Projektarbeit

PROVIRENT

Professional **Vi**deo **Rent**al Software

Student:	Stefan Forstner
Student:	Remo Griesch
Student:	Philipp Schneider
Fachbereich:	Automatisierung und Informatik
Fachrichtung:	Kommunikationsinformatik
Betreuer:	Prof. Dr. Sigurd Günther
Abgabedatum:	20. Dezember 2005

Inhaltsverzeichnis

1 Einführung **3**
- 1.1 Grundlegendes . . . 3
  - 1.1.1 Betreuender Professor . . . 3
  - 1.1.2 Studenten . . . 3
  - 1.1.3 Kapitelarbeit . . . 3
- 1.2 Motivation . . . 4
- 1.3 Ideen zur Projektarbeit . . . 4
  - 1.3.1 Tippspiel . . . 4
  - 1.3.2 Videosoftware . . . 5
- 1.4 Zielsetzung . . . 5

2 Konzepte und Aufbau **6**
- 2.1 Aufbau . . . 6
  - 2.1.1 Beschreibung des Gesamtsystems . . . 6
  - 2.1.2 das Kundenmodul im Detail . . . 9
  - 2.1.3 das Versandmodul im Detail . . . 11
  - 2.1.4 das Verwaltungsmodul im Detail . . . 15
- 2.2 geplante Module und Versionen . . . 15

3 Technologien **17**
- 3.1 Versionsverwaltung . . . 17
  - 3.1.1 Concurrent Versions System - CVS . . . 21
  - 3.1.2 Subversion - SVN . . . 22
- 3.2 Entwicklungsumgebung . . . 25
  - 3.2.1 Eclipse . . . 25
- 3.3 grafische Benutzerschnittstellen in Java . . . 27
  - 3.3.1 Abstract Window Toolkit - AWT . . . 27
  - 3.3.2 Swing . . . 27
  - 3.3.3 Standard Widget Toolkit und JFace . . . 30
- 3.4 Apache Struts . . . 31
- 3.5 Hibernate . . . 38

4 Implementierung **42**
4.1 Versionsverwaltung mit Subversion 42
4.2 Management Anwendung mit SWT 47
4.2.1 Log4j 47
4.2.2 PropertyResourceBundle-Klasse 47
4.2.3 Locale-Klasse 47
4.2.4 Aufbau der grafischen Benutzeroberfläche 48
4.2.5 SWTResourceManager-Klasse 49
4.2.6 ManagementGui-Klasse 49
4.2.7 CompositeDVD-Klasse 50
4.2.8 DialogDVD-Klasse 53
4.2.9 StatusLineStyledText- und ColorDef-Klasse 54
4.3 Java-Web-Anwendung mit Struts 55
4.3.1 Login-Formular 55
4.3.2 Erzeugung der ActionForm Bean 56
4.3.3 Erzeugung der Action-Klasse 56
4.3.4 Einträge in die *struts-config.xml* 57
4.4 Persistenzschicht mit Hibernate 58
4.4.1 Erzeugen der Hibernate-Mapping-Dateien 58
4.4.2 Generieren der Mapping-Klassen aus den XML-Dateien . . . 58
4.4.3 Generieren des Datenbankschemas 59
4.4.4 Erzeugen der Hibernate-Konfiguration 59
4.4.5 Entwicklung der Klasse *Database* für die Interaktion mit der Datenbank 60

5 Fazit **62**

Literaturverzeichnis **64**

Kapitel 1

Einführung

1.1 Grundlegendes

1.1.1 Betreuender Professor

Hochschule Harz
Prof. Dr. Sigurd Günther
Friedrichstr. 57- 59
38855 Wernigerode
sguenther@hs-harz.de

1.1.2 Studenten

Remo Griesch	Stefan Forstner	Philipp Schneider
E.Thälmannstr.248	Strasse der Jugend 22	Kastanienring 16
06528 Blankenheim	04880 Dommitsch	04316 Leipzig
Romeodied@gmx.de	fossiossi@web.de	provirent@phil-schneider.de

1.1.3 Kapitelarbeit

Hier wird aufgeschlüsselt, welcher Student welchen Teil der Dokumentation geschrieben hat.Kapitel 5 auf Seiten 62–63 wurde von allen Studenten bearbeitet.

Remo Griesch bearbeitete die folgenden Kapitel:

- Abschnitt 3.2 auf Seiten 25–26;
- Abschnitt 3.3 auf Seiten 27–30;
- Abschnitt 4.2 auf Seiten 47–54

Stefan Forstner bearbeitete folgende Kapitel:

- Abschnitt 3.4 auf Seiten 31–37;
- Abschnitt 3.5 auf Seiten 38–41;
- Abschnitt 4.3 auf Seiten 55–57;
- Abschnitt 4.4 auf Seiten 58–61

Philipp Schneider bearbeitete folgende Kapitel:

- Kapitel 1 auf Seiten 3–5;
- Kapitel 2 auf Seiten 6–15;
- Abschnitt 3.1 auf Seiten 17–23;
- Abschnitt 4.1 auf Seiten 42–46

1.2 Motivation

Im Rahmen des Studiums an der Fachhochschule Harz in Wernigerode muss jeder Student des Studiengangs *Kommunikationsinformatik* eine Projektarbeit abgeben. Dies bedeutet, daß der Student eine Aufgabe (meist Programmieraufgabe) alleine oder in einem kleinen Team bewältigen muss. Die Professoren der Hochschule bieten dabei viele interessante Projektarbeiten an, sind jedoch auf offen für eigene Vorschläge der Studenten.
Da in den Teamprojekten[1] *Labmin*[2] und *German Team Sony Aibo*[3] eine interessante Aufgabe von den Studenten gelöst wurde, sollte das dort erlernte Wissen vertieft und weiter ausgebaut werden.

1.3 Ideen zur Projektarbeit

1.3.1 Tippspiel

Die erste Idee dieser Projektarbeit war die Umsetzung eines Tippspiels in Java, passend zu den damaligen Fussball-Europameisterschaft in Portugal. Diese Idee wurde im JavaMagazin[4] in mehreren Ausgaben aufgegriffen und verschiedene Ansatzmöglichkeiten diskutiert. Die Idee unseres Tippspiel war dabei eine Webanwendung mit Datenbankanbindung. Nutzer dieses Systems sollten sich in verschiedenen Tippgemeinschaften, mit je einem Tippgemeinschaftsverwalter, zusammen finden

[1] Auch das Teamprojekt ist Bestandteil des Studiums. Beim Teamprojekt müßen mehrere Studenten (7-15) gemeinsam eine Programmieraufgabe umsetzen.
[2] `http://labmin.de.vu`
[3] `http://www.der-baer.com/projects.htm`
[4] [Frotscher 2004a][Frotscher 2004b][Frotscher 2004c]

und gemeinsam die EM 2004 tippen. Das Tippspiel sollte jedoch nicht nur auf die EM 2004 zugeschnitten sein, sondern auch für andere Fußballereignisse tauglich sein. Zusätzlich kam von unserer Seite die Idee, eine Webanwendung zur Verwaltung der Bundesligaergebnisse. Ein Tippspielsystem sollte dann auf diese Daten zurückgreifen und so ein Bundesligatippspiel darstellen können.
Dieser Gedanke wurde jedoch aus verschiedenen Gründen verworfen. Zum einen war es nicht unsere Idee, sondern die des Javamagazin's und zum anderen wussten wir nicht sofort was bei diesem System alles zu realisieren war. Die grobe Funktionsweise war allen klar, jedoch fehlte bei diesem System das gewisse etwas.

1.3.2 Videosoftware

Da jeder von uns schon einmal ein Video in einer Videothek ausgeliehen, kam uns der Gedanke einer Onlinevideothek. Solche Videotheken gibt es mittlerweile wie bspw. Amango[5], Netleih[6], Invdeo[7] und Verleihshop [8]. Bei genauer Betrachtung dieser Onlinevideotheken, fragten wir uns wie solch eine Videothek technisch funktioniert. Da wir gerade auf der Suche nach einem idealen Projekt waren, hatten wir damit eins gefunden.
Es solle versucht werden eine Online-Videothek mit entsprechenden Modulen zu realisieren.

1.4 Zielsetzung

Zielsetzung dieses Projektes ist dabei Erfahrung mit verschiedenen neuen Technologien zu sammeln und selbständig an einem Projekt zu arbeiten. Sowohl die eigene Gedanken, Ideen, Planung und auch Realisierung dieses Projektes sollten uns auf eine spätere Eigenverantwortung im Berufsleben vorbereiten. Das Projekt sollte dabei keine vollständige und fehlerfreie Implementierung darstellen. Uns war bewußt, dass wir nur einen einfachen Prototypen einzelner Module realisieren könnten.

[5] `http://www.amango.de`
[6] `http://www.netleih.de`
[7] `http://www.invdeo.de/`
[8] `http://www.verleihshop.de`

Kapitel 2

Konzepte und Aufbau

2.1 Aufbau

2.1.1 Beschreibung des Gesamtsystems

Bei einer klassischen Videothek besucht der Kunde das Ladengeschäft der Videothek und stöbert dabei nach Videos, die er gerne an diesen Abend schauen möchte. Dabei muss die Videothek eine möglichst größe Ladenfläche besitzen um die Videos dem Kunden zu präsentieren. Nachdem der Kunde sich für ein Video entschieden hat, nimmt er entweder die leere Verpackung oder ein Plastikschild mit einer Nummer zum Verleihschalter der Videothek. Nachdem der Kunde seine Kundenkarte vorzeigt und durch sein Passwort oder seine Unterschrift verifiziert wurde, sucht der Mitarbeiter anhand einer Nummer in der Leerverpackung oder des Plastikschildes das entsprechende Video heraus, markiert dieses Video im System und gibt es dem Kunden. Dies ist der klasische Ablauf in einer Videothek.
Bei einer Online-Videothek kann der Kunde, durch den Versand der Videos, keine Videos für den gleichen Abend ausleihen. Er ist gezwungen, sich einige Tage vorher für ein oder mehrere Videos zu entscheiden. Der Ablauf unterscheidet sich von einer klassischen Videothek. Der Kunden „besucht“ die Webseite der Videothek und sucht im Angebot nach Filmen die er sich ausleihen möchte. Nachdem die Verfügbarkeit überprüft wurde, legt er Videos in seinem Warenkorb ab. Durch Eingabe seines Benutzernamens und das zugehörige Passwort wird der Kunde verifiziert. In dem Lager der Videothek nimmt ein Mitarbeiter die Bestellung über einen Monitor oder eine ausgedruckte Liste entgegeben und bearbeitet die Bestellung. Dabei sucht dieser die Videos für den Kunden heraus, nimmt die Videos in das System auf und versendet die Videos zu dem Kunden per Post. Der Kunde erhält seine gewünschten Videos, kann diese sich anschauen und schickt diese nach einer bestimmten Zeit an die Videothek zurück.
Das hier gewünschte System soll eine komplette Videothek ersetzen. Die Online-

Videothek benötigt nur noch ein Lager für die zu verleihenden Videos und wenige Mitarbeiter für den Versand der Videos und die Verwaltung der Videothek. Die Software wird dabei wie in Abbildung 2.1 auf der nächsten Seite zu sehen ist, von zwei verschiedenen Personenkreisen benutzt, dem Kunden und dem Mitarbeiter. Der Kunde kann die in der Abbildung dargestellten Aktionen ausführen, wie bspw. betrachten und bestellen von Videos. Der Mitarbeiter kann dabei das System verwalten und Bestellungen der Kunden bearbeiten.

Nach einiger Überlegung wurde festgestellt, das die Software aus drei anstatt zwei Modulen bestehen muss, wie in Abbildung 2.2 auf Seite 9 zu sehen ist. Bei den Mitarbeitern der Online-Videothek muss in Verwaltung und Versand/Lager unterschieden werden, da diese unterschiedlichen Aufgaben von unterschiedlichen Mitarbeitern bearbeitet werden. Das **Kundenmodul** ist die Internetpräsenz der Videothek und repräsentiert das Unternehmen nach aussen. Auf dieser dynamischen Webseite kann der Kunde die vorhanden Videos durchstöbern und detaillierte Informationen zu den Videos erhalten. Nach erfolgreicher Anmeldung im System kann der Kunde die Verfügbarkeit des jeweiligen Videos kontrollieren und auf Wunsch Videos ausleihen. In einem zusätzlichen Menüpunkt kann er seine bestellten Videos betrachten und sich ggf. Rechnungen ausdrucken. Das **Versandmodul** stellt die benötigte Software für das Lager und den Versand zur Verfügung. Mit deren Hilfe kann ein Mitarbeiter der Online-Videothek Videos für den Versand vorbereiten. D.h. der Mitarbeiter bekommt eine Liste mit Bestellungen von Kunden (elektronisch oder auf Papier) und arbeitet diese ab. Damit der Mitarbeiter nicht jedes mal die Kundennummer und Nummern der Videos eintippen muss, wird seine Arbeit durch Barcodes und Barcodescanner unterstützt. Mit dessen Hilfe markiert er Videos für einen bestimmten Kunden und eine bestimmte Bestellung und versendet diese. Dem System teilt der Mitarbeiter dadurch mit, das bestimmte Videos nicht mehr verfügbar sind und von einem bestimmten Kunden ausgeliehen wurde. Rechnungen, Versandetiketten und eventuelle Lieferscheine werden dabei automatisch mit Hilfe eines Druckers erstellt. Zusätzlich bietet das Versandmodul die Möglichkeit, zurück gekommene Videos der Kunden wieder in das System aufzunehmen. Somit wurde das Video wieder vom Kunden zurückgegeben und es kann im System als vorhanden/ausleihbar markiert werden oder gleich an den nächsten Kunden weitergeschickt werden. Das **Verwaltungsmodul** hilft den Mitarbeitern in der Verwaltung bei der Organisation der Online-Videothek. Es können Kundendaten und Rechnungen betrachtet und ggf. gedruckt werden. Das Videosortiment kann bearbeitet und inhaltliche Änderungen (z.B. Sonderangebote) an dem Kundenmodul vorgenommen werden. Weiterhin besteht die Möglichkeit ausführliche Reports & Statistiken zu erstellen und zu betrachten.

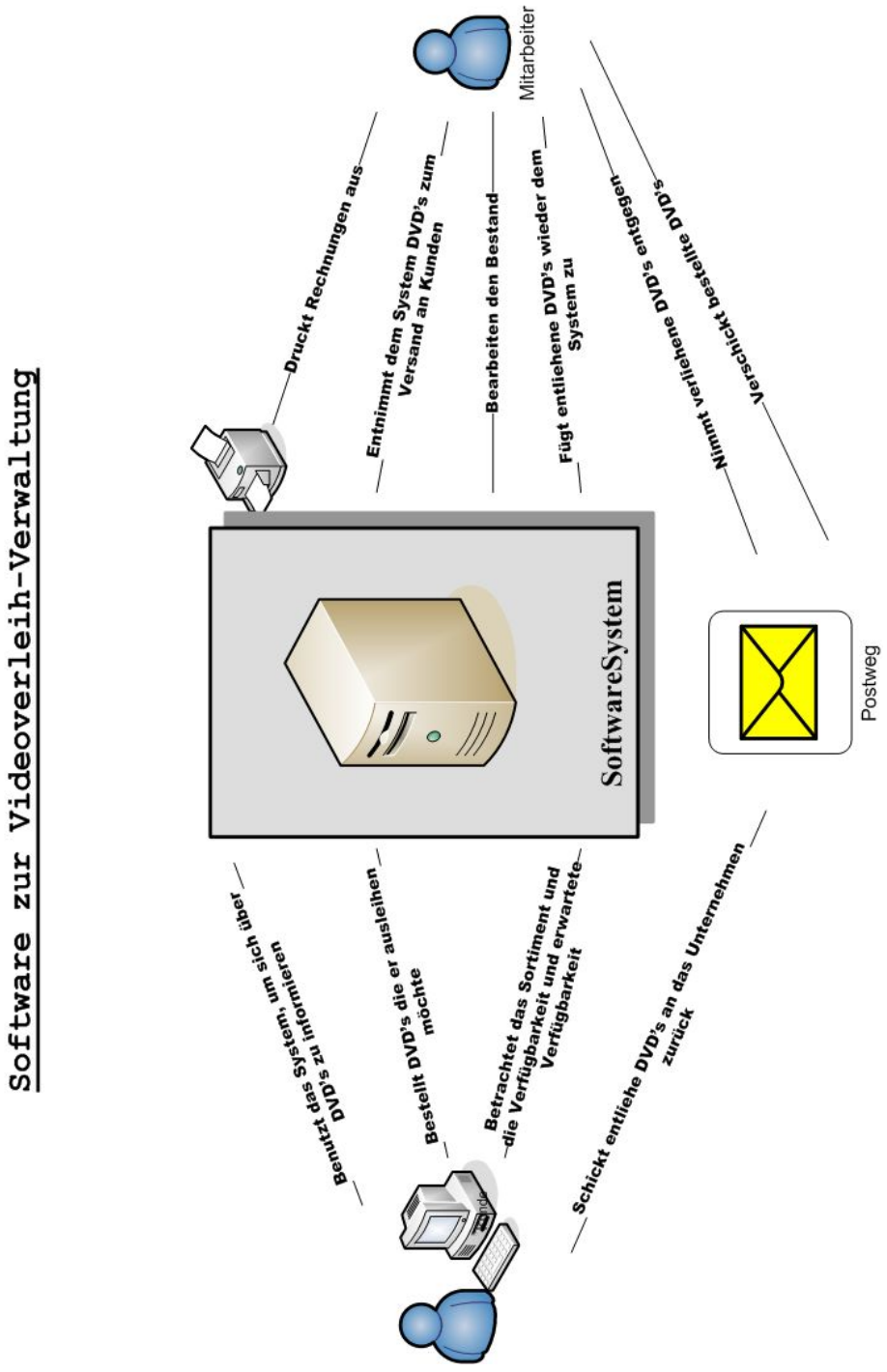

Abbildung 2.1: Erste Gedanken zu der Videosoftware

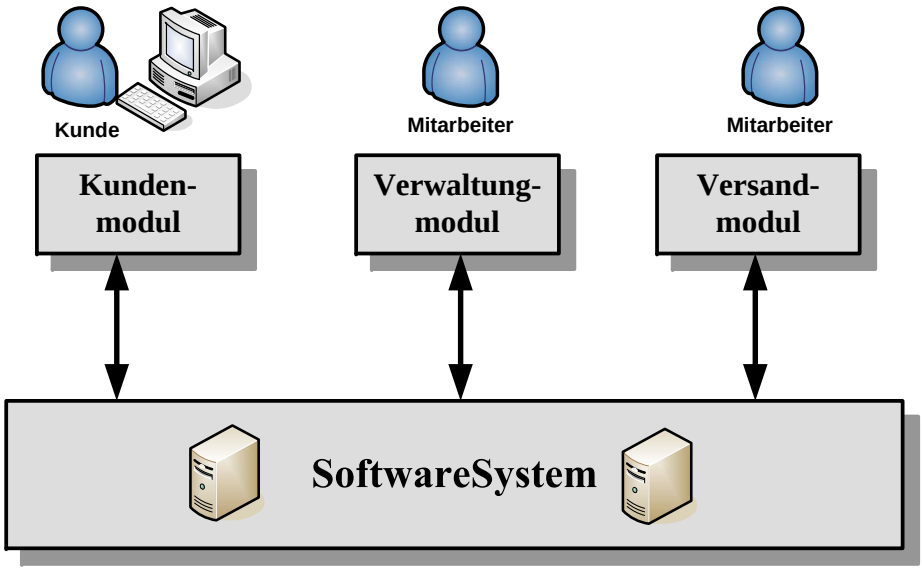

Abbildung 2.2: Die drei Module der Software

2.1.2 das Kundenmodul im Detail

Das Kundenmodul ist das wichtigste Element der Anwendung, denn dieses Modul wird vom Kunden verwendet und dieser bestimmt über Erfolg oder Misserfolg der Online-Videothek und damit über die Anwendung. Das Kundenmodul wird, wie bereits erklärt, als Webanwendung entwickelt. Somit muss der Nutzer keine spezielle Software auf seinem Rechner verwenden und kann die Software somit überall verwenden. Grundsätzlich wird bei der Anwendung zwischen zwei verschiedenen Kundengruppen unterschieden: dem *Interessenten* und dem *angemeldeten Kunden*. Ein *Interessent* ist ein Internetnutzer, der sich über das Angebot der Online-Videothek interessiert, ohne bisher ein Video ausgeliehen zu haben. Ist der Interessent von dem Angebot überzeugt und möchte ein Video ausleihen, muss er sich im System registrieren (siehe 1. Abbildung 2.3 auf Seite 11). Dazu gehört neben einem Benutzernamen und Passwort auch seine vollständige Adresse, für den Versand der Videos bzw. Rechnungen, und seine Bankverbindung bzw. Kreditkartendaten, für die Bezahlung. Ist der Kunde in das System eingeloggt, ist er ein angemeldeter Kunde. Je nach Kundengruppe präsentiert sich die Webseite mit anderen Funktionalitäten. Zuerst sollen die Funktionen und Möglichkeiten eines Interessenten erklärt werden. Der Interessent kann sich auf der Webseite der Online-Videothek sowohl über das Angebot an Videos (siehe 2. Abbildung 2.3 auf Seite 11), als auch über die Online-Videothek informieren. Über verschiedene Webseiten erhält er Einblick in die für ihn interessante Funktionsweise der Online-Videothek und

allgemeinen Daten, wie Impressum, Kontaktdaten und Allgemeine Geschäftsbedingungen (siehe 3. Abbildung 2.3 auf der nächsten Seite). Über eine intelligente Suche, Navigationselementen und den Top-Listen (siehe 5. Abbildung 2.3 auf der nächsten Seite) kann er nach Videos suchen bzw. stöbern (siehe 4. Abbildung 2.3 auf der nächsten Seite). Die Videos sind dabei geordnet. Der Interessent kann detaillierte Informationen über einzelne Filme erhalten (siehe 2. Abbildung 2.3 auf der nächsten Seite). Dabei kann er sich über die Features des Filmes informieren, Kommentare bzw. Bewertungen lesen und ähnliche Filme betrachten. Die verwendete Suche wird als intelligente Suche bezeichnet, weil diese Suche ähnliche bzw. verwandte Filme findet und sowohl im Titel, als auch in der Beschreibung der Filme nach dem Stichwort sucht. Es könnte auch realisiert werden, das die Suchergebnisse nach Trefferquoten geordnet werden bzw. das Suchfeld automatisch das Wort nach häufigen Suchbegriffen ergänzt.
Ist der User eingeloggt hat er die gleiche Funktionsweise wie ein Interessent, nur mit erweiterten Funktionen. In der Ergebnissliste der Suche oder der Anzeigeliste der Kategorien, welche beide gleich aufgebaut sind, befinden sind neben jedem Video Elemente zum Bestellen (siehe 1. Abbildung 2.4 auf Seite 12) und zum Überprüfen der Verfügbarkeit (siehe 2. Abbildung 2.4 auf Seite 12). Jedes Video, also jeder Film, ist in einer gewissen Anzahl vorhanden. Mit Hilfe der Verfügbarkeit kann der Kunde überprüfen, ob ein Video dieses Filmes für ihn zum Ausleihen vorhanden ist, bzw. wann das nächste Video wieder verfügbar ist. Je nach Status der Verfügbarkeit kann der Kunde das Video in seinem Warenkorb legen und damit ausleihen, oder vormerken lassen. Bei einer Vormerkung wird der Kunde entweder per Email informiert, dass das Video vorhanden ist, oder das Video wird schnellstmöglich ihn versendet. Hat der Kunde Videos in seinen Warenkorb abgelegt, kann dieser am Ende seiner Bestellung zur Kasse gehen und somit die Videos ausleihen. Weiterhin hat der Kunde die Möglichkeit eine so genannte *Wunschliste* zu Erstellen (siehe 3. Abbildung 2.4 auf Seite 12). Diese Liste kann eine bestimmte Anzahl an Videos aufnehmen, die der Kunde schauen möchte. Das System arbeitet diese Liste automatisch ab, indem es dem Kunden eine bestimmte Anzahl an Videos mit einmal zusendet und bei Rücksendung durch den Kunden die nächsten Filme automatisch an den Kunden sendet. Somit hat der Kunde z.B. die Möglichkeit 50 Filme zu bestimmen, die er gerne anschauen möchte. Das System sendet ihm immer zwei Filme mit einmal zu, die er sich anschauen kann. Sendet er diese zwei Filme zurück, bekommt er die nächsten zwei Filme, bis die Liste abgearbeitet wurde. Weiterhin bietet das Kundenmodul dem Kunden die Möglichkeiten, ältere und aktuelle Bestellungen bzw. Vorbestellungen zu betrachten (siehe 4. Abbildung 2.4 auf Seite 12), Rechnungen auszudrucken (siehe 5.) und allgemeine Einstellungen an seinem Profil vorzunehmen (siehe 6.). Das Kundenmodul soll dabei einfach und ohne Bedienungsanleitung zu bedienen sein. Der Kunden soll sich schnell zu recht

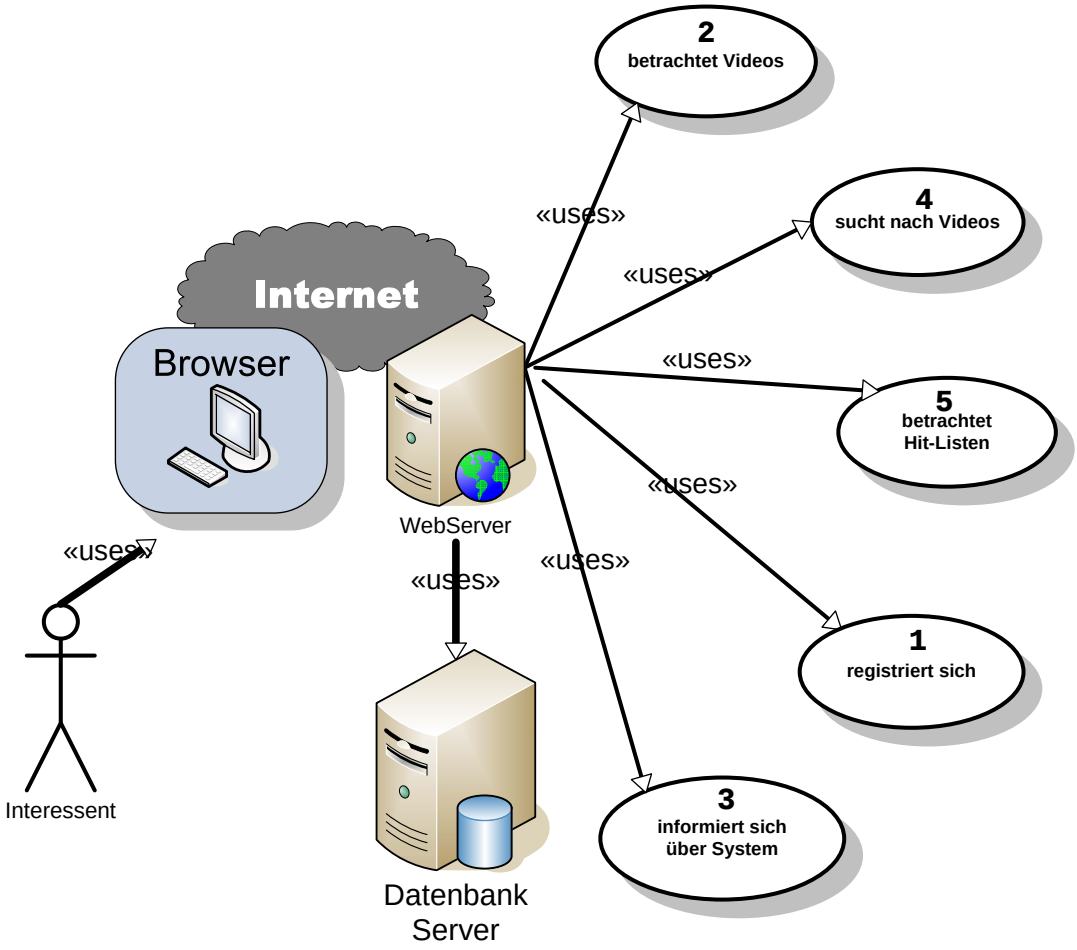

Abbildung 2.3: UseCase Kundenmodul Interessenten

finden und schnell zu seinem Ziel, einer Bestellung, kommen.

2.1.3 das Versandmodul im Detail

Das Versandmodul ist der Bestandteil der Videothek in der die Arbeit geschieht. Hier werden Videos herrausgesucht, verpackt und an den Kunden versendet. Dabei wird der Mitarbeiter soweit wie Möglich von der Technik unterstützt. Bei dieser Technik handelt es sich hauptsächlich um Barcodescanner. Ein Barcodescanner ist ein Lesegerät, das mit Hilfe eines Lasers einen Strichcode auf einer Verpackung oder einem Blatt einliest. Dieser Strichcode kann dabei je nach Art und Weise eine Nummer oder eine Wort repräsentieren. Diese Strichcodes sind aus dem heu-

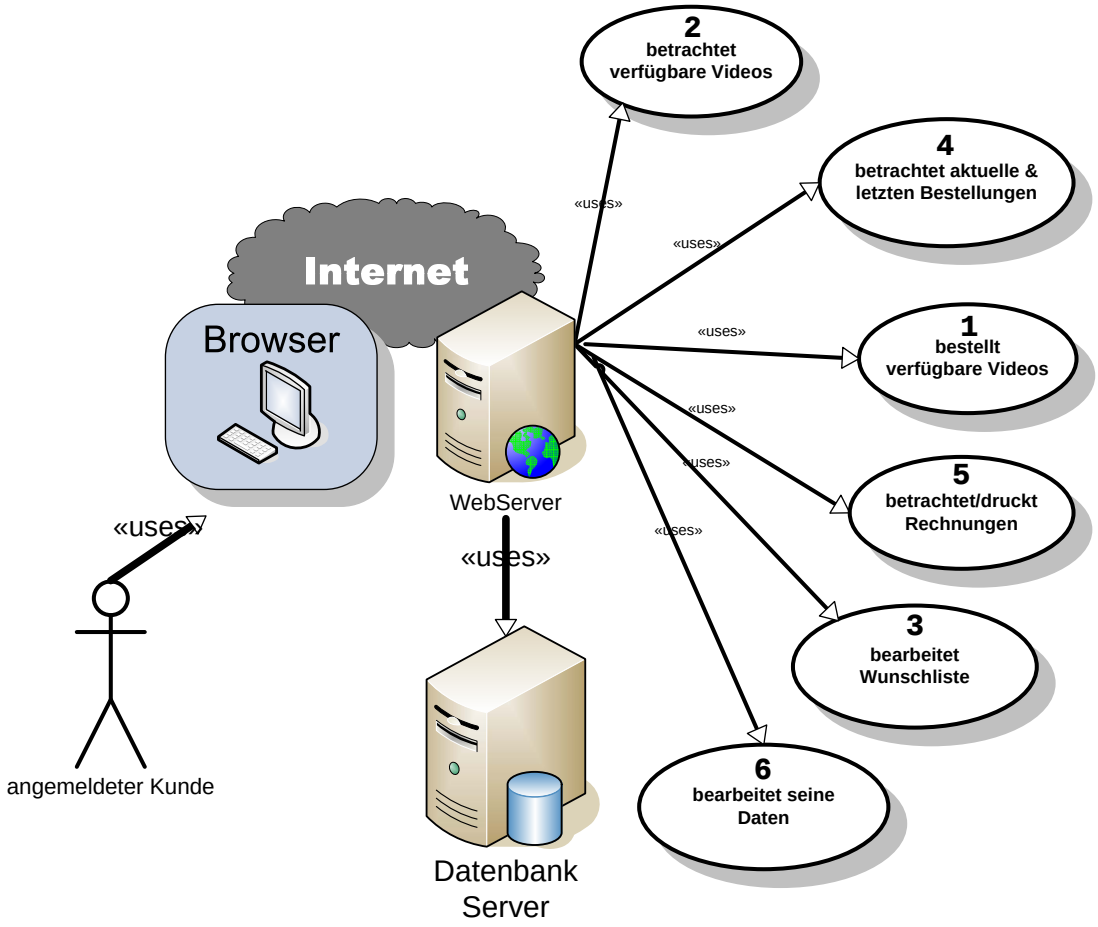

Abbildung 2.4: UseCase Kundenmodul angemeldete Kunden

tigen Alltag nicht wegzudenken, denn sie sind auf jeder Verpackung vorhanden und werden in Supermärkten als Preisschild verwendet. Dabei hat jedes Produkt eine eindeutige Nummer, zu der mit Hilfe einer Datenbank ein Preis zugeordnet wird. Neben Strichcodes gibt es noch zweidimensionale Codes, die mehr Informationen speichern können. An einem Zentralen Rechner werden die Bestellungen der Kun-

Abbildung 2.5: Abbildung eines Barcodescanners

den ausgedruckt. Solch ein Ausdruck besteht aus einer Seite, auf der oben zwei Adressetiketten und unten einem Lieferschein vorhanden sind. Das erste Adressetikett ist das für den Versand zu dem Kunden, das auf den Umschlag der Bestellung geklebt wird. Das zweite ist dasjenige, welches der Kunde auf dem Umschlag zur Rücksendung der Video klebt. Der Lieferschein ist sowohl für den Mitarbeiter als auch für den Kunden wichtig. Der Mitarbeiter liest mit Hilfe eines Barcodescanners die Bestellnummer ein. Aus dieser Nummer kann der Computer sowohl auf den Kunden als auch auf die Videos schließen, die versendet werden sollen. Der Mitarbeiter sucht, mit Hilfe einer weiteren Software den Lagerort der jeweiligen Videos heraus und scannt deren eindeutige Nummer. Damit sind die Videos im System nicht mehr verfügbar und dem Kunden zugeordnet. Anschließend verpackt er die Videos in einen entsprechenden Umschlag und verschickt die Videos mit dem Ausdruck an dem Kunden. Bei diesem Vorgang gibt es mehrere Möglichkeiten, wie der Mitarbeiter vorgehen kann. Damit der Mitarbeiter nicht für jede Bestellung durch das Lager gehen muss, kann das System eine Liste mit Videos erstellen, für die nächsten 10 Bestellungen. Somit sucht der Mitarbeiter diese Videos heraus und bearbeitet nacheinander diese zehn Bestellungen. Eine weitere Möglichkeit wäre, das der Mitarbeiter und das Lager durch ein automatisiertes Lager ersetzt wird. Dabei übernimmt ein Roboter die Aufgabe des Suchen und Finden der Videos. Schickt der Kunde die Videos an die Online-Videothek zurück, nimmt ein Mitarbeiter die Videos entgegen. Zuerst scannt er dabei die Bestellnummer und die eindeutige Nummer des jeweiligen Videos. Zusätzlich muss er den Zustand der Videos überprüfen und im System eintragen. Anschließend sind die Videos wieder

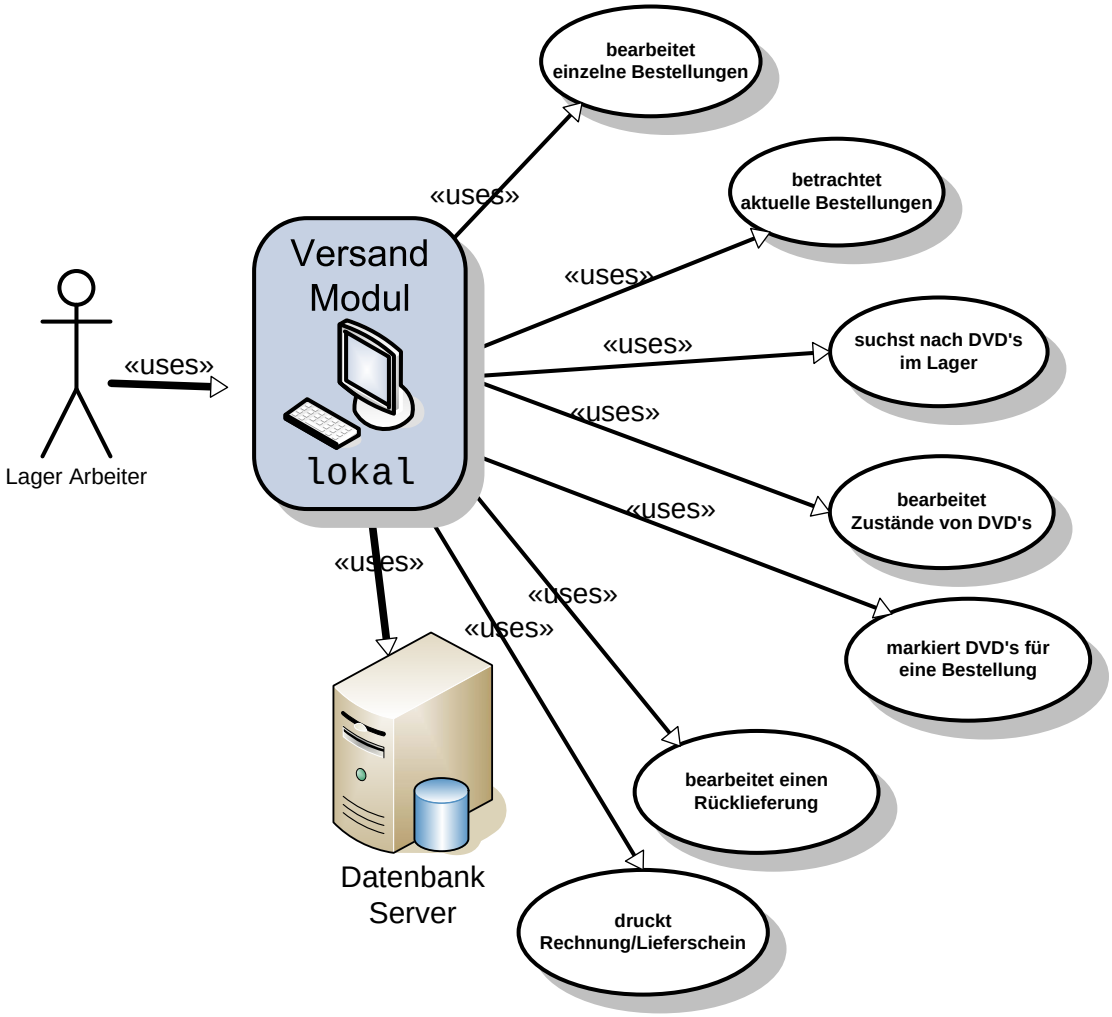

Abbildung 2.6: UseCase Versandmodul

im System verfügbar und können wieder eingeordnet werden oder an den nächsten Kunden versendet werden.
Das komplette Lager könnte durch spezielle Maschinen vollständig automatisiert werden. Diese Maschinen würden dann automatisch das Suchen und das Verpacken der Videos übernehmen. Dabei würden wieder Barcodescanner zum Einsatz kommen, um die Bestellnummern und Videonummers automatisch einzulesen. Solch ein System würde sich aber nur bei einer sehr grosser Online-Videothek rentieren, da der Anschaffungspreis solcher Maschinen enorm ist.

2.1.4 das Verwaltungsmodul im Detail

Das Verwaltungsmodul ist für die Verwaltung der Online-Videothek in Büroräumen gedacht. Die Mitarbeiter des Unternehmens haben dabei Einblick in die verschiedenen Daten der Online-Videothek und können diese, sollten sie die benötigten Rechte besitzen, verändern. Unter zu Hilfenahme des Verwaltungsmodul können neue Videos in System aufgenommen werden. Dazu wird zuerst der jeweilige Film hinzugefügt und danach die einzelnen Videos dieses Filmes. Dabei werden sich wiederholende Daten wie Darsteller oder Genre in eigenen Elementen gespeichert. Nachdem der Mitarbeiter mit dem Hinzufügen von neuen Videos fertig ist, werden diese automatisch im Kundenmodul verfügbar sein. Mit Hilfe dieses Moduls können auch Kundendaten betrachtet oder verändert werden. Somit kann z.B. eine Telefonhotline oder der Kundensupport Fragen der Kunden zu einzelnen Bestellungen beantworten. Für die Geschäftsleitung können ausführliche Statistiken erstellt werden.

2.2 geplante Module und Versionen

Da dieses Projekt nach ersten Überlegungen und Planungen nicht nur ein kleiner Projekt ist, wurde beschlossen, zuerst das Verwaltungsmodul, dann das Kundenmodul und danach das Versandmodul zu implementieren. Das Verwaltungsmodul ist das Herzstück der Online-Videothek. Hier werden die Daten erstellt und verwaltet, die von den beiden anderen Modulen verwendet werden. Das Verwaltungsmodul und das Versandmodul, soll dabei eine Anwendung auf einem beliebigen Rechner innerhalb des Firmennetzwerks sein. Das Kundenmodul hingegen soll eine weltweit verwendbare Webanwendung sein. Mit der Realisierung des Verwaltungsmodul wurde zuerst begonnen. Im Laufe der Entwicklung des Verwaltungsmoduls wurde das Ausmaße des Projektes sichtbar. Der Umfang des Projektes war größer als angenommen. Die Einarbeitungszeit in die Technologien, die Koordination der Zusammenarbeit und die größe des Projektes führten zu einer sehr langen Entwicklungszeit des Verwaltungsmoduls. Das Kundenmodul wurde dadurch nur teilweise und anschaulich realisiert werden. Auf eine Realisierung des Versandmoduls wurde komplett verzichtet.

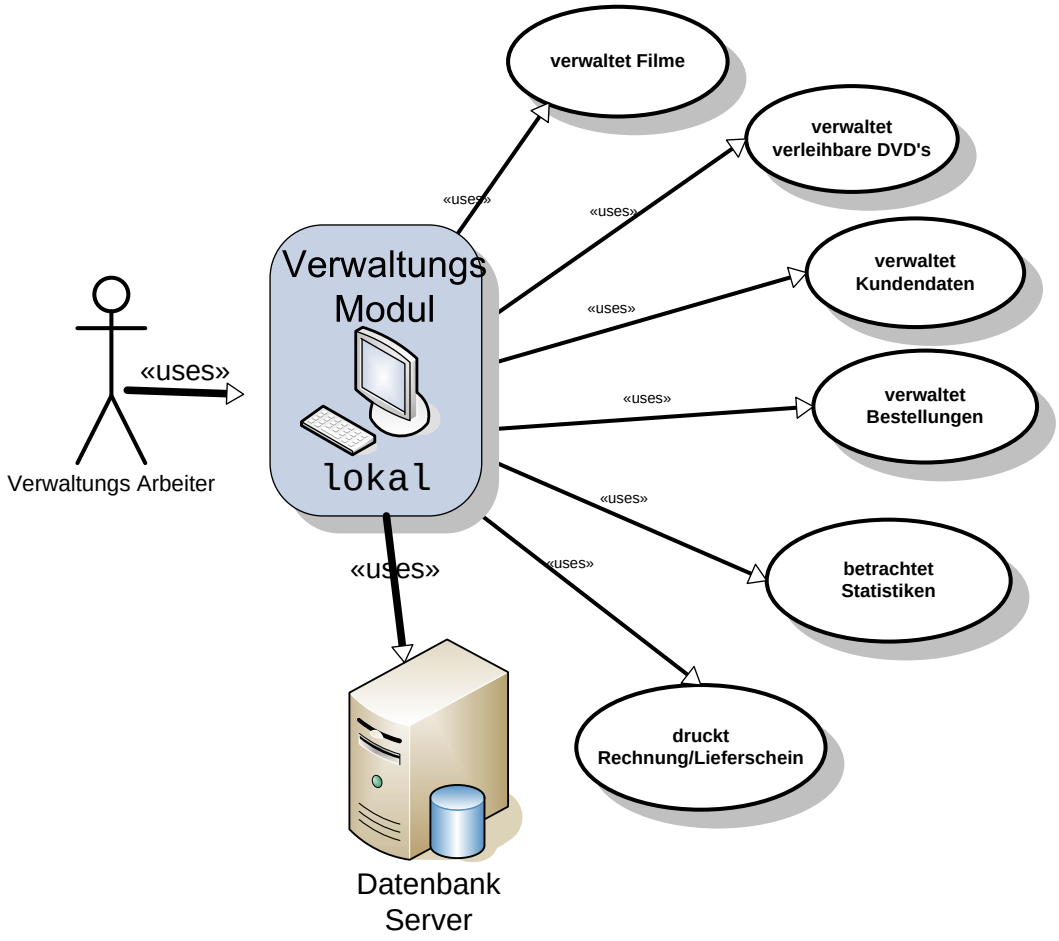

Abbildung 2.7: UseCase Verwaltungsmodul

Kapitel 3

Technologien

3.1 Versionsverwaltung

Eine Versionsverwaltung ist ein System der Softwareentwicklung zur Versionierung und Kontrolle des gemeinsamen Zugriff's auf Quelltexte. Mit Hilfe einer solchen Software werden laufende Änderungen an Quelltexten oder ähnlichen Dateien erfasst und in einem Archiv mit Zeitstempel und Benutzerkennung erfasst. Ein Archiv, auch Repository genannt, ist dabei meist eine Datenbank oder ein systemabhängiges Dateiformat. Mit Hilfe einer Versionsverwaltung können mehrere Benutzer (Entwickler) an einem Projekt, sogar an einer Datei gleichzeitig arbeiten. Ohne den Einsatz einer Versionsverwaltung ist es sehr schwierig in einem Team eine Software zu entwickeln. Jeder Entwickler muss immer den aktuellen Stand der Quelltexte besitzen und müsste sich mit den anderen Entwickler absprechen, welche Dateien von wem bearbeitet werden. Eine gleichzeitige Bearbeitung einer Datei wäre nur mit großem Aufwand möglich, wie in Abbildung 3.1 auf der nächsten Seite beispielhaft zu sehen ist. Weiterhin ist es in Fehlerfällen schwierig auf ältere Projektstände zurückzugreifen. Da in der heutigen Zeit viele Projekte global, von mehreren Entwicklern unter Zeitdruck entwickelt werden, ist der Einsatz einer Versionsverwaltung zwingend notwendig. Eine Versionsverwaltung übernimmt dabei folgende Aufgaben:

- frühere Versionen reproduzierbar
- kontinuierliche Generationsfolge sicherstellen
- Platzsparende Speicherung
- Schutz vor unberechtigtem Zugriff
- Kostenreduzierung

Jeder Entwickler muss eine *Arbeitskopie* des Projektes vom Repository anfordern. Dieser Vorgang wird als *„Checking out“* bezeichnet und einmalig am Anfang er-

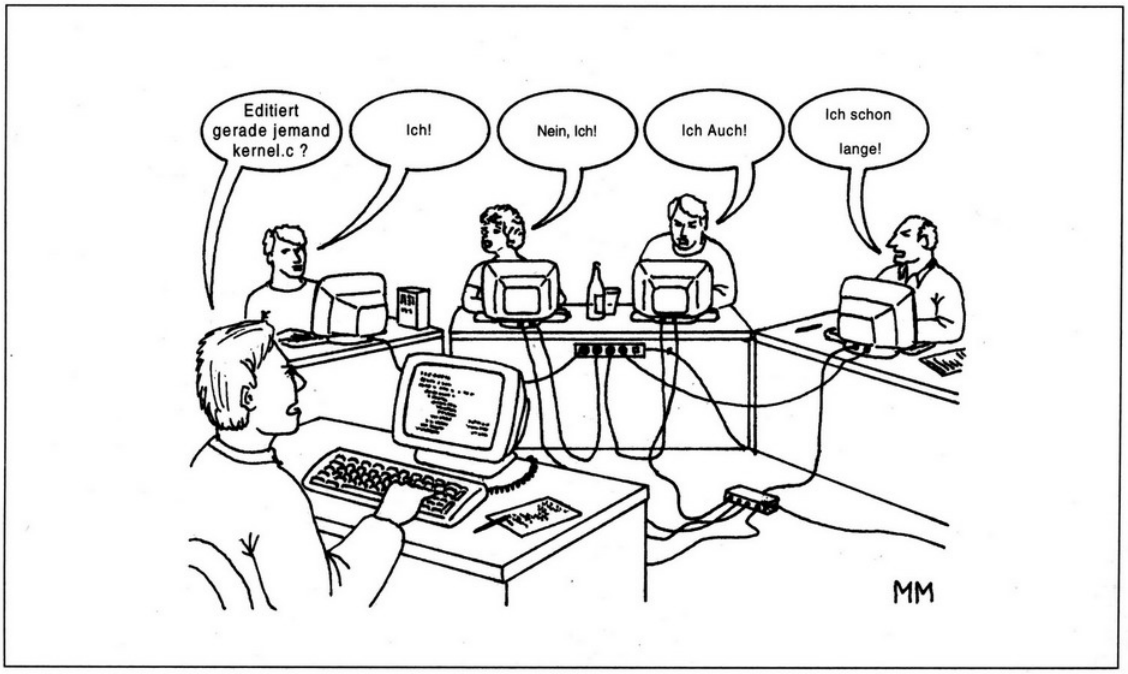

Abbildung 3.1: Softwareentwicklungs-Praxis [Herold u. Meyer 1995, Bild 2.1]

folgen. Danach aktualisiert er seine Arbeitskopie auf den aktuellsten Stand des Repository. Bei der Speicherung (*„Commit“*) der neuen Zusände der Quelltexte in dem Repository werden den einzelnen Dateien oder dem aktuellen Projektstand eine aufsteigende Nummer vergeben. Somit weiß der Entwickler, dass der sich gerade in *Revision* 13.2 befindet. Da jede Revision in dem Repository gespeichert wird, ist es dem Entwickler möglich, auf ältere Revisionen zurückzugreifen und somit ältere Dateien betrachten. Dies ist bspw. notwendig um Änderungen nach zu vollziehen bei eventuellen aktuellen Fehlern. Weiterhin müssen wichtige z.Z. nicht benötigte Teile des Quelltextes nicht auskommentiert werden, sondern können gelöscht werden. Somit können *frühere Versionen reproduziert* werden. Durch die automatische Vergabe der Revisionsnummern, wird weiterhin die *kontinuerliche Generationsfolge* sichergestellt. Der Entwickler ist gezwungen vor dem Speichern seiner Quelltexte in dem Repository, die aktuelle Revision aus dem Repository zu laden. Denn nur dadurch wird sichergestellt, dass nach der Speicherung der Revision 13.2 die Revision 13.3 folgt. Möchten bspw. zwei Entwickler eine Datei editieren und letztendlich zum Repository hinzufügen, müssen beide vor dem Speichern die aktuellste Version laden. Abbildung 3.2 auf der nächsten Seite veranschaulicht den dargestellten Sachverhalt. Dabei möchten sowohl *EntwicklerA* als auch *EntwicklerB* eine Datei *QuellcodeA.txt* bearbeiten. Zum gleichen Zeitpunkt aktualisieren beide ihre lokale

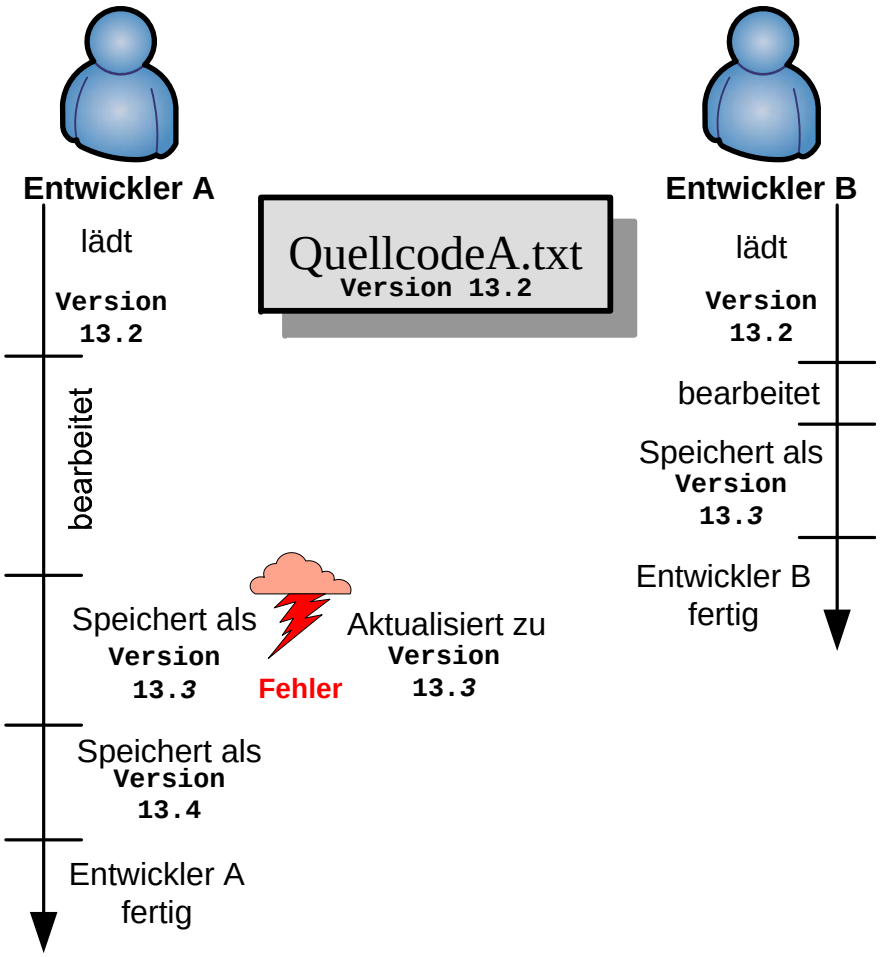

Abbildung 3.2: Kontinuierliche Entwicklung

Arbeitskopie zur aktuellen Revision *13.2* und beginnen die Datei zu verändern. EntwicklerB ist nach einer kurzen Zeit mit seinen Änderungen fertig und möchte diese im Repository speichern. Diese Aktion wird erfolgreich vom Versionsverwaltungsserver quittiert, indem die Revision 13.3 vergeben wird und somit die aktuelle Revision der Datei *QuellcodeA.txt* ist. EntwicklerB ist mit seiner Arbeit an dieser Datei später fertig. Nach einer Weile möchte EntwicklerA seine Änderungen im Repository speichern und versucht dies. Der Versionsverwaltungsserver wird dem EntwicklerA mitteilen, dass seine lokale Kopie mit der Versionsnummer 13.2 nicht mehr die aktuellste ist und zwingt den EntwicklerA zuerst die aktuelle Revision der zu speichernden Datei zu laden. Nachdem der EntwicklerA dies getan hat, muss er eventuelle Konflikte zwischen beiden Dateiversionen beheben und kann danach die Datei im Repository mit der Revisionnummer 13.4 speichern. Da im Laufe der

Bild 2.2: »Das neue Telefonbuch ist so dünn, weil es nicht mehr alle Namen enthält, sondern nur noch die Unterschiede zur Vorjahresausgabe.«

Abbildung 3.3: Speicherbedarf einer Versionsverwaltung [Herold u. Meyer 1995, Bild 2.2]

Projektzeit viele Änderungen an dem Projekt im Repository gespeichert werden, ist der Speicherplatz auf dem Server enorm. Der Versionsverwaltungsserver speichert jedoch nicht jede Revision komplett in seinem Repository. Die erste Revision einer Datei muss komplett gespeichert werden, für alle darauf folgenden Revisionsstände werden nur die Unterschiede zur Vorgängerversion gespeichert, siehe auch Abbildung 3.3. Der Versionsverwaltungsserver kann aus diesen Informationen jeden Revisionsstand wieder herstellen und bietet somit eine *platzsparende Möglichkeit zur Speicherung* der Revisionen. Ein Versionsverwaltungsserver bietet über verschiedene Sicherheitsmechanismen *Schutz vor unberechtigtem Zugriff.* Dabei muss sich jeder Entwickler über einen definierten Mechanismus authentisieren, meist anhand eines Benutzernamens mit einem dazugehören Passwort. Dazu werden abgestufte Zugriffsrechte für verschiedene Projektmitglieder auf einzelne Projektteile vergeben, meist anhand der Zuständigkeit der Projektmitglieder. Bei der Speicherung eines neues Versionsstandes im Repository wird immer der Benutzer mit gespeichert, damit ist es möglich, nachzuvollziehen, welcher Benutzer

welche Änderungen vorgenommen hat. Ein weitere wichtige Aufgabe einer Versionsverwaltung ist die *Kostenreduzierung*. Durch die bereits erwähnten Aufgaben einer Versionsverwaltung verringert sich der organisatorische Aufwand des Projektes, Fehler werden vermieden und ein weltweit aktives Team kann gleichzeitig an dem Projekt arbeiten [Fogel u. Bar 2002; Herold u. Meyer 1995; Rechenberger 1994; Wikipedia 2005].
Versionsverwaltungen können, wie andere Software Produkte heutzutage auch, in zwei verschiedene Gruppen eingeteilt werden: die kommerziellen Produkte und die Open-Source Produkte. Bei der kommerziellen Versionsverwaltungssystemen gibt es bspw. Microsoft's Visual SourceSafe[1] oder Borland's Starteam[2]. In den folgenden zwei Unterabschnitten werden zwei bekannte Open Source Versionsverwaltungen beschrieben.

3.1.1 Concurrent Versions System - CVS

Bei CVS handelt es sich um eine Versionsverwaltung mit einer Client-Server Architektur. Die Entwicklung von CSV begann mit Ansammlung von Shell-Skripten, die in einer Newsgroup 1986 von Dick Grune veröffentlicht wurden. Im Jahre 1989 begann Brian Berliner mit späterer Hilfe von Jeff Polk mit der eigentlichen Entwicklung von CVS. Das Concurrent Versions System ist ein Kommandozeilen orientiertes System, auf das mit Hilfe verschiedener Client-Interfaces zugegriffen werden kann. Im Laufe der Zeit hat sich CVS zu *der* Versionsverwaltung von Open-Source Projekten entwickelt und wird bspw. von sourceforge.net eingesetzt. Innerhalb des Archiv/Repository existieren mehrerer *Module*, die eine Sammlung von Dateien sind und dabei meist ein einzelnes Projekt darstellen. Dadurch, dass jede Datei eine eigene fortlaufende Revisionsnummer besitzt, wurde das *Tagging* eingeführt. Mit Hilfe des *Tagging* kann jeder Datei zu einem definierten Zeitpunkt ein einheitliches Tag vergeben werden. Zu einem Zeitpunkt X, besitzt bspw. `DateiA.txt` die Revision *1.13* und `DateiB.txt` die Revision *1.78*. Mit Hilfe eines Tag mit bspw. dem Namen *Alpha 0.18* erhalten beide Dateien dieses zusätzliche Tag, auf das zu einer späteren Version zurückgegriffen werden kann. Das Tagging bietet in der Softwareentwicklung eine gute Möglichkeit wichtige Abschnitte des Quellcodes zu markieren, jedoch kann es vorkommen, dass diese Abschnitte separat weiterentwickelt werden müssen. Dies ist z.B. der Fall, wenn eine Software ausgeliefert wird, an dieser jedoch weiterentwickelt wird. Dabei ist es notwendig zwei getrennte Entwicklungszweige fortzuführen. Einen Zweig um bei der ausgelieferten Version Fehler zu beheben und kleinere Folgeversionen zu veröffentlichen und einen Zweig um die Software weiter zuentwickeln. Diese Vorgehensweise wird als

[1] `http://msdn.microsoft.com/ssafe`
[2] `http://www.borland.com/starteam/`

Branching bezeichnet und hilft den Entwicklern bei der Entwicklung seinen Produktes auf unterschiedlichen Entwicklungszweigen. Der Hauptzweig eines Moduls wird als *trunk*[3] bezeichnet und besitzt die Revision *HEAD*. *Merging* bezeichnet das Zusammenführen zweier Entwicklungszweige zu einem einzigen Zweig. Dies kann notwendig sein, um bspw. den Wartungszweig einer ausgelieferten Software mit dem Tag *Version 1.0* mit dem Entwicklungszweig zur Version mit dem Tag *Version 2.0* zusammen zuführen. Mit Hilfe verschiedener *Trigger* können Skripte oder Programme im Repository ausgeführt werden. Die Trigger können vor und/oder nach dem Speichern der Dateien aufgeführt werden und dabei Einfluss auf das Speichern nehmen. So kann bspw. vor dem Speichern der Quellcode nach bekannten Bugs mit Hilfe verschiedener Programme[4] gesucht werden. Im Fehlerfall wird der Quellcode nicht gespeichert und der Entwickler erhält eine Fehlermeldung. Nach dem Speichern der Dateien könnte ein Skript eine automatische Email an alle Projektmitglieder schicken und diese über die gespeicherten Änderungen informieren. CVS hat jedoch einige Nachteile: Zum einen können Dateien nicht umbenannt werden. Um den Dateinamen zu verändern, muss diese Datei aus dem Modul entfernt und mit dem neuen Namen hinzugefügt werden. Dabei beginnt jedoch die Revision von vorne zu zählen und somit kann auch nicht auf die ältere Version zurückgegriffen werden. Zum anderen können auch Verzeichnisse nicht verschoben bzw. umbenannt werden. Um ein Verzeichnis zu verschieden, muss jede einzelne Datei in dem Verzeichnis aus dem Modul gelöscht und an neuer Stelle als neue Datei hinzugefügt werden. Diese Mißstände führten dazu, dass einige Softwareentwickler mit der Entwicklung einer neuer Versionsverwaltung mit dem Namen *Subversion* begonnen haben.[Wikipedia 2005; Fogel u. Bar 2002; Price 2005]

3.1.2 Subversion - SVN

„Subversion (SVN) ist eine Open-Source-Software zur Versionsverwaltung“, die oft als Nachfolger von CVS bezeichnet wird[Wikipedia 2005]. SVN ist ein von CVS unabhängiges Projekt und wird seit Anfang 2000 bei CoolabNet entwickelt. Seit dem 31.August 2001 ist Subversion „self-hosting“, d.h. der Quellcode zu Subversion wird mit Subversion verwaltet. Am 23.Februar 2004 wurde die erste stabile Version 1.0 veröffentlicht. Bei der Entwicklung von Subversion wurde auf eine ähnliche Bedienung wie CVS geachtet, damit der Umstieg von CVS auf SVN für viele Entwickler einfach ist. Weiterhin unterstützt Subversion fast alle Funktionen von CVS, versucht jedoch dessen Nachteile zu vermeiden. In Subversion hat nicht jede Datei eine eigene Revision, sondern das gesamte Projekt eine einheitliche Versionsnummer, ähnlich einem Tag in CVS. Änderungen an einem Projekt werden

[3] engl. Stamm

[4] Checkstyle – `http://checkstyle.sourceforge.net/`
Findbugs – `http://findbugs.sourceforge.net/`

meist an mehreren Dateien vorgenommen, die nun alle die gleiche Revision besitzen. Im Unterschied zu CVS benötigt Subversion neben der lokalen Arbeitskopie noch eine weitere zweite Kopie. Dadurch verdoppelt sich der benötigte Speicherbedarf bei dem Client, jedoch hat dies zum Vorteil, dass Subversion dadurch weniger Netzwerkzugriffe benötigt und bei einem *commit* nur die Unterschiede zur Vorgängerversion überträgt, nicht wie CVS die kompletten Dateien. Weiterhin ist das Umbenennen und Verschieben von Dateien möglich, die Verwaltung von Verzeichnissen und Metadaten. Die *commits* sind bei SVN atomar, d.h. Änderungen werden entweder komplett oder gar nicht ins Repository aufgenommen und Verbindungsabbrüche führen nicht zu inkonsistenten Zuständen. Der Zugriff auf den Subversionserver, der das Repository in einer Berkeley Datenbank oder seit Version 1.1 im Dateisystem speichert, kann über ein eigenes Protokoll, über die Secure Shell oder über ein WebDav-Modul für den Apach2 Webserver erfolgen. Abbildung 3.4 auf der nächsten Seite verdeutlicht nochmal die gesamte Subversion Architektur. Dabei ist zu sehen, dass über Kommandozeilenparameter oder einer grafischen Oberfläche auf die Client-Bibliothek zugegriffen werden kann. Diese Bibliothek kann über verschiedene Methoden auf das SVN Repository zugreifen. Dabei gibt es die direkte lokale Zugriffsart oder der Zugriff über ein Netzwerk/Internet. Das Repository speichert die Dateien bzw. die Unterschiede der Dateien letztendlich in einer Berkeley Datenbank oder auf dem Dateisystem. Subversion bietet somit fast die gleiche Funktionalität die CVS, behebt dessen Nachteile und bietet dazu noch einige Extrafunktionen. [Wikipedia 2005; Collins-Sussman u. a. 2005]

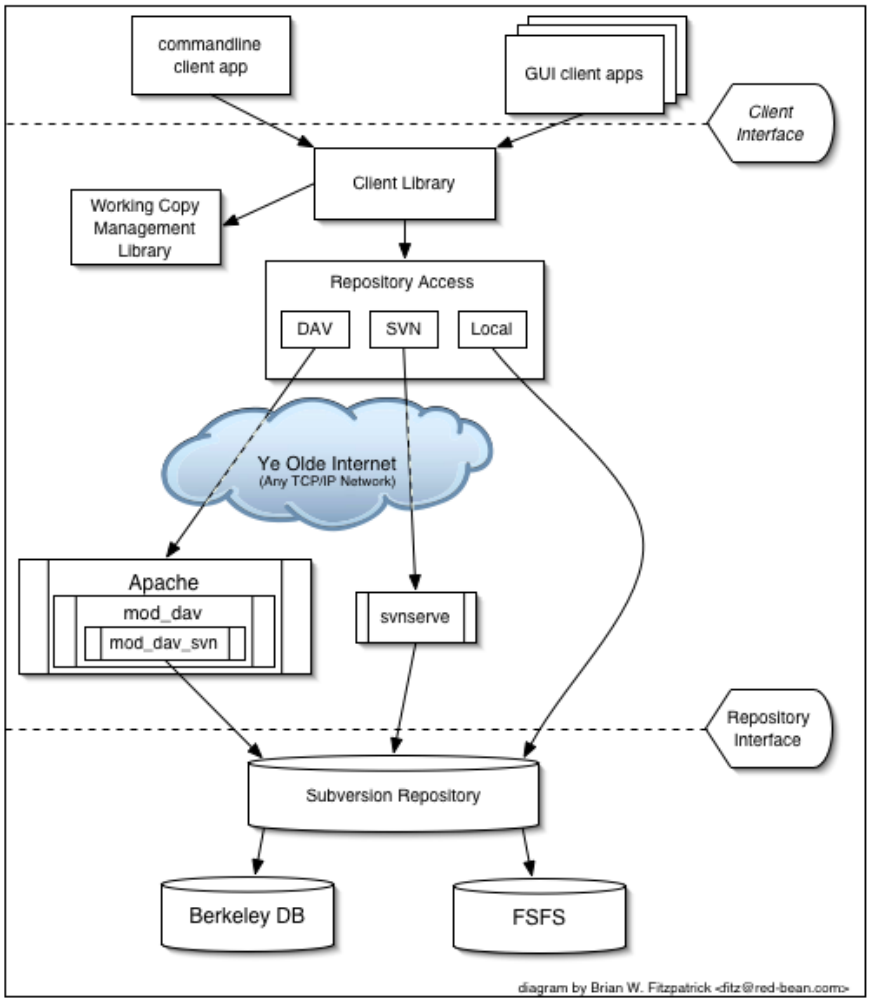

Abbildung 3.4: Subversion's Architektur [Collins-Sussman u. a. 2005, Kap. 1]

3.2 Entwicklungsumgebung

3.2.1 Eclipse

Eclipse ist eine Open-Source-Framework, das meist als Entwicklungsumgebung genutzt wird und auf 11 verschiedenen Systemen und Architekturen verwendet werden kann. Darunter befinden sich unter anderem Windows, Linux, MAC OS X, etc. Eclipse ist der Nachfolger von Visual Age for Java von IBM. Der Quellcode von Eclipse wurde 2001 von IBM freigegeben. Eclipse bildet nur den Kern, die Funktionalität wird über Plug-ins realisiert. Je nach verwendeten Plug-ins stellt sich somit die gesamte Funktionalität der Entwicklungsumgebung zusammen. Sowohl Eclipse als auch die Plug-ins sind in Java implementiert. Die Plug-Ins sind teilweise frei verwendbar und teilweise kommerziell. Dabei werden neben Java auch andere Sprachen wie zum Beispiel: C, C++. Perl und PHP unterstützt. Das dynamische Hinzufügen und Entfernen von Plug-ins und die daraus entstehende Gesamtfunktionalität der Eclipse-Entwicklungsumgebung bezeichnet man als Rich Client Platform, das wiederum auf den OSGi-Standard basiert.
Der OSGi-Standard (Open Services Gateway Initiative) ist ein Industriekonsortium, das ein Framework, für die Vernetzung von intelligenten Embedded-Geräten und das Bereitstellen von Diensten zur Laufzeit ermöglicht, entwickelt hat. Eclipse verwendet das OSGi-Framework außerhalb dieser Embedded-Ausrichtung.
Für die GUI-Erstellung wurde in Eclipse SWT verwendet, da SWT auf die nativen GUI-Komponenten des jeweiligen Betriebssystems zugreift ist Eclipse, trotz Java, nicht als plattformunbhängig zu betrachten.
Der Aufbau von Eclipse setzt sich dabei aus folgenden Subsystemen zusammen, die

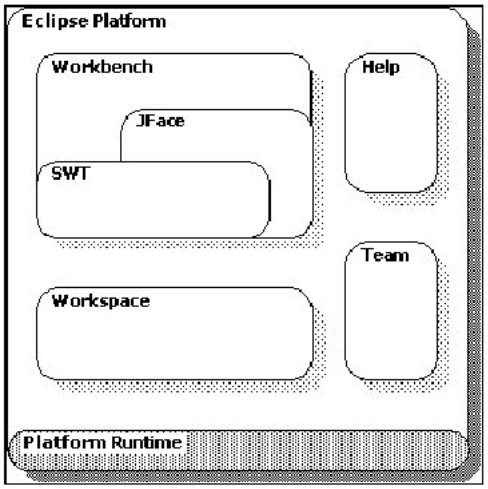

Abbildung 3.5: Aufbau der Eclipse-Plattform

teilweise aus einem oder mehreren Plug-ins bestehen wie in Abbildung 3.5 auf der vorherigen Seite zu sehen ist. Die einzelnen Subsysteme definieren Erweiterungs-Schnittstellen um deren Funktionalität der Plattform zugänglich zu machen, wie in Tabelle 3.1 auf Seite 26 zu sehen ist.

Tabelle 3.1: Subsysteme von Eclipse

Subsystem	**Funktion**
Plattform-Runtime	Definiert die Erweiterungs-Schnittstellen und das Plug-in Model. Es findet dynamisch Plug-ins und enthält Informationen zu den Plug-ins und deren Schnittstellen in einem Register. Die eingefügten Plug-ins werden gestartet, wenn der Anwender sie über die Plattform aufruft.
Resource Management	Definiert die API für das Erzeugen und Verwalten von Projekten, Dateien und Verzeichnissen, die von Tools erzeugt wurden und im Dateisystem abgelegt sind.
Workbench (UI)	Implementiert das Bedienfeld um mit der Plattform zu arbeiten. Definiert Erweiterungs-Schnittstellen für weitere UI-Komponenten, um vorhandene Funktionalitäten aus bereits bestehenden Ansichten und Menüs zu erweitern oder zu verwenden. Als Grundlage dienen dazu JFace und SWT, die im Kapitel SWT beschrieben sind.
Help System	Definiert Erweiterungs-Schnittstellen für Plug-ins, um eine Hilfe oder andere Informationen als durchsuchbare Bücher bereitzustellen.
Team Support	Definiert ein Team-Programmierungsmodell zum Verwalten von Ressourcen.
Debug Support	Definiert ein sprachunabhängiges Debug-Modell und UI-Klassen für das Erstellen von Debuggern und Launchers.
Other Utilities	Andere nützliche Plug-ins unterstützen Funktionen für das Vergleichen von Ressourcen, das Erzeugen von Konfigurationen über XML-Files und das dynamische Updaten der Plattform über einen Server.

[Wikipedia 2005, Eclipse] [Ecl 2005]

3.3 grafische Benutzerschnittstellen in Java

3.3.1 Abstract Window Toolkit - AWT

Die Abkürzung AWT steht für Abstract Window Toolkit. AWT stellt eine Standard-API für die Erzeugung und Darstellung von plattformunabhängigen Benutzerschnittstellen für grafische Oberflächen in Java zur Verfügung. AWT verwendet die nativen GUI-Komponenten des jeweiligen Betriebssystem zur Darstellung der grafischen Oberflächen. Da diese nativen GUI-Komponenten teilweise mit umfangreichen Betriebssystemressourcen verbunden sind, wird AWT auch als Heavyweight-Framework bzw. die einzelnen GUI-Elemenete als Heavyweight-Components bezeichnet.
Seit dem JDK (Java Development Kit) 1.0 steht AWT als Grafik-Bibliothek für die Anwender zur Verfügung. Dabei lässt sich AWT in 4 Gruppen unterteilen:

- Grafische Primitivoperationen zum Zeichnen von Linien oder Füllen von Flächen und zur Ausgabe von Text
- Methoden zur Steuerung des Programmablaufs über Nachrichten durch Tastatur-, Maus- oder Fensterereignisse.
- Portables Design für Dialogboxen zur Darstellung der Dialogelemente und die Kommunikation mit dem Anwender
- Fortgeschrittene Grafikfunktionen für die Bearbeitung und Darstellung von Bitmaps, sowie zur Ausgabe von Sound

Aufgrund der hohen Performance-Ansprüche von AWT, besonders für große GUI-Anwendungen, wurde es im JDK 1.1 von SUN massiv verändert. Dabei wurden viele Fehler behoben und auch viele Methodennamen verändert. Des Weiteren wurde das Event Handling-Modell, also der Transfer von GUI-Ereignissen, komplett überarbeitet. Dazu wurde AWT die Fähigkeit gegeben GUI-Ereignisse an beliebige Objekte zu übergeben und dort abzuarbeiten. Aufgrund dieser Fähigkeit erreicht man eine klare Trennung zwischen Benutzeroberfläche und Applikationslogik.[Krüger 2003]

3.3.2 Swing

Swing ist eine weitere Grafikbibliothek die im JDK 1.1 als Add-on verfügbar war und seit Version 1.2 fester Bestandteil des Java Development Kit ist. Swing wurde von SUN entwickelt um die grafische Benutzeroberfläche von Java Programmen zu verbessern, da man folgende Eigenschaften von AWT als nachteilig ansah:

- Aufgrund der Verwendung der jeweiligen nativen GUI-Elemente des eingesetzten Betriebssystems, war es sehr schwer ein einheitliches Look-an-Feel (Aussehen) zu realisieren.

- Um das Verhalten der verschiedenen GUI-Elemente der einzelnen Betriebssysteme annähernd anzugleichen war sehr viel Portierungsaufwand notwendig.
- Des Weiteren gibt es in AWT nur eine Grundmenge an Dialogelementen mit denen sich aufwendige grafische Oberflächen nur mit sehr viel Zeitaufwand realisieren lassen.

Das Ziel der Entwickler war es diese Nachteile mit Swing zu beseitigen.
Um einen Teil der Nachteile zu beseitigen, greift Swing nur noch eingeschränkt auf die jeweiligen nativen GUI-Elemente zu. Die restlichen GUI-Elemente werden von Swing selbst erzeugt und gezeichnet.

- Aufgrund dieser Vorgehensweise fällt ein Großteil der plattformspezifischen Besonderheiten weg.
- Des Weiteren entfällt die unterschiedliche Bedienung und Darstellung zwischen den verschiedenen Betriebssystemen, so dass ein Anwender auf allen Betriebssystemen dasselbe Aussehen vorfindet.
- Zusätzlich ist SWING nicht mehr angewiesen auf den kleinsten gemeinsamen Nenner der GUI-Elemente der Betriebsysteme aufzubauen, sondern kann unabhängig davon komplexere GUI-Elemente bereitstellen ohne auf das jeweilige Betriebssystem zu achten. Außerdem stehen in Swing dem Programmierer viel mehr Dialogelemente zur Verfügung als das bei AWT der Fall war.

Leichtgewicht-Komponenten

Die GUI-Elemente in Swing werden als Lightweight-Components bezeichnet, da sie nicht mehr über die betriebssystemspezifische Klassen erstellt werden müssen, sondern durch Komponenten-Klassen mit Hilfe von Primitivoperationen erzeugt werden.
Mit Hilfe der Leichtgewicht-Komponenten war es zusätzlich möglich, Tooltips (kleine Fenster mit nützlichen Informationen) unabhängig vom verwendeten Grafiksystem und einem Debug-Modus (Debug-Grafik) zum Finden von Fehlern während der Erzeugung von grafischen Oberflächen zu implementieren.

Pluggable Look-and-Feel

Eine der spektakulärsten Eigenschaften von Swing ist das Look-and-Feel der GUI-Elemente zur Laufzeit zu verändern. Dabei hat der Anwender die Möglichkeit zwischen einer bestimmten Menge an Look-and-Feels auszuwählen. Des Weiteren ist es möglich auch eigene Look-and-Feels zu erstellen.
Die Entscheidung, welches Look-and-Feel verwendet werden soll, muss dazu nicht während der Design-Phase entschieden werden. Man kann dem Anwender ein

Menü zur Verfügung stellen, indem er während der Laufzeit das ihm passende Look-and-Feel auswählt. Das Umschalten erfolgt praktisch ohne weiteren Aufwand des Programmes, es wird von einem User-Interface-Manager erledigt, der alle notwendigen Maßnahmen ausführt.

Das Model-View-Controller Prinzip

Beim Model-View-Controller Prinzip wird nicht die gesamte Funktionalität in eine Klasse gepackt sondern in drei Bestandteile zerlegt:

- Das Modell enthält die Daten des Dialogelementes und speichert seinen Zustand.
- Der View ist für die grafische Darstellung der Komponente verantwortlich.
- Der Controller wirkt als Verbindungsglied zwischen dem Modell und dem View und stößt bei Ereignissen die notwendigen Maßnahmen zur Veränderung von Modell und View an.

Eine wichtige Eigenschaft bei diesem Prinzip ist, dass ein Modell mehrere Views gleichzeitig haben kann. Dazu werden bei Veränderungen am Modell die Views benachrichtigt und aktualisieren sich entsprechend.
Bei Swing sind der View und der Controller in einer Klasse untergebracht, dadurch wird die Komplexität reduziert. Außerdem ist in den meisten Fällen der Controller so einfach strukturiert, dass es sich gar nicht lohnt ihn in einer zusätzlichen Klasse zu implementieren.
Die hier genannten Eigenschaften bringen aber auch Nachteile mit sich:

- Swing-Anwendungen sind ressourcenhungrig. Da alle Komponenten selbst gezeichnet werden müssen, erfordert es eine höhere CPU-Leistung und Hauptspeicher.
- Im JDK 1.2 gab es Speicherprobleme mit Swing-Elementen, da diese den Speicher, den sie belegten, nicht wieder komplett freigaben und sich dadurch der verfügbare Speicher mit der Zeit verringerte.
- Für Applet-Programmierer gab es keine Browser mit eingebauter Swing-Unterstützung, dies muss dem Anwender durch einen zusätzlichen Download (der enstsprechenden Bibliothek) und Installationsschritt zugemutet werden. Eine andere Alternative wäre auf AWT zurückzugreifen.

Aufgrund der steigenden Rechenleistung der PCs und der im JDK 1.3 verbesserten Swing-Grafikbibliothek konnten jedoch viele der Probleme behoben werden. [Krüger 2003]

3.3.3 Standard Widget Toolkit und JFace

SWT ist ebenfalls eine Grafikbibliothek für Java und steht für Standard Widget Toolkit. Im Gegensatz zu den anderen genannten Grafikbibliotheken ist SWT von IBM und nicht von SUN entwickelt wurden. SWT wurde im Jahr 2001 für die Entwicklungsumgebung Eclipse entwickelt. Im Gegensatz zu Swing verwendet SWT die nativen grafischen Elemente des Betriebssystems und schließt sich damit AWT an.
Des Weiteren ist SWT, im Gegensatz zu AWT und Swing, nicht als plattformunabhängig zu betrachten. Dies resultiert daraus, dass SWT die entsprechende Bibliothek für den betriebssystemabhängigen Code für die GUI-Elemente zur Laufzeit benötigt. Zusätzlich werden die Java-Klassen der SWT-Bibliothek benötigt, die auf die betriebsystemabhängige Bibliothek zugreifen.
Leider leidet SWT aufgrund des eben beschriebenen Sachverhaltes auf einigen nicht Windows-Plattformen an Effizienzproblemen, da bestimmte Funktionalitäten eventuell nicht verfügbar sind und dadurch emuliert werden müssen. Aufgrund des nicht vorhersagbaren Zeitpunktes an dem nicht mehr verwendete Objekte durch den Garbage Collector freigegeben werden, hat IBM implementiert, dass alle erstellten Objekte selbst freigegeben werden müssen. Durch diese Vorgehensweise wird sicher gestellt, dass der komplette Speicher, den ein SWT-Objekt belegt, wieder freigegeben wird. Dies bedeutet für einen Programmierer zusätzlichen Aufwand, da der Programmierer dafür verantwortlich ist, erzeugte Objekte wieder freizugeben.
JFace ist ein UI-Toolkit und setzt aus den gelieferten Basiskomponenten von SWT komplexere Widgets zusammen. JFace kann daher überall eingesetzt werden, wo auch SWT zur Verfügung steht.[Wikipedia 2005, SWT] [Ramachandran 2003]

Krueger2003 SWTBasic EclipseISV

3.4 Apache Struts

Struts basiert auf dem Model 2 (MVC 2) und ist ein leistungsstarkes und frei erhältliches Framework. Hierbei wird die Präsentationsebene durch JavaServer Pages konstruiert, der Model-Teil durch JavaBeans übernommen und Servlets als Controller dieses Frameworks eingesetzt. Das Struts Framework Projekt wurde nach [Apa 2005] im Mai 2000 von Craig R. McCalahan ins Leben gerufen und versuchte die Vorteile von Java Servlets und JavaServer Pages zu vereinen. Zu Beginn wurde ein Model-View-Controller Framework für die Java-Welt erarbeitet, welches im Juli 2001 von der Apache Software Foundation (ASF) unter dem Namen Struts 1.0 veröffentlicht wurde. Die ASF ist eine ehrenamtlich arbeitende Organisation zur Förderung der Apache-Softwareprojekte. Sie entstand im Juni 1999 aus der Apache Group und soll den Apache Open-Source-Software Projekten organisatorische, juristische und finan-zielle Unterstützung zur Verfügung stellen. Die ASF ist eine nicht kommerzielle Organisation aus Entwicklern, die an Open-Source-Softwareprojekten arbeiten. Auch Struts ist ein Open-Source Projekt und unterliegt der Apache Software License[5].

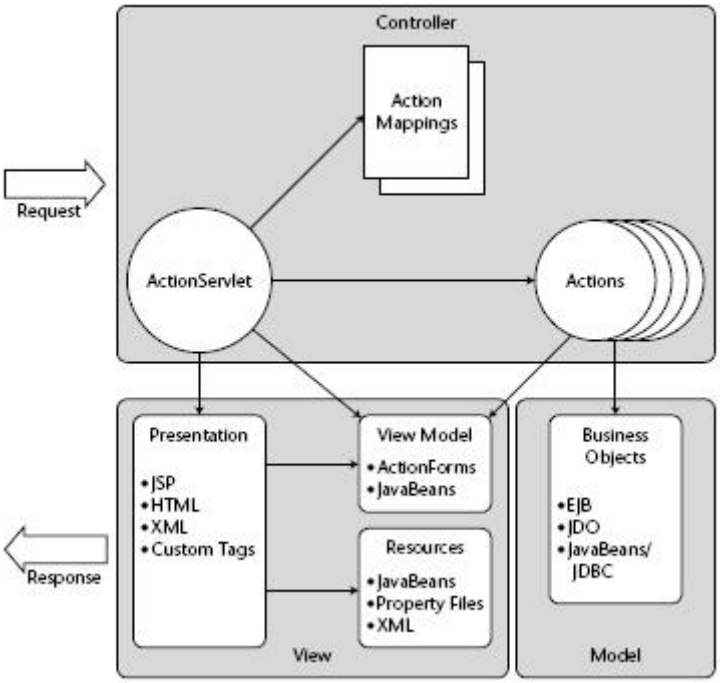

Abbildung 3.6: Architektur von Struts [Dudney u. a. 2004, Bild 1.5]

Wie bereits erwähnt, verfolgt Struts das Model2-Konzept und lässt sich somit

[5] `http://www.apache.org/licenses/`

in die drei Komponenten (Model, View und Controller) unterteilen. Abbildung 3.6 auf der vorherigen Seite gibt einen Überblick der Architektur von Struts, welche diese Komponenten vereint. Diese einzelnen Komponenten werden in den folgenden Abschnitten detailliert beschrieben.

Die Strutskomponente Model

Das Model von Struts wird durch JavaBeans implementiert. Je nach Funktionalität können sie in drei Kategorien unterteilt werden:

- Beans für den Systemzustand
- Beans für die Anwendungslogik
- ActionForm-Beans

Die Beans für den Systemzustand repräsentieren die Zustandsinformationen des Systems über deren Attribute. Der interne Zustand der Anwendung wird somit durch eine oder mehrere Beans und den Attributen dargestellt. Im Beispiel der Provirent-Anwendung lässt sich der Einkaufskorb durch ein Bean darstellen, da es beinhaltet, was ein Kunde für seine Bestellung ausgewählt hat. Zur Darstellung des aktuellen Zustands können die zugehörigen „get“- und „set“-Methoden aufgerufen werden. Zum Beispiel kann die Anzahl der ausgewählten Artikel, die durch das Attribut anzahlArtikel dargestellt wird, über den Aufruf der Methode getArtikelAnzahl() abgefragt werden.
Die Anwendungslogik kann durch JavaBeans ergänzt werden. Damit eine Wiederverwendung der Anwendungslogik gewährleistet werden kann, sollten die JavaBeans möglichst so implementiert werden, dass sie unabhängig von der Umgebung der Anwendung ausgeführt werden können. Zum Beispiel sollte die Logik zum Speichern von Bestellungen in die Datenbank ausgelagert werden. Hierbei müssen die Funktionalitäten für den Zugriff auf die Datenbank korrekt implementiert werden. Diese Methoden der JavaBeans können dann sowohl in der Struts-Applikation als auch in anderen Umgebungen, wo Datenbankzugriffe gebraucht werden, aufgerufen werden.
Die ActionForm-Beans dienen zur Behandlung eines Formulars einer Webanwendung. Für jedes Eingabeformular ist ein entsprechendes ActionForm-Bean vorgesehen. Dieses ermöglicht das Zwischenspeichern von Formulareingabedaten, wobei jedes Eingabefeld einem Attribut der Bean entspricht. Mit der Verwendung von ActionForm Beans ist es möglich, auf die Daten des Formulars in verschiedenen Bereichen der Anwendung zuzugreifen. Eine solche Bean ist von der Klasse *ActionForm* abgeleitet und kann neben den „get“- und „set“-Methoden optional noch zwei spezielle Methoden besitzen: *validate*() und *reset*(). Die Methode validate() dient dazu, Eingabedaten aus dem Formular zu validieren. Die Attribute eines

Formulars lassen sich durch die Methode reset() zurücksetzen. Aufgrund der zwingend erforderlichen Namensgleichheit zwischen den Formularelementen und den Attributen der ActionForm Beans wird die Kommunikation zwischen dem Bean und dem HTML-Formular sichergestellt.

Die Strutskomponente View

Die View ist für die Präsentation der Daten zuständig, die meist durch JavaServer Pages umgesetzt wird. Die JSPs können neben HTML, XML und JavaScript, auch zur Laufzeit dynamisch generierten Code enthalten. Da es möglichst vermieden werden sollte, Java-Code in die JSP einzubauen werden so genannte Tags eingesetzt. Die JSP-eigenen Tags werden durch die umfangreichen Struts Tag-Bibliotheken (Taglibs) erweitert, wodurch eine größere Funktionalität erreicht werden kann. Wie in Quellcode 3.1 zu erkennen ist, wird jeder einzelnen Bibliothek ein Präfix zugeordnet, damit der Compiler diese auseinander halten kann.

```
1 <\%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" \%>
2 <\%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" \%>
3 <\%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" \%>
4 <\%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" \%>
```

Quellcode 3.1: Integration der StrutsTaglibs

Neben diesen Bibliotheken lassen sich auch noch eine Reihe anderer Tag-Bibliotheken, wie die Java Standard Tag Library (JSTL), integrieren. Wenn die Funktionalitäten dann immer noch nicht ausreichen sollten, besteht auch die Möglichkeit, eigene Bibliotheken zu erstellen und zu benutzen. Mit der Verwendung solcher Taglibs wird nach [Cavaness 2004] für eine deutliche Senkung der Entwicklungszeit und die daraus resultierende Steigerung der Produktivität erreicht. Auch die Fehlerbehandlung und die Kommunikation mit den ActionForm Beans wird dadurch vereinfacht.
Darüber hinaus ist man in der Lage, die Web-Anwendung in mehreren Sprachen zu unterstützen. Dabei muss die JSP nicht in mehreren Sprachen auf dem Server hinterlegen werden, sondern es können hierfür die vom Struts-Framework bereitgestellten Message-Tags verwendet werden.

```
1 <title>
2   <bean:message key="provirent.title"/>
3 </title>
```

Quellcode 3.2: Struts Message-Tag

An dem Präfix *bean* in Quellcode 3.2 ist zu erkennen, dass dieses Tag der *struts-bean.tld* Bibliothek angehört. Das Attribut *key* verweist auf ein Element einer zentral definierten Datei, in der alle Texte einer Sprache enthalten sind. Es handelt sich

hierbei um eine Properties-Datei mit Key-Value Paaren, die für jede der unterstützten Sprachen vorliegt. Die Schlüssel werden in den Message-Tags angegeben und verweisen auf die entsprechenden Texte (Values) der jeweiligen Sprache, die in jeder dieser Properties-Dateien enthalten sind. Dies hat den Vorteil, dass keine Texte mehr in den Quellcode der JSP geschrieben werden müssen, sondern durch den Einsatz der Message-Tags in der gewünschten Sprache zu Laufzeit eingefügt werden. Der Name und der Pfad der Properties-Datei sind frei wählbar, müssen aber in der Konfigurationsdatei des Frameworks definiert werden. Es muss darauf geachtet werden, dass eine sprachenabhängige Properties-Datei der Namenskonvention entspricht, d.h. im Format `<name_der_Message_Datei>_xx.properties`, wobei das *xx* für den zweistelligen Code der jeweiligen Länder bzw. Sprachen steht (z.B. *en* für Englisch, d*e* für Deutsch).
Für die Interaktion von JSPs mit den ActionForm Beans wurden die Standard HTML-Tags in den Struts Taglibs um gewisse Funktionalitäten erweitert. Prinzipiell kommen alle wichtigen HTML-Elemente darin vor. Zum Beispiel lässt sich mit Hilfe des Tags `<html:form\>` ein HTML-Formular erstellen. Über das Pflichtattribut *action* wird angegeben, wie nach dem Absenden der Formulardaten damit verfahren werden soll. Hierbei kann eine URL angegeben werden. Es ist aber gängiger, keine Web-Ressourcen, sondern eine Action-Klasse aufzurufen, was durch die Endung **.do* gekennzeichnet wird und dem Controller mitteilt, in der struts-config.xml nach dem entsprechenden Pfad zu suchen. Ein solcher Pfad stellt die Verknüpfung von ActionForm Beans und Action Klassen der JSP dar. Anhand dieser Zuordnungen können die Formulardaten der ActionForm Bean zugewiesen und validiert werden.
Wenn durch die *validate*()-Methode ein Fehler festgestellt wird, kann dem Benutzer eine entsprechende Fehlermeldung über den Tag `<html:errors\>` mitgeteilt werden. Dieser Tag wird nur aktiv, wenn die mit dem Attribut property definierte Fehlermeldung während der Validierung erzeugt wurde.
Zusätzlich zu den oben genannten Tags der Struts-Taglibs stehen dem Struts-Framework noch eine Reihe weiterer Tags zur Verfügung. Es können auch die Taglibraries der JSTL Spezifikation[6] eingesetzt werden.

Die Strutskomponente Controller

Der gesamte Ablauf einer Struts-Anwendung wird über den zentral im Struts-Framework vorliegenden Controller gesteuert. Dessen Aufgabe ist es, HTTP-Requests vom Client zu empfangen, diese auszuwerten und zu entscheiden, welche Maßnahme als nächstes durchgeführt werden soll. Wenn z.B. kein Verarbeitungsschritt mehr notwendig sein sollte, so wird die Anfrage direkt an die JSP weitergeleitet,

[6] `http://java.sun.com/products/jsp/jstl/index.jsp`

ansonsten an die spezifische Action-Klasse. Besonders vorteilhaft ist nach [Cavaness 2004] die an einem zentralen Punkt liegende Ablaufsteuerung der Anwendung durch den Controller. Bei daran notwendigen Änderungen muss nicht die ganze Anwendung sondern nur ein relativ kleiner Bereich des Programms angepasst werden. Der Controller umfasst folgende Komponenten:

- die Klasse ActionServlet
- die Datei struts-config.xml
- die Klasse ActionMapping
- verschiedene Klassen, die sich von der Klasse Action ableiten

Jede Anwendung enthält genau ein ActionServlet, das alle Requests des Benutzers verarbeitet. Dieses Servlet sucht in der Struts-Konfigurationsdatei nach der Action für den gerade zu bearbeitenden Request. Darüber hinaus erzeugt und verwendet es ActionForm Beans für das Speichern und Validieren von Daten aus HTML-Formularen und ActionForward Klassen für die Fortsetzung des Programmflusses. Das Servlet wird von der Klasse *org.apache.struts.action.ActionServlet* abgeleitet und wird in der Konfigurationsdatei *web.xml* registriert.
Bei einem Request an die Web-Applikation, dessen URL mit *.do endet, schaltet sich das Struts-Framework ein, indem das ActionServlet die Anfrage an die in der URL angegebene Action weiterleitet. Dies geschieht über ActionMappings, welche in der *struts-config.xml* angegeben sind.
Ein solches Mapping wird für das Abbilden von speziellen Ereignissen auf die zuständigen Action-Klassen benötigt, was in Struts durch Einträge in die XML-Datei realisiert wird. Der Ablauf der Anwendung kann somit sehr einfach verändert werden, da lediglich eine Datei angepasst werden muss. Dadurch wird die Datei struts-config.xml zum tatsächlichen Mittelpunkt des Frameworks. Ein weiterer Vorteil der Zentralisierung der ActionMappings liegt darin, dass der Ablauf einer Applikation besser zu verstehen ist, wenn dieser nicht im Quellcode versteckt ist. In Quellcode 3.3 wird eine solche Konfiguration dargestellt.

```
1  <struts-config>
2
3    <form-beans>
4
5      <form-bean name="LoginForm"
6          type="de.hsharz.provirent.customer.form.LoginForm"/>
7
8    </form-beans>
9
10   <global-forwards>
11
12     <forward name="index" path="/index.do"/>
13     <forward name="login" path="/Login.do"/>
14     <forward name="logout" path="/Logout.do" />
15
16   </global-forwards>
17
```

```
18    <action-mappings>
19
20      <action path="/index" forward="provirent.index"/>
21
22      <action path="/Login" input="vkb.scharf.admin.login"
23          type="de.hsharz.provirent.customer.action.LoginAction" name="LoginForm"
24          validate="true"/>
25
26    </action-mappings>
27
28    <message-resources parameter="MessageResources"/>
29
30  </struts-config>
```

Quellcode 3.3: Auszug aus struts-config.xml

Im Bereich `<form-beans>` wird die Zuordnung der HTML-Formulare zu den entsprechenden ActionForm Beans vorgenommen. Über das Attribut *name* wird dem ActionForm eine Bezeichnung gegeben, unter der es aufgerufen bzw. initialisiert werden kann, während *type* den vollständigen Namen der zugehörigen Java-Klasse angibt.
Der Abschnitt `<global-forwards>` ist dazu notwendig, logische Namen bestimmten URLs (JSPs oder Actions) zuzuordnen. Der Entwickler hat dann die Möglichkeit, im Quellcode diese Namen mit dem Vorteil anzusprechen, dass Änderungen an den URLs nur noch in der Konfigurationsdatei vorgenommen werden müssen.
Mit `<action-mappings>` wird die Zuordnung bestimmter Request-URLs auf die Action-Klassen definiert. Dabei wird für jedes Ereignis ein eigenes `<action>` Element angelegt. Dessen Attribut *path* gibt die URL der Action an. Nach Quellcode 3.3 auf der vorherigen Seite wird z.B. das ActionMapping */login* durch den Aufruf der URL http://localhost:8080/customer/Login.do aufgerufen. Über das Attribut *type* wird der vollständige Pfad der Action-Klasse angegeben, während *name* den Namen der zugeordneten ActionForm Bean festlegt. Ein weiteres Attribut *input* gibt die URL einer JSP oder Action an, die für die Auslösung des ActionEvents verantwortlich ist.
Mit dem Element `<forward>` definiert man, ähnlich wie bei den Global Forwards, eine mögliche URL, die nach dem Ausführen der Action-Klasse aufgerufen werden kann. Ein solcher Forward wird einer bestimmten Action zugeordnet, somit kann sie auch nur aus dieser Action-Klasse aufgerufen werden. Der Vorteil der Action-Mappings ist, dass man jede beliebige Action-Klasse aufrufen und an jede JSP oder andere Action weiterleiten kann.
Action-Klassen sind die Schnittstellen zwischen den Benutzeranfragen und der Geschäftslogik. Bei deren Erstellung spielen nach [Goodwill 2004] folgende Faktoren eine Rolle:

- Sie wird von der Klasse *org.apache.struts.action.Action* abgeleitet.
- Sie implementiert die *execute*()-Methode und enthält die spezifische Logik.

- Die neue Action muus kompiliert werden und sich danach in dem Verzeichnis der Web-Applikation befinden (meistens */WEB-INF/classes*).
- Es muss ein neues `<action>` Element in der Konfiguration erstellt werden, dass die neue Action abbildet.

Die *execute*()-Methode einer Action-Klasse wird aufgerufen, sobald diese die Kontrolle erhalten hat. Nach [Goodwill 2004] wird daraufhin die benutzerdefinierte Geschäftslogik ausgeführt und die Anfrage weitergereicht. Innerhalb dieser Methode werden alle Operationen, die zum Durchführen des Requests nötig sind, durchgeführt. Nach [Wolff 2004] und [Robinson 2004] sind das hauptsächlich:

- Authentifizierung des Userstatus (Zugriff auf die Session)
- Validierung der Formulareingaben
- Zugriff auf die Geschäftslogik
- Aufbereitung der Ergebnisse aus der Geschäftslogik, so dass diese angezeigt wer-den können, und Aktualisieren der ActionForm
- Rückgabe eines ActionForward-Objektes, das die nachfolgende Aktion angibt

Bei der Erstellung von Action-Klassen muss darauf geachtet werden, dass diese thread-safe und reentrant (wiedereintrittsfähig) sind. Sie sollten also in einer Multi-Thread Umgebung richtig arbeiten können. Aus diesem Grund sollten nur lokale Variablen und Methoden zum Einsatz kommen. Darüber hinaus kann es zu Skalierungsproblemen kommen, wenn für jeden Benutzer Ressourcen, wie z.B. Datenbank-Verbindungen, erzeugt und verwendet werden. Solche Ressourcen sollten über Pools vergeben werden.

3.5 Hibernate

Hibernate ist ein Open Source-Persistenz-Tool, das basierend auf so genannten Mappings das Bindeglied zwischen JavaBeans und einer Datenbank darstellt. Seit September 2003 gehört das Hibernate-Projekt zur JBoss Group und liegt in der aktuellen Version 3.0 kostenlos zum Download[7] bereit. Zur Zeit werden 16 Datenbanken unterstützt, worunter unter anderem Oracle, DB2, MySQL sowie PostgreSQL zählen. Zu den weiteren Besonderheiten zählen die Hibernate Query Language, Native SQL Queries sowie Lazy- und Outer-Join Fetching zur Steigerung der Performance. Ferner lässt sich Hibernate problemlos in alle bekannten J2EE Application Server integrieren.
Hibernate stellt den Entwicklern ein umfangreiches Werkzeug für die Realisierung einer leistungsfähigen Persistenzschicht zur Verfügung. Hierbei werden grundlegende Mechanismen für das Laden, Speichern, Aktualisieren und Löschen von Java-Objekten, sowie deren Beziehungen, bereitgestellt.
Das Abbilden von Java-Objekten auf eine entsprechende Datenbank erfolgt auf einem äußerst flexiblen Weg, da sich diese Java-Klassen und die entsprechenden Konfigurationsdateien sehr einfach aus einem bestehenden Datenbankschema generieren lassen. Auch der umgekehrte Weg (Top-Down), d.h. die Generierung eines Datenbankschemas aus bestehenden Java-Klassen, lässt sich einfach realisieren.

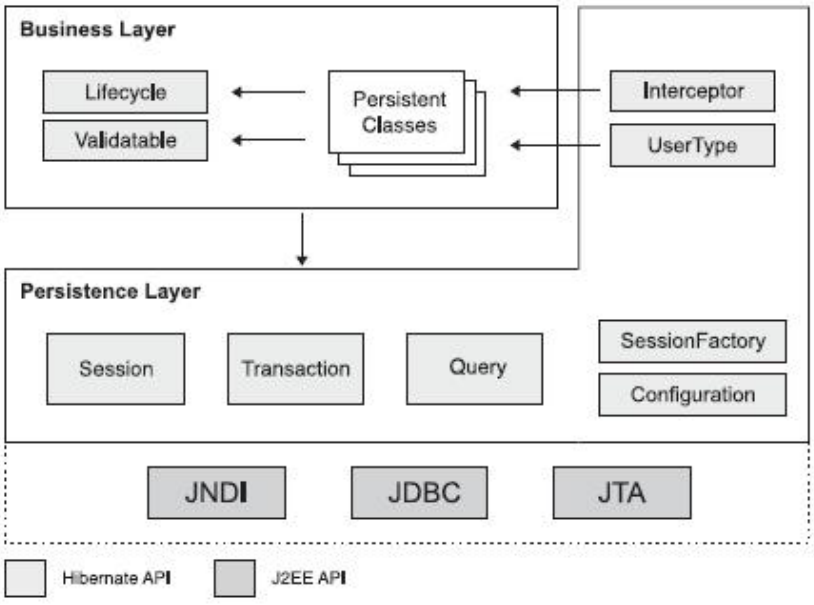

Abbildung 3.7: Architektur von Hibernate [Bauer 2004, Bild 2.1]

[7] `http://www.hibernate.org`

Abbildung 3.7 auf der vorherigen Seite stellt die Rollen der wichtigsten Schnittstellen der Business- und Persistenzschicht von Hibernate dar. Dabei agiert die Businessschicht als ein Client der Persistenzschicht. In manchen Anwendungen werden diese beiden Schichten aber auch nicht getrennt dargestellt. Hibernate ermöglicht nach [Bauer 2004] auch die Verwendung von bestehenden Java APIs, wie z.B. JDBC[8], JTA[9] oder JNDI[10]. JDBC bietet abstrakte Funktionalitäten analog zu relationalen Datenbanken und erlaubt es, fast jede Datenbank über einen JDBC Treiber mit Hibernate verwenden zu können. JNDI und JTA ermöglichen Hibernate die Integration in J2EE Applikationsservern.

Über XML-basierte Mapping-Dateien wird das objektrelationale Abbilden der Java-klassen für Hibernate-Anwendungen definiert. Eine solche Mapping-Datei wird mit dem Dateinamen-Suffix *.hbm.xml* versehen und wird generell für jede persistente Klasse erzeugt. In Quellcode 3.4 wird das Prinzip der Mapping-Dateien dargestellt.

```
1  <hibernate-mapping>
2    <class name="de.hsharz.provirent.objects.Bill" table="BILL">
3
4      <id name="billId" type="int" column="BILLID">
5        <meta attribute="scope-set">public</meta>
6        <meta attribute="use-in-equals">true</meta>
7        <generator class="native"/>
8      </id>
9
10     <many-to-one name="customer" class="de.hsharz.provirent.objects.Customer">
11       <meta attribute="use-in-tostring">true</meta>
12     </many-to-one>
13
14     <property name="pdfFile" type="binary">
15       <column name="pdffile" sql-type="BLOB" />
16     </property>
17
18     <property name="pdfFileSize" type="int">
19       <meta attribute="use-in-tostring">true</meta>
20     </property>
21
22   </class>
23 </hibernate-mapping>
```

Quellcode 3.4: Mapping-Datei von Hibernate

In den Mapping-Dateien wird die Zuordnung der einzelnen Attribute (Properties) zu den entsprechenden Tabellenspalten der zugrunde liegenden Datenbank und auch Beziehungen zu anderen persistenten Java-Klassen (Relationen) festgelegt. Die folgende XML-Elemente sollten dabei verwendet werden:

8 Java Database Connectivity
9 Java Transaction API
10 Java Naming Directory Interface

- class: Name der Java-Klasse und deren Zuordnung zur korrespondierenden Tabelle der Datenbank
- id: Attribut(e) der Klasse für den Primärschlüssel
- property: Zuordnung der einzelnen Spalten der Datenbanktabelle zu den Properties der Java-Klasse mit zusätzlichen Angaben über den zu mappenden Datentyp und das Erlauben von Null-Werten
- many-to-one: Darstellung einer n:1 Beziehung mit Zuordnung der Spalte aus der Datenbanktabelle zu einer entsprechenden Property und Angabe des Objekttyps der Beziehung

Außer diesen gibt es noch weitere Attribute. Über deren Bedeutung informieren die Hibernate-Webseiten.
Standardmäßig wird Hibernate über ein zentrales XML-Dokument konfiguriert. Der Name dieser Datei wird meist mit der Endung .cfg.xml gebildet. Darin werden solche Konfigurationen wie Deklaration der Datenbankverbindung, des Hibernate-Dialektes (abhängig vom verwendeten DBMS), sowie zusätzlichen Optionen festgelegt. Ferner können auch die Ressourcen der einzelnen Mapping-Dateien angegeben werden, um diese der Java-Applikation bekanntzumachen.
Vor der Verwendung von Hibernate als Persistenzmechanismus in einer Anwendung muss die Hibernate-Umgebung initialisiert werden. Hierbei wird die Klasse *SessionFactory* in der Geschäftslogik der Anwendung geladen. Mit Hilfe dieser Klasse lässt sich eine Session-Instanz erzeugen, die als ein Bindeglied zwischen der Datenbank und der Anwendung fungiert. Nur über diese Session ist die Interaktion mit den Datenbankobjekten möglich. Darunter sind die so genannten „CRUD-Methoden“ (create, retrieve, update, delete) oder Queries (Abfragen mit HQL[11]) zu verstehen. Quellcode 3.5 illustriert das Speichern des persistenten Objekts *Dvd* in die Datenbank.

```
1 try {
2   Dvd dvd = (Dvd) session.save(new Dvd());
3 } catch (HibernateException e) {
4   logger.error("Objekt konnte nicht gespeichert werden", e);
5 }
```

Quellcode 3.5: Save-Methode der Hibernate-API

Hibernate besitzt auch eine Transaktionsschnittstelle. So lassen sich über eine Transaktions-Instanz, die über das Session-Objekt erzeugt werden kann, Transaktionen durch geeignete Methoden (*begin*/*commit*/*rollback*) abgrenzen. Diese Schnittstelle ist insofern erweiterbar, dass sie leicht mit anderen Systemen integriert werden kann. Weiterhin stehen dem Entwickler als Transaktionsstrategien sowohl op-

[11] Hibernate Query Language

timistisches als auch pessimistisches Locking zur Verfügung.
Ein weiteres Merkmal von Hibernate ist die Möglichkeit der automatischen Generierung von Primärschlüsseln, wobei an die 10 verschiedenen Möglichkeiten, wie z.B. Sequenzen, im Vordergrund stehen.
Darüber hinaus stellt Hibernate verschiedene Abfragesprachen zur Verfügung. Hierbei sind die Hibernate Query Language (HQL), Query By Criteria und Query By Example zu nennen. HQL ist an SQL angelehnt, beherrscht aber auch objektorientierte Konzepte wie Vererbung und Assoziationen. Die Anfragen werden dabei in Zeichenketten abgelegt und Hibernate übergeben. Dieses Konzept lässt sich in den Referenz-Dokumenten[12] von Hibernate genauer betrachten. Bei der Verwendung von Query By Criteria werden keine Zeichenketten benutzt, sondern eine Anfrage setzt sich aus einzelnen Ausdrücken zusammen, die zu einer so genannten *CriteriaQuery* hinzugefügt werden. Demnach wird die Syntax der Abfragen bereits zur Übersetzungszeit durch den Compiler und nicht erst zur Laufzeit überprüft. Die Query By Example Schnittstelle nutzt das Konzept von Query By Criteria. Hierbei wird eine mit entsprechenden Suchdaten versehene Beispielklasse einer *CriteriaQuery* übergeben, woraufhin diese alle Klassen zurückliefert, die den Eigenschaften der übergebenen Klasse entsprechen.

[12] siehe [Hib 2005]

Kapitel 4

Implementierung

4.1 Versionsverwaltung mit Subversion

Als Versionsverwaltung wurde Subversion verwendet. Dies hat mehrere Gründe: Zum einen war Subversion zu diesem Zeitpunkt eine neue und z.T. auch unbekannte Technologie, die somit ihren Reiz hatte. Zum anderen wurde Subversion als Nachfolger von CVS angepriesen und sollte viele Nachteile eliminieren. Einer der wichtigen Vorteile von Subversion ist die geringe Netzwerklast. Bei einem „Commit“ werden nur die Unterschiede zur Vorgängerversion übertragen und nicht wie bei CVS jede Datei komplett. Um Subversion verwenden zu können, wurde ein Rechner mit einem installieren Subversion Server benötigt, der möglichst 24h im Internet verfügbar ist. Um mit einem Client auf einen Subversion Server zuzugreifen, gibt es verschiedene Übertragungsprotokolle: Ein eigenes Subversion protokoll (svn), eine Kombination aus Secure Shell und dem eigenen Protokoll (ssh+svn), über http oder über https. Die sicherste Methode ist über ssh+svn, jedoch erfordert diese die Installation eines SSH Server, was unter Linux, dank verschiedener Anleitungen, einfach geht, jedoch unter Windows nicht so einfach zu realisieren ist. Unter Windows wurde zu Testzwecken ein Apache2 Webserver mit einem Zertifikat zur Übertragung von Daten per https installiert. Dieser Webserver wurde mit dem Subversion Server kombiniert und jeglicher Datentransfer für Subversion erfolgte über https und dessen eingestellen Port. Der Nachteil war jedoch, dass keiner die Möglichkeit hatte, einen PC 24h online zur Verfügung zu stellen. So wurde nach einiger Suche im Internet der Open Source Anbieter Berlios Developer[1] entdeckt. Dort können Open-Source Projekte ihre Quellcodes in einer Subversion Versionsverwaltung mittels ssh+svn kostenlos speichern. Bei Berlios ist das Repository bereits erstellt, bei unseren Testsystem musste dies mit einem einfachen Befehl manuell erfolgen:

[1] `http://developer.berlios.de/`

```
svnadmin create f:/subversion/phil/daten
```

Durch diesen Befehl wurde in dem angegebenen Verzeichnis ein neues Repository erzeugt, das im Dateisystem auf dem Server die folgende Verzeichnisstruktur wie in Abbildung 4.1 zu sehen ist, erzeugt. Im Verzeichnis `conf` befinden sich zwei

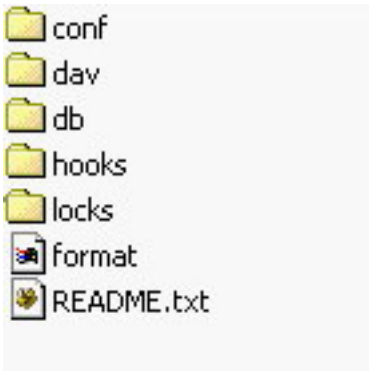

Abbildung 4.1: Verzeichnisstruktur des Repository auf dem Server

wichtige Dateien. In der `svnserve.conf`, wie in Quellcode 4.1 zu sehen, sind grundlegende Eigenschaften wie Zugangsberechtigungen, Begrüßung und Passwortdatei festgelegt. In der Passwortdatei, wie in Quellcode 4.2 zu sehen, werden Benutzernamen und Passwörter für die Benutzer dieses Repository vergeben. Diese Methode der Authentisierung ist jedoch nicht zu empfehlen, da die Passwörter in dieser Datei lesbar sind und diese Datei, zusätzlich zu anderen bereits vorhandenen Zugangslisten, aktuell gehalten werden muss.

```
1 [general]
2 anon-access = none
3 auth-access = write
4 realm = Repository for my personal diplomarbeit
5 password-db = user
```

Quellcode 4.1: Datei /conf/svnserve.conf

```
1 [users]
2 pschneider = geheim
```

Quellcode 4.2: /conf/user

Eine bessere Möglichkeit der Authentisierung wäre die Überprüfung, bevor der Benutzer Zugang zum System erhält. Bei *ssh+svn* erfolgt die Authensisierung durch ssh und bei https erfolgt diese über htaccess, in beiden Fällen wird dabei wahrscheinlich auf eine vorhandene Authentisierungsinfrastruktur zurückgegriffen.

In unserem Projekt wurde als Client für den Zugriff auf das Subversionreposi-

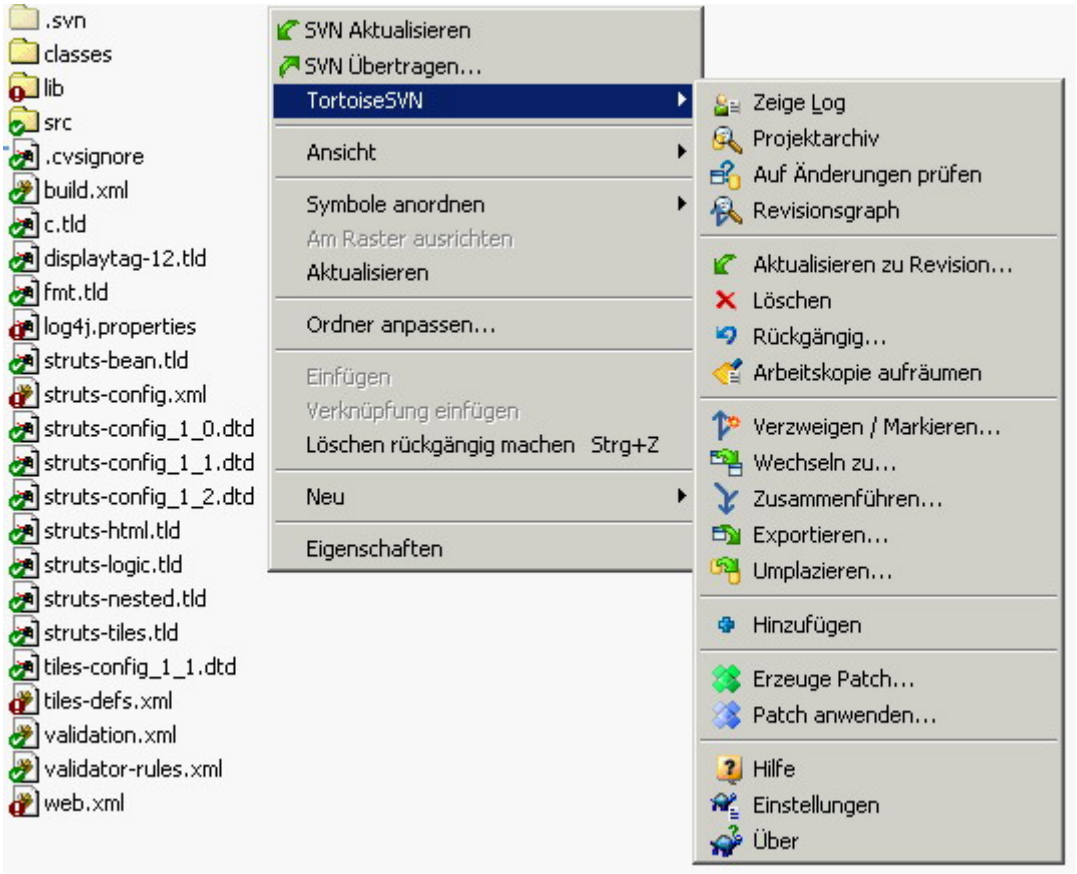

Abbildung 4.2: Ansicht des Clients TortoiseSVN

tory TortoiseSvn[2] verwendet. Dabei handelt es sich wie in Abbildung 4.2 auf der vorherigen Seite zu sehen, um eine Windows-Shellerweiterung, die sich in den Explorer integriert. Ist ein Verzeichnis oder eine Datei Teil eines Projektes, das in einer Subversionversionsverwaltung gespeichert wird, sind diese durch kleine farbige Symbole markiert. Diese Symbole geben den Zustand der Dateien an, ob diese aktuell sind, verändert wurden oder ob eventuelle Konflikte bestehen. Weiterhin können damit auch alle anderen Aktionen wie Log Datei betrachten, Rückgänging, Tagging/Merging und Verschiebungen wie in Abbildung 4.2 auf der vorherigen Seite zu sehen, vorgenommen werden. Mit TortoiseSVN ist es auch möglich nur ein bestimmtes Verzeichnis des meist sehr strukturiertes Projektes auszuchecken und zu bearbeiten. Dazu wird `TortoiseSVN` ⇒ `Wechseln zu` , wie in Abbildung 4.3 zu sehen, ausgewählt. In dem sich öffnenden Dialog wird das entsprechende Verzeichnis im Repository gewählt und die lokale Arbeitskopie wird automatisch auf die entsprechenden Verzeichnise und Dateien aktualisiert.
In den Einstellungen des Subversionservers können auch wie in Unterabschnitt 3.1.2

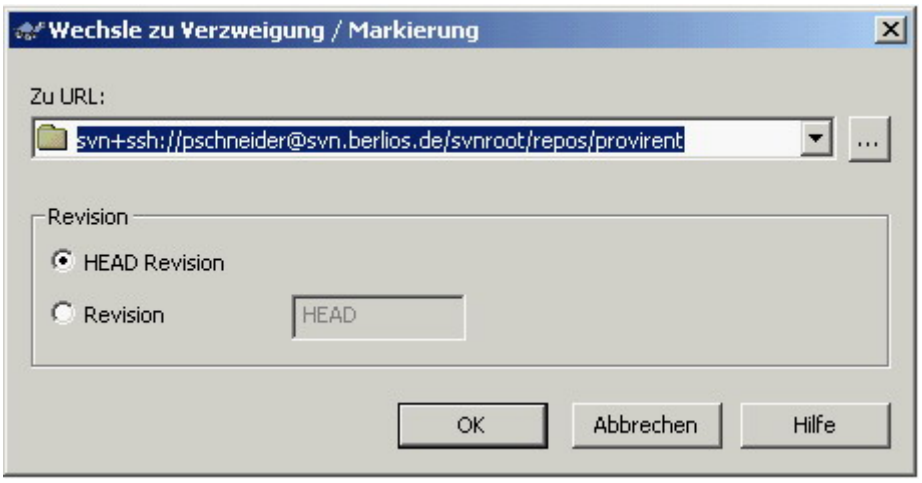

Abbildung 4.3: Ansicht des Fensters zum wechseln der Arbeitskopie

auf Seite 22 bereits beschrieben, automatisch Email verschickt werden. Die in Quellcode 4.3 auf der nächsten Seite gezeigte Email soll verdeutlichen, welche Informationen mit diesem Mechanismus verschickt werden können. Im Betreff der Email befindet sich die neue Revisionnummer und das Verzeichnis in dem die Änderungen stattgefunden haben. Am Anfang der Email ist der Name des Authors, das Datum, eine Auflistung der geänderten Dateien und die Logmeldung zu sehen. Am Ende befinden sich die kompletten Änderungen als Text.

[2] `http://tortoisesvn.tigris.org/`

```
Author: rgriesch
Date: 2005-12-02 21:04:22 +0100 (Fri, 02 Dec 2005)
New Revision: 276
Modified:
   project_src/provirent_hibernate/src/de/hsharz/provirent/management/gui/
       ManagementGui.java
Log:
Menu Rechnung fehlt  - sollte aber aus Zeitgr?\195?\188nden
                                  nicht mehr implementiert
                                     werden !!
ManagementGui        - Reiter Film war aktiv, obwohl
                       Fenster kunden angezeigt wurde
                       ( Fehler behoben )
ManagementGui        - Button Beenden im Menu "Datei"
                       wurde Funktionalit?\195?\164t eingef?\195?\188gt
Modified: project_src/provirent_hibernate/src/de/hsharz/provirent/management/gui/
    ManagementGui.java
===================================================================
--- project_src/provirent_hibernate/src/de/hsharz/provirent/management/gui/
    ManagementGui.java   2005-11-30 15:04:55 UTC (rev 275)
+++ project_src/provirent_hibernate/src/de/hsharz/provirent/management/gui/
    ManagementGui.java   2005-12-02 20:04:22 UTC (rev 276)
@@ -412,9 +412,12 @@
                exitMenuItem = new MenuItem(fileMenu, SWT.CASCADE);
                exitMenuItem.setText(l.getString("menu.file.exit"));
-
+               exitMenuItem.addSelectionListener(new SelectionAdapter() {
+                       public void widgetSelected(SelectionEvent evt) {
+                           shell.close();
+                       }
+               });
        }
-
        /**
         * init the View Menu
         */
@@ -484,7 +487,7 @@
                viewCustomerMenuItem = new MenuItem(viewMenu, SWT.CHECK);
                viewCustomerMenuItem.setText(l.getString("menu.view.customer"));
-               viewCustomerMenuItem.setSelection(false);
+               viewCustomerMenuItem.setSelection(true);
                viewCustomerMenuItem.addSelectionListener(new SelectionAdapter() {
                        public void widgetSelected(SelectionEvent evt) {
                                if (tabItemCustomer == null || tabItemCustomer.
                                    isDisposed()) {
@@ -599,7 +602,7 @@
                viewMovieMenuItem = new MenuItem(viewMenu, SWT.CHECK);
                viewMovieMenuItem.setText(l.getString("menu.view.movie"));
-               viewMovieMenuItem.setSelection(true);
+               viewMovieMenuItem.setSelection(false);
                viewMovieMenuItem.addSelectionListener(new SelectionAdapter() {
                        public void widgetSelected(SelectionEvent evt) {
                                if (tabItemMovie == null || tabItemMovie.
                                    isDisposed()) {
```

Quellcode 4.3: Auszug aus einer automatisch erzeugten Email

4.2 Management Anwendung mit SWT

Dieses Kapitel beginnt mit der Vorstellung allgemeiner Komponenten, die während der Implementierung verwendet wurden. Danach wird auf den Aufbau der grafischen Benutzeroberfläche und dazu auf einzelne Klassen eingegangen, um einen Überblick über die gesamte Implementierung zu vermitteln. Aufgrund dessen, dass dieses Projekt noch nicht beendet wurde, sind nicht alle hier erwähnten Funktionalitäten bereits vollständig realisiert.

4.2.1 Log4j

Log4j[3] wurde während der Implementierung der einzelnen Klassen genutzt um evtl. Debug- und Fehlermeldungen über eine gemeinsame Schnittstelle zu verwalten. Dabei unterstützt Log4j Hierarchien, das bedeutet man kann Logger-Ausgaben einer bestimmten Hierarchie zuordnen und dadurch entscheiden, welche Ausgaben man im Moment benötigt.
Aufgrund dieser Debug- und Fehlermeldungen konnte das Programmverhalten bei Bedienungsfehlern entsprechend angepasst werden. Des Weiteren konnte auf diese Weise der Programmablauf verfolgt und evtl. Fehler in der Programmierung beseitigt werden.

4.2.2 PropertyResourceBundle-Klasse

Die PropertyResourceBundle-Klasse befindet sich java.util package. Um die Möglichkeit zu haben, vor und während des Programmablaufs die Sprache im Programm zu ändern, sind, anstatt des eigentlichen Namens von Buttons oder der Inhalte von Hilfetexten, entsprechende Bezeichner definiert wurden. Diese Bezeichner befinden sich mit dem zugehörigen Inhalten in verschiedenen Textdateien. Mit Hilfe der PropertyResourceBundle-Klasse können nun die entsprechenden Dateien ausgelesen werden und ersetzen die definierten Bezeichner mit dem eigentlichen Inhalt. Dazu wird die Datei ausgelesen und ein entsprechendes ResourceBundle erzeugt, über den man Zugriff auf die definierten Bezeichner und deren Inhalt hat. Es ist möglich verschiedene Sprachen zu unterstützen, indem man für jede Sprache entsprechende Dateien erzeugt, bei denen die selben Bezeichner definiert sind, der Inhalt aber in der entsprechenden Sprache vorliegt.

4.2.3 Locale-Klasse

Mit Hilfe der Locale-Klasse (java.util package) ist man in der Lage sich an bestimmte geographische, politische oder kulturelle Eigenschaften bestimmter Ge-

[3] `org.apache.log4j`

biete anzupassen. Im Zusammenhang mit diesem Projekt wird Locale dazu verwendet, die entsprechende Zeitzone und, wenn vom Programm unterstützt, die dortige Sprache als Standard-Sprache zu übernehmen. Das Einstellen der Region,Sprache,etc. muss aber durch den Benutzer erfolgen.

4.2.4 Aufbau der grafischen Benutzeroberfläche

Für die grafische Benutzeroberfläche wurde auf die SWT-Bibliothek zurückgegriffen. Die Entwicklungsumgebung Eclipse baut auch auf SWT auf. Aus diesem Grund stand die SWT-Bibliothek ohne weiteren Aufwand zur Verfügung. Auf die einzelnen Klassen von SWT wird hier nicht eingegangen, da sich der Umfang des Dokumentes erheblich erhöht. Alle verwendeten grafischen Komponenten für Fenster, Buttons, Textfelder, etc. wurden aus der SWT-Bibliothek verwendet. Genauere Informationen können über die API abgefragt werden.
Die grafische Oberfläche baut sich aus einem Hauptfenster (mit Menüleiste) und einer Statusleiste auf. Das Hauptfenster erlaubt es, zwischen den einzelnen Unterfenstern (Menüpunkten) zu navigieren. Die Statusleiste informiert über entsprechende Meldungen bei Fehlern oder erfolgreiches Abschließen von Aktionen. Die Untermenüs werden über das Hauptfenster aufgerufen, erhalten gleichzeitig den Fokus und werden damit dem Benutzer angezeigt. Um zwischen den verschiedenen Untermenüs zu navigieren, kann zum einen das entsprechende Menü aufgerufen werden und zum anderen wird jedes geöffnete Menü als Reiter dargestellt. Über diesen Reiter kann der Fokus auf das gewünschte Menü gegeben werden. In einzelnen Untermenüs werden für kritische Eingaben entsprechende Dialogboxen verwendet, die die Bedienung vereinfachen und verhindern, dass ein Benutzer falsche Angaben machen kann.
Der oben vorgestellte Aufbau wird durch folgenden Klassen realisiert:

1. SWTResourceManager (realisiert das korrekte Freigeben der grafischen Objekte, siehe Unterabschnitt 3.3.3 auf Seite 30)
2. ManagementGui (bildet das Hauptfenster, mit Menü und Statusleiste / ist die zentrale Verwaltungsstelle der Untermenüs)
   (a) CompositeActors (Untermenü für Schauspieler)
   (b) CompositeCustomer (Untermenü für Kunden)
   (c) CompositeDirectors (Untermenü für Regisseure)
   (d) CompositeDVD (Untermenü für vorhandene DVDs / Lagerbestand,Status)
      i. DialogDVD (Dialoge für einfachere Bedienung des DVD-Menüs)
   (e) CompositeFormate (Untermenü für Audio- und Videoformate)

(f) CompositeGenre (Untermenü für Genre)
(g) CompositeImage (Untermenü für Cover,Poster und Bilder zu Filmen)
(h) CompositeLanguage (Untermenü für unterstützte Untertitel und Sprachen)
(i) CompositeMovie (Untermenü für Filme)

i. DialogDVD (Dialoge für einfachere Bedienung des DVD-Menüs)

(j) CompositeOrder (Untermenü für Bestellungen)
(k) CompositePayment (Untermenü für Rechnungen)
(l) CompositeStatus (Untermenü für den Zustand von DVDs)

3. StatusLineStyledText (definiert das Aussehen der Statusleiste bei Fehlern oder anderen Ereignissen)
4. SWTCalendarDialog (bietet eine grafische Kalenderoberfläche, zur einfacheren Eingabe von Datum- und Zeitangaben)
5. ColorDef (definiert Farben für festgelegte Ereignisse)

4.2.5 SWTResourceManager-Klasse

Diese Klasse ist verantwortlich dafür, das grafische Objekte (Fenster, Buttons, etc.) wieder freigegeben werden. Dazu wird bei einem nicht mehr verwendeten Objekt die dispose()-Methode aufgerufen und somit das Objekt und der damit verbundene Speicherplatz wieder freigegeben.

4.2.6 ManagementGui-Klasse

Die ManagementGui-Klasse erzeugt das Hauptfenster, indem die anderen Untermenüs angezeigt werden. Des Weiteren wird eine Menüleiste und eine Statusleiste implementiert.
Dazu wird als Erstes das Hauptfenster erzeugt, der Titel des Fensters über Locale l ermittelt und an das Fenster übergeben. Die Größe des Fensters ist in Konstanten festgelegt, da aufgrund von Bildschirmeinstellungen die jeweilige Auflösung sich von System zu System unterscheiden kann.

```
1 display = Display.getDefault();
2 shell = new Shell(display);
3 shell.setText(l.getString("mainwindow.title"));
4 //set size
5 shell.setSize(MAIN_WINDOW_WIDTH, MAIN_WINDOW_HEIGHT);
6 shell.setLocation(MAIN_WINDOW_X, MAIN_WINDOW_Y);
```

Anschließend wird die Menüleiste mit den entsprechenden Menüpunkten erzeugt und initialisiert.

```
rootMenu = new Menu(shell, SWT.BAR);
shell.setMenuBar(rootMenu);
//init the other menu's
initFileMenu();
initViewMenu();
initHelpMenu();
```

Das `ViewMenu` beinhaltet alle implementierten und auswählbaren Untermenüs. Die Bezeichnung der einzelnen Menüpunkte ist wieder in Locale `l` festgelegt. Nachdem der Menüpunkt erzeugt wurde, werden die einzelnen Untermenüpunkte erstellt. Danach wird festgelegt, ob die Untermenüs bereits nach dem Start aktiv sein sollen. Der folgende Quellcodeabschnitt zeigt diese Vorgehensweise mit dem Menüpunkt `menu.view` und dem Untermenü `menu.view.format`.

```
viewMenuItem = new MenuItem(rootMenu, SWT.CASCADE);
viewMenuItem.setText(l.getString("menu.view"));
viewMenu = new Menu(viewMenuItem);
viewMenuItem.setMenu(viewMenu);
viewVideoFormatMenuItem = new MenuItem(viewMenu, SWT.CHECK);
viewVideoFormatMenuItem.setText(l.getString("menu. view.format"));
viewVideoFormatMenuItem.setSelection(false);
```

Darauf wird dem Untermenüpunkt ein `SelectionListener` zugeordnet, der die entsprechende Behandlung festlegt, wenn der Untermenüpunkt ausgewählt wurde. Bevor das Hauptfenster angezeigt werden kann, wird vorher noch das Layout des Fensters festgelegt und die Statusleiste initialisiert.

```
//init the MainComposite
compositeMain = new Composite(compositeRoot, SWT.EMBEDDED);
GridLayout composite3Layout = new GridLayout();
...
//StatusLine
initStatusLine();
```

Das fertige Hauptfenster sieht wie in Abbildung 4.4 auf der nächsten Seite aus.

4.2.7 CompositeDVD-Klasse

Die CompositeDVD-Klasse legt das Aussehen und Verhalten des DVD-Menüpunktes fest. Wird der DVD-Menüpunkt aufgerufen, sorgt die ManagementGui-Klasse dafür, dass das DVD-Fenster den Fokus erhält, ein DVD-Reiter erstellt und zu den bereits vorhandenen Reitern hinzugefügt wird. Zusätzlich müssen die ganzen Eigenschaften, die ein DVD-Objekt besitzt, mit Hilfe des Menüpunktes bearbeitet werden können. Dazu wird eine Reihe von Buttons, Labels, Tables, Comboboxen, etc. benötigt.

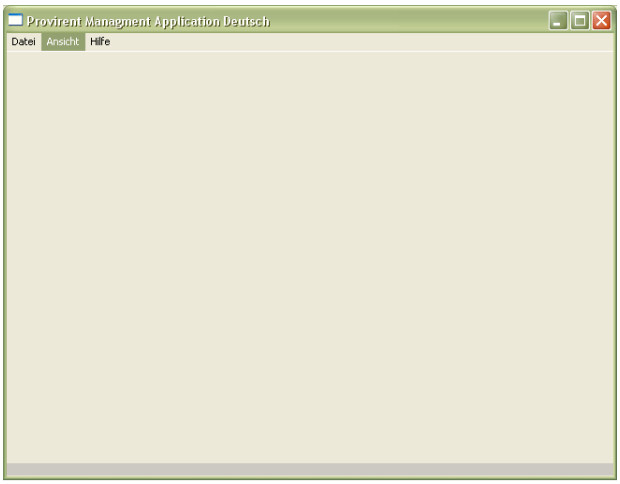

Abbildung 4.4: Hauptfenster der Management Anwendung

Die grafische Oberfläche wird im DVD-Fenster in eine linke und rechte Hälfte unterteilt. In der linken Hälfte werden alle DVD-Objekte angezeigt, die sich in der Datenbank befinden. Des Weiteren befindet sich auf der linken Seite ein Suchfeld um die evtl. Massen an DVD-Objekten einfacher zu handhaben. Auf der rechten Seite des DVD-Fensters befindet sich die Editor-Umgebung. Mit Hilfe der verschiedenen Textfelder, Buttons, Comboboxen im Editor können nun existierende DVDs bearbeitet und neue DVDs in die Datenbank eingefügt werden.

```
1 if (mode_dvd == ManagementGui.MODE_ADD)   {
2   Database.saveObject(localdvd);
3   ...
4 } else if (mode_dvd == ManagementGui.MODE_EDIT)   {
5   Database.updateObject(localdvd);
6 }
```

Das DVD-Fenster sieht wie in Abbildung 4.5 auf der nächsten Seite aus:

Wie oben beschrieben, werden auf der linken Seite des Fenster die in der Datenbank vorhandenen Objekte angezeigt und darunter befindet sich das Suchfeld. Auf der rechten Seiten befinden sich mehrere Buttons, Textfelder, etc., die je nach gewünschter Tätigkeit aktiviert oder deaktivert sind. Auf diese Art und Weise lassen sich Benutzerfehler verhindern, da je nach gewünschter Tätigkeit nur die in diesem Zustand möglichen Aktionen (Buttons,Textfelder,etc.) funktionstüchtig sind.
Wird das DVD-Fenster geschlossen, wechselt der Fokus zum nächsten vorhandenen Reiter, ansonsten ist nur das Hauptfenster sichtbar.
Die anderen Untermenüs sind je nach Funktionalität komplexer oder weniger komplexer aufgebaut. Trotzdem sind sie im Großen und Ganzen identisch zum DVD-

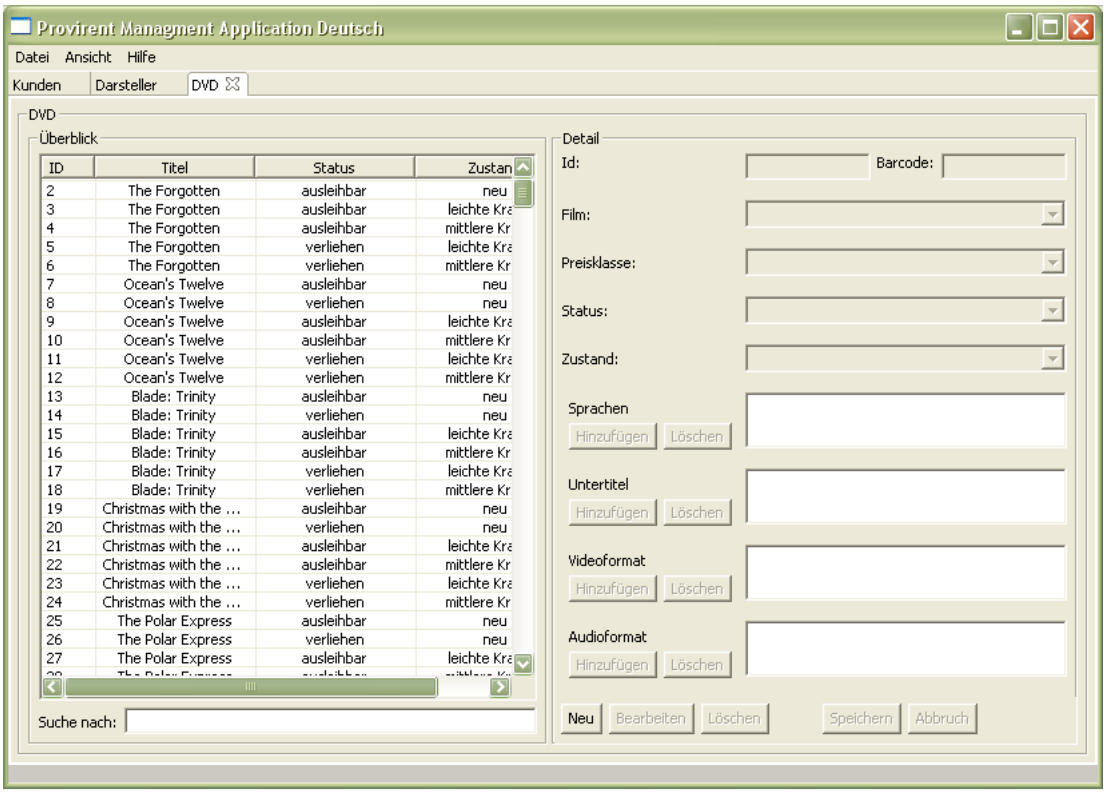

Abbildung 4.5: DVD-Fenster

Fenster. Aus diesem Grund werden sie hier nicht weiter behandelt.

4.2.8 DialogDVD-Klasse

Zur einfacheren Handhabung wurden für bestimmte Felder, die zum Beispiel mehrere Werte erlauben oder von Benutzern unterschiedlich eingegeben werden könnten, Dialogfenster erstellt, um die Eingabe durch den Benutzer einheitlich zu halten. Die DialogDVD-Klasse realisiert mehrere Dialogfenster unter anderem für die Auswahl der Sprache und der Untertitel, die eine DVD unterstützt. Wie der folgende Quellcodeausschnitt zeigt, wird dazu der gewünschte Dialogtyp übergeben und dementsprechend behandelt.

```
1 if (type == TYPE_LANGUAGE) {
2   groupDialogDescription.setText(l.getString("dvd.dialog.title.language"));
3 } else if (type == TYPE_SUBTITLE) {
4   groupDialogDescription.setText(l.getString("dvd.dialog.title.subtitle"));
5 ...
```

Die folgende (siehe Abbildung 4.6) Grafik zeigt das Dialogfenster um die Sprachen, die eine DVD unterstützt, auszuwählen:

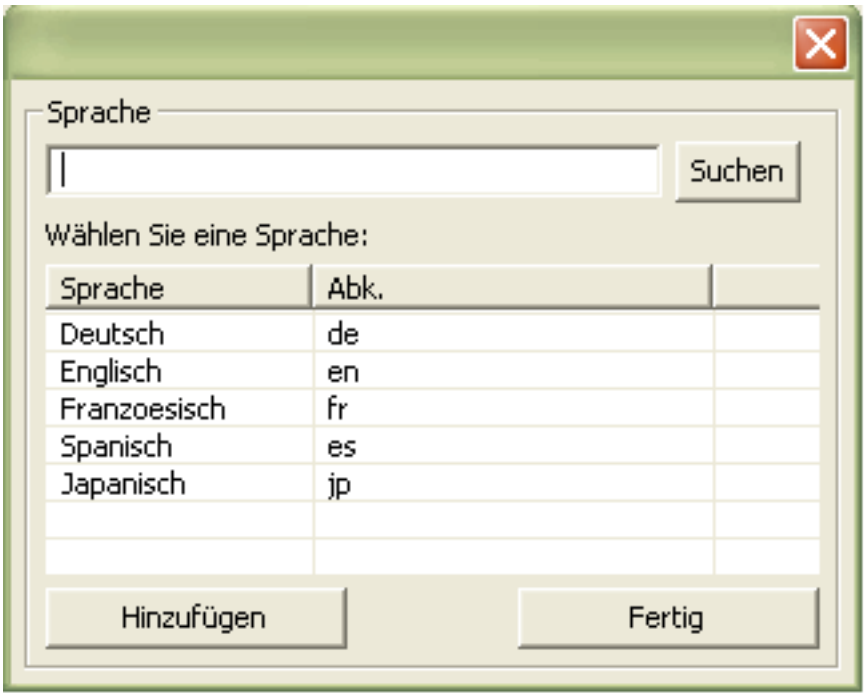

Abbildung 4.6: DVD-Dialogfenster für Sprachen

Je nach verwendetem Dialog steht dem Benutzer ein Suchfeld zur Verfügung und eine Auswahl an erlaubten Werten. Aus diesen kann über das Dialogfenster

eine bestimmte Menge ausgewählt werden. Die Menge wird über eine Liste an die aufrufende Klasse zurückgegeben und von ihr für das entsprechende Feld übernommen. Doppelte Eingaben werden dabei durch das Ausblenden bereits ausgewählter Werte verhindert.
Die DialogMovie-Klasse geht in ähnlicher Weise vor und wird deshalb hier nicht weiter behandelt.
Die SWTCalendarDialog-Klasse realisiert ein Dialogfenster für Datum-Eingaben, dazu existiert in der SWT-Bibliothek eine dafür bereits entwickelte Klasse. Diese SWTCalender-Klasse wird in der SWTCalendarDialog-Klasse verwendet und muss als Argument das anzuzeigende Dialogfenster erhalten.

```
1 swtcal = new SWTCalendar(shell);
```

Die folgende Grafik (siehe Abbildung 4.7) zeigt ein solches Dialogfenster:

Abbildung 4.7: DialogFenster für Datum-Eingaben

4.2.9 StatusLineStyledText- und ColorDef-Klasse

Beide Klassen realisieren ein einheitliches Aussehen der Statusleiste bzw. der Farben für die verschiedenen Menüs. In der ColorDef-Klasse sind verschiedene Farben definiert, um bei Fehlermeldungen oder Ereignissen ein einheitliches Farbschema zu haben. Die StatusLineStyledText-Klasse nutzt dieses Farbschema für entsprechende Ausgaben auf der Statuszeile.

4.3 Java-Web-Anwendung mit Struts

Die Webanwendung ist dafür gedacht, dass Kunden von überall die Möglichkeit (mit der Voraussetzung eines Internetzugangs) haben sollen, nach DVDs im Datenbestand zu suchen, Informationen darüber abzurufen und diese zu bestellen.
Der aktuelle Stand dieser Webanwendung ist insofern implementiert, dass sich bereits Kunden in das System einloggen können. Anhand dieser Funktionalität wird auf die Implementierung unter Verwendung des Web-Frameworks Apache Struts eingegangen.
Folgende Schritte waren dabei notwendig:

- Login-Formular in JSP integrieren
- ActionForm Bean für das Formular erzeugen
- Action-Klasse für das Login erzeugen
- ActionForm und Action in struts-config.xml registrieren

4.3.1 Login-Formular

Um den Kunden die Möglichkeit zu geben, sich am System anzumelden, gibt es ein entsprechendes Formular. Darin muss der Benutzer seinen Benutzernamen sowie sein persönliches Passwort eingeben mit Bestätigung seiner Eingaben über den Login-Button soll diese Anfrage mit den Daten in der Datenbank verglichen werden und bei Erfolg eine dem Benutzer angepassten Seite geladen werden.
Das Login-Formular wird in Quellcode 4.4 mit den entsprechenden Tags der Struts-Taglibrary gebildet. Das `<html:form>`-Tag grenzt darin das Formular ein. Über das Attribut *action* wird dem Servlet signalisiert, dass die in der *struts-config.xml* definierte Action mit dem Pfad */Login* beim Klick des Submit-Buttons ausgeführt werden soll.
Mit der Verwendung von `<html:text>`-Tags werden dem Benutzer Eingabefelder erzeugt, worin dieser seine Daten zum Einloggen eintragen kann. Über das Attribut *property* wird auf das entsprechende Attribut in der ActionForm Bean referenziert. Das bedeutet bei einer Betätigung des Login-Buttons, der durch das Tag `<html:submit` dargestellt wird, dass die Instanz der ActionForm Bean mit den eingegebenen Daten aus dem Formular versehen wird.

```
1 <html:form action="/Login" method="post">
2   <tr>
3     <td>
4       <label class="login_label"><bean:message key="main.login.username"/>:</label
            >
5       <html:text property="username" styleClass="login" style="font-weight: bold;"
            size="13"/>
6     </td>
7     <td>
```

```
8         <label class="login_label"><bean:message key="main.login.passwort"/>:</label
              >
9         <html:password property="password" styleClass="login" size="14" />
10      </td>
11      <td>
12         <html:submit title="main.login.submit" styleId="submit_search"/>
13      </td>
14    </tr>
15  </html:form>
```

Quellcode 4.4: Login-Formular in der JSP

4.3.2 Erzeugung der ActionForm Bean

Die Klasse zur Aufbewahrung der Daten eines Formulars besteht prinzipiell aus den Attributen und ihren Getter- bzw. Setter-Methoden. Dementsprechend wurde die Klasse *LoginForm* mit den Attributen *username* und *password* implementiert. Die Methoden zum Erhalt bzw. Setzen der Attribute wurden in diesem Fall noch durch zwei weitere Methoden zum Validieren der Eingaben bzw. zum Zurücksetzen der Formulareingaben komplettiert.

4.3.3 Erzeugung der Action-Klasse

Zur Ausführung der Geschäftslogik, in diesem Anwendungsfall des Logins, war es notwendig die Klasse *LoginAction*, die von der Klasse *Action* aus dem Struts-Framework erbt, zu implementieren. Die Methode *execute*() spielt dabei die größte Rolle, da diese zwingend implementiert werden musste. Darin wird die Logik ausgeführt, um den Kunden zu identifizieren und authentifizieren. In dieser Methode (siehe Quellcode 4.5) werden zunächst die Daten aus der ActionForm Bean ausgelesen, d.h. Benutzername und Passwort. Unter Verwendung der Klasse *Database*, die im Kapitel Abschnitt 4.4 beschrieben wird, werden die Benutzerdaten mit denen aus der Datenbank verglichen. Bei erfolgreichem Ergebnis wird das aus der Datenbank geladene *Customer*-Objekt (gleichzusetzen mit dem Kunden des Systems) in die Session der Webanwendung gespeichert und eine entsprechende Antwort (Response) an den Browser gesendet.

```
1  public ActionForward execute(
2                          ActionMapping mapping, ActionForm form,
3                          HttpServletRequest request,
4                          HttpServletResponse response)
5          throws Exception {
6
7    LoginForm lf = (LoginForm) form;
8
9    boolean existUser = false;
10
11   //    Laden von Benutzername und Passwort
12   String aktUsername = lf.getUsername();
```

```
13    String aktPassword = lf.getPassword();
14
15    Customer user = null;
16
17    List users = Database.getUsersByLogin(aktUsername, aktPassword);
18
19    //über die Benutzerliste iterieren
20    while (users.iterator().hasNext()) {
21
22      existUser = true;
23
24      //Benutzer aus Liste laden
25      user = (Customer) it.next();
26    }
27
28    //            wenn Benutzer nicht existiert
29    if (!existUser) {
30
31      //Eingabeformular laden
32      return mapping.getInputForward();
33
34    } else {
35
36      request.getSession().setAttribute(StrutsConstant.SESSION_USER_KEY, user);
37      return mapping.findForward(StrutsConstant.FWD_SUCCESS);
38
39    }
40  }
```

Quellcode 4.5: execute()-Methode der LoginAction

4.3.4 Einträge in die *struts-config.xml*

Die Zuordnung von AcctionForm Bean und Action-Klasse erfolgt in der Struts-Konfigurationsdatei. Zum einen musste die Form Bean registriert werden:

```
1  <form-bean name="LoginForm" type="de.hsharz.provirent.customer.form.LoginForm"/>
```

Ferner wurde hier ein entsprechendes Action-Mapping definiert:

```
1  <action path="/Login"
2          input="provirent.index"
3          type="de.hsharz.provirent.customer.action.LoginAction"
4          name="LoginForm"
5          validate="true" >
6    <forward name="success" path="/index.do"/>
7  </action>
```

In diesem Mapping erfolgt zum einen die Zuordnung der Form Bean *LoginForm* zur Action *LoginAction* und zum anderen wird der Forward, also das Weiterleiten des Requests, bei erfolgreichem Login definiert.

4.4 Persistenzschicht mit Hibernate

Die Implementation der Persistenzschicht unter Verwendung von Hibernate wurde in mehreren Schritten vollzogen:

1. Erzeugen der Hibernate-Mapping-Dateien
2. Generieren der Mapping-Klassen aus den XML-Dateien
3. Generieren des Datenbankschemas
4. Erzeugen der Hibernate-Konfiguration
5. Entwicklung der Klasse *Database* für die Interaktion mit der Datenbank

Diese Schritte werden im Folgenden etwas genauer beschrieben.

4.4.1 Erzeugen der Hibernate-Mapping-Dateien

Wie in Kapitel Abschnitt 3.5 beschrieben, sind die Mapping-Dateien für das Abbilden eines Datenbankobjekts auf eine Java-Klasse notwendig.
Zunächst wurde in der Phase der Implementierung ein entsprechendes Datenmodell mit allen benötigten Entitäten und Beziehungen erzeugt. Daraus wurden die entsprechenden Hibernate-Mapping-Dateien gebildet, indem hier unter Angabe von Name, Typ, Name der korrespondierenden Tabellenspalte der Datenbank und einigen anderen zum Teil auch optionalen Attributen alle Properties eines persistenten Objekts definiert wurden. Wichtig ist auch, dass neben den Properties mit einfachen Datentypen auch sämtliche Beziehungen zu anderen Objekten bekannt gemacht werden mußten.

4.4.2 Generieren der Mapping-Klassen aus den XML-Dateien

Nachdem alle Mapping-Dateien erzeugt wurden, war es notwendig daraus die entsprechenden Java-Klassen erzeugen zu lassen. Hierfür wurde das Tool zur Code-Generierung von Hibernate verwendet. Die Automatisierung dieses Prozesses wurde in ein Ant Build-Skript eingefügt, indem das Tool mit den entsprechenden Optionen von diesem Skript aus ausgeführt werden kann.
Wie in Quellcode 4.6 ersichtlich, wird über das Element *taskdef* das Tool in das Skript integriert und über *target* die Anweisungen zur Ausführung definiert. Mithilfe des Ant-Plugins für Eclipse konnte dieses Skript zur Ausführung gebracht werden.

```
1     <!-- Teach Ant how to use Hibernate's code generation tool -->
2     <taskdef name="hbm2java"
3              classname="net.sf.hibernate.tool.hbm2java.Hbm2JavaTask"
4              classpathref="project.class.path"/>
5
6     <!-- Generate the java code for all mapping files in our source tree -->
```

```
7     <target name="codegen"
8             description="Generate Java source from the O/R mapping files">
9         <hbm2java output="${source.root}">
10          <fileset dir="${source.root}">
11            <include name="**/*.hbm.xml"/>
12          </fileset>
13        </hbm2java>
14    </target>
```

Quellcode 4.6: Erzeugung der persistenten Java-Klassen über Apache Ant

4.4.3 Generieren des Datenbankschemas

Für die Verwaltung der Provirent-Datenbank wurde die Verwendung des DBMS Firebird vorgezogen. Dabei wurde ein entsprechender Server auf den Entwicklungsrechnern installiert. Mithilfe des Firebird-Clients FlameRobin war der projektexterne Zugriff auf diesen Datenbankserver und dementsprechend auf die Datenbank möglich.
Zuvor musste jedoch erst das Datenbankschema generiert werden. Hierbei wurde das Hibernate-Tool zur Generierung von Datenbankschemata verwendet. Wie bei der Generierung der persistenten Java-Klassen wurden die Anweisungen für die Generierung in ein Ant Build-Skript, wie in Quellcode 4.7 zu sehen, integriert.

```
1     <!-- Generate the schemas for all mapping files in our class tree -->
2     <target name="schema" depends="compile"
3             description="Generate DB schema from the O/R mapping files">
4
5       <!-- Teach Ant how to use Hibernate's schema generation tool -->
6       <taskdef name="schemaexport"
7                classname="net.sf.hibernate.tool.hbm2ddl.SchemaExportTask"
8                classpathref="project.class.path"/>
9
10      <schemaexport properties="\${class.root}/hibernate.properties" quiet="no"
            text="no" drop="no" delimiter=";">
11        <fileset dir="\${class.root}">
12          <include name="**/*.hbm.xml"/>
13        </fileset>
14      </schemaexport>
15    </target>
```

Quellcode 4.7: Erzeugung des Datenbankschemas über Apache Ant

4.4.4 Erzeugen der Hibernate-Konfiguration

Die Konfiguration des Hibernate-Frameworks bedurfte es, zum einen die Eigenschaften der Datenbankverbindung zu deklarieren und zum anderen die verschiedenen Mapping-Dateien dem Framework bekannt zu machen.
Die Datenbankverbindung wurde in einer Properties-Datei (siehe Quellcode 4.8)

beschrieben. Dabei war es unter anderem notwendig, folgende Eigenschaften zu definieren:

- Dialekt der Datenbank
- Datenbanktreiber
- URL der Datenbank
- Benutzername und Passwort für die DB

```
1
2 hibernate.dialect=net.sf.hibernate.dialect.FirebirdDialect
3 hibernate.connection.driver_class=org.firebirdsql.jdbc.FBDriver
4 hibernate.connection.url=jdbc:firebirdsql:localhost:c:/video
5 hibernate.connection.username=SYSDBA
6 hibernate.connection.password=masterkey
7 hibernate.jdbc.use_streams_for_binary=true
```

Quellcode 4.8: hibernate.properties

Zusätzlich dazu musste noch eine Klasse implementiert werden, die Sessions für das Hibernate-Framework bereitstellt. Mithilfe der Klasse *HibernateUtil* wurden nicht nur Methoden zum Erzeugen und Beenden von Sessions angeboten, sondern auch das objektrelationale Mapping der Provirent-Datenbank für Hibernate sichtbar gemacht. Hierbei wird eine SessionFactory erzeugt, dessen Konfiguration die Namen aller persistenten Klassen besitzt und somit Zugriff darauf hat.

4.4.5 Entwicklung der Klasse *Database* für die Interaktion mit der Datenbank

Nachdem nun das Hibernate-Framework alle entsprechenden Einstellungen erhalten hat, war es möglich darüber mit der Datenbank zu interagieren. Um jedoch eine zentrale Klasse mit allen Operationen zu erhalten, wurde die Klasse *Database* implementiert. Hier befinden sich z.B. Methoden zum Laden, Speichern, Aktualisieren oder Löschen von einzelnen oder mehreren Datenbankobjekten.
Die Methode *getSingleActor*() in Quellcode 4.9 lädt einen Schauspieler anhand seiner ID aus der Datenbank und gibt diesen als Objekt der Klasse *Actor* zurück.

```
1  public static Actor getSingleActor(final int id) {
2          if (logger.isDebugEnabled()) {
3                  logger.debug("getSingleActor() - start. int filter= " + id);
4          }
5          //init the returnlist
6          Actor returnobject = null;
7
8          Session s = null;
9          Transaction tx = null;
10         try {
11                 //get new Session and begin Transaction
12                 s = HibernateUtil.currentSession();
```

```
13
14                  returnobject = (Actor) s.get(Actor.class, new Integer(id));
15
16          } catch (Exception e) {
17                  logger.error("getSingleActor() - Error while trying to do
                        Transaction", e);
18
19          } finally {
20                  try {
21                          // No matter what, close the session
22                          HibernateUtil.closeSession();
23                  } catch (HibernateException e1) {
24                          logger.error("getSingleActor() - Could not Close the
                                Session", e1);
25                  }
26          }
27
28          if (logger.isDebugEnabled()) {
29                  logger.debug("getSingleActor() - end");
30          }
31          return returnobject;
32  }
```

Quellcode 4.9: Methode zum Laden eines Schauspielers aus der DB

Kapitel 5

Fazit

Mit Hilfe dieses Projektes wurde bei allen Projektmitglieder Erfahrungen im Bereich Softwareplanung und Softwareentwicklung gesammelt. Bei diesem Projekt wurde eine eigenständige Idee in die Realität umgesetzt, wobei alle Projektmitglieder dabei intensiv und mit viel Interesse an einer möglichst guten Realisierung mitarbeiteten.

Technologisch gesehen haben sich die Projektmitglieder in Themengebiete gewagt, die ihnen bis dahin zum Teil relativ fremd waren. So wurden im Bereich vom objektrelationalen Mapping überhaupt die ersten Erfahrungen gesammelt. Dazu wurde die Erkenntnis gewonnen, dass Hibernate ein sehr vorteilhaftes und umfassendes Werkzeug ist, um die Persistenzschicht von der Businessschicht unabhängig zu gestalten.
So wurde erreicht, dass diese Persistenzschicht vollständig implementiert werden konnte und für den Zugriff aus dem Kundenmodul und dem Verwaltungsmodul zur Verfügung steht.

Weiterhin ist festzustellen, dass die Implementierung des Projekts noch nicht abgeschlossen ist, so dass z.B. noch grundlegende Funktionalitäten des Kundenmoduls umgesetzt werden müssen. Trotzdem konnte und kann hier die Verwendung des Webframeworks Apache Struts dazu beitragen, dass mit relativ wenig Aufwand und Entwicklungszeit die Entwicklung einer 3-schichtigen Webanwendung ermöglicht wird. Dies hat den Vorteil, dass Design, Businesslogik und Daten von einander getrennt dargestellt und umgesetzt werden konnten.

Durch die Verwendung von Subversion als Versionsverwaltung wurde der Umgang mit einer neuen Technologie schnell zur Gewohnheit. Dadurch ist es für die Projektmitglieder zum Alltag geworden, Änderungen sofort für alle zu speichern und mit aussagekräftigen Kommentaren zu versehen. Durch die netzwerksparende

Übertragung von Daten, war auch der Einsatz von Modems möglich. Durch den automatischen Versand von Emails wurden andere Projektmitglieder schnell und kompakt über Änderungen informiert. Die installation auf einem Windowstestrechner mit einer gesicherter HTTPS-Verbindung gestaltete sich als unkompliziert, wie die Installation einer gesicherten ssh+svn Verbindung auf einem Fedore Core 3 Linux Rechner, dank gut beschriebener Anleitungen.

Aufgrund der SWT-Bibliothek konnte eine einfach zu bedienende Benutzeroberfläche erstellt werden, mit der alle notwendigen Objekte für eine Online-Videothek einfach editiert und erstellt werden können. Aufgrund der verwendeten Reiter für jedes Untermenü kann schnell zwischen den einzelnen Menüs umgeschaltet und Veränderungen vorgenommen werden. Die verschiedenen Dialogfenster erleichtern die Eingabe für den Benutzer und zugleich die Bearbeitung im Quellcode.

Dieses Projekt könnte auch als Grundlage für andere Studentenprojekte dienen. Dabei müßen die Studenten sich nicht in dieses spezielle Projekt einarbeiten, sondern können ein selbständiges Modul entwickeln. So könnten Studenten bspw. ein Suchmodul mit Apache Lucence entwickeln oder ein Barcodemodul zur Verarbeitung von Barcodes entwickeln. Weiterhin könnte ein Modul zur Erstellung von Rechnungen oder anderen Reports mit Eclipse Birt erstellt werden, ein Modul zur Verwaltung der Standorte/Lagerorte der DVD's oder ein Modul zur Berechnung der kürzesten Wegstrecke bei der Zusammenstellung einer bzw. mehrerer Bestellungen. Jedes dieser Module könnte dann mit geringen Aufwand in die Online-Videothek integriert werden.

Literaturverzeichnis

Ecl 2005
Eclipse-Tutorial, 2005. `http://download.eclipse.org/eclipse/downloads/drops/R-3.1-200506271435/org.eclipse.jdt.doc.isv.3.1.pdf.zip`

Hib 2005
Hibernate Reference Documentation. 3.0. http://www.hibernate.org/hib_docs/v3/reference/en/html_single/, 2005

Apa 2005
Apache Software Foundation: *Struts - User Guide*. 2005

Bauer 2004
Bauer, Gavin: *Hibernate in Action*. Manning Publications, 2004

Cavaness 2004
Cavaness, Chuck: *Programming Jakarta Struts*. 2nd. O'Reilly Media Inc., 2004

Collins-Sussman u. a. 2005
Collins-Sussman, Ben ; Fitzpatrick, Brian W. ; Pilato, C. M.: *Version Control with Subversion*, 2005

Dudney u. a. 2004
Dudney, Bill ; Lehr, Jonathan ; Mattingly, LeRoy: *Mastering JavaServer Faces*. Wiley Publishing, 2004. – ISBN 0471462071

Fogel u. Bar 2002
Fogel, Karl ; Bar, Mashe: *Open Source-Projekte mit CVS*. mitp-Verlag, 2002

Frotscher 2004a
Frotscher, Thilo: Der Ball ist rund. In: *Javamagazin* 04 (2004), S. 95

Frotscher 2004b
FROTSCHER, Thilo: Das naechste Spiel ist immer das schwerste. In: *Javamagazin* 05 (2004), S. 89

Frotscher 2004c
FROTSCHER, Thilo: Das Runde muss ins Eckige. In: *Javamagazin* 06 (2004), S. 60

Goodwill 2004
GOODWILL, Richard: *Professional Jakarta Struts*. Wiley Publishing Inc., 2004

Herold u. Meyer 1995
HEROLD, Helmut ; MEYER, Manfred: *SCCS und RCS. Versionsverwaltung unter UNIX*. Addison-Wesley, München, 1995. – ISBN 3893197540

Krüger 2003
KRÜGER, Guido: *Handbuch der Java Programmierung*. Addison-Wesley, 2003 (3-8273-2120-4)

Price 2005
PRICE, Derek R.: *Version Management with CVS*, 2005. `http://ximbiot.com/cvs/manual/cvs-1.11.21/cvs.html`

Ramachandran 2003
RAMACHANDRAN, Shantha: *SWT-Basic-Tutorial*, 2003. `http://www.cs.umanitoba.ca/~eclipse/2-Basic.pdf`

Rechenberger 1994
RECHENBERGER, Christoph: *Konzepte und Verfahren für die Software-Versionsverwaltung*. Universitätsverlag Rudolf Trauner, 1994

Robinson 2004
ROBINSON, Ellen: *Jakarta Struts for Dummies*. Wiley Publishing, Inc., 2004

Wikipedia 2005
WIKIPEDIA: *Wikipedia - Enzyklopädie*. Version: 2005. `http://de.wikipedia.org`. – Online Ressource

Wolff 2004
WOLFF, Michael: *Struts Gepackt*. Mitp-Verlag, 2004