alternativer Ablauf: Keiner
Nachbedingungen: Die angegebenen Einträge wurden zur Kartei hinzugefügt.

Name: Eintrag ändern
Beschreibung: Der Benutzer ändert einen bestehenden Eintrag der Kartei.
Akteure: Benutzer
Vorbedingungen: Ein gefundener und angezeigter Eintrag.
Ablauf: Der Benutzer kann für alle Felder des angezeigten Eintrags neue Werte angeben. Diese ersetzen die alten Werte. Danach wird der Eintrag in der Kartei geändert.
Alternativer Ablauf: Keiner
Nachbedingungen: Der geänderte Eintrag wurde zur Kartei hinzugefügt und ersetzt den ursprünglichen Eintrag.

Name: Eintrag löschen
Beschreibung: Der Benutzer löscht einen bestehenden Eintrag der Kartei.
Akteure: Benutzer
Vorbedingungen: Ein gefundener und angezeigter Eintrag.
Ablauf: Der Benutzer löscht den momentan angezeigten Eintrag. Dieser Eintrag scheint nun in Suchergebnissen nicht mehr auf.
Alternativer Ablauf: Keiner
Nachbedingungen: Der Eintrag wurde als gelöscht markiert und scheint in Suchen nicht mehr auf. Der Eintrag wurde dadurch nicht endgültig entfernt.

Name: Eintrag drucken
Beschreibung: Der Benutzer druckt einen bestehenden Eintrag der Kartei.
Akteure: Benutzer, Drucker
Vorbedingungen: Ein gefundener und angezeigter Eintrag.
Ablauf: Der Benutzer wählt den momentan angezeigten Eintrag zum Drucken aus. Der Eintrag wird am Standarddrucker ausgedruckt.
Alternativer Ablauf: Keiner
Nachbedingungen: Keine

Name: Kartei drucken
Beschreibung: Der Benutzer druckt eine Kartei. Er kann ein Format für den Ausdruck definieren. Weiters kann er ein Ziel für den Ausdruck angeben.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer definiert das Ausgabeformat für den Ausdruck. Er gibt einen Dateinamen einer Datei mit extrahierten Einträgen einer Kartei an. Er gibt den Dateinamen der Zieldatei für den Ausdruck an. Weiters gibt er die Breite für den Ausdruck an. Danach werden die Einträge in die Zieldatei entsprechend der Formatangabe gedruckt.
Alternativer Ablauf: Wurde kein Dateiname für die Datei mit extrahierten Einträgen angegeben, werden alle Dateien der geöffneten Kartei gedruckt.

Es wurde kein Dateiname für die Zieldatei angeben, sondern eine Angabe mit „|“ und einem Kommando gemacht. Das Ergebnis des Ausdrucks wird an dieses Kommando übergeben.
Nachbedingungen: Keine

Name: Karteiabbild erzeugen
Beschreibung: Der Benutzer erzeugt ein Abbild der geöffneten Kartei, das alle Informationen zu den bestehenden Einträgen enthält.
Akteure: Benutzer, Drucker
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer erzeugt ein Abbild der geöffneten Kartei. Dieses beinhaltet alle bestehenden Einträge der Kartei, weiters alle Indexeinträge mit Suchbegriffen zu den Einträgen. Diese sind nach den Suchfeldern aufsteigend geordnet. Das Abbild wird am Standarddrucker ausgedruckt.
Alternativer Ablauf: Keiner
Nachbedingungen: Keine

Name: Kartei komprimieren
Beschreibung: Der Benutzer komprimiert die Kartei.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer komprimiert die Kartei. Zuvor gelöschte Einträge werden endgültig entfernt und der zuvor belegte Speicherplatz zurückgewonnen.
Alternativer Ablauf: Keiner
Nachbedingungen: Zuvor gelöschte Einträge wurden endgültig aus der Kartei entfernt und können nicht mehr wiederhergestellt werden.

Name: Suchindex neu erzeugen
Beschreibung: Der Benutzer erzeugt den Suchindex neu.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer erzeugt den Suchindex neu.
Alternativer Ablauf: Keiner
Nachbedingungen: Der Suchindex wurde neu erzeugt. Eine Suche liefert die Suchergebnisse in alphabetisch aufsteigender Reihenfolge.

Name: Einträge extrahieren
Beschreibung: Der Benutzer extrahiert ausgewählte Einträge in eine Datei.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer gibt zuerst einen Namen für die Datei an, in die extrahiert werden soll. Wird kein Name angegeben, wird der Standardname „temp.ext“ verwendet. Danach verwendet der Benutzer den Anwendungsfall „Einträge suchen“, um zu extrahierende Einträge auszuwählen. Nach Abschluss dieser Auswahl werden die gefundenen Einträge in die zuvor angegebene Datei geschrieben.
Alternativer Ablauf: Keiner
Nachbedingungen: Keine

Nicht-funktionale Anforderungen:

Eine mögliche nicht-funktionale Anforderung ist die Portabilität unter Unix-basierten Systemen. Es werden keine weiteren nicht-funktionalen Anforderungen identifiziert.

4.4.3 Design Recovery

Aufrufgraph

Um die interne Struktur von Cardfile zu erfassen, wird ein interprozeduraler Kontrollflussgraph (Aufrufgraph) der Anwendung erstellt. Da die strukturelle Aufteilung in viele Module mit nur wenig enthaltenen Funktionen besteht, wird der Aufrufgraph auf Modulebene erstellt. Auf dieser hierarchischen Ebene erhält man eine sinnvolle Repräsentation. Der Aufrufgraph ist in Abbildung 4.3 dargestellt.

Da die Funktionen einiger Module eine hohe Anzahl an Aufrufen aufweisen, wird für diese Module die Darstellung der Abhängigkeiten im Graphen entfernt. Dadurch lässt sich das Strukturverständnis erhöhen, da die Abhängigkeiten nun graphisch vom Entwickler besser erfasst werden können. Der reduzierte Aufrufgraph ist in Abbildung 4.4 dargestellt.

Basierend auf dieser Darstellung wird der modulinterne Quellcode betrachtet. Dadurch wird ein Verständnis für das Programm entwickelt, das we-

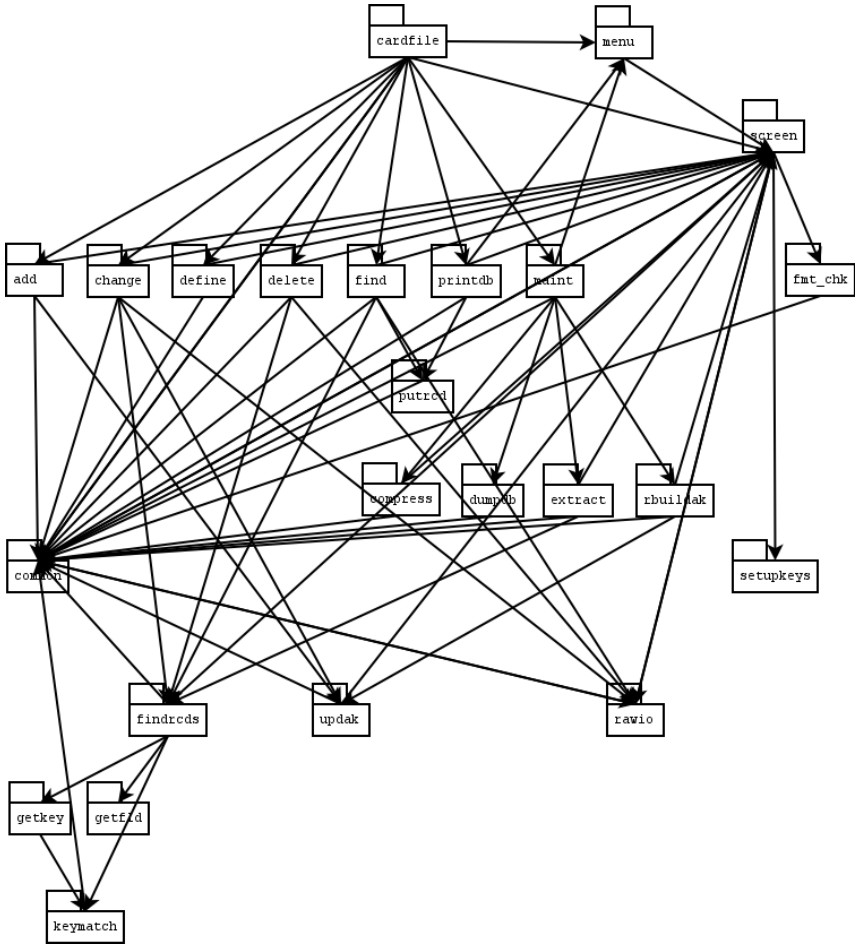


Abbildung 4.3: Aufrufgraph auf Modulebene

sentlich für die folgenden Schritte ist. Weitere Erkenntnisse bezüglich struktureller Aufteilung, Programmablauf und Verhalten werden im Folgenden beschrieben.

- **Ablauf der Anwendung**
  Der Ablauf wie bei der Verwendung der Anwendung ist aus dem Graphen ersichtlich. Ausgehend vom Modul cardfile, das die main-Funktion enthält, gelangt man entweder zu define oder zu den Karteifunktionen, die über das Menü ausgewählt werden. Dort kann auch über das Modul maint zu den Wartungsfunktionen verzweigt werden. Funktionen, die auf Suchergebnissen operieren, sind durch die Verwendung des Moduls findrcds ersichtlich. Diese finden sich in den Modulen change, delete, find und extract. Weiters sind Funktionen, die drucken, durch die Abhängigkeit vom Modul putrcd ersichtlich. Funktionen, die ein Neuerzeugen der Indexdateien verwenden, sind durch die Abhängigkeit zum Modul

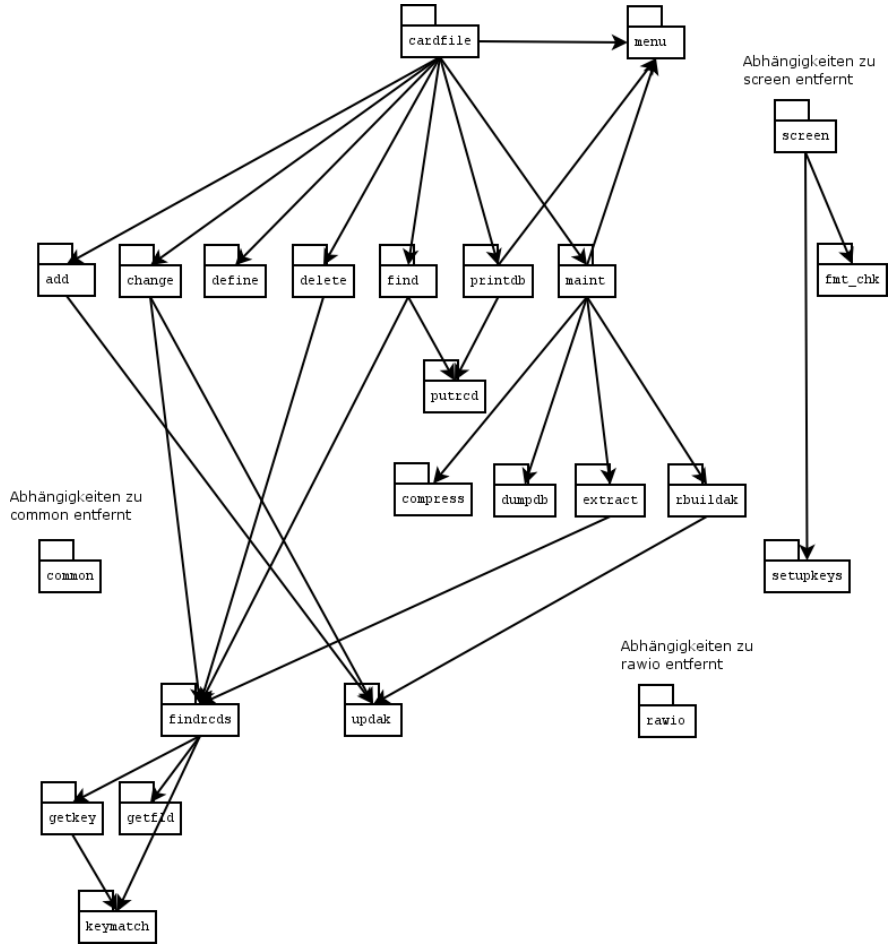


Abbildung 4.4: Reduzierter Aufrufgraph auf Modulebene

updak erkennbar.

- **Suche**
  Die Suche nach Einträgen wird über die Funktion findrcds des gleichnamigen Moduls durchgeführt. Diese verwendet dafür 3 weitere Module, stellt selbst aber die einzige Abhängigkeit zu suchbasierten Funktionen dar. Module, die sie verwenden, sind change, delete, find und extract.

- **Drucken**
  Funktionen, die drucken, sind durch die Abhängigkeit zum Modul putrcd ersichtlich. Die Funktion „Find“ der Anwendung ermöglicht das Drucken einzelner gefundener Einträge. Die Funktion „Printdb“ der Anwendung ermöglicht das formatierte Drucken der Kartei. Die entsprechenden Module find und printdb weisen Abhängigkeiten zu putrcd auf.

- **Indexdateien**
  Die Anwendung beinhaltet Funktionen, nach deren Ausführung die Indexdateien neu erzeugt werden müssen, um eine sortierte Indexreihenfolge zu gewährleisten. Diese ändern die Anzahl der Einträge in der Kartei. Sie sind durch Abhängigkeiten von Funktionen des Moduls updak zu identifizeren. Die Funktion „Add“ der Anwendung fügt Einträge hinzu. Die Funktion „Change“ macht dies ebenfalls, wobei der originale Eintrag als gelöscht markiert wird. Nach der Durchführung einer dieser Funktionen werden die Indexdateien automatisch neu erzeugt. Ein vom Benutzer direkt ausgelöstes Erzeugen ist mittels der Wartungsfunktion „Rebuild AK's“ möglich. Weiters sollte die automatische Erzeugung auch nach dem Komprimieren einer Kartei stattfinden, was, wie aus dem Graphen ersichtlich, nicht der Fall ist. Dies muss vom Benutzer selbst über die entsprechende Wartungsfunktion gemacht werden.

- **Menüauswahl**
  Das Modul menu übergibt Menüoptionen an screen und nimmt das Auswahlergebnis entgegen. Es beinhaltet selbst keine Definition der Menüoptionen. Diese werden im Modul cardfile für die Karteifunktionen und im Modul maint für die Wartungsfunktionen definiert. Beide verwenden das Modul menu, was aus dem Graphen ersichtlich ist.

- **Ein- und Ausgabe**
  In der gleichnamigen Funktion im Modul screen findet sich der graphisch strukturelle Teil der Benutzerschnittstelle. Diese wird von verwendenden Funktionen mit Daten für die Darstellung aufgerufen, wobei die Struktur Sdata für die Datenübergabe zuständig ist. screen verwendet exklusiv 2 weitere Module. Einzelne Nachrichten können mittels der Funktionen msg und help des Moduls common angezeigt werden. Diese schreiben die Nachricht zeichenorientiert und unabhängig von der screen-Funktion. Das Modul rawio beinhaltet Funktionen zur zeichenweisen Ein- und Ausgabe. Die hier erwähnten Module haben hohe Abhängigkeiten zu anderen Modulen, da sie die Benutzerschnittstelle implementieren und diese häufig verwendet wird.

Ergebnisse

In diesem Schritt wurde zunächst eine Übersicht über die Anwendung gewonnen. Die bereits vorhandene Dokumentation wurde gelesen, die Anwendung wurde verwendet und der Quellcode wurde durchgesehen. Danach wurde Cardfile redokumentiert. Die Anforderungen wurden rekonstruiert und ein Aufrufgraph wurde erstellt, der die interne Struktur verdeutlichen soll. Diese erzeugte Dokumentation vertiefte das Programmverständnis. Besonders die Graphendarstellung ermöglichte eine gezielte Betrachtung des Quellco-

des und vermittelte Verständnis für den internen Ablauf der Anwendung. Zusätzlich zu Dokumentationszwecken und Programmverstehen, kann der Graph für den nächsten Schritt als Eingabe für automatisches Clustering verwendet werden.

Der Aufrufgraph und die hier gewonnenen Erkenntnisse über die Anwendung werden im nächsten Schritt verwendet, um Cardfile zu restrukturieren. Die Anforderungen werden beim Forward-Engineering zum Zielsystem als Basis für dessen Anforderungen weiterverwendet.

4.5 Schritt 2 - Restrukturierung

4.5.1 Remodularisierung

Bei der Remodularisierung werden die strukturellen Einheiten von Cardfile so zu Gruppen zusammengefasst, das zusammengehörende Einheiten in ein und dieselbe Gruppe platziert werden. Dadurch werden ähnliche Einheiten zusammengefasst und verschiedene Teile getrennt. Das Ergebnis ist eine technologisch und funktional zum ursprünglichen Cardfile äquivalente Variante, dessen interne Struktur jedoch besser aufgeteilt und dadurch leichter verständlich ist. Durch die verbesserte Struktur werden folgende Schritte vereinfacht.

Für die Remodularisierung werden die Ergebnisse aus dem vorherigen Schritt verwendet. Um eine Einteilung automatisch zu treffen, wird hier die Dominanzanalyse verwendet. Der erzeugte Aufrufgraph stellt die Basis dafür dar. Nachdem automatisch gruppiert wurde, wird das Ergebnis analysiert und gegebenenfalls modifiziert, um zu einer guten Einteilung zu gelangen. Dafür ist das zuvor entwickelte Programmverständnis wesentlich.

Automatische Einteilung mittels Dominanzanalyse

Wie zuvor wird auch hier der Quellcode auf der hierarchischen Ebene der Module betrachtet. Die Dominanzanalyse liefert für den Aufrufgraphen folgenden Dominanzbaum.

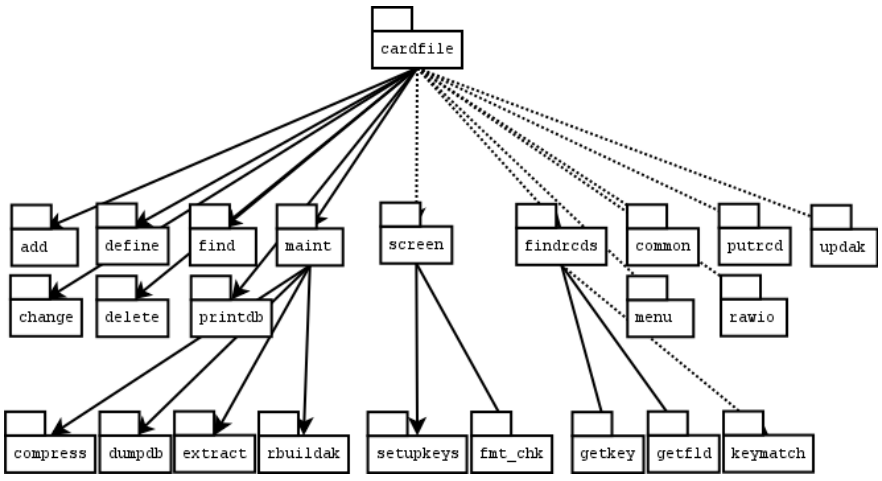


Abbildung 4.5: Dominanzbaum auf Modulebene

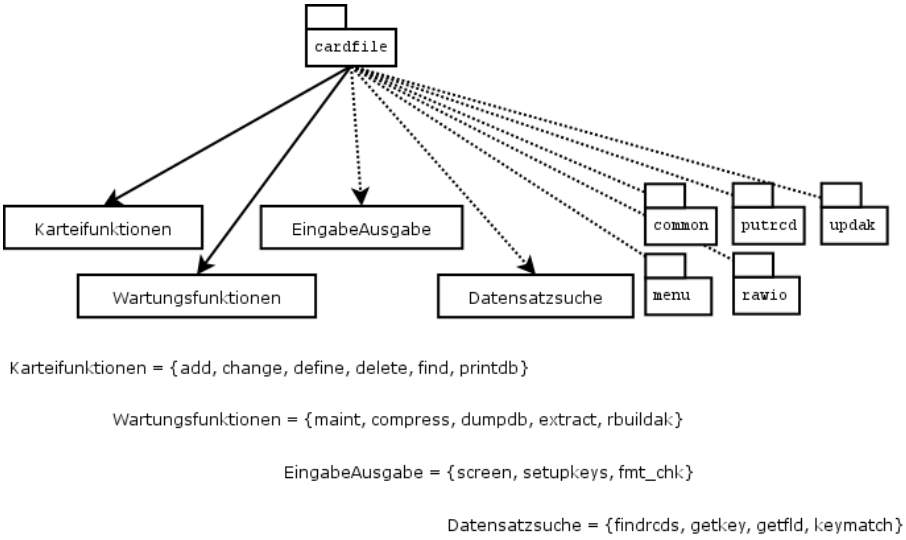


Abbildung 4.6: Dominanzbaum mit zu Gruppen zusammengefassten Modulen

Obwohl das Modul keymatch von findrcds nicht stark dominiert wird, wird es hier der Gruppe Datensatzsuche zugewiesen, da es zu diesem Teilsystem gehört und dieses nur über das Modul findrcds als dessen Schnittstelle verwendet werden kann.

Zusätzliche Restrukturierungen

Da der Algorithmus auf Modulebene durchgeführt wurde, sollen nun die einzelnen Gruppen für sich betrachtet werden. Es soll überprüft werden, ob eine gute Einteilung erreicht wurde. Das Maß dafür ist das Programmverständnis der Entwickler. Da auf hierarchisch hoher Ebene gruppiert wurde, sind unter Umständen modulare oder prozedurale Aufteilungen angebracht. Weiters könnten Gruppen zerlegt werden, falls die Einteilung zu allgemein getroffen wurde.

Nach der Betrachtung der Gruppen werden folgende Restrukturierungen durchgeführt.

- **Modul cardfile**
  Die main-Funktion des Moduls cardfile wird in 2 Teilfunktionen aufgespalten. Die erste Teilfunktion behält weiter den Namen main und beinhaltet das Öffnen einer Kartei und alternatives Aufrufen der Definition. Sie dient also der Karteiverwaltung. Die zweite Teilfunktion beinhaltet die Definition für das Auswahlmenü der Karteifunktionen. Diese Funktion erhält den Namen cardfileMenu. Sie ist der Funktion

maint des gleichnamigen Moduls, dass das Auswahlmenü für die Wartungsfunktionen implementiert, sehr ähnlich. Diese Funktion und das Modul wird in maintMenu umbenannt, um diese Ähnlichkeit zu berücksichtigen. Weiters wird für die Funktion cardfileMenu ein neues Modul cardfileMenu erstellt und diese dorthin verschoben. Damit weisen beide Menüdefinitionen die gleiche Struktur auf.

- **Gruppe Karteifunktionen**
  Nachdem die main-Funktion aufgeteilt wurde, wird die Gruppe Karteifunktionen neu betrachtet. Der Dominanzbaum hat sich verändert, da das Modul cardfile, das die Wurzel darstellt, aufgeteilt wurde. Zwischen diesem und den Modulen der Karteifunktionen befindet sich nun das Modul cardfileMenu. Dieses weist eine starke Dominanzrelation zu den Karteifunktionen auf, wobei aber das Modul define nicht mehr enthalten ist. Dieses wird weiterhin von cardfile stark dominiert. Aufgrund dieser neuen Betrachtung wird das Modul define aus der Gruppe Karteifunktionen entfernt und in eine neue Gruppe Karteiverwaltung verschoben. In diese wird weiters die Wurzel selbst, das Modul cardfile, verschoben. Die Gruppe Karteifunktionen ist nun strukturell äquivalent zur Gruppe Wartungsfunktionen und beinhaltet jene Module, die auf einer geöffneten Kartei operieren. Die Verwaltung der Karteien wird durch die Module der Gruppe Karteiverwaltung durchgeführt, die für das Öffnen von vorhandenen Karteien und die Definition von neuen Karteien zuständig sind.

- **Funktionen auf Einträgen**
  Eine Aufrufbeziehung, die nicht statisch entschieden werden kann, findet sich in den Funktionen, die auf gefundenen Einträgen einer Suche operieren. Diese sind das Ändern, das Löschen, das Anzeigen und die Extraktion von Einträgen. Sie werden durch den Benutzer ausgelöst, der mittels der Suche einzelne Einträge durchgeht und bei Bedarf eine der Funktionen darauf anwendet. Die Entscheidung, welche Funktion zur Anwendung kommt, erfolgte zuvor durch den Benutzer, indem er einen der Menüeinträge „Change“, „Delete“, „Find“ oder „Extract“ auswählte. Die jeweils zugehörige Funktion wird durch die Suchfunktion aufgerufen. Dieser Aufruf erfolgt über einen Zeiger, den die Suchfunktion als Parameter erhielt. Welche Funktion tatsächlich zur Anwendung kommt, ist von der zuvor getroffenen Auswahl des Benutzers abhängig. Um diese Aufrufbeziehung zu repräsentieren, werden die Funktionen processc, processd, display und write_rcd in einer Gruppe zusammengefasst und von den Modulen change, delete, find und extract in 4 neue Module changeRecord, deleteRecord, displayRecord und extractRecord verschoben. changeRecord erhält auch die zum Ändern gehörige Funktion doadd. Betrachtet man den Dominanzbaum für diese Aufrufabhängigkeit nach dieser Restrukturierung, ergibt sich eine starke

Dominanzrelation dieser Funktionen zur Suchfunktion findrcds. Diese wird durch eine Gruppe Datensatzfunktionen repräsentiert.

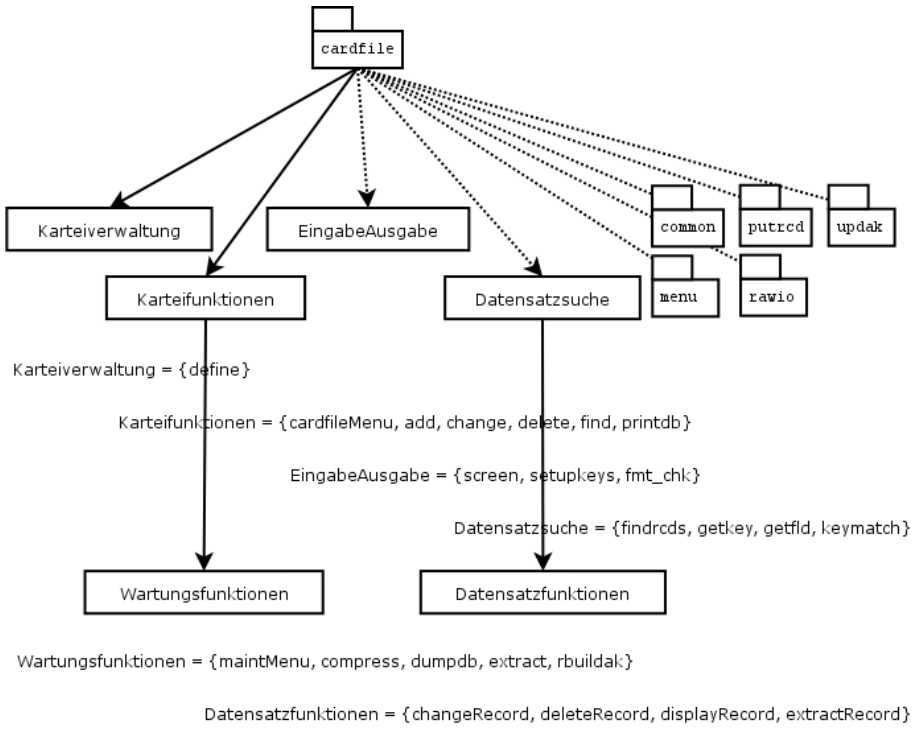


Abbildung 4.7: Dominanzbaum nach durchgeführten Restrukturierungen

Umgesetzt wird die Gruppeneinteilung mittels Verzeichnissen des Dateisystems, da hierarchisch höher liegende Strukturierungsmöglichkeiten hier nicht zur Verfügung stehen.

Ergebnisse

Die Dominanzanalyse ermöglichte hier eine automatische Gruppeneinteilung für die strukturellen Einheiten von Cardfile. Danach wurde das Ergebnis kontrolliert und zusätzliche Aktionen durch die Entwickler durchgeführt, um zu einer guten Einteilung zu gelangen. Eine gute Einteilung erhöht die Verständlichkeit des Gesamtsystems und liefert Teilsysteme mit starkem inneren Zusammenhalt und geringer Kopplung dieser Systeme untereinander.

Diese Einteilung unterstützt die Transformation in eine objektorientierte Umgebung, da starker Zusammenhalt und geringe Kopplung auch Kriterien des objektorientierten Designs sind. Weiters werden dadurch die Migrationsmethoden unterstützt, da die Teilsysteme aufgrund dieser Kriterien gute Kandidaten für eine schrittweise Migration darstellen.

Es ist ersichtlich, dass eine gute Einteilung in Gruppen nicht vollständig automatisch getroffen werden kann und mittels zuvor entwickeltem Programmverständnis von den Entwicklern durchgeführt werden sollte. Wissen um die interne Struktur, Ablauf und Anwendungsmöglichkeiten des Systems haben hohe Wichtigkeit. Dennoch sind Clustering-Methoden ein wesentliches Hilfsmittel, da sie automatisch Ergebnisse liefern. Dies ist vor allem bei umfangreichen Systemen, für die es keine Dokumentation gibt, wichtig. Automatische Ergebnisse können durch die Entwickler überprüft und gegebenenfalls verbessert werden.

4.5.2 Schichtentrennung

Bei der Schichtentrennung wird versucht, das remodularisierte Cardfile weiter zu einer linearen Schichtenstruktur zu restrukturieren. Dies soll als Vorbereitung für die Transformation zur Zielarchitektur dienen und dessen Durchführung erleichtern. Die strukturellen Einheiten von Cardfile werden nun auf die Implementierung von Benutzerschnittstellen und Datenbankzugriffen hin untersucht. Damit soll eine Einteilung in die 3 Schichten Präsentation, Anwendungslogik und Datenhaltung erreicht werden. Dadurch können durch Restrukturierung die 3 Schichten bereits hier getrennt und linear angeordnet werden.

Abtrennung der Benutzerschnittstelle

Hier werden strukturelle Einheiten gesucht, die Benutzerschnittstellenfunktionalität enthalten. Diese werden in die Schicht Präsentation verschoben. Da Cardfile eine zeichenorientierte Benutzerschnittstelle aufweist, können Einheiten der Präsentation über die Eigenschaft der Ausgabe von Zeichen am Bildschirm und der Eingabe von Zeichen über die Standardeingabe gefunden werden.

Funktionalität der Benutzerschnittstelle wird in den Modulen screen, rawio, setupkeys, common und cardfile gefunden. Das Modul common enthält mit msg und help 2 Funktionen für das Anzeigen von Nachrichten am Bildschirm. Die dritte in diesem Modul enthaltene Funktion getfield hat allerdings nichts mit der Benutzerschnittstelle zu tun. Deshalb wird dieses Modul aufgespalten und die 2 Funktionen msg und help in ein neues Modul messages verschoben, das eindeutig der Präsentation zugeordnet werden kann. In dieses Modul wird auch die Funktion usage des Moduls cardfile verschoben, die einen Verwendungshinweis am Bildschirm ausgibt und damit wie msg und help der Nachrichtenausgabe dient. Die Gruppe EingabeAusgabe wird ebenfalls aufgespalten. Die Module screen, rawio und setupkeys werden der Präsentationsschicht zugeordnet. Das Modul fmt_chk verbleibt im Anwendungsteil

von Cardfile. Der Gruppenname EingabeAusgabe wird in beiden Schichten verwendet, um weiterhin diese Gruppeneinteilung zu erhalten.

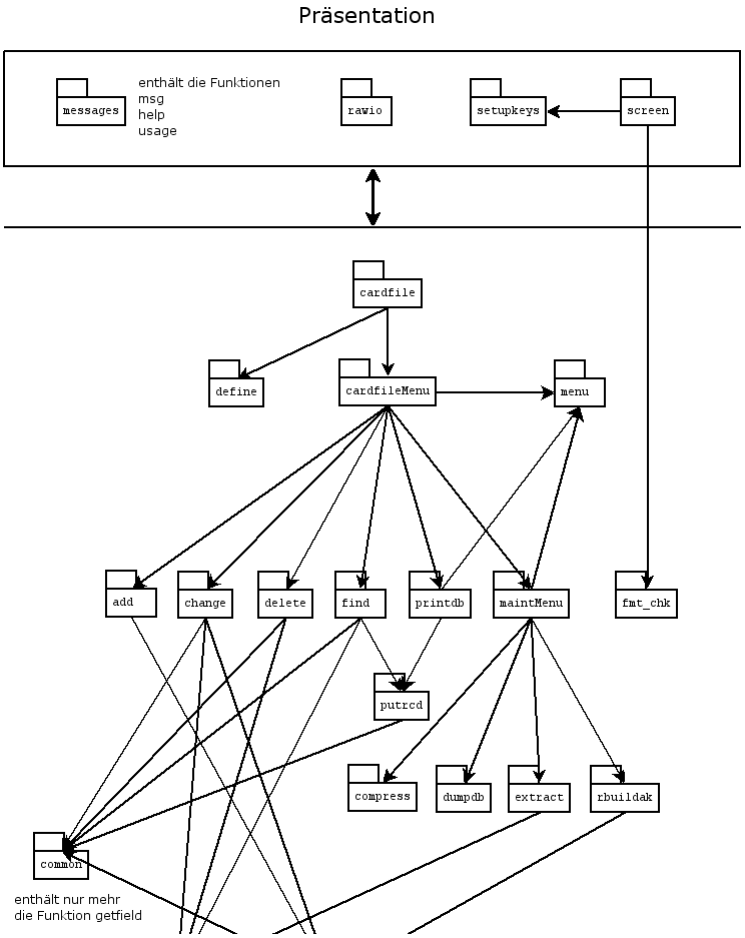


Abbildung 4.8: Aufrufgraph auf Modulebene mit abgetrennter Präsentation

Abbildung 4.8 zeigt nicht den gesamten Aufrufgraphen, sondern nur den relevanten Teil. Weiters werden, wie zuvor, die Abhängigkeiten zu screen und rawio nicht dargestellt. Die hohen Abhängigkeiten vom Modul common bestehen nun nicht mehr. In diesem Modul findet sich nur mehr die Funktion getfield. Oft verwendet wird die Funktion msg, die vom Modul common in das Modul messages verschoben wurde. Dieses Modul weist dadurch nun hohe Abhängigkeiten auf.

Nach diesen Restrukturierungen findet sich in der Schicht Präsentation die Implementierung der Benutzerschnittstelle von Cardfile. Diese entspricht der Client-seitigen Präsentation. Die Aufteilung wurde wie zuvor mittels Verzeichnissen umgesetzt.

Logische Architektur

Betrachtet man die bisher durchgeführten Restrukturierungen in Hinsicht auf die Zielarchitektur, erkennt man, dass durch die Abtrennung der Benutzerschnittstelle diese bereits erreicht wurde. Die Transformation im folgenden Schritt setzt diese in HTML-Seiten um, die für den Client als Benutzerschnittstelle dienen und gemeinsam mit diesem die Präsentationsschicht darstellen. Die Datenbank von Cardfile war ursprünglich bereits von der Anwendung getrennt, da sie mittels Textdateien realisiert ist. Durch die Abtrennung der Datenbankzugriffe wurde innerhalb der Anwendung eine Kapselung der Verwendung der Datenhaltung erreicht. Dieser Teil verbleibt aber auch am Zielsystem innerhalb der Anwendung selbst, während die Präsentation getrennt von dieser mittels HTML umgesetzt wird. Es liegt deshalb nahe, die 3-schichtige Architektur selbst und die Repräsentation dieser innerhalb der Anwendung zu unterscheiden. Die Aufteilung in Präsentation, Anwendungslogik und Datenhaltung wird als physische Architektur bezeichnet. Diese stellt weiterhin die Architektur einer Web-Anwendung dar. Die abgetrennte Benutzerschnittstelle von Cardfile entspricht der Präsentation, während die Textdateien die Datenhaltung darstellen. Diese getrennten Teile werden innerhalb der Anwendung strukturell repräsentiert. Bereits durchgeführt wurde die Abtrennung der Datenbankzugriffe. Diese Kapseln die Verwendung der Datenhaltung und ermöglichen Transparenz bezüglich deren Technologie.

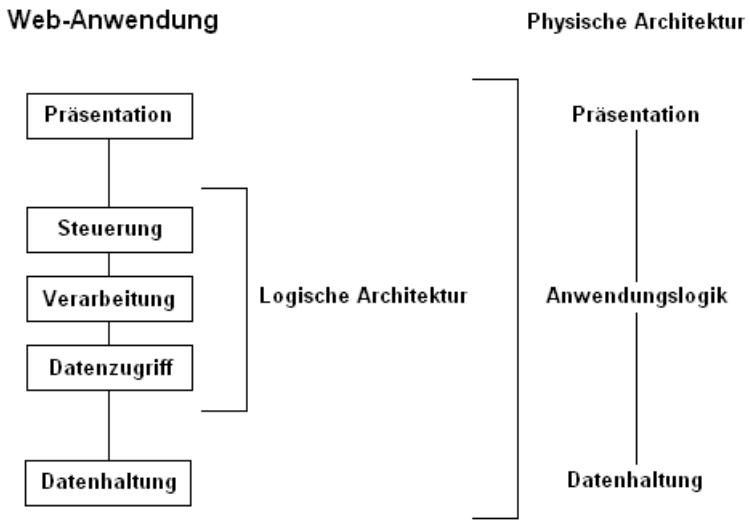


Abbildung 4.9: Physische und logische Schichten

Um die 3 Schichten innerhalb der Anwendung zu repräsentieren, wird die logische Architektur dafür wie folgt festgelegt. Diese 3 Teile der Anwendung sind wie die physischen Schichten linear angeordnet.

- **Steuerung**
  Die Steuerungsschicht stellt die Verbindung zur Präsentationsschicht her, die die Benutzerschnittstelle implementiert. Sie nimmt die Eingabe von Daten durch den Benutzer entgegen, kontrolliert die Darstellung der Benutzerschnittstelle und dient der Navigation, die der Benutzer durchführt.

- **Verarbeitung**
  In der Verarbeitungsschicht ist die Kernfunktionalität der Anwendung enthalten. Sie wird von der Steuerung verwendet, um diese Funktionalität für den Benutzer verfügbar zu machen. Dabei verwendet sie die Datenzugriffsschicht, um auf die Datenhaltung zuzugreifen, die sie für die Erfüllung ihrer Aufgaben benötigt.

- **Datenzugriff**
  In der Datenzugriffschicht sind Funktionen für den Zugriff auf die Datenhaltung zu finden. Diese schirmen die Verarbeitungsschicht vor der physischen Realisierung der Persistenz ab.

  Strukturelle Einheiten für den Datenzugriff greifen auf die Datenhaltung zu, wobei sie dessen Technologie direkt verwenden. Diese Einheiten schirmen die Logik vor dieser technologische Abhängigkeit ab und stellen Funktionen für den Datenzugriff in der Implementierungssprache zur Verfügung.

Wrapping der Datenhaltung

Hier werden strukturelle Einheiten gesucht, die Datenbankzugriffe enthalten. Diese werden in die Schicht Datenzugriff der Anwendungslogik verschoben. Da die Datenbank von Cardfile mittels Textdateien realisiert ist, können Einheiten der Datenhaltung über die Eigenschaft des Zugriffs auf diese Dateien gefunden werden.

Diese Zugriffe auf die Textdateien werden mittels Wrapping von der Anwendung getrennt. Dafür werden 3 neue Module, Cardfile, Definition und Ak, erstellt, die Funktionen für die Manipulation dieser 3 Teile der Datenbank zur Verfügung stellen. Diese Funktionen umhüllen die tatsächlichen Zugriffe auf die Dateien in der Implementierungssprache, um diese Vorgänge von verwendenden Einheiten abzuschirmen. Die dort erfolgten direkten Zugriffe werden entfernt, daraus Funktionen für Cardfile, Definition und Ak erstellt, und die entfernten Teile durch Aufrufe dieser Funktionen ersetzt. Dabei wird das Mo-

dul getkey entfernt. Die darin enthaltenen Funktion getkey wird in findKey umbenannt und in das Modul Ak verschoben. Weiters wird das Modul keymatch dem Datenzugriff zugeordnet. Um Abhängigkeiten des Datenzugriffs vom Rest der Anwendungslogik zu vermeiden, wird ein Teil der Funktion field_match in eine neue Funktion matchValue des Moduls Cardfile extrahiert. keymatch wird damit nur mehr in der Datenzugriffsschicht verwendet.

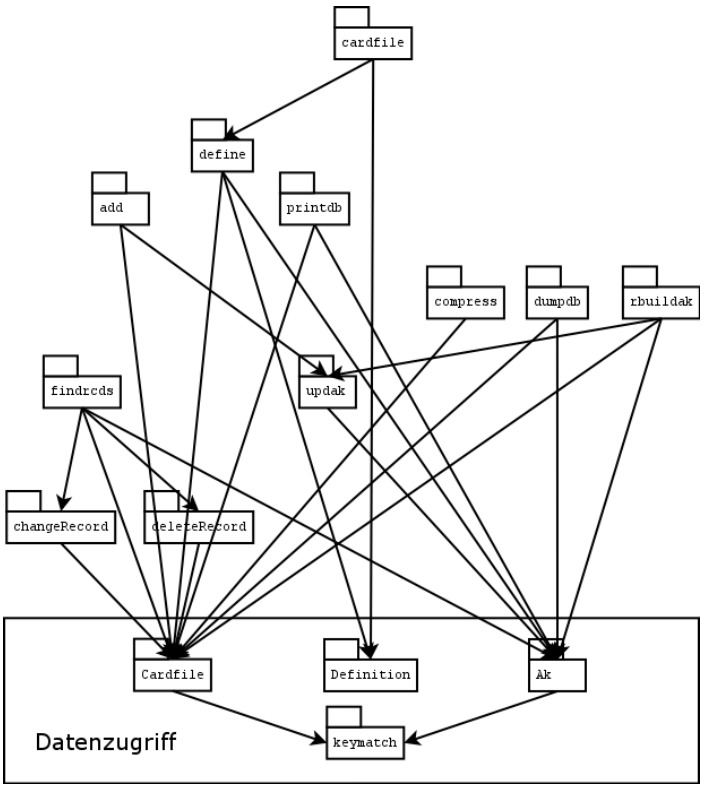


Abbildung 4.10: Aufrufgraph auf Modulebene mit getrenntem Datenzugriff

Abbildung 4.10 zeigt nicht den gesamten Aufrufgraphen, sondern nur jene Module, die Abhängigkeiten zum Datenzugriff aufweisen.

Durch das Wrapping wurde die Verwendung der Datenhaltung innerhalb der Anwendungslogik gekapselt. Die 3 Module repräsentieren die Schicht für den Datenzugriff. Zu der zuvor abgetrennten Präsentation bestehen keine Abhängigkeiten dieser Schicht. Nur die Anwendungslogik greift darauf zu.

Trennung von Steuerung und Verarbeitung

Bisher wurde die Präsentation von der Anwendungslogik getrennt. Innerhalb der Anwendungslogik wurde strukturell eine Datenzugriffsschicht umgesetzt. Der nun verbleibende Teil soll strukturell in Steuerung und Verarbeitung aufgeteilt werden.

Eine Identifikation von Programmteilen für Steuerung und Verarbeitung muss aufgrund des Fehlens genauer Kriterien nach Ermessen des Entwicklers erfolgen. Die Steuerung kann über die Eingabe und Ausgabe von Daten in Zusammenhang mit der bereits abgetrennten Präsentation erfolgen. Die Verarbeitung verwendet diese Daten und liefert Ergebnisse zurück, wobei sie die Datenzugriffsschicht verwendet. Diese Kriterien sind allerdings nur eine grobe Vorgabe. Deshalb kann eine automatische Identifikation hier nicht erfolgen.

Für die Trennung von Steuerung und Verarbeitung sind umfassende Restrukturierungen notwendig. Die meisten Funktionen der Anwendungslogik werden dabei in 2 Teile aufgetrennt. Die Modulnamen sowie die Gruppeneinteilung werden für beide Teile beibehalten, wobei jeweils ein Modul in Steuerung und Verarbeitung positioniert wird. Die ursprünglichen Funktionsnamen werden für die Steuerung beibehalten, die Teile der Verarbeitung werden in neu erstellte Funktionen verschoben.

Die Funktionen der Gruppe Datensatzfunktionen changeRecord, deleteRecord, displayRecord und extractRecord werden aufgetrennt. Die Verarbeitung erhält die Module changeRecord, deleteRecord und printRecord. displayRecord weist keine Vearbeitung auf.

Bei der Gruppe Datensatzsuche verbleibt nur der Steuerungsteil der Funktion findrcds in der Steuerung. Die eigentliche Suche von findrcds, sowie die Module getfld und keymatch, die von dieser verwendet werden, werden in die Verarbeitung verschoben.

Das Modul fmt_chk wird der Steuerung zugewiesen. Hier findet keine Verarbeitung statt. Da dies das einzige Modul der EingabeAusgabe in der Anwendungslogik ist, besteht diese Gruppe nur in der Steuerung.

Die Module change, delete und find der Gruppe Karteifunktionen enthalten nur Steuerung, da die Verarbeitung durch die Datensatzsuche und die Datensatzfunktionen durchgeführt wird. Sie werden deshalb ohne Aufteilung der Steuerung zugewiesen. Weiters stellt das Modul cardfileMenu, das die Menüauswahl implementiert, einen reinen Steuerungsmechanismus dar. Die Funktionen der Module add und printdb müssen jedoch aufgeteilt werden.

Die Gruppe Karteiverwaltung beinhaltet momentan nur das Modul define. Dieses muss aufgeteilt werden. Weiters wird der Verarbeitung in dieser Grup-

pe ein Teil der main-Funktion des Modul cardfile verschoben. Dieser Teil dient der Suche nach bestehenden Karteien und liefert der Steuerung deren Definition zurück. Da diese Vorgänge der Karteiverwaltung dienen, wird ein Modul cardfile mit dieser Funktionalität in dieser Gruppe der Verarbeitungsschicht erstellt.

Die Gruppe Wartungsfunktionen ist der Gruppe Karteifunktionen strukturell sehr ähnlich. Das Modul extract verwendet die Datensatzsuche und Datensatzfunktionen und beinhaltet nur Steuerung. Das Modul maintMenu implementiert das Wartungsmenü und wird ebenfalls der Steuerung zugewiesen. Die Module compress, dumpdb und rbuildak werden aufgeteilt, da sie Steuerung und Verarbeitungsvorgänge beinhalten.

Lineare Anordnung

Nach der Durchführung sind die 3 logischen Schichten Steuerung, Verarbeitung und Datenzugriff strukturell getrennt. Es besteht jedoch keine lineare Anordnung. Innerhalb der Steuerung sind Aufrufe des Datenzugriffs vorhanden. Weiters verwendet die Verarbeitung die Präsentationsschicht direkt. Diese Aufrufe erfolgen nicht über die Steuerung. Um eine lineare Anordnung zu erreichen, sind weitere restrukturierende Schritte notwendig.

Delegation zur Steuerung

Um zu verhindern, dass die Verarbeitung direkt die Präsentationsschicht verwendet, werden diese Aufrufe zur Steuerung delegiert. Dafür werden sie aus den Funktionen herausgetrennt und durch Rückgabewerte ersetzt. Die Steuerung wird so verändert, dass diese Rückgabewerte bei Aufrufen der Verarbeitung ausgewertet werden und der zuvor herausgetrennte Präsentationsaufruf erfolgt.

Wrapping des Datenzugriffs

Um eine direkte Verwendung des Datenzugriffs durch die Steuerung zu verhindern, werden diese Aufrufe mittels Wrapping durch dafür erstellte Funktionen der Verarbeitung ersetzt. Im Zuge dessen wird auch die Verwendung der Dateireferenzen verändert. Diese verbleibt in der Verarbeitung und wird nicht mehr direkt von der Steuerung verwendet. Für jede Referenz wird jeweils eine Funktion für die Initialisierung und das Schließen erstellt. Diese werden von der Steuerung verwendet und benötigen keine übergebene Referenz mehr. Um dies zu erreichen, werden externe Variablen für die Referenzen verwendet. Diese Funktionen werden in 2 neu erstellte Module Cardfile und Ak der Verarbeitung positioniert. Diese bilden eine neue Gruppe DBWrapper innerhalb der Verarbeitungsschicht. Funktionen des Moduls Definition

werden nicht von der Steuerung aus aufgerufen. Deshalb werden hier keine Funktionen für Wrapping benötigt.

Nach diesen Restrukturierungen sind die 3 logischen Schichten Steuerung, Verarbeitung und Datenzugriff linear angeordnet. Die Präsentation wird nur mehr von der Steuerung verwendet. Die Datenhaltung steht über den Datenzugriff zur Verfügung. Die Kernfunktionalität verbleibt in der Verarbeitung, wobei diese zwischen Steuerung und Datenzugriff angeordnet ist und mit beiden Schichten in Verbindung steht.

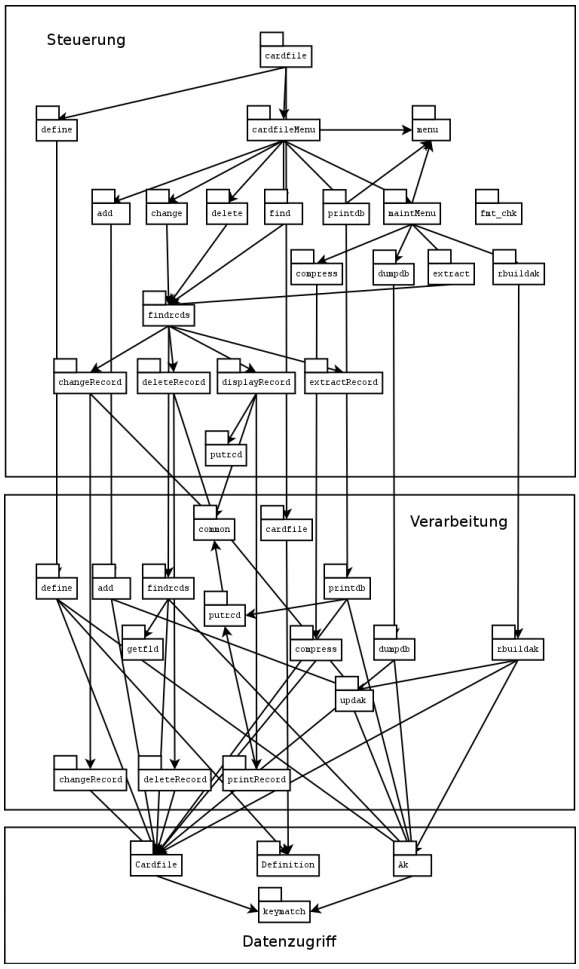


Abbildung 4.11: Aufrufgraph auf Modulebene mit getrennten logischen Schichten

In der Abbildung 4.11 sind die Abhängigkeiten auf Modulebene der nun getrennten logischen Schichten dargestellt. Es ist erkennbar, dass sich durch die Trennung hier eine hohe Aufteilung der strukturellen Programmeinheiten ergibt. Die Verarbeitung ist ähnlich strukturiert wie die Steuerung, wobei jene

Einheiten nicht vertreten sind, die nur Steuerungsfunktionalität enthalten. Die Anzahl der Einheiten hat mit der Trennung stark zugenommen. Dadurch verschlechtert sich die Lesbarkeit. Weiters sind die Gruppen nicht dargestellt. Um einen besseren Überblick über die aktuelle Struktur zu erhalten, werden diese in Abbildung 4.12 farblich innerhalb der logischen Schichten dargestellt.

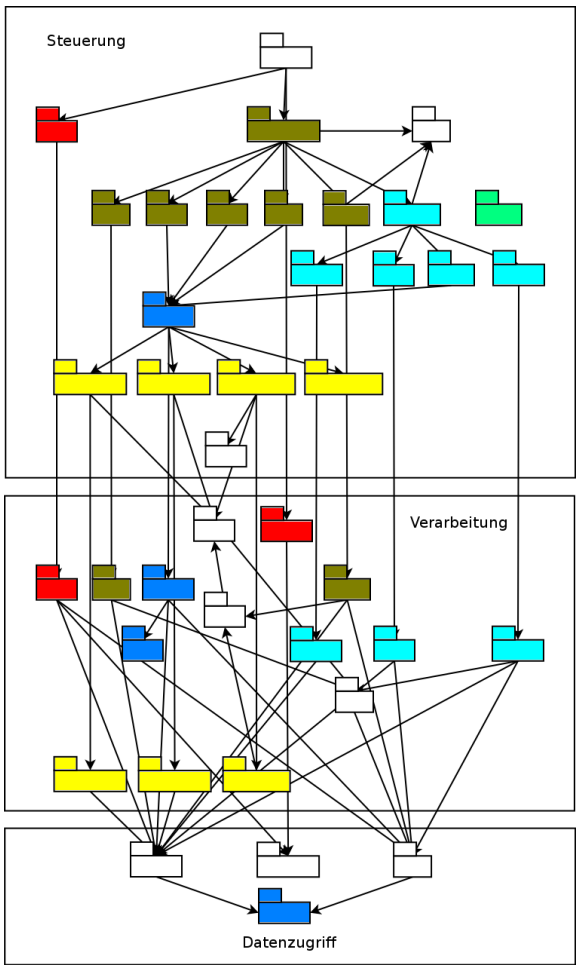


Abbildung 4.12: Logischen Schichten und Gruppen

Damit sind die vorbereitenden Restrukturierungen des Ausgangssystems abgeschlossen. Die 3 physischen Schichten Präsentation, Anwendungslogik und Datenhaltung wurden getrennt. Weiters wurde innerhalb der Anwendungslogik zu den 3 logischen Schichten Steuerung, Verarbeitung und Datenzugriff restrukturiert. Diese wurden danach linear angeordnet. Innerhalb dieser Schichtenstruktur wurde die Gruppeneinteilung beibehalten.

Ergebnisse

Hier werden die Ergebnisse der Trennung der einzelnen physischen und logischen Schichten betrachtet. Die Ordnung entspricht der Reihenfolge der Durchführung.

Physische und logische Schichten

Die wesentliche Erkenntnis dieses Schrittes war, dass die physische Architektur von der logischen Architektur unterschieden werden sollte. Bei der Prozessdefinition wurde angegeben, dass für die Schichtentrennung Einheiten mit Benutzerschnittstellen und solche mit Datenbankzugriffen als Schichten vom Rest des Systems strukturell getrennt werden. Bei der Anwendung der Trennung auf Cardfile ergibt sich dabei aber eine Abtrennung der physischen Schicht Präsentation und eine Abtrennung der logischen Datenzugriffe auf die bereits getrennte physische Datenbank. Während die Präsentation getrennt von der Anwendungslogik umgesetzt wird, verbleibt der Datenzugriff aber im Zielsystem innerhalb der Anwendung selbst und stellt dort einen strukturell getrennten Teil dar. Weiters wurde bei der Behandlung der Präsentationsschicht in der Prozessdefinition die Steuerung besprochen. Diese stellt aber, wie der Datenzugriff, einen Teil der Anwendungslogik dar. Die physische Präsentationsschicht wird im Zielsystem nur durch HTML-Seiten repräsentiert. Nur diese werden vom Client auf dessen Seite direkt verwendet. Eine getrennte Berücksichtigung von physischer und logischer Architektur für den Prozess ist sinnvoll, da dies bei der Restrukturierung des Ausgangsystems nicht unbedingt ersichtlich ist, aber im Zielsystem so umgesetzt werden soll.

Abtrennung der Benutzerschnittstelle

Da die Eigenschaften für die Identifikation von Einheiten mit Benutzerschnittstellenfunktionalität klar festgelegt waren, konnten diese Einheiten problemlos gefunden werden. Es ist ersichtlich, dass hohe Abhängigkeiten von diesen Einheiten zu Einheiten der Anwendungslogik bestehen. Der Grund dafür ist die häufige Verwendung der Benutzerschnittstellenfunktionen beim Ablauf des Programms. Diese werden für die Interaktionen mit dem Benutzer verwendet, wobei sich der Ablauf der Anwendungslogik selbst linear darstellt und nur durch diese Verwendungen unterbrochen wird.

Datenzugriff – Wrapping der Datenhaltung

Um die Implementierung der physischen Zugriffe auf die Datenhaltung transparent zu gestalten, wurden diese Zugriffe in Funktionen eingeschlossen und zu den 3 Modulen Cardfile, Definition und Ak zusammengefasst. Wie bei der

Benutzerschnittstelle war eine Identifikation einfach, da die Kriterien dafür zuvor festgelegt wurden. Im Gegensatz zur Präsentation mussten hier aber intraprozedurale Restrukturierungen durchgeführt werden, da die Datenbankzugriffe mit der Anwendungslogik vermischt waren und nicht durch eigene strukturelle Einheiten repräsentiert wurden. Dadurch erschwerte sich auch die Restrukturierung, da im Gegensatz zur Identifikation eine klare Trennung der Programmteile innerhalb der Prozeduren schwierig war. Die Beurteilung, welche Quellcodeteile zum Datenzugriff gehören, musste mittels Programmverständnis durch den Entwickler erfolgen. Das Ergebnis kapselt die Zugriffe selbst, während die zugehörigen Dateireferenzen weiterhin in der Anwendungslogik verbleiben. Eine technologieunabhängige Verwendung der Datenhaltung ist im Ausgangssystem auch durch Wrapping kaum zu erreichen.

Trennung von Steuerung und Verarbeitung

Der schwierigste Teil der Schichtentrennung war die Trennung der Steuerung von der Verarbeitung. Die Gründe dafür waren das Fehlen von Kriterien für die Identifikation und die Notwendigkeit der intraprozeduralen Restrukturierung, da diese 2 Schichten, wie der Datenzugriff, stark miteinander vermischt waren. Das Zusammentreffen dieser 2 Eigenschaften erschwerte diesen Teil ernorm. Einzig das zuvor entwickelte Programmverständnis konnte als Maß für Identifikation und Trennung verwendet werden.

Lineare Anordnung

Die lineare Anordnung der Schichten war nach der strukturellen Trennung einfach durchzuführen. Um sie zu ermöglichen, mussten allerdings auch ungünstige Strukturen verwendet werden. Da die Dateireferenzen nun nicht mehr als Eingangsparameter der Steuerung bestehen, mussten sie als externe Variablen, die global verwendet werden, umgesetzt werden. Dies beschränkt sich aber auf die Verarbeitungsschicht. Innerhalb von Steuerung und Datenzugriff findet keine Verwendung dieser Variablen statt.

Das Durchführen von intraprozeduralen Restrukturierungen und das Auftrennen von Funktionen ist ein sehr fehleranfälliger Prozess. Vor allem bei der Trennung von Steuerung und Verarbeitung, bei der keine klaren Kriterien vorhanden waren, führte dies zu Fehlern. Ein ausführliches Testen zur Verifikation der Funktionalität war notwendig.

4.6 Schritt 3 - Forward Engineering

Nach dem Abschluss der Restrukturierungen werden dessen Ergebnisse nun verwendet, um die Transformation von Cardfile zur Java EE Web-Anwendung durchzuführen. Dafür werden wie beim traditionellen Prozess der Software-Entwicklung Anforderungsanalyse, Design und Implementierung durchlaufen, wobei die Ergebnisse der vorherigen Schritte dafür als Grundlage verwendet werden.

Zuerst werden die Anforderungen von Cardfile herangezogen, die am Beginn der Fallstudie rekonstruiert wurden. Diese werden, wie im Prozess beschrieben, als Grundlage für die Anforderungen an Web-Cardfile verwendet. Die einzelnen Anwendungsfälle werden auf ihre Eignung bezüglich des Zielsystems untersucht und gegebenenfalls modifiziert. Neue Funktionalität kann hier ebenfalls eingebracht werden.

Nach der Festlegung der Anforderungen wird das Design erstellt. Dafür werden die bisherigen Ergebnisse verwendet. Für die Anwendungslogik wird ein objektorientiertes Design erstellt. Dabei können die logischen Schichten getrennt behandelt werden. Verwendende Schichten müssen nur die Dienste kennen. Einzelne Zieleigenschaften können innerhalb der Schichten umgesetzt werden. Die Java EE stellt für diese logischen Schichten unterschiedliche Technologien zur Verfügung. Für die Benutzerschnittstelle, die die Präsentation repräsentiert, wird eine äquivalente HTML-basierte Variante geplant. Für die Datenhaltung wird ein relationales Modell entworfen, das den selben Datenbestand wie Cardfile halten kann, aber die Vorteile des relationalen Datenbanksystems nutzt.

Das Forward Engineering wird mit der Transformation abgeschlossen. Dabei wird das Design in eine Implementierung von Web-Cardfile umgesetzt. Diese ist funktional äquivalent zu Cardfile. Der Datenbestand wird übernommen und damit das Testen zur Verfifikation der Äquivalenz durchgeführt.

4.6.1 Anforderungen

Funktionale Anforderungen

Hier werden die funktionalen Anforderungen für das Zielsystem festgelegt. Dafür werden die einzelnen Anwendungsfälle untersucht, die für das Ausgangssystem rekonstruiert wurden. Diese werden hinsichtlich ihrer Eignung für eine Umsetzung im Zielsystem betrachtet und werden dann für jeden Anwendungsfall einzeln entweder direkt übernommen, für das Zielsystem modifiziert oder entfallen.

Migrierbare Anforderungen

Die folgenden Anwendungsfälle werden direkt für das Zielsystem übernommen. Eine genaue Beschreibung ist in Kapitel 4.4.2 zu finden.

- Einträge suchen
- Eintrag hinzufügen
- Eintrag ändern
- Eintrag löschen
- Kartei komprimieren

Nicht zu migrierende Anforderungen

Die folgenden Anwendungsfälle werden im Zielsystem nicht umgesetzt.

- Suchindex neu erzeugen

  Dieser Anwendungsfall ist für das Zielsystem nicht mehr angebracht. Er ergibt sich aus der verwendeten Datenbanktechnologie des Ausgangssystems. Da diese für das Zielsystem nicht mehr verwendet wird, wird der Anwendungsfall entfernt.

Geänderte Anforderungen

Hier werden Anforderungen beschrieben, die zwar auf ursprünglichen Anwendungsfällen basieren, aber für das Zielsystem abgeändert werden. Die Änderungen fallen dabei in die Bereiche „Nicht zu migrierende Anforderungen“ und „Neue Anforderungen“. Der Einfachheit halber werden hier die originalen Anwendungsfälle entsprechend modifiziert beschrieben. Änderungen sind dabei kursiv angegeben.

Name: Kartei öffnen
Beschreibung: Der Benutzer öffnet eine bestehende Kartei.
Akteure: Benutzer
Vorbedingungen: Keine
Ablauf: Der Benutzer öffnet eine bestehende Kartei. Er gibt dafür den Namen der Kartei an. Weiters gibt er an, ob die Kartei nur lesbar geöffnet werden soll. *Während des Vorgangs werden zu seiner Information alle momentan bestehenden Karteien der Anwendung aufgelistet.*
Alternativer Ablauf: Die angegebene Kartei existiert nicht. Der Anwendungsfall „Neue Kartei erstellen“ wird ausgelöst.
Nachbedingungen: Die angegebene Kartei wurde geöffnet.

Dieser Anwendungsfall wird um die Anzeige aller momentan bestehenden Karteien erweitert. Diese Funktionalität ist nicht im Ausgangssystem vorhanden, da dort die Dateien, die Karteien darstellen, über das Betriebssystem aufgelistet werden können. Da dies für das Zielsystem nicht möglich ist, wird diese Funktionalität Teil der Anwendung.

Name: Neue Kartei erstellen
Beschreibung: Der Benutzer erstellt eine neue Kartei. Er definiert die einzelnen Felder und legt ihre Eigenschaften fest.
Akteure: Benutzer
Vorbedingungen: Eine Kartei mit dem angegebenen Namen existiert nicht.
Ablauf: Der Benutzer definiert nacheinander alle Felder für die Einträge der Kartei. Für jedes Feld können dabei folgende Eigenschaften festgelegt werden.

- Name des Feldes
- Maximale Länge eines Wertes in diesem Feld
- Eine Angabe, ob ein Wert eingegeben werden muss
- Eine Angabe, ob nach diesem Feld gesucht werden kann
- Trennzeichen für die Trennung mehrerer Wertangaben
- Ein Format als regulärer Ausdruck, dem eine Wertangabe entsprechen muss

Alternativer Ablauf: Keiner
Nachbedingungen: Die definierte Kartei wurde erstellt.

Bei diesem Anwendungsfall werden alle Angaben zur Positionierung entfernt, da diese nicht mehr angebracht sind. Die zeichenorientierte Benutzerschnittstelle des Ausgangssystems wird im Zielsystem durch äquivalente HTML-Seiten ersetzt. Deshalb ist eine Positionierung wie im Ausgangssystem nicht mehr möglich und auch nicht sinnvoll.

Name: Eintrag drucken
Beschreibung: *Der Benutzer erzeugt eine Druckansicht eines bestehenden Eintrags der Kartei.*
Akteure: Benutzer
Vorbedingungen: Ein gefundener und angezeigter Eintrag.
Ablauf: *Der Benutzer wählt den momentan angezeigten Eintrag für die Druckansicht aus. Der Eintrag wird auf einer neuen Seite in einer Druckansicht angezeigt.*
Alternativer Ablauf: Keiner
Nachbedingungen: Keine

Bei diesem Anwendungsfall entfällt das direkte Drucken eines Eintrags, da dies im Zielsystem nicht umsetzbar ist. Die Web-Anwendung hat keinen direkten Zugriff auf den Drucker des Clients. Statt dem Druck wird eine Seite geöffnet, die den Eintrag in einer Druckansicht darstellt. Diese kann dann mittels Web-Browser gedruckt werden. Weiters wird der Akteur Drucker entfernt.

Name: Kartei drucken
Beschreibung: Der Benutzer druckt eine Kartei.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: *Der Benutzer erzeugt einen Ausdruck der geöffneten Kartei. Diese beinhaltet alle bestehenden Einträge der Kartei. Die Einträge werden auf einer neuen Seite in einer Druckansicht angezeigt.*
Alternativer Ablauf: *Keiner*
Nachbedingungen: Keine

Bei diesem Anwendungsfall entfällt das direkte Speichern der extrahierten Einträge in eine Datei, da dies im Zielsystem nicht sinnvoll ist. Die Web-Anwendung würde die Datei mit den extrahierten Einträgen am Server speichern. Statt dem direkten Speichern wird eine Seite geöffnet, die die Einträge darstellt. Diese kann dann mittels Web-Browser weiterverwendet werden. Weiters entfällt das Weiterleiten an ein Kommando, da dies im Zielsystem ebenfalls nicht sinnvoll ist. Die Web-Anwendung hat keinen direkten Zugriff auf Kommandos des Client-Betriebssystems. Dieser alternative Ablauf wird deshalb aus dem Anwendungsfall entfernt. Die Angabe eines Formats für den Ausdruck wird ebenfalls entfernt, da HTML sich nicht für Formatierungen eignet. Weiters wird das Drucken einer extrahierten Datei entfernt, da dies nur im Kontext der Formatierungsmöglichkeit sinnvoll ist, die im Zielsystem aber nun nicht mehr vorhanden ist. Somit verbleibt nur das Erzeugen einer

Druckansicht für die bestehenden Einträge der geöffneten Kartei.

Name: Karteiabbild erzeugen
Beschreibung: Der Benutzer erzeugt ein Abbild der geöffneten Kartei, das alle Informationen zu den bestehenden Einträgen enthält.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer erzeugt ein Abbild der geöffneten Kartei. Dieses beinhaltet alle bestehenden Einträge der Kartei, weiters alle Indexeinträge mit Suchbegriffen zu den Einträgen. Diese sind nach den Suchfeldern aufsteigend geordnet. *Das Abbild wird auf einer neuen Seite in einer Druckansicht angezeigt.*
Alternativer Ablauf: Keiner
Nachbedingungen: Keine

Bei diesem Anwendungsfall entfällt das Drucken am Standarddrucker, da dies im Zielsystem nicht umsetzbar ist. Die Web-Anwendung hat keinen direkten Zugriff auf den Drucker des Clients. Statt dem direkten Druck wird eine Seite geöffnet, die den Eintrag druckbar darstellt. Diese kann dann mittels Web-Browser gedruckt werden. Weiters wird der Akteur Drucker entfernt.

Name: Einträge extrahieren
Beschreibung: Der Benutzer extrahiert ausgewählte Einträge.
Akteure: Benutzer
Vorbedingungen: Eine geöffnete Kartei.
Ablauf: Der Benutzer verwendet den Anwendungsfall „Einträge suchen“, um zu extrahierende Einträge auszuwählen. *Nach Abschluss dieser Auswahl werden die gefundenen Einträge auf einer neuen Seite angezeigt.*
Alternativer Ablauf: Keiner
Nachbedingungen: Keine

Bei diesem Anwendungsfall entfällt das direkte Speichern der extrahierten Einträge in eine Datei, da dies im Zielsystem nicht sinnvoll ist. Die Web-Anwendung würde die Datei mit den extrahierten Einträgen am Server speichern. Statt dem direkten Speichern wird eine Seite geöffnet, die die extrahierten Einträge darstellt. Diese kann dann mittels Web-Browser gespeichert werden.

Nicht-funktionale Anforderungen

Nebenläufige Verwendung

Web-Cardfile soll durch mehrere Benutzer gleichzeitig verwendbar sein.

Die ursprüngliche Anforderung der Portabilität unter Unix-basierten Systemen ist hier nicht mehr zutreffend. Web-Cardfile ist plattformunabhängig über Web-Browser verwendbar. Die Anwendung selbst wird innerhalb der serverseitigen Web-Umgebung ausgeführt.

4.6.2 Design

Hier wird das Design für Web-Cardfile erstellt. Dies wird basierend auf den Ergebnissen der vorherigen Schritte gemacht und setzt die festgelegten Anforderungen an Web-Cardfile um. Da das ursprüngliche Cardfile sich in allen für die Web-Anwendung gewünschten Eigenschaften unterscheidet, muss sowohl ein objektorientiertes Design als auch die Umsetzung aller Eigenschaften einer Web-Anwendung geplant werden.

Dabei wird jede physische und logische Schicht für sich betrachtet. Das Design muss zwar für die gesamte Web-Anwendung erstellt werden, aber einzelne Eigenschaften können aufgrund der vorbereitenden Restrukturierung lokal in den zuständigen Schichten geplant werden. Der Zugriff auf die Datenbank erfolgt nur über die Datenzugriffsschicht. Diese wird nur von der Verarbeitung verwendet. Die Verarbeitung wird nur von der Steuerung verwendet. In der Steuerung wird auch die ereignisbasierte Verarbeitung der Web-Anwendung umgesetzt. Die Eingaben und Ausgaben für Benutzerschnittstelle werden ebenfalls nur dort geplant. Generell muss beim Design einer Schicht nur die verwendete Schicht berücksichtigt werden. Dies gilt sowohl für die physischen als auch für die logischen Schichten. Durch die lineare Anordnung beschränkt sich die Verwendung in jeder Schicht auf eine einzige angrenzende Schicht.

Im Folgenden wird das Design für die Schichten entsprechend ihrer linearen Anordnung beschrieben. Innerhalb der logischen Schichten wird ein objektorientiertes Design erstellt. Dafür werden die Gruppen als Basis verwendet, wobei diese nur lokal in der zugehörigen Schicht betrachtet werden. Für jede der Schichten wird für die Umsetzung mittels Java EE eine geeignete Technologie ausgewählt.

Präsentation - JavaServer Faces

Die zeichenorientierte Benutzerschnittstelle von Cardfile wird hier äquivalent in eine HTML-basierte Benutzerschnittstelle umgesetzt. Jeder Eingabebildschirm wird als eine JSF-Seite implementiert. Eingabemöglichkeiten in Cardfile werden mit Formulareingabefeldern realisiert. Dabei wird eine Schaltfläche für das Senden benötigt, die hinzugefügt wird. Optionen reiner Auswahl ohne Benutzereingabe, die zur Navigation verwendet werden, werden mittels Hyperlinks umgesetzt.

Die Anzeige von Nachrichten, die in Cardfile mittels eigener Funktionen am Bildschirm dargestellt werden, wobei die zuvor gemachte Ausgabe nicht gelöscht wird und die Nachricht sozusagen hinzugefügt wird, wird innerhalb der Seite berücksichtigt und die Anzeige dynamisch von der Web-Anwendung gesteuert. Es wird nur die Anzeigemöglichkeit für Nachrichten innerhalb der Seite implementiert. Der Inhalt selbst wird von der Anwendung über die Steuerung zur Verfügung gestellt.

Die HTML-Seiten werden Black-Box nach der Vorgabe von Cardfile umgesetzt. Die Implementierung wird nicht mittels HTML direkt gemacht, sondern es werden JSF-Komponenten dafür verwendet. Der Anwendungsserver erzeugt bei einer Anfrage daraus dynamisch entsprechenden HTML-Quellcode, welcher danach zum Client gesendet wird.

Steuerung - Backing Beans

Die Steuerung stellt die Verbindung der Präsentation mit der Verarbeitung her. Sie ermöglicht dem Client die Verwendung der EJBs. Sie erhält von ihm die Eingabedaten, löst die Verarbeitung der Daten mittels Methoden der EJBs aus und vermittelt das Ergebnis zurück an den Client. Dabei kontrolliert sie auch den Ablauf der Anwendung.

Die Steuerung von Web-Cardfile wird mittels Backing Beans umgesetzt, die die Dateneingabe über die HTML-Seiten entgegen nehmen, die Verarbeitung auslösen und dynamisch die Erzeugung von HTML-Seiten als Antwort steuern. Weiters werden sie für die Validierung von Eingaben verwendet, da dies mit JSF direkt umsetzbar ist. Für die Verarbeitung verwenden sie Methoden der EJBs.

Um die Funktionalität von Cardfile damit umsetzen zu können, müssen Funktionen, die die Benutzerschnittstelle verwenden, aufgespalten werden. Die Umsetzung mittels JSF und Backing Beans wird so realisiert, dass der erste Teil der Funktion zunächst im Konstruktor der zuständigen Backing Bean ausgeführt wird. Dieser steuert dynamisch die Erzeugung der entsprechenden HTML-Seite, die dann zum Client gesendet wird. Der Client kann damit

Eingaben in ein Formular machen und dieses senden oder einem Hyperlink folgen. Erhält die Web-Anwendung das Ergebnis dieser Benutzeraktion, wird eine weitere Methode der Backing Bean ausgeführt, die den zweiten Teil der Funktion implementiert. Damit wird die Umsetzung der aufgespaltenen Funktion und die Integration der JSF-Seite in die Anwendung erreicht.

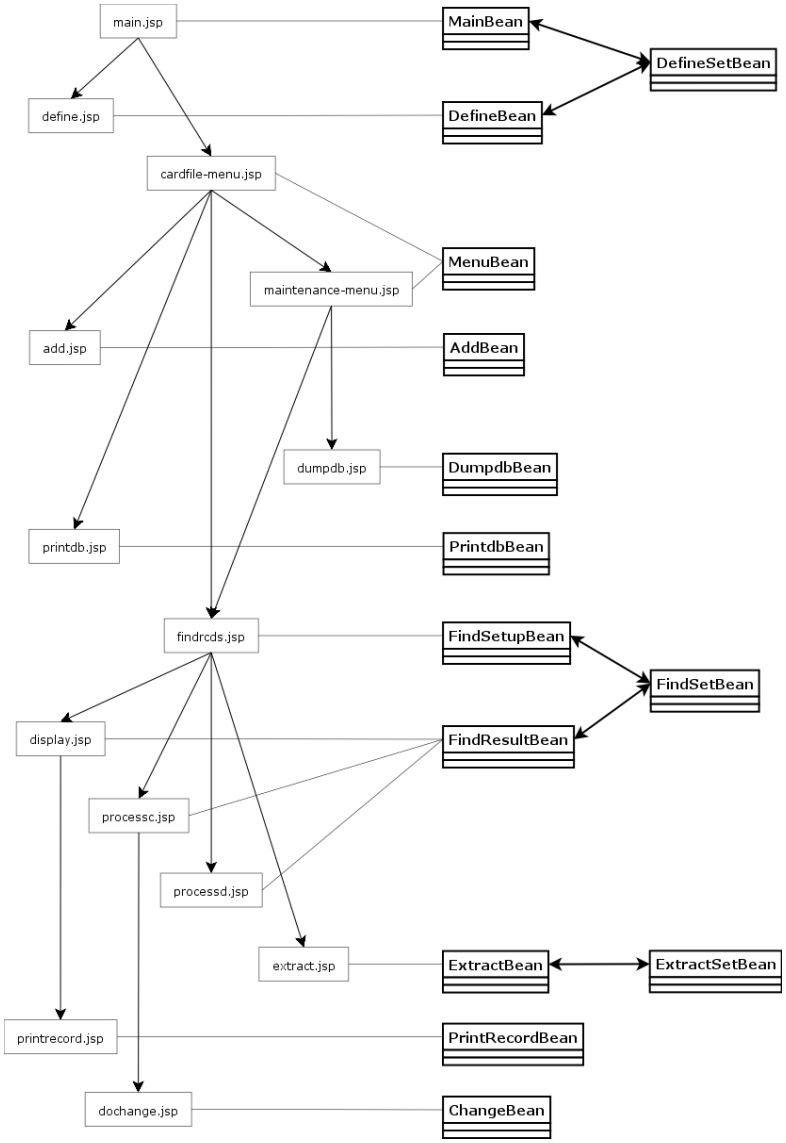


Abbildung 4.13: JSF-Seiten und Backing Beans

Für das Design der Backing Beans werden die Gruppen als Basis verwendet. Das Modul cardfile, dass die main-Funktion enthält, wird als MainBean umgesetzt und stellt weiterhin den Eintrittspunkt in den Ablauf dar.

Die Karteiverwaltung enthält das Modul define. Dieses wird als DefineBean umgesetzt.

Die Menüs der Gruppen Karteifunktionen und Wartungsfunktionen werden direkt als JSF-Seiten umgesetzt. Obwohl sie bei der Schichtentrennung der Steuerung zugewiesen wurden und hier auch so umgesetzt werden können, ist es einfacher, sie direkt mittels JSF zu realisieren. Da auch auf den Menüseiten Nachrichten angezeigt werden, wird eine MenuBean erstellt, die dafür zuständig ist.

Bei der Gruppe Karteifunktionen werden 2 Arten von Funktionalität unterschieden. Die Module add und printdb beinhalten Funktionen, die direkt das Hinzufügen und Drucken steuern. Die Funktionen der Module change, delete und find bereiten jedoch nur die Suche nach Datensätzen vor und übergeben die Datensatzfunktionen. Die Umsetzung der Gruppe Karteifunktionen erfolgt durch die Backing Beans AddBean, PrintdbBean und FindSetupBean, wobei letztere Bean die Suche und deren Steuerung, die den Modulen change, delete und find entspricht, umsetzt. Um das Ergebnis des Druckens anzuzeigen, das im Gegensatz zu Cardfile nun nicht mehr zum Drucker gesendet wird, wird eine PrintdbBean erstellt. Weiters wird für das Drucken einzelner Einträge über die Suche eine PrintRecordBean erstellt.

Wie bei den Karteifunktionen wird auch bei der Gruppe Wartungsfunktionen nach dem Bezug zur Suche unterschieden. Das Modul extract bereitet die Suche vor, während die Module compress, dumpdb und rbuildak direkt die entsprechende Funktionalität umsetzen. Die Funktionalität von extract wird deshalb ebenfalls in der FindSetupBean implementiert. Um das Ergebnis des Extrahierens anzuzeigen, das im Gegensatz zu Cardfile nun nicht mehr in eine Datei geschrieben wird, wird eine ExtractBean erstellt. dumpdb wird in einer DumpdbBean umgesetzt. Da für compress keine Benutzerschnittstelle benötigt wird und nur Nachrichten angezeigt werden, wird dessen Funktionalität vom Wartungsmenü direkt über die MenuBean aufgerufen. Die Ergebnisnachrichten werden dann, entsprechend der Anzeige in Cardfile, dort angezeigt. Das Modul rbuildak wird nicht weiter verwendet. Der Anwendungsfall wurde für Web-Cardfile entfernt und dessen Funktionalität wird hier nicht umgesetzt.

Die Gruppe Datensatzsuche, die nur das Modul findrcds enthält, wird in der FindResultBean umgesetzt. Dort wird die eigentliche Suche im Konstruktor ausgeführt und das Ergebnis in der zugeordneten JSF-Seite dargestellt.

Ebenfalls in der FindResultBean werden die Datensatzfunktionen behandelt. deleteRecord wird dort direkt umgesetzt, da es keine Benutzerschnittstelle benötigt. Der Eintrag kann direkt bei der Anzeige des Suchergebnis gelöscht werden. Um die unterschiedlichen Datensatzfunktionen auf dem Suchergebnis zu ermöglichen, wird für jede Datensatzfunktion eine eigene JSF-Seite

erstellt, über die die entsprechende Funktionalität ausgelöst werden kann. Wie zuvor beim Design der Menüs ist es hier einfacher, sie direkt mittels JSF zu realisieren.

Hinzufügen und Ändern von Einträgen verwenden die Funktion fmt_chk des gleichnamigen Moduls der Gruppe EingabeAusgabe. Diese wird in einer Klasse Regex als deren Methode umgesetzt und einem util-Paket zugeordnet. Regex wird von AddBean und ChangeBean für die Überprüfung der Eingaben verwendet.

Problematisch beim Erstellen des Designs für die Backing Beans sind Schleifen mit Benutzerschnittstellen in Cardfile. Das Durchlaufen dieser steht im Gegensatz zur ereignisbasierten Verarbeitung. Die Unabhängigkeit der Ereignisse widerspricht Durchläufen durch eine Schleife. Um eine Umsetzung zu ermöglichen, müssen die Schleifenvariablen ebenfalls in der Sitzung gespeichert werden. Für jedes Ereignis, das die Schleife verwendet, werden sie dann als Ausgangszustand für die aktuelle Iteration herangezogen. Dieser Schleifenzustand wird mittels JavaBeans realisiert. Deren Lebensdauer wird auf eine Sitzung festgelegt. Damit stehen diese über alle Ereignisse eines Benutzers zur Verfügung und können für die Verbindung der einzelnen Ereignisse entsprechend der Schleifeniterationen verwendet werden. Anwendungsfälle, die Schleifen mit Benutzerschnittstellen verwenden, sind „Neue Kartei erstellen“, „Einträge suchen“ und „Einträge extrahieren“. Für diese werden die JavaBeans DefineSetBean, FindSetBean und ExtractSetBean erstellt.

Die Sitzungen für Benutzer werden vom Anwendungsserver selbst implementiert und verwaltet. Sie müssen nicht vom Entwickler umgesetzt werden. Dieser hat aber die Möglichkeit, die Lebensdauer von Objekten anzugeben. Er kann beispielsweise angeben, dass ein Objekt während des gesamten Sitzungsverlaufs existieren soll, oder die Lebensdauer auf eine einzelne Anfrage beschränken. Um die Verarbeitung verwenden zu können, muss für jede Sitzung eine Referenz auf die Enterprise Beans des Benutzers in der Steuerung gehalten werden. Dafür wird eine JavaBean CardfileBean verwendet, die eine Enterprise Bean CardfileController als Attribut zugewiesen bekommt.

Verarbeitung – Enterprise Beans

Die Verarbeitung, die die Kernfunktionalität der Anwendung darstellt, wird mittels Enterprise Beans umgesetzt. Das objektorientierte Design dafür ergibt 2 Beans. Der Zugriff der Steuerung auf die Enterprise Beans erfolgt über die 2 Schnittstellen CardfileManager und CardfileController.

Die CardfileManagerBean stellt der Anwendung die Funktionalität zum Verwalten der Karteien zur Verfügung. Sie ist als zustandslose Session Bean implementiert und beinhaltet keine Attribute. Ihre Methoden beinhalten das

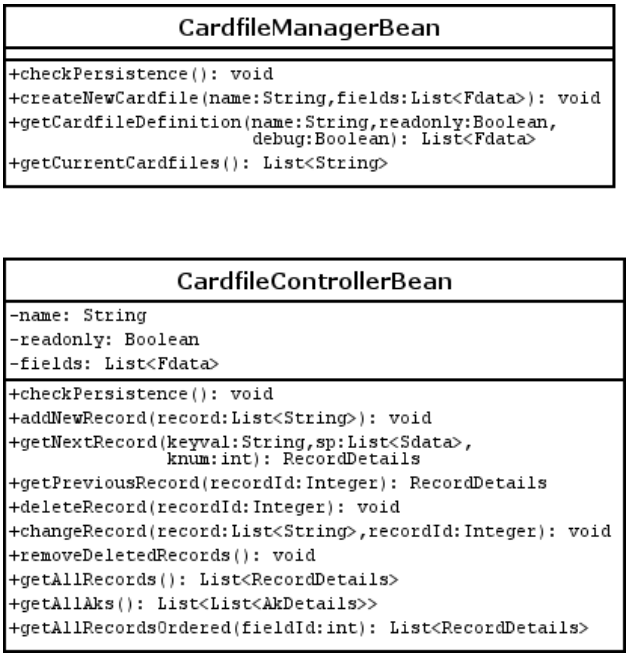


Abbildung 4.14: Enterprise Beans

Finden und Erzeugen von Karteien. Unter Verwendung des Namens kann eine Kartei damit gesucht werden. Ist sie in der Datenbank vorhanden, wird die Definition der Kartei zurückgeliefert, die diese Kartei repräsentiert. Weiters können neue Karteien erzeugt werden, die sich noch nicht in der Datenbank befinden.

Die CardfileControllerBean repräsentiert eine bestehende Kartei. Sie ist als zustandsbehaftete Session Bean implementiert und wird durch den Namen der Kartei und die Eigenschaft readonly als Attribute dargestellt. Methoden dieser Bean operieren auf einer Kartei und manipulieren deren Einträge. Funktionen wie Hinzufügen, Suchen, Ändern und Löschen von Einträgen sind hier zu finden. Weiters ist hier Funktionalität für die Wartungsfunktionen compress und dumpdb umgesetzt, die ebenfalls auf einer Kartei operieren.

Da die 2 Methoden getNextRecord und getPreviousRecord für die Suche den nächsten und den vorherigen Eintrag liefern, muss ein Zustand für die aktuelle Suche gehalten werden. Dies wird mittels der Attribute currentAkList und offset gemacht, die für diesen Zweck der CardfileControllerBean hinzugefügt werden. Die Notwendigkeit dafür besteht durch die Übernahme des Suchalgorithmus und der nacheinander folgenden Anzeige der Einträge.

Gemeinsam von Steuerung und Verarbeitung verwendete Klassen werden der Verarbeitung zugewiesen, damit die lineare Abhängigkeit der Schichten eingehalten wird. Die Steuerung verwendet die Verarbeitung. Die Verarbei-

tung soll nicht von der Steuerung abhängig sein. Mit der Zuordnung von gemeinsam verwendeten Klassen zur Verarbeitung wird dies erreicht.

Durch diese 2 Enterprise Beans wurde der Kern der Anwendung umgesetzt. Als mittlere Schicht verwenden sie die Entities des Datenzugriffs, um ihre Aufgaben mittels der Datenhaltung zu erfüllen. Sie werden von den Backing Beans der Steuerungsschicht verwendet, für die sie diese Aufgaben ausführen. Dabei erhalten sie von ihr Daten für die Verarbeitung und liefern Informationen zurück, die für die Anzeige in der Präsentation bestimmt sind.

Datenzugriff – Entities

Die Datenzugriffsschicht wird mittels Entities umgesetzt. Diese stellen die Datenhaltung von Web-Cardfile der Verarbeitung als Klassen und technologieunabhängig zur Verfügung. Die Entities ermöglichen eine objektorientierte Verwendung des Datenbestands. In dieser Schicht finden sich die meisten Änderungen zu Cardfile, da ein unterschiedliches Datenmodell zugrunde liegt. Eine Wiederverwendung der Funktionen der Module Cardfile, Definition und Ak ist hier aufgrund des geänderten Datenmodells für die Datenhaltung von Web-Cardfile nicht möglich.

Datenhaltung – PostgreSQL

Für die Datenhaltung von Web-Cardfile wird ein neues Datenmodell erstellt. Da nun ein relationales Datenbanksystem verwendet wird, sollen dessen Vorteile gegenüber der Datenhaltung von Cardfile genutzt werden. Basierend auf dieser wird ein äquivalentes Modell erstellt, das die Zusammenhänge der Textdateien relational darstellt und denselben Datenbestand wie Cardfile speichern kann. Die Möglichkeiten zur Datenspeicherung ändern sich also nicht. Nur die Strukturierung wird geändert.

Migration

Im Zuge der Transformation zu Web-Cardfile werden keine Teilsysteme auf Implementierungsebene wiederverwendet. Die gesamte Anwendung soll in die Zielsprache Java transformiert werden. Die Wiederverwendung findet also nur auf struktureller und architektureller Ebene statt. Untersucht werden dabei die Umsetzungsmöglichkeiten der Teilsysteme, die durch die Gruppen repräsentiert werden. Eine Anbindung von Teilsystemen auf Implementierungsebene ist nicht Untersuchungsgegenstand dieser Arbeit und wird nicht durchgeführt.

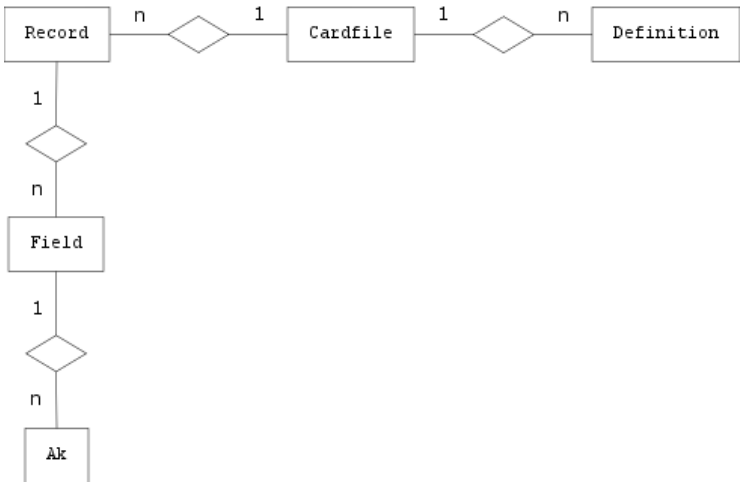


Abbildung 4.15: Datenmodell als ER-Diagramm

Weiters wird die Migration als inkrementelle Umsetzung der zuvor festgelegten Teilsysteme als Objekte in der Zielumgebung durchgeführt. Cardfile und Web-Cardfile arbeiten zu keinem Zeitpunkt zusammen. Dies entspricht der Big Bang Methode. Nach Abschluss der Transformation wird der alte Datenbestand in die Datenbank von Web-Cardfile übernommen. Mit dieser Datenbasis wird dann das abschließende Testen durchgeführt, um die funktionale Äquivalenz zu gewährleisten.

Die Arbeit beschränkt sich darauf, im Zuge des Web-Reengineering Prozesses die Möglichkeiten für die Anwendung der Migrationsmethoden zu schaffen und diese zu unterstützen, da dies für die Ablösung von Legacy Systemen hohe Wichtigkeit hat. Eine Durchführung der Migrationsmethoden selbst wird nicht behandelt.

Ergebnisse

Die Ergebnisse werden einzeln für jede physische und logische Schicht angegeben. Die Reihenfolge entspricht der linearen Anordnung.

Präsentation

Das Design für die Präsentation konnte relativ problemlos durchgeführt werden. Eingabefelder und Aktionen, die diese verwenden, werden mit Formularen umgesetzt. Für die Navigation innerhalb der Anwendung werden Hyperlinks verwendet. Nicht direkt umgesetzt werden Tastenkombinationen für

Aktionen. Diese werden durch Hyperlinks ersetzt.

Steuerung

Bei der Steuerung war zunächst die Zieleigenschaft ereignisbasierte Verarbeitung umzusetzen. Dafür wurden die Funktionen an den Stellen der Interaktionen mit dem Benutzer aufgetrennt und daraus Methoden der Backing Beans erstellt. Diese dienen der Dateneingabe und Datenausgabe über die JSF-Seiten. Das Design dafür hatte nicht viel Spielraum, da nur eine solche Anordnung der ursprünglichen Funktionalität in Zusammenhang mit JSF ein verwendbares Ergebnis brachte. Diese Aufteilung steht aber im Gegensatz zur Gruppeneinteilung des restrukturierten Cardfiles, da dafür Gruppen zusammengefasst und auch aufgeteilt werden mussten. Die Gruppeneinteilung war hier für die Objektfindung nicht sehr hilfreich. Das liegt daran, dass die Gruppen anhand der Aufrufbeziehungen gebildet wurden. Im Gegensatz dazu mussten die Backing Beans anhand der Daten, die mit der jeweiligen JSF-Seite in Verbindung stehen, erstellt werden, da sie deren Attribute darstellen. Dies war notwendig, um die ereignisbasierte Verarbeitung umzusetzen und steht im Gegensatz zur Gruppierung anhand der Aufrufe.

Bei manchen Teilen der Steuerung konnten die Vorteile der Zieltechnologie verwendet werden, um diese einfacher umzusetzen. Die Menüs können beispielsweise direkt mittels JSF erstellt werden. Weiters bieten JSF einen Mechanismus zur Validierung von Eingaben in Formularfelder an, der ebenfalls zur Vereinfachung beiträgt.

Schlecht umsetzbar in Web-Anwendungen sind Schleifen im ursprünglichen System, innerhalb derer Benutzereingaben stattfinden, da zusätzlicher Aufwand für eine Realisierung notwendig ist. Dabei müssen alle an der Schleife beteiligten Variablen für die Sitzung des Benutzers gespeichert werden, um die einzelnen Ereignisse als Iterationen zu einer Schleife verbinden zu können. Diese müssen zuerst identifiziert und danach entsprechend in die Web-Anwendung integriert werden. Deshalb wurde für Web-Cardfile für jede einzelne Schleife eine eigene JavaBean für die Zustandsspeicherung verwendet, um so die Zuständigkeiten zu trennen.

Von Vorteil war die zuvor erfolgte Trennung der logischen Schichten, da aufgrund dieser hier nur der für die Steuerung der Präsentation notwendige Teil umgesetzt werden konnte und so eine bessere strukturelle Aufteilung entstand. Die Backing Beans sind hier nur für die Steuerung gedacht. Die Verarbeitung soll mittels Enterprise Beans erfolgen. Die Restrukturierung brachte hier also Vorteile.

Verarbeitung

Für die Verarbeitung wurde ebenfalls ein objektorientiertes Design auf Ba-

sis der Gruppen erstellt. Wie zuvor bei der Steuerung brachte eine direkte Umsetzung der Gruppen als Klassen weniger gute Ergebnisse. In diesem Fall mussten Gruppen zwar nur zusammengefasst, und nicht getrennt, werden. Aber die Objekte wurden schließlich basierend auf den verwendeten Variablen von Cardfile erstellt, da dies eine bessere Aufteilung ergab. Die Gruppen waren nur deshalb hilfreich, da auch sie im Kontrollfluss gemeinsame Variablen verwenden. Bei gemeinsamer gruppenübergreifender Verwendung, wie beispielsweise beim Karteinamen, wurde sie zusammengefasst. Das Ergebnis basiert also auf den Variablen, die nun die Attribute der Klassen darstellen.

Die Funktionen von Cardfile wurden als Methoden für die Klassen übernommen. Dabei wurden die ursprünglichen Signaturen so geändert, dass Parameter, die Attribute der Klasse darstellen, daraus entfernt wurden.

Wie zuvor bei der Steuerung erwies sich die zuvor erfolgte Trennung der logischen Schichten auch hier als Vorteil, da hier nur der für die Verarbeitung notwendige Teil umgesetzt werden konnte. Das vereinfachte auch das Design der Enterprise Beans.

Datenzugriff

Das Design für die Entities des Datenzugriffs war sehr einfach, da es durch das neue Datenmodell von Web-Cardfile vorgegeben war.

Datenhaltung

Für die Datenhaltung von Web-Cardfile wurde das ursprüngliche Datenmodell geändert, um die Vorteile des relationalen Datenbanksystems nutzen zu können. Die lineare Suche in den Textdateien von Cardfile hatte wesentliche Nachteile bezüglich Geschwindigkeit und Effizienz. Um diese zu überwinden, wurde ein relationales Modell auf Basis der originalen Datenhaltung von Cardfile erstellt, das genau denselben Datenbestand halten kann, aber relational verwendet wird. Das Design des neuen Modells war relativ einfach, da auch das ursprüngliche Modell sehr einfach ist.

4.6.3 Transformation

Die Transformation setzt nun das zuvor erstellte Design in eine Implementierung um. Das Ergebnis ist die Web-Anwendung Web-Cardfile, die funktional äquivalent zum Ausgangssystem Cardfile ist.

Die im Design festgelegten Schichten sind wie in der restrukturierten Variante von Cardfile linear angeordnet. Abhängigkeiten bestehen nur zwischen in der Architektur angrenzenden Schichten. Durch das zuvor festgelegte Design

kann die Transformation schichtenweise modular und vertikal oder horizontal durchgeführt werden. Einzelne Teilsysteme können nacheinander in die Java EE Umgebung auf der Implementierungebene transformiert werden. Die einzelnen Schichten können unabhängig von einander umgesetzt werden. Es ist möglich, die Funktionalität entsprechend einzelner Anwendungsfälle vertikal über alle Schichten zu implementieren. Damit sind Forward und Reverse Strategien durchführbar. Für Web-Cardfile wird eine Forward Strategie gewählt, bei der vertikal die einzelnen Anwendungsfälle entsprechend dem Design umgesetzt werden. Nach Implementierung aller Anwendungsfälle ist die Transformation abgeschlossen.

Für Web-Cardfile wird entschieden, nur Struktur und Architektur direkt wiederzuverwenden. Innerhalb der Funktionen, die in der Zielumgebung Methoden von Objekten darstellen, wird in der Zielsprache Java neu implementiert. Auf eine automatische Übersetzung wird verzichtet. Die Funktionalität innerhalb dieser Methoden wird direkt übernommen. Eine Ausnahme stellen die Methoden der Enterprise Beans dar. Da die Datenbank neu entworfen wurde, muss die Verarbeitung daran angepasst werden. Der Datenzugriff selbst basiert nun auf den Tabellen. Die darin enthaltenen Reihen werden durch Entities repräsentiert. Um die Datenhaltung für die Anwendungslogik funktional äquivalent zu Cardfile verwenden zu können, muss die Datenspeicherung entsprechend der neuen Struktur geändert werden.

Nach Abschluss der Transformation wird der alte Datenbestand aus der bei Cardfile beiliegenden Datenbank übernommen. Mit diesem wird Web-Cardfile ausführlich getestet, um die funktionale Äquivalenz zu Cardfile zu gewährleisten.

Ergebnisse

Das Ergebnis der Transformation, die Implementierung des Zielsystems, wird betrachtet und Probleme bei der Umsetzung aufgezeigt.

Unabhängigkeit der Anwendungslogik von der Datenhaltung

In der Prozessdefinition wurde behauptet, dass die Datenhaltung unabhängig von der Anwendungslogik betrachtet werden kann. Dies stimmt aber nur teilweise. Es trifft zu, dass die Verarbeitung der Anwendungslogik sich der persistenten Datenspeicherung nicht bewusst sein muss, da der Datenzugriff die Verwendung der Technologie für die Datenhaltung kapselt. Da sich aber in diesem Fall das Datenmodell geändert hat, muss die Verarbeitung daran angepasst werden. Es besteht also keine vollständige Unabhängigkeit. Dies wäre nur der Fall bei direkter Wiederverwendung des ursprünglichen Modells, wobei dessen Nachteile auch übernommen werden würden. Da das Zielsystem

diese Nachteile gegenüber dem Legacy System aber nicht mehr aufweisen soll, ist eine Änderung des Modells angebracht.

Umsetzung der Schichten

Die Schichtentrennung des restrukturierten Cardfiles erwies sich hier teilweise als schlecht im Zusammenhang mit der Zieltechnologie. Dort wurde die Suche nach Datensätzen innerhalb der Datenzugriffsschicht gemacht. Dafür wurden die Schlüssel mit dem Suchbegriff verglichen. Dieser Suchbegriff kann ein regulärer Ausdruck sein. Dies lässt sich aber mit der Zieltechnologie nicht umsetzen, da Entities nicht den benötigten Umfang der regulären Ausdrücke für eine funktional äquivalente Suche zur Verfügung stellen. Deshalb mussten Einheiten des Datenzugriffs wieder zur Verarbeitung verschoben werden. Dort kann der Vergleich funktional äquivalent in der Implementierungssprache Java durchgeführt werden. Da die Technologie bei der Restrukturierung noch nicht feststand, konnte dies zu diesem Zeitpunkt nicht berücksichtigt werden.

Neuimplementierung

Die gewählte Transformation ist eine funktional äquivalente Neuimplementierung des ursprünglichen Cardfiles. Wiederverwendet wurden die Anforderungen, weiters Struktur und Architektur der restrukturierten Variante. Der Quellcode innerhalb der strukturellen Einheiten wurde nicht weiterverwendet, sondern neu geschrieben. Die Ergebnisse dabei waren gut und die Implementierung konnte aufgrund der Vorbereitungen einfach durchgeführt werden. Nachteile ergaben sich aber im hohen Aufwand und in der Fehleranfälligkeit. Das funktional äquivalente Neuimplementieren ist ein im Vergleich zum Umfang der Anwendung sehr zeitaufwendiger Vorgang. Weiters ergaben sich bei der Durchführung häufig Fehler als Abweichungen von der ursprünglichen Funktionalität. Deshalb war das Testen auch hier sehr wichtig, um die Äquivalenz zu gewährleisten.

Metriken

Um einen messbaren Vergleich zu erhalten, werden hier Kennzahlen für die 3 Varianten von Cardfile, dem Originalsystem, der restrukturierten Variante und dem Zielsystem Web-Cardfile, angegeben. Die Anzahl der Einheiten gibt bei den C-Implementierungen die Anzahl der Module und Header an. Bei der Java EE Anwendung entspricht sie der Anzahl der Java Klassen und JSF-Seiten. Die Anzahl der Quellcodezeilen sind als logische Lines of Code (LOC) angegeben.

	Einheiten (Module)	Logische LOC
Cardfile Gesamt	27	4343

Tabelle 4.1: Metriken – Ausgangssystem Cardfile

	Einheiten (Module)	Logische LOC
Anwendungslogik	45	4112
Steuerung	22	1962
Verarbeitung	19	1583
Datenzugriff	4	567
Präsentation	4	1493
Cardfile Gesamt	49	5605

Tabelle 4.2: Metriken – Restrukturiertes Cardfile

Sowohl die Anzahl an Modulen als auch die LOC haben sich im Vergleich zum originalen Cardfile stark erhöht. Der Grund dafür ist die Schichtentrennung, die viele neue strukturelle Einheiten ergab.

	Einheiten (Klassen)	Logische LOC
Anwendungslogik	39	5416
Steuerung	16	2711
Verarbeitung	18	1644
Datenzugriff	5	1061
Präsentation	14	
Web-Cardfile Gesamt	53	

Tabelle 4.3: Metriken – Zielsystem Web-Cardfile

Vergleicht man die Zahlen des restrukturierten Cardfiles mit denen von Web-Cardfile, erkennt man allgemein eine reduzierte Anzahl an Einheiten in der Anwendungslogik und eine erhöhte Anzahl an LOC.

Die Unterschiede bei der Anzahl der Module und Klassen ergeben sich dadurch, dass nach der strukturell stark aufgeteilten restrukturierten Cardfile-Variante beim Design von Web-Cardfile Strukturen wieder zusammengefasst wurden, um das objektorientierte Design zu erhalten. Nachdem durch die Restrukturierung die Gruppen nach Schichten zerlegt wurden, wurden bei Web-Cardfile innerhalb der Schichten strukturelle Einheiten zu Objekten

zusammengefasst. Dadurch reduziert sich hier die Anzahl der Einheiten geringfügig.

Dies kann man bei der Steuerung sehen, da die ursprünglichen Funktionen dort in den Backing Beans für die JSF-Seiten ablauforientiert zusammengefasst wurden. Die LOC der Steuerung erhöhen sich allerdings, da diese bei Web-Cardfile einen höheren Aufwand in der Implementierung hat. Eingabe, Ausgabe und Steuerung von JSF-Seiten mittels Backing Beans erfordern hier ein erhöhtes Maß an Funktionalität.

Die Verarbeitung weist keine wesentlichen Änderungen sowohl bei der Anzahl der Einheiten als auch den LOC auf. Die Klassen unterscheiden sich jedoch gänzlich von den Modulen des restrukturierten Cardfiles, da wiederum zu Objekten zusammengefasst wurde. Dabei ergaben sich aber aus einzelnen Strukturen auch neue Klassen, die bei Cardfile in Modulen zusammengefasst vorkommen. Es wurde also einerseits zusammengefasst und andererseits getrennt. Weiters wurden Exceptions hinzugefügt. Dadurch bleibt die Anzahl sowohl der Einheiten als auch der LOC ungefähr gleich.

Bei den Einheiten für den Datenzugriff ist sowohl die Anzahl als auch die Aufteilung gleich geblieben. Die LOC haben sich bei Web-Cardfile allerdings stark erhöht, da ein vollkommen unterschiedlicher Persistenzmechanismus verwendet wird. Die Verwendung der Datenbank über Entities erfordert wesentlich mehr Programmcode als die Dateizugriffe von Cardfile.

Weiters erhöht sich die Anzahl an Einheiten für die Präsentation, da im Gegensatz zu Cardfile bei Web-Cardfile für jede angezeigte Benutzerschnittstelle eine eigene JSF-Seite verwendet wird. Eine Angabe der LOC ist hier für einen Vergleich nicht sinnvoll, deshalb entfällt sie in der Tabelle.

Nachfolgend ist der Zeitaufwand für die Durchführung der einzelnen Schritte angegeben.

	Stunden
Reverse Engineering	35
Restrukturierung	40
Forward Engineering	75

Tabelle 4.4: Metriken – Zeitaufwand der Durchführung

Der Großteil des Zeitaufwandes bei der Restrukturierung ergab sich bei der Trennung der Steuerung von der Verarbeitung. Einerseits war die Identifikation aufgrund von fehlenden Kriterien schwierig. Andererseits führte das Restrukturieren hier aufgrund der häufigen Notwendigkeit von intraproze-

duralen Trennungen zu vielen Fehlern.

Den größten Zeitaufwand hatte das Forward Engineering. Der Grund dafür war zu einem wesentlichen Teil die funktional äquivalente Neuimplementierung. Die zuvor durchgeführte Restrukturierung erleichterte das Design und ergab dadurch eine Zeitersparnis. Die vollständige Neuimplementierung ohne Wiederverwendung auf Quellcodeebene erforderte den höchsten Zeitaufwand aller Schritte.

Kapitel 5

Erkenntnisse

In diesem Kapitel werden die Ergebnisse der Fallstudie in Zusammenhang mit der Prozessdefinition betrachtet. Die dabei gewonnenen Erkenntnisse werden verwendet, um die folgenden Punkte zu untersuchen.

- Welche Eigenschaften für das Zielsystem sind beim Reengineering zu Web-Anwendungen umzusetzen und wie kann diese Umsetzung im Zuge des Prozesses durchgeführt werden.
- Die Umstellung auf die mehrschichtige Architektur der Web-Anwendung wird möglichst früh im Prozessverlauf berücksichtigt. Die Auswirkungen einer solchen Vorgangsweise auf die Durchführung des gesamten Web-Reengineering Prozesses werden betrachtet und als Vorteile und Nachteile zusammengefasst.

5.1 Architektureller Ansatz

5.1.1 Physische und logische Schichten

Es ist wichtig bei der Schichtentrennung physische und logische Schichten zu unterscheiden. In der Fallstudie war physisch die Datenhaltung bereits im Ausgangssystem getrennt, da die Textdateien, die die Datenhaltung darstellen, kein Teil des Quellcodes sind. Die Präsentation aber war noch im Quellcode enthalten. Nach der Abtrennung dieser ergaben sich die 3 physischen Schichten. Danach wurde die verbleibende Anwendung in die 3 logischen Schichten Steuerung, Verarbeitung und Datenzugriff aufgeteilt, die ebenfalls linear angeordnet sind. Diese stellen die logischen Schichten dar und repräsentieren die 3-schichtige Architektur innerhalb der Anwendungslogik. Die Steuerung steht mit der Präsentation in Verbindung. Der Datenzugriff verwendet direkt die persistente Datenhaltung, in diesem Fall die

Textdateien. Die Verarbeitung beinhaltet die Kernfunktionalität und befindet sich zwischen Steuerung und Datenzugriff. Später wurden im Zuge des Forward Engineerings die 3 logischen Schichten im Zielsystem direkt in der Anwendungssprache in strukturell getrennten Teilen implementiert. Diese sind innerhalb der Anwendung selbst linear angeordnet. Die Präsentation wurde mit einer geeigneten Technologie wie HTML oder JSF implementiert. Ebenso die Datenbank, die im physisch getrennten Datenbankserver gehalten wird. Dadurch wurden physische und logische Schichten im Zielsystem umgesetzt.

Diese Aufteilung sollte bereits bei der Restrukturierung berücksichtigt werden. In dieser Fallstudie war die Präsentation nicht mit einer getrennten Technologie implementiert und deshalb ergab die Aufteilung nach physischen und logischen Schichten die 4 strukturellen Teile Präsentation, Steuerung, Verarbeitung und Datenzugriff in der Quellsprache und als fünften Teil die Datenbank als Textdateien. Diese Aufteilung entsprach der gewünschten Struktur für das Zielsystem nach der Transformation. Versuche, den Quellcode in nur 3 Teile entsprechend den Schichten Präsentation, Anwendungslogik und Datenhaltung aufzuteilen, lieferten keine guten Ergebnisse, da in diesem Fall die Steuerung nicht von der Verarbeitung getrennt wurde. Der Grund dafür sind die Ziele für die Schichtentrennung in der Prozessdefinition. Dort wurde angegeben, dass strukturelle Einheiten mit Benutzerschnittstellen und solche mit Datenbankzugriffen vom Rest des Systems als 3 Schichten getrennt werden sollen. Führt man die Schichtentrennung so durch, werden die Steuerung und die Verarbeitung nicht getrennt. Wird das Design dann aufgrund dieser Struktur erstellt, erhält man keine 3 logischen Schichten am Zielsystem und Steuerung sowie Verarbeitung werden in den Backing Beans implementiert. Die Kopplung ist dadurch höher und Wartbarkeit und Erweiterbarkeit niedriger.

Physische Trennung:

Die physische Trennung der 3 Schichten muss durchgeführt werden. Selbst wenn eine Implementierung mit der selben Technologie wie für die Anwendungslogik erfolgt, wie es beispielsweise bei der Verwendung von Servlets für die Präsentation der Fall wäre, muss dennoch auf die HTML-basierte Benutzerschnittstelle und die ereignisbasierte Verarbeitung umgestellt werden. Eine Trennung der Datenbank von der Anwendung bestand in der Fallstudie bereits. Eine Trennung der Schichten bereits im Ausgangssystem hilft bei der Wiederverwendung der Anwendungslogik in Zusammenhang mit dem Design der Zielanwendung, da nur mehr für diese relevanten Teile des Ausgangssystems betrachtet werden müssen. Automatische Methoden können hier verwendet werden, da Kriterien für die Findung vorhanden sind. Der Wiederverwendungswert von Präsentation und Datenhaltung kann im Rahmen dieser

Arbeit nicht ermessen werden. Bei der Fallstudie wurde die zeichenorientierte Benutzerschnittstelle nicht direkt wiederverwendet, da ein zu großer technologischer Unterschied zu JSF bestand. Ebenfalls wurde das Datenmodell verworfen und ein neues relationales Modell für die Zieldatenbank erstellt. Für beide Schichten musste jedoch die Vorgabe der funktionalen Äquivalenz berücksichtigt werden. Das bedeutet, dass die JSF-Seiten äquivalente Möglichkeiten zur Eingabe und Ausgabe bieten, und das neue Datenmodell den Datenbestand des ursprünglichen Modells halten kann. Das wurde durch die Wiederverwendung der Anwendungsfälle und des Datenbestands erreicht.

Logische Trennung:

Im Gegensatz zur physischen Trennung ist eine logische Trennung der Schichten innerhalb der Anwendungslogik nicht zwingend notwendig. Es hat sich aber im Rahmen der Fallstudie gezeigt, dass sie zu Vorteilen für das Design der Zielanwendung führen kann. Diese ergeben sich durch die stärkere Aufteilung der strukturellen Einheiten nach Zuständigkeiten, was eine getrennte Behandlung bei der Umsetzung der Zieleigenschaften ermöglicht. Durch die Trennung ergab sich eine vorteilhafteres Design für die Zielanwendung und die Erstellung vereinfachte sich.

Wären die logischen Schichten bei der Fallstudie nicht getrennt worden, wäre die gesamte Anwendungslogik in den Backing Beans umgesetzt worden, was eine eindeutig schlechtere Struktur liefern würde. Schlechtere Wartbarkeit und Erweiterbarkeit wären die Folge.

Vorteile der logischen Trennung:

- Die Trennung der logischen Schichten bewirkt eine stärkere Aufteilung in Teilsysteme, die dann jeweils für sich betrachtet werden können. Weiters kann dadurch die Umsetzung weiterer Zielsystemeigenschaften lokal in der Schicht durchgeführt werden, die dafür zuständig ist. Dadurch wird das Design für das Zielsystem unterstützt und vereinfacht.
- Die Auftrennung der Funktionen für die Umstellung auf die ereignisbasierte Verarbeitung geschieht nur innerhalb der Steuerung. Wurden die logischen Schichten zuvor getrennt, müssen beim Forward Engineering nur Einheiten dieser Schicht dafür betrachtet werden.
- Die Verwendung der Datenzugriffsschicht geschieht nur innerhalb der Verarbeitung. Wird das zugrunde liegende Datenmodell geändert, müssen auch in der Verarbeitung Änderungen vorgenommen werden. Wurden die logischen Schichten zuvor getrennt, müssen für diese Änderungen nur Einheiten der Verarbeitung bei Design und Transformation betrachtet werden.

- Durch die Kapselung der Datenzugriffe in der Datenzugriffsschicht wird die Verwendung der Datenhaltung transparent. Eine Änderung der Datenbanktechnologie beim Forward Engineering kann lokal in dieser Schicht erfolgen.

Nachteile der logischen Trennung:

- Die Trennung der logischen Schichten im Ausgangssystem kann sehr schwierig durchzuführen sein. Dies ist abhängig von der Zerlegbarkeit des Ausgangssystems. Schwierigkeiten ergeben sich vor allem bei Notwendigkeit von intraprozeduralen Trennungen. Weiters ist die Identifikation von Teilsystemen der Steuerung und der Verarbeitung aufgrund fehlender Kriterien schwierig.
- Aufgrund oben genannter Schwierigkeiten kann sich ein hoher Aufwand für die Durchführung dieses Schrittes ergeben.
- Die Restrukturierungen zur Trennung der logischen Schichten im Ausgangssystem haben sich bei der Fallstudie als sehr fehleranfällig erwiesen. Fehler traten vor allem bei intraprozeduraler Trennung auf.

5.1.2 Objekte und Schichten

Wie bereits beim Design von Web-Cardfile beschrieben, hat sich bei der Fallstudie eine direkte Verwendung der Gruppen für das objektorientierte Design nicht als vorteilhaft herausgestellt. Das lag daran, dass die Gruppen basierend auf dem Aufrufgraphen von Cardfile erstellt wurden. Die Einteilung lieferte zwar sinnvolle Teilsysteme, die sich für eine modulare Migration gut eignen. Es hat sich aber gezeigt, dass ein objektorientiertes Design eher basierend auf den Daten erstellt werden sollte. Eine allgemeine Aussage im Rahmen dieser Arbeit kann aber nicht gemacht werden. Allerdings wird diese Annahme durch die Betrachtung der COREM-Methode unterstützt, die das objektorientierte Design auch anhand der verwendeten Daten erstellt. Im Gegensatz zu den Objekten hat sich der Aufrufgraph günstig für die Umsetzung der Schichten und der ereignisbasierten Verarbeitung erwiesen. Da bei den Ereignissen die Schichten linear durchlaufen werden, eignet sich eine ablauforientierte Betrachtung gut für die Schichtentrennung und das Design der logischen Schichten.

Allgemein ergibt sich ein Widerspruch zwischen der datenbasierten Sichtweise für die Objekte und der ablaufbasierten Sichtweise für die 3-schichtige Architektur.

Objekte – Vergleich mit COREM:

Im Gegensatz zu COREM wurde das objektorientierte Design für das Zielsystem erschwert. Das liegt daran, dass als Ausgangsbasis für die Restrukturierungen der Aufrufgraph von Cardfile verwendet wurde. Dieser repräsentiert den Kontrollfluss durch die Anwendung und wurde anhand der Aufrufbeziehungen erstellt. Die Gruppeneinteilung wurde anhand des Graphen erstellt. Obwohl die Aufteilung sinnvoll getroffen wurde, ist sie nicht optimal für die Erstellung des objektorientierten Designs. Besser wäre dafür eine Einteilung anhand der verwendeten Daten gewesen. Die COREM-Methode basiert auf diesem Ansatz und ordnet die Funktionen danach den Daten zu, um die Objekte zu erhalten. In der Fallstudie wurde der umgekehrte Weg gegangen und die Funktionen der Gruppen danach anhand der Datenverwendung umgruppiert. Dies geschah aber erst beim Design. Im Falle der Verarbeitung von Web-Cardfile wurden dafür Gruppen zusammengefasst, was relativ unproblematisch war. Anders bei der Steuerung. Dafür mussten Funktionen aufgetrennt werden und die Teile wurden unterschiedlichen Objekten zugeordnet. Dies war notwendig aufgrund der Umstellung auf die ereignisbasierte Verarbeitung. Die Gruppeneinteilung war hier für die Objektfindung nicht hilfreich und die Objekte wurden schließlich anhand der zugrunde liegenden Eingangs- und Ausgangsdaten der Präsentation für den Benutzer erstellt. Die 2 Zieleigenschaften Objektorientierung und ereignisbasierte Verarbeitung widersprachen sich hier im Sinne des Designs.

Ablauf – Vergleich mit MORPH:

Im Gegensatz zu MORPH wurde hier nicht nur die Verarbeitung selbst umgestellt, sondern auch die Struktur in der Anwendung daran angepasst. Durch die Trennung der Steuerung vom Rest der Anwendungslogik konnte dies bereits im Ausgangssystem vorbereitet werden und beim Design und der Transformation auch gut umgesetzt werden. Die Aufteilung in die physischen und logischen Schichten erzielte in diesem Fall einen Vorteil für die Umsetzung. Das liegt daran, dass die Schichten den Kontrollfluss durch die Anwendung repräsentieren und die ereignisbasierte Verarbeitung ein Teil davon ist.

5.2 Transformation

5.2.1 Restspuren des Ausgangssystems

Web-Cardfile haftet teilweise noch der ursprüngliche Programmablauf und die Architektur an. Das sieht man gut bei der Umsetzung der Suche. Die Vorgabe war ein funktionales Äquivalent. Um dies zu erreichen, mussten teilweise Nachteile in der Verarbeitung in Kauf genommen werden.

Die Suche am Ausgangssystem lieferte nach dem Suchalgorithmus nacheinander einzeln Einträge für die Präsentation. Nach dem Starten der Suche wird der erste gefundene Eintrag dargestellt. Mit der Enter-Taste kann dann jeweils zum nächsten gefundenen Eintrag gewechselt werden. Während des gesamten Vorganges befindet sich das System vom Ablauf her innerhalb der Schleife des Suchalgorithmus, in der auch die Präsentation stattfindet. Dieses Verhalten kann natürlich in der Web-Anwendung so nicht umgesetzt werden. Hier müssen die Ereignisse, die der Benutzer auslöst, nach dem Senden einer Antwort abgeschlossen sein. Eine Schleife, in der Benutzereingaben stattfinden, kann nicht realisiert werden.

Um den Suchalgorithmus funktional äquivalent umzusetzen, wurden die Ergebnisse einer Anfrage gespeichert, um sie für die nächste Anfrage als Ausgangszustand zu verwenden. Das bedeutet, dass das Ereignis „Suche nächsten Eintrag“ einen Eintrag aufgrund der zuvor geänderten Variablen liefert. Die Verarbeitung jeder Anfrage ist aber für sich abgeschlossen. Das Ereignis implementiert einen Durchlauf der ursprünglichen Schleife mit einer Benutzereingabe. Die nach dem Durchlauf geänderten Variablen werden gespeichert, um sie für die nächste Anfrage verwenden zu können. Damit wurde die Schleife aufgelöst und der Suchalgorithmus kann übernommen werden.

Durch diese Umsetzung wurde zwar funktionale Äquivalenz erreicht, aber Restspuren der ursprünglichen Schleifenverarbeitung sind noch zu finden. Daraus ergeben sich Nachteile in der Verarbeitung. Die Suche kann die Vorteile der relationalen Datenbank nicht nutzen, da die Einträge zwar einzeln präsentiert, aber als Menge der Suchergebnisse aus der Datenbank geholt werden. Weiters muss die aktuelle Position innerhalb der Suche gespeichert werden, um den entsprechenden Eintrag darstellen zu können. Das Resultat davon ist, dass die Suche teilweise im Quellcode und teilweise mittels Datenbankabfragen implementiert werden musste. Dadurch ergibt sich eine schlechtere Performanz im Vergleich zu einer vollständig datenbankbasierten Suche. Es wird zwar das neue Datenmodell verwendet, aber das ursprüngliche Datenmodell wird teilweise in der Anwendungslogik daraus rekonstruiert. Es war nicht möglich, sich für die Suche auf die Datenbank zu beschränken, da ein Äquivalent damit nicht erreichbar war.

5.2.2 Funktionale Äquivalenz und Legacy Nachteile

Während der Suchalgorithmus funktional äquivalent umgesetzt werden konnte, wurde dies beim Ändern von Einträgen nicht erreicht. Der ursprüngliche Vorgang war ein lineares Durchsuchen des Suchergebnisses, wobei immer ein einzelner Eintrag angezeigt wurde. Wurde dabei ein Eintrag verändert, wurde dieser ans Ende der Suchergebnisse gestellt und konnte dort wieder bearbeitet werden. Diese Funktionalität konnte nicht äquivalent umgesetzt werden,

da das entworfene Datenmodell dies nicht unterstützt. Die Suche selbst wird relational durchgeführt. Der geänderte Eintrag wird in das Suchergebnis einsortiert und steht deshalb nicht unbedingt am Ende. Die Möglichkeit, dies vollständig funktional äquivalent umzusetzen besteht zwar, aber dafür muss das ursprüngliche Datenmodell mit der relationalen Datenbank nachgebildet werden. Dadurch werden alle damit verbundenen Nachteile übernommen. Es ist deshalb sinnvoll, dann auf vollständige Äquivalenz zu verzichten, wenn Nachteile der Ausgangssystems übernommen werden und die Abweichungen nur gering sind. Dies gilt jedoch nicht allgemein. In diesem Fall bewirkte es nur eine geringfügige Änderung bei der Präsentation für den Benutzer und hatte keine Auswirkungen auf den Datenbestand. Deshalb war diese Abweichung akzeptabel. In anderen Fällen kann es sinnvoller sein, die Nachteile zu übernehmen, um eine Änderung zu vermeiden. Da ein solcher Fall bei der Fallstudie nicht auftrat, kann dafür hier kein Beispiel gegeben werden.

Allgemein kann die funktionale Äquivalenz im Gegensatz zur Überwindung der Nachteile des Legacy Systems stehen. Es muss abgewogen werden, in welchem Maß Äquivalenz sinnvoll ist und welche Nachteile vermieden werden sollen.

5.2.3 Entscheidung für die Zieltechnologie

Nicht nur die Technologie des Ausgangssystems schränkt den Entwickler ein. Auch die Verwendung der Zieltechnologie erfordert gewisse Kompromisse. Zuvor wurde beschrieben, dass die Suche aufgrund der Anforderungen nicht rein datenbankbasiert umgesetzt werden kann. Ein weiterer Grund dafür ist, dass bei einer Suche die Vergleiche, die zur Ergebnismenge führen, mittels regulärer Ausdrücke durchgeführt werden. Der durch den Benutzer angegebene Suchbegriff wird also nicht als Text, sondern als regulärer Ausdruck interpretiert. Diese Funktionalität konnte in der Zieltechnologie nicht in der Datenzugriffsschicht umgesetzt werden, da die Entities nicht die Möglichkeit dazu bieten.

Die Zieltechnologie sollte sorgfältig gewählt werden. Zu Beginn ist noch nicht bekannt, ob und welche Teile des Ausgangssystems damit nicht gut umsetzbar sind. Wie bei der Prozessdefinition angegeben, sollte eine Entscheidung erst dann getroffen werden, wenn die Ergebnisse des Restrukturierungsschrittes vorliegen und ein entsprechend umfangreiches Programmverständnis vorhanden ist. Dadurch sind zu erwartende Probleme schon ersichtlich und man kann dies bei der Auswahl der Zieltechnologie berücksichtigen.

5.3 Reengineering allgemein

5.3.1 Hoher Aufwand einer Neuentwicklung

Der Aufwand für eine funktional äquivalente Neuentwicklung hat sich im Zuge der Fallstudie im Vergleich zum Funktionsumfang von Web-Cardfile als sehr hoch erwiesen. Das Ergebnis weist dafür gute Wartbarkeit und Erweiterbarkeit auf und hat weitgehend die Nachteile des Legacy Systems Cardfile überwunden. Die Datenbasis kann übernommen werden und Web-Cardfile kann das ursprüngliche Cardfile ersetzen.

Für eine Reduzierung des hohen Aufwandes kann man sich darauf beschränken, wiederverwendbare Teilsysteme zu extrahieren und diese in einem neuen System weiterzuverwenden. Das Thema der Extraktion von Teilsystemen zur Wiederverwendung als Komponenten ist in der Literatur häufig vertreten. Als Beispiel sei hier [ABB+02] genannt.

5.3.2 Abhängigkeit von der Ausgangstechnologie

Semi-automatische Reengineering-Methoden und Remodularisierungstechniken sind stark paradigmenabhängig und weniger von der Sprache selbst. Das kommt daher, dass sich Abhängigkeiten einerseits aus den Zusammenhängen der strukturellen Einheiten ergeben. Selbst bei intraprozeduraler Analyse sind die Unterschiede der Anweisungen verschiedener Sprachen nur syntaktischer Natur. Dadurch gibt es für Reverse Engineering und Restrukturierung keinen großen Unterschied zwischen Sprachen desselben Paradigmas. Andererseits ergeben sich aufgrund des Paradigmas Abhängigkeiten, die berücksichtigt werden müssen. Ein Beispiel stellt die Möglichkeit der Vererbung dar, die selbst aber nicht sprachspezifisch ist und in vielen Sprachen vorkommt.

Der Vorteil semi-automatischer Methoden ergibt sich erst durch die Implementierung durch Werkzeuge. Diese bleiben natürlich sprachabhängig, da der Quellcode dabei geparst werden muss. Mittels Werkzeugunterstützung können die Methoden auch für umfangreiche Systeme eingesetzt werden, was einen sinnvollen praktischen Einsatz erst ermöglicht. Durch den Werkzeugeinsatz ergibt sich die Abhängigkeit der Methoden von der Implementierungssprache.

5.4 Der Web-Reengineering Prozess

5.4.1 Einteilung in Teilschritte

Der Prozess wurde so festgelegt, dass voneinander abgegrenzte Teilschritte nacheinander durchlaufen werden. Die Ergebnisse jeden Schrittes werden als Ausgangsbasis für den jeweils nächsten verwendet. In jedem Schritt müssen von den Entwicklern Entscheidungen getroffen werden. Zum Zeitpunkt einer Entscheidung haben sie aber oft kein Wissen über die zukünftigen Auswirkungen. In vielen Fällen gibt es keine optimale Lösung und eine Bewertung ist aufgrund eines fehlenden Maßstabs schwierig. Deshalb ist es auch schwierig, gute Entscheidungen zu treffen. Die Ergebnisse der Methoden und verwendeter Werkzeugunterstützung bieten Informationen für besseres Programmverstehen und Ansatzpunkte für diese Entscheidungen. Beispielsweise kann geringere Kopplung auf eine verbesserte Struktur hinweisen. Getroffen werden müssen die Entscheidungen aber von den Entwicklern. Hier wirken sich vor allem gutes Programmverständnis und Erfahrung aus.

Die Aufteilung in Teilschritte mit festgelegten Ergebnissen ist sinnvoll, kann jedoch bei fehlender Erfahrung Schwierigkeiten verursachen. Liefert der Prozess aufgrund der Entscheidungen in einem späteren Schritt eine eindeutig schlechte Lösung, kann eine Korrektur angebracht sein. Dann müssen aber die Ergebnisse aller bisherigen Schritte ab der Korrektur neu durchlaufen werden, was den Aufwand sehr stark steigen lässt.

5.4.2 Abstraktionsgrad

Der Web-Reengineering Prozess wurde so weit wie möglich für das Zielsystem Web-Anwendung konkretisiert. Dabei wurde an die Zielsystemeigenschaften angepasst und Möglichkeiten zur Umsetzung beschrieben. Weiters wurden Methoden für den konkreten Einsatz angegeben. Die Eigenschaften des Ausgangssystems wurden dabei offen gelassen. Dadurch muss auf einem höheren Abstraktionsniveau verblieben werden. Der Prozess konzentriert sich auf die Struktur und Architektur von Ausgangssystem und Zielsystem. Dabei wird versucht, das Ausgangssystem in Hinsicht auf diese Eigenschaften dem gewünschten Zielsystem so weit wie möglich anzunähern. Durch das Offenlassen der Eigenschaften des Ausgangssystems muss aber auf einem solchen Abstraktionsniveau verblieben werden, dass eine konkrete und praktisch direkt einsetzbare Definition nicht möglicht ist. Der Prozess soll deshalb mehr im Sinne des traditionellen Software-Entwicklungsprozesses gesehen werden, der die zu erreichenden Ziele definiert und angibt, welche Aktivitäten dafür notwendig sind, aber die konkrete Umsetzung offen lässt.

5.4.3 Migration von Teilsystemen

Beim Übergang von Ausgangssystem zu Zielsystem ist die modulare Aufteilung wesentlich. Abgesehen von Migrationsmethoden, die das Zielsystem komplett getrennt entwickeln, wird die Migration durch die Transformation von zusammenhängenden Teilsystemen in die Zielumgebung durchgeführt. Weist das Ausgangssystem eine dafür günstige Struktur auf, wird dieser Vorgang vereinfacht. Durch Remodularisierung und Schichtentrennung des Ausgangssystems wird eine solche Struktur erreicht. Nachdem sich die beschriebenen Migrationsmethoden am Datenbestand orientieren, ist eine schichtenweise Migration von Vorteil. Die Forward und Reverse Migration Methoden profitieren von einer solchen Aufteilung des Ausgangssystems. Innerhalb der Schichten kann weiter unterteilt werden und so schichtenweise sinnvolle Teilsysteme migriert werden. Obwohl die Anwendung der Migrationsmethoden im Rahmen dieser Arbeit nicht untersucht wurde, zeigten sich diese Vorteile bei der getrennten funktional äquivalenten Neuimplementierung des Zielsystems ebenfalls.

Kapitel 6

Schlussfolgerungen

6.1 Zusammenfassung

Um das Problem der Legacy Systeme zu überwinden, sollen diese in modernisierte Äquivalente transformiert werden. Diese weisen gute Wartbarkeit und Erweiterbarkeit auf und können das Legacy System ablösen. Als Zielanwendung bieten sich dafür Web-Anwendungen an, die aufgrund moderner Technologien viele Vorteile mit sich bringen.

In dieser Arbeit wurde für die Untersuchung des Themas Reengineering zu Web-Anwendungen ein durchgehender Web-Reengineering Prozess definiert. Dieser orientiert sich an den bekannten Reengineering-Aktivitäten Reverse Engineering, Restructuring und Forward Engineering und setzt diese als Teilschritte in eine feste Reihenfolge. Dabei wurden diese allgemeinen Tätigkeiten für das Zielsystem Web-Anwendung konkretisiert. Die zu erreichenden Zielsystemeigenschaften wurden festgelegt und Aktivitäten angegeben, mit denen diese erreicht werden können. Für eine konkrete Umsetzung wurden beispielhaft bekannte Methoden dafür vorgestellt.

Der Web-Reengineering Prozess verwendet einen architekturellen Ansatz. Er konzentriert sich auf die Transformation zur Architektur der Web-Anwendungen, die sich aus mehreren Schichten zusammensetzt. Präsentation, Anwendungslogik und Datenhaltung sind linear angeordnet. Weiters wird die Transformation in eine objektorientierte Umgebung unterstützt. Dieser architekturelle Ansatz wurde dahingehend untersucht, wie weit Struktur und Architektur der Zielanwendung bereits durch Restrukturierung im Ausgangssystem umgesetzt werden kann. Eine solche Vorbereitung an einem möglichst frühen Zeitpunkt im Prozess soll die Transformation zum Zielsystem unterstützen und vereinfachen. Um diesen Ansatz genauer und mittels einer praktischen Durchführung zu untersuchen, wurde eine Fallstudie verwendet.

Durch die Ergebnisse der Fallstudie hat sich gezeigt, dass eine Schichtentrennung im Ausgangssystem sowohl Vorteile als auch Nachteile mit sich bringt. Zunächst ist es angebracht, bei der Restrukturierung physische und logische Schichten zu unterscheiden. Während die physischen Schichten die verteilte 3-Schichten-Architektur darstellen, repräsentieren die logischen Schichten Steuerung, Verarbeitung und Datenzugriff die Architektur innerhalb der Anwendunglogik. Eine solche Sichtweise war hilfreich bei der Schichtentrennung im Ausgangssystem, da dadurch eine sinnvolle Trennung in Teilsysteme ermöglicht wurde. Durch diese Trennung wurden Design und Implementierung des Zielsystems vereinfacht, da eine stärkere Aufteilung erreicht wurde und die Zielsystemeigenschaften getrennt nach zuständigen logischen Schichten umgesetzt werden konnten. Die Nachteile zeigten sich in der Schwierigkeit, die Trennung im Ausgangssystem durchzuführen. Aufgrund ungenauer Kriterien war eine Identifikation der logischen Schichten vor allem bezüglich Steuerung und Verarbeitung sehr schwierig. Weiters sind der hohe Aufwand und die Fehleranfälligkeit der Restrukturierungen zu berücksichtigen.

Im Zuge der vorbereitenden Restrukturierungen wurde auch eine Gruppeneinteilung getroffen und strukturell umgesetzt. Diese sollte das Design der Objekte für das Zielsystem unterstützen. Dabei wurde die Einteilung aufgrund des Aufrufgraphen gemacht. Während sich dieser hilfreich für die Behandlung der 3-Schichten-Architektur und der ereignisbasierten Verarbeitung herausgestellt hat, war die Nützlichkeit für das objektorientierte Design eher gering. Es hat sich gezeigt, dass eine Transformation in eine objektorientierte Umgebung besser anhand der verwendeten Daten geplant wird. In der Fallstudie ergab sich ein Gegensatz bei der Umsetzung von Objekten und Schichtenarchitektur.

Bei der Durchführung des Reengineerings müssen Einschränkungen sowohl der Ausgangstechnologie als auch der Zieltechnologie berücksichtigt werden. Weiters ist zu beurteilen, wie genau die funktionale Äquivalenz umgesetzt werden muss, da es möglich ist, dass dadurch Nachteile des Legacy Systems im Zielsystem übernommen werden müssen.

Eine Orientierung des Reengineerings an der Architektur ist sinnvoll, da diese grundlegende Vorgaben für die Zielanwendung macht und ein Erreichen dieser für die Realisierung notwendig ist. Die architekturelle Sichtweise schafft tieferes Verständnis für das gesamte System und unterstützt dadurch das Reengineering. Deshalb sollte die Zielarchitektur bereits bei der Betrachtung des Ausgangssystems berücksichtigt werden.

6.2 Ausblick

In dieser Arbeit wurde die Architektur des Zielsystems durch Restrukturierung bereits im Ausgangssystem umgesetzt. Nicht untersucht wurde der umgekehrte Weg, bei dem zuerst in die Zielsprache übersetzt wird und danach im Zielsystem die Restrukturierungen vorgenommen werden. Dies hätte den Vorteil, eine automatische Übersetzung für die gesamte Anwendung verwenden zu können. Dadurch würde sich eine Zeitersparnis ergeben. In Hinsicht auf die damit verbundenen Nachteile vor allem bei umfangreichen Systemen ist eine solche Vorgangsweise aber nicht empfehlenswert. Weiters müssen alle Schritte trotzdem durchgeführt werden, um zu einem äquivalenten Ergebnis zu gelangen. Der Unterschied besteht in der Verlagerung ins Zielsystem und der Durchführung in der Zielsprache. Die tatsächliche Zeitersparnis und die Qualität des Ergebnisses wären dabei zu überprüfen.

Ein weiterer Ansatz wäre, nur die physischen Schichten im Ausgangssystem zu trennen und die gesamte Anwendungslogik dann zu Objekten zu transformieren. Danach erst erfolgt die logische Schichtentrennung im Zielsystem. Vor allem da in dieser Arbeit bei der Fallstudie ein Widerspruch zwischen objektorientiertem Design und dem Design der Schichten und der ereignisbasierten Verarbeitung auftrat, wäre ein solcher Ansatz interessant. Der Schwerpunkt würde hier eher auf den Objekten liegen.

Diese Arbeit beschränkte sich auf imperative Sprachen für das Ausgangssystem. Eine weitere Möglichkeit wäre die Untersuchung des Reengineering zu Web-Anwendungen für Ausgangssysteme, die in deklarativen Sprachen implementiert wurden.

Der Prozess arbeitet auf hohem Abstraktionsniveau und muss konkretisiert werden. Ein Ansatzpunkt für weitere Untersuchungen ist die Kombination von mehreren konkreten Methoden, die unterschiedliche Zieleigenschaften umsetzen.

Literaturverzeichnis

[ABB+02] Andriessens, Christoph, Markus Bauer, Holger Berg, Jean-François Girard, Michael Schlemmer und Olaf Seng: *Strategien zur Migration von Altsystemen in komponenten-orientierte Systeme.* Technischer Bericht, Technische Universität Kaiserslautern, 2002.

[Ben95] Bennett, Keith: *Legacy Systems: Coping with Success.* IEEE Software, 12(1):19–23, 1995.

[BLW+97] Bisbal, Jesús, Deirdre Lawless, Bing Wu, Jane Grimson, Vincent Wade, Ray Richardson und Donie O'Sullivan: *An Overview of Legacy System Migration.* In: *APSEC* [IEE97], Seite 529.

[BLWG99] Bisbal, Jesús, Deirdre Lawless, Bing Wu und Jane Grimson: *Legacy Information Systems: Issues and Directions.* IEEE Software, 16(5):103–111, 1999.

[BR04] Bennicke, Marcel und Heinrich Rust: *Programmverstehen und statische Analysetechniken im Kontext des Reverse Engineering und der Qualitätssicherung.* Technischer Bericht, Universität Oldenburg, 2004.

[BS93] Brodie, Michael L. und Michael Stonebraker: *DARWIN: On the Incremental Migration of Legacy Information Systems.* Technischer Bericht S2K-93-25, GTE Laboratories Inc., March 1993.

[BS95] Brodie, Michael L. und Michael Stonebraker: *Migrating legacy systems: gateways, interfaces and the incremental approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[CI90] Chikofsky, Elliot J. und James H. Cross II: *Reverse Engineering and Design Recovery: A Taxonomy.* IEEE Software, 7(1):13–17, 1990.

[CV95] Cimitile, A. und G. Visaggio: *Software salvaging and the call dominance tree.* Journal of Systems and Software, 28(1):117–127, 1995.

[DA00] Davis, Kathi Hogshead und Peter H. Aiken: *Data Reverse Engineering: A Historical Survey.* In: *WCRE '00: Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, Seite 70, Washington, DC, USA, 2000. IEEE Computer Society.

[DLM+05] Ducasse, Stèphane, Michele Lanza, Andrian Marcus, Jonathan I. Maletic und Margaret-Anne D. Storey (Herausgeber): *Proceedings of the 3rd International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2005, September 25, 2005, Budapest, Hungary.* IEEE Computer Society, 2005.

[FKKQ05] Falke, Raimar, Raimund Klein, Rainer Koschke und Jochen Quante: *The Dominance Tree in Visualizing Software Dependencies.* In: Ducasse, Stèphane et al. [DLM+05], Seiten 83–88.

[Fow99] Fowler, Martin: *Refactoring - Improving the Design of Existing Code.* Addison-Wesley, Reading/Massachusetts, 1999.

[Fow07] Fowler, Martin: *Refactoring.* http://www.refactoring.com/, 2007.

[IEE97] IEEE Computer Society: *4th Asia-Pacific Software Engineering and International Computer Science Conference (APSEC '97 / ICSC '97), 2-5 December 1997, Clear Water Bay, Hong Kong.* IEEE Computer Society, 1997.

[IEE98] IEEE – The Institute of Electrical and Electronics Engineers, Inc., New York, NY: *IEEE Std 1219-1998: IEEE Standard for Software Maintenance*, 1998.

[JW99] Jahnke, Jens H. und Jörg Wadsack: *Integration of Analysis and Redesign Activities in Information System Reengineering.* In: *CSMR '99: Proceedings of the Third European Conference on Software Maintenance and Reengineering*, Seite 160, Washington, DC, USA, 1999. IEEE Computer Society.

[KE00] Koschke, Rainer und Thomas Eisenbarth: *A Framework for Experimental Evaluation of Clustering Techniques.* In: *IWPC '00: Proceedings of the 8th International Workshop on Program Comprehension*, Seite 201, Washington, DC, USA, 2000. IEEE Computer Society.

[KG95] Klösch, Rene und Harald Gall: *Objektorientiertes Reverse Engineering.* Springer-Verlag, Berlin Heidelberg New York, 1995.

[KWC98] Kazman, Rick, Steven G. Woods und S. Jeromy Carrière: *Requirements for Integrating Software Architecture and Reengineering Models: CORUM II.* In: *WCRE '98: Proceedings of the Working Conference on Reverse Engineering (WCRE'98)*, Seite 154, Washington, DC, USA, 1998. IEEE Computer Society.

[Lam06] Lampe, David J.: *Cardfile - Index card catalog.* http://www.dplace.com/cardfile/, 2006.

[LL03] Lundberg, Jonas und Welf Löwe: *Architecture Recovery By Semi-Automatic Component Identification.* In: *Software Composition (SC'03) - a workshop affiliated with the Europian Joint Conference on Theory and Practice of Software (ETAPS'03), Poland*, April 2003.

[Luc01] Lucia, Andrea De: *Program slicing: Methods and applications.* In: *First IEEE International Workshop on Source Code Analysis and Manipulation*, Seiten 142–149. IEEE Computer Society Press, Los Alamitos, California, USA, November 2001.

[Mac01] Maciaszek, Leszek A.: *Requirements Analysis and System Design: Developing Information Systems with UML.* Addison-Wesley Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England, 2001.

[MM00] Moore, Melody M. und Lilia Moshkina: *Migrating Legacy User Interfaces to the Internet: Shifting Dialogue Initiative.* In: *WCRE '00: Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, Seite 52, Washington, DC, USA, 2000. IEEE Computer Society.

[oCS06] Computer Science, Department of: *Rigi: a visual tool for understanding legacy systems.* http://www.rigi.csc.uvic.ca/, 2006.

[Ost04] Osterrieder, Christian: *Komponentenmodelle für Web-Anwendungen.* Diplomarbeit, Universität Salzburg, Juli 2004.

[SM07] Sun Microsystems, Inc.: *Java Enterprise Edition.* http://java.sun.com/javaee/, 2007.

[SPL03] Seacord, Robert C., Daniel Plakosh und Grace A. Lewis: *Modernizing Legacy Systems: Software Technologies, Engi-*

neering Process and Business Practices. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[SR99] Siff, Michael und Thomas Reps: *Identifying Modules via Concept Analysis.* IEEE Trans. Software Eng., 25(6):749–768, 1999.

[Tur99] Turau, Volker: *Techniken zur Realisierung Web-basierter Anwendungen.* Informatik Spektrum, 22(1):3–12, 1999.

[WLB+97a] Wu, Bing, Deidre Lawless, Jesús Bisbal, Jane Grimson, Ray Richardson und Donie O'Sullivan: *The Butterfly Methodology : A Gateway-free Approach for Migrating Legacy Information Systems.* In: *ICECCS '97: Proceedings of the 3rd IEEE Conference on Engineering of Complex Computer Systems*, Seiten 200–205, Villa Olmo, Como, Italy, September 1997. IEEE Computer Society.

[WLB+97b] Wu, Bing, Deidre Lawless, Jesús Bisbal, Jane Grimson, Ray Richardson und Donie O'Sullivan: *Legacy System Migration : A Legacy Data Migration Engine.* In: *DATASEM '97: Proccedings of the 17th International Database Conference*, Seiten 129–138, Brno, Czech Republic, Oktober 1997. Ed. Czechoslovak Computer Experts.

[WLB+97c] Wu, Bing, Deirdre Lawless, Jesus Bisbal, Jane Grimson, Vincent Wade, Donie O'Sullivan und Ray Richardson: *Legacy Systems Migration : A Method and its Tool-Kit Framework.* In: *APSEC* [IEE97], Seiten 312–321.

[Zdu02] Zdun, Uwe: *Reengineering to the Web: A Reference Architecture.* In: *CSMR '02: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, Seite 164, Washington, DC, USA, 2002. IEEE Computer Society.

[ZK04] Zou, Ying und Kostas Kontogiannis: *Reengineering Legacy Systems Towards Web Environments.* In: *Managing Corporate Information Systems Evolution and Maintenance*, Seiten 138–146. Idea Group Publishing, Hershey, 2004.