Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Theoretische und Technische Informatik

Fachgebiet Prozessinformatik

Studienarbeit

„Erstellen eines Backup - Moduls unter Verwendung gängiger Design Patterns aus dem ResMedLib Framework.“

Bearbeiter:	Dirk Behrendt
Matrikelnummer:	26726
Matrikel:	M 97
Studiengang:	Informatik
Wissenschaftlicher Betreuer:	Dipl. Ing. Christian Heller

Ilmenau, 28.01.2003

Inhaltsverzeichnis

Abbildungsverzeichnis

Vorwort

Hauptaufgabe des theoretischen Teils der Studienarbeit ist die Erstellung eines Überblicks über gängige Entwurfsmuster der Softwareentwicklung. Der praktische Teil der Studienarbeit beinhaltet die Implementierung eines Backup - Moduls nach dem Model - View - Controller - Muster.

Im *Kapitel 1* wird die Programmiersprache Java besprochen. Nach einem kurzen historischen Abriss, wird auf die Sprachmerkmale von Java und auf die Java API eingegangen werden.

Das Thema Entwurfsmuster ist Gegenstand des *2. Kapitels*. Es werden die gängigen Entwurfsmuster - Kataloge von Frank Buschmann und Erich Gamma beleuchtet und ausgewählte Muster näher beschrieben. Größere Aufmerksamkeit erfährt das Model - View - Controller - Konzept, welches auch im ResMedLib Framework Verwendung findet und bei der Implementierung des Moduls anzuwenden war.

Kapitel 3 beschäftigt sich mit dem Res Medicinae Projekt. Dabei wird auf die Einsatzmöglichkeiten und die Architektur des ResMedLib Frameworks näher eingegangen.

Der praktische Teil der Studienarbeit wird im *Kapitel 4* besprochen. Es wurde ein Backup Tool implementiert, welches auf der Grundlage des ResMedLib Frameworks im Rahmen des Projektes Res Medicinae entstand. Die Funktionalität des Moduls wird beschrieben. Neben einigen Screenshots, wird auf die Umsetzung und Implementierung eingegangen.

Eine Zusammenfassung und ein Ausblick werden in *Kapitel 5* gegeben.

Kapitel 1

Grundlagen

1.1 Java

In diesem Kapitel wird ein kurzer geschichtlicher Abriss stattfinden, bevor auf die Sprachmerkmale und die Java API eingegangen wird.

1.1.1 Geschichte von Java

1990 wollte eine Gruppe von Programmierern bei SUN eine an C++ angelehnte, jedoch bei weitem einfachere und für die Programmierung spezieller Anwendungen geeignete Programmiersprache entwickeln und im Green Project von SUN anwenden. 1993 erkannte das Projekt die Eignung von Java für den Einsatz im World Wide Web. Das Java Projekt erhielt bei SUN einen höheren Stellenwert und der Erfolg von Java rettete SUN vor dem Bankrott. Später entwickelte SUN den Browser "HotJava", welcher Javacode interpretieren konnte. Inzwischen haben IBM, Borland, Oracle, Microsoft und andere Firmen Java lizenziert, um eigene Anwendungen damit zu entwickeln.

1.1.2 Sprachmerkmale von Java

Die Autoren von Java haben ein Diskussionspapier (White Paper) verfasst, welches Entwicklungsziele und Realisierungen erläutert.
In [Hor99] werden folgende 11 Merkmale aufgelistet.

(1) Einfach
Die Syntax von Java ist eine bereinigte Version der C++ Syntax. Für C++ Programmierer ist der Übergang zu Java relativ einfach. So entfallen z. B. Header - Dateien, Zeigerarithmetik oder das Überladen von Operatoren.

(2) Objektorientiert
Heutzutage ist eine moderne Programmiersprache undenkbar ohne Objektorientierung. Im Gegensatz zu C++ bietet Java keine Mehrfachvererbung.

(3) Verteilt
Die Netzwerkfähigkeiten von Java sind gut ausgebaut und leicht einzusetzen. Servlets machen die serverseitige Verarbeitung in Java sehr effizient.

(4) Robust
Der Java - Compiler erkennt viele Probleme, die bei anderen Sprachen nur zur Laufzeit aufgetreten sind. Java benötigt keine Zeiger für Konstrukte wie Strings oder Arrays. Dennoch kann man bei Bedarf auf die Leistungsfähigkeit von Zeigern zugreifen, so z. B. bei verketteten Listen.

(5) Sicher
Java ist für verteilte und Netzwerk - Umgebungen vorgesehen. Somit stehen Sicherheitsaspekte im Vordergrund. Virusfreie und gegen unbefugte Eingriffe gesicherte Systeme lassen sich mit Java konstruieren.

(6) Architekturneutral
Der Java - Compiler generiert einen Bytecode, welcher auf den meisten heutigen Computerarchitekturen arbeitet. Jedoch gehen Bytecodes zu Lasten der Ausführungsgeschwindigkeit.

(7) Portabel
Sowohl die Größen der einfachen Datentypen als auch die dafür definierten arithmetischen Operationen sind genau festgelegt. Die Bibliotheken definieren portable Schnittstellen. So

gibt es z. B. eine abstrakte Window - Klasse mit entsprechenden Implementierungen für Unix, Windows und den Macintosh.

(8) Interpretiert
Der Java - Interpreter kann Java - Bytecode direkt auf jeder Maschine ausführen, auf die der Interpreter portiert wurde. Die Übersetzung erfolgt relativ langsam.

(9) Hohe Leistung
Just - In - Time Compiler kompilieren die Bytecodes einmalig in nativen Code, speichern die Ergebnisse zwischen und rufen sie bei Bedarf wieder ab. Somit werden Schleifen-konstruktionen erheblich beschleunigt. Der Just - In - Time Compiler bewirkt bei bestimmten Programmen eine 10- bis 20fache Geschwindigkeitssteigerung und ist schneller als der Java - Interpreter.

(10) Multithreaded
Vorzüge des Multithreading sind bessere interaktive Reaktion und besseres Echtzeitverhalten. Im Vergleich zu anderen Sprachen ist dieses in Java relativ einfach zu realisieren. Java überträgt die Implementierung des Multithreading auf das zugrunde liegende Betriebsystem.

(11) Dynamisch
Java zielt darauf ab, sich an eine entwickelnde Umgebung anzupassen. Es lassen sich die Typinformationen zur Laufzeit unkompliziert ermitteln. Java lädt seine Klassen erst, wenn sie benötigt werden.

1.1.3 Java API

Als *API* bezeichnet man das *Application Programming Interface*, also die Programmier-schnittstelle einer Klasse, eines Pakets oder einer ganzen Bibliothek. In Abbildung 1.1 sind einige API's dargestellt.

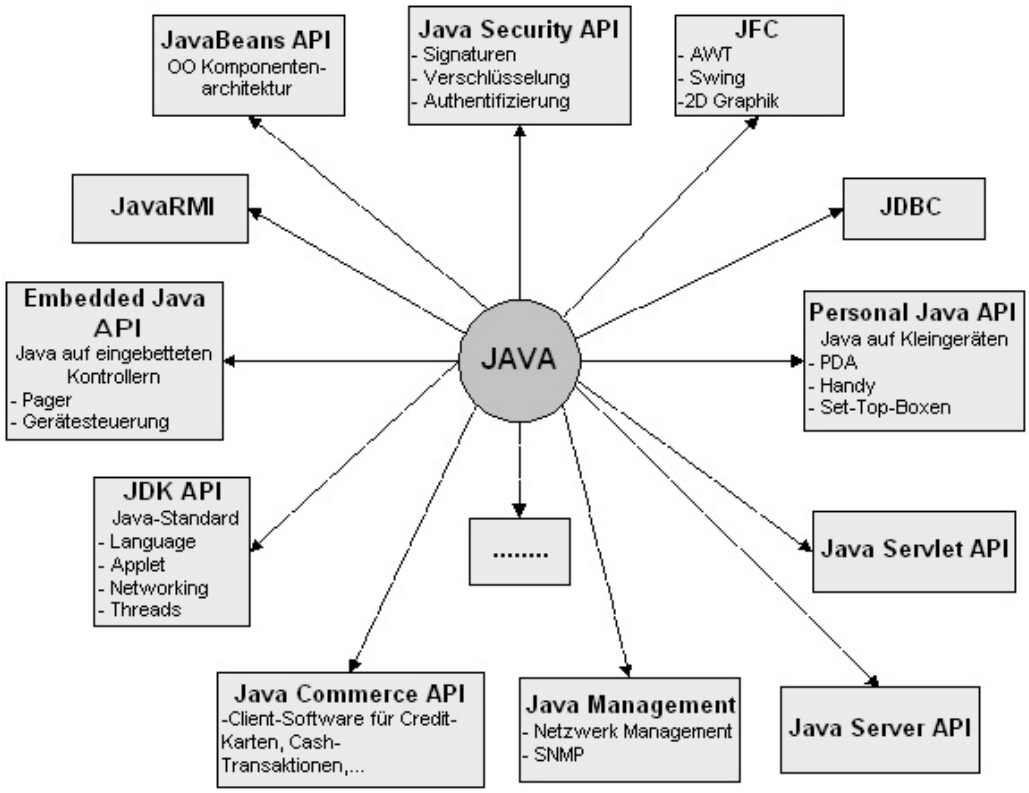

Abbildung 1.1: Ein Überblick über die Java - API

Der Erfolg von Java liegt nicht zuletzt an der engen Verbindung der Sprache zum Internet.

Applets

Applets sind vollständige Java Programme und werden über das Internet verbreitet und können mit Browsern gestartet werden. Dazu wurde HTML um das Applet - Tag erweitert. So kann kompilierter Java Code in normale Web - Seiten eingebunden werden. Ein Java - fähiger Browser enthält eine *JVM (Virtuelle Java - Maschine)* und die Laufzeitbibliothek. Applets ist es nicht gestattet, Dateioperationen auf dem lokalen Rechner durchzuführen oder externe Programme zu starten.

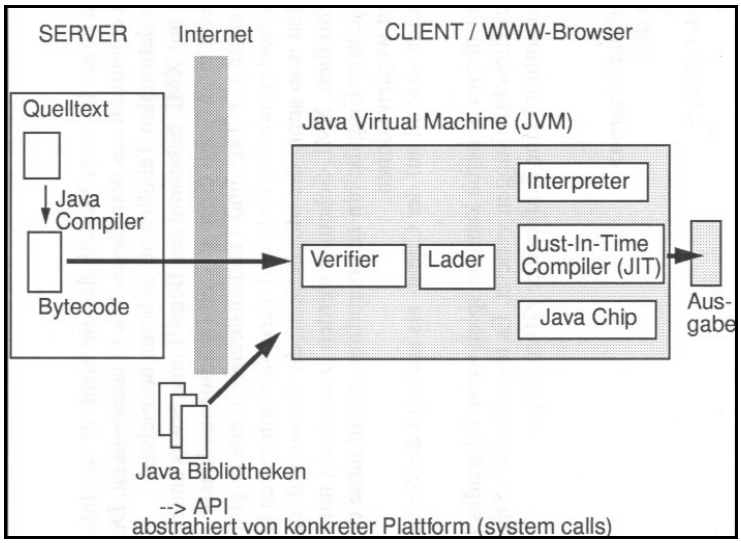

Abbildung 1.2: Ein Blick in die JVM

Die Java - Laufzeitbibliothek bietet umfassende grafische Fähigkeiten. Diese sind im Wesentlichen plattformunabhängig. Somit bietet sich die Möglichkeit der Entwicklung portabler Programme mit *GUI (Graphical User Interface)* - Fähigkeiten. Diese Fähigkeiten werden unter dem Begriff *JFC (Java Foundation Classes)* zusammengefasst. Die drei wichtigsten Komponenten sind das *AWT (Abstract Window Toolkit),* das *Swing Toolset* und das *Java 2D API*.

AWT

Das AWT stellt elementare Grafik- und Fensterfunktionen auf der Basis der auf der jeweiligen Zielmaschine verfügbaren Fähigkeiten zur Verfügung. AWT Komponenten werden als *heavyweight* bezeichnet, da sie direkt aus dem Betriebsystem kontrolliert werden. Da alle Fenster- und Dialogelemente durch das Betriebsystem bereitgestellt werden, ist es sehr schwierig, ein plattformübergreifendes Look - and - Feel zu realisieren. Im AWT gibt es nur eine Grundmenge an Dialogelementen. Aufwendige Benutzeroberflächen sind nur mit sehr viel Zusatzaufwand realisierbar. Bereits kurze Zeit nach dem das AWT eingeführt wurde, begann man über Verbesserung nachzudenken und mit der Entwicklung von Swing zu beginnen.

Swing

Swing benutzt nur noch in eingeschränkter Weise plattformspezifische GUI - Ressourcen. Fast alle GUI – Elemente werden von Swing selbst erzeugt. Swing Komponenten werden auch als *lightweight* bezeichnet, da sie nur innerhalb des Swing Frameworks existieren.

Pluggable Look - and - Feel

Es besteht die Möglichkeit das Aussehen und die Bedienung einer Anwendung zur Laufzeit umzuschalten. Einem Windows Anwender stehen drei verschiedene Look - and - Feel Modi zur Verfügung.

Das Model - View - Controller - Prinzip (MVC)

Dies ist die wichtigste Errungenschaft von Swing. Anstatt den gesamten Code in eine Klasse zu packen, werden beim MVC - Konzept drei unterschiedliche Bestandteile eines grafischen Elements unterschieden. Eine ausführliche Beschreibung dieses Konzepts wird in Abschnitt 2.4 vorgenommen.

Neben den Vorteilen zieht die Verwendung von Swing auch Probleme mit sich. Swing - Anwendungen sind ressourcenhungrig. Viel CPU - Leistung und Hauptspeicher wird erforderlich, da alle Komponenten selbst gezeichnet werden müssen.

JDBC
JDBC (Java Database Connectivity) ist ein standardisiertes Java - Datenbank - Interface und bietet den Zugriff auf relationale Datenbanken. Die JDBC - Architektur zeichnet sich durch ihre datenbankneutrale Zugriffsschnittselle aus. So ist es zum Beispiel möglich eine JDBC - Anwendung, welche für eine Oracle Datenbank entwickelt wurde, durch einfaches Austauschen des Datebanktreibers mit DB2 einzusetzen.

Beans
Als Beans werden eigenständige, wieder verwendbare Softwarekomponenten zum Aufbau von Applets und Applikationen bezeichnet. Diese werden typischerweise als grafische Komponenten zur Verfügung gestellt und können mittels eines GUI - Editors interaktiv zu komplexen Anwendungen zusammengesetzt werden. Mit Beans steht eine plattform-übergreifende Komponentenarchitektur zur Verfügung.

JSP
Mit Hilfe von JSP (Java Server Pages) kann man dynamische Webseiten erzeugen. Dabei wird ein reguläres HTML - Dokument geschrieben und Code für dynamische Teile in speziellen Tags in das HTML - Dokument eingebettet.

Servlets
Servlets sind Programme, welche auf einem Webserver laufen, um sowohl mit Clients als auch Datenbank- und Applikations - Servern eine Kommunikation aufnehmen zu können.

Kapitel 2

Entwurfsmuster (Design Patterns)

2.1 Was sind Entwurfsmuster?

Der Entwurf wieder verwendbarer objektorientierter Software ist schwer. Ein Entwurf muss den vorliegenden spezifischen Anforderungen genügen, jedoch auch allgemein genug sein, um zukünftigen Problemen und Anforderungen entgegen treten zu können. Man muss nicht jedes Problem von Grund auf neu angehen, sondern kann Lösungen verwenden, die zuvor erfolgreich eingesetzt wurden.

Folgender Vergleich aus der Medizin soll dies verdeutlichen:
Wenn ein Patient mit einer Bewegungseinschränkung an einem Gelenk die Physiotherapie aufsucht, so ist die Vorgehensweise der Physiotherapeuten in der Regel immer die gleiche. Zuerst erfolgt die Anamnese. Dann wird der Patient gebeten Aussagen zum Schmerzverhalten zu tätigen. Also das zum Beispiel die rechte Schulter beim Heben des Arms schmerzt. Anschließend erfolgt die Untersuchung spezieller Strukturen, wie der Muskeln. Nach diesen Maßnahmen kann die Ursache für den Schmerz eingegrenzt werden und eine geeignete Therapie beginnen.

In vielen objektorientierten Systemen lassen sich wiederkehrende Muster von Klassen und kommunizierenden Objekten finden. Entwurfsmuster vereinfachen die Wiederverwendung von erfolgreichen Entwürfen und Architekturen. Mit Hilfe von Entwurfsmustern kann das Wissen erfahrener Software Entwickler genutzt werden. Jedoch muss man nicht für jedes Problem ein Entwurfsmuster verwenden. Der damit verbundene Aufwand ist stets in Betracht zu ziehen. Ein Entwurfsmuster beschreibt ein häufig auftretendes Entwurfsproblem und präsentiert ein erprobtes Schema zu seiner Lösung. Dieses Lösungsschema spezifiziert die beteiligten Komponenten, ihre Zuständigkeiten, ihre Beziehungen und ihre Kooperationsweise.

Ein Muster unterliegt einem dreiteiligen Schema:

Kontext
Dieser enthält die Beschreibung der Situation, in der das Problem auftritt.

Problem
Beschreibung des Problems, welches im Kontext immer wieder auftritt.

Lösung
Aufzeigen einer erprobten Lösung für das Problem.

Inzwischen gibt es mehrere hundert Entwurfsmuster. Die wichtigsten Entwurfsmuster - Kataloge sind: [Uni02]

- "Entwurfsmuster" von Gamma et al., Addison - Wesley ([Gam96])
- "Pattern - orientierte Software-Architektur" von Frank Buschmann, Addison - Wesley ([Bus98])
- "Patterns in JAVA Volume 1 - 3" von Mark Grand, Wiley ([Gra98], [Gra99], [Gra01])

Auf den Entwurfsmuster - Katalog von Mark Grand wird in dieser Arbeit nicht näher eingegangen. Der Autor hat bis heute 3 Bände dazu veröffentlicht, in denen über 130 Muster besprochen werden.

2.2 Entwurfsmuster - Katalog von Frank Buschmann

Im Buch von Frank Buschmann unterscheidet man Architekturmuster, Entwurfsmuster und Idiome.

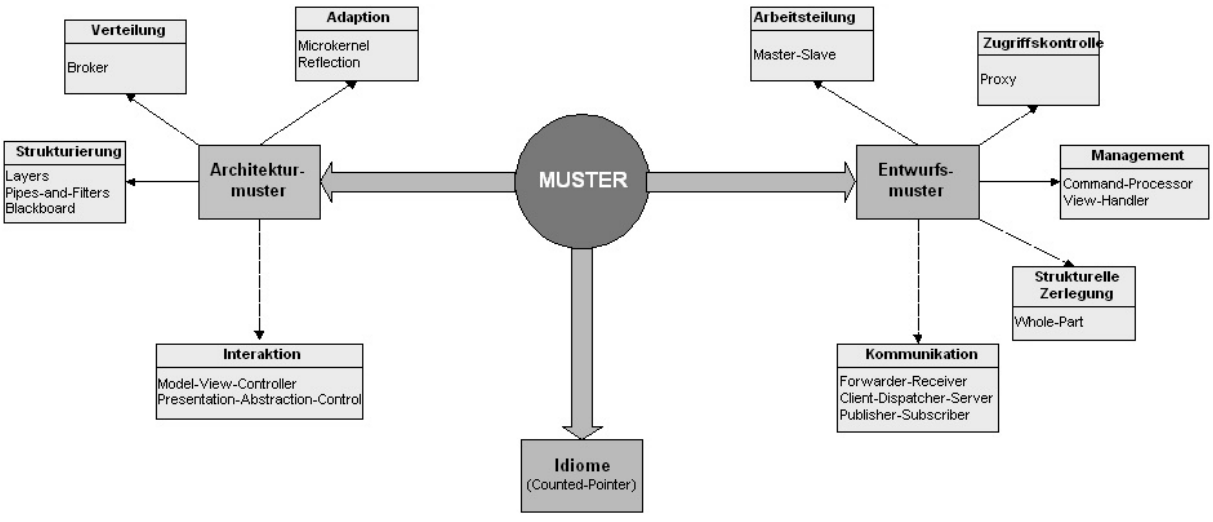

Abbildung 2.1: Entwurfsmuster - Katalog von Frank Buschmann

2.2.1 Architekturmuster

Ein Architekturmuster spiegelt die Strukturierung eines Softwaresystems wieder. Dieses beinhaltet eine Menge von Subsystemen und Beziehungen zwischen diesen. Die Auswahl eines Architekturmusters ist eine Grundsatzentscheidung im Entwurf eines Softwaresystems. Architekturmuster befinden sich auf der höchsten Abstraktionsebene. So lassen sich grundsätzliche Strukturen einer Anwendung spezifizieren.

Die Architekturmuster gruppiert man in vier Kategorien:

Strukturierung
Diese Kategorie beinhaltet eine sinnvolle Zerlegung einer Aufgabe in kooperierende Teilaufgaben.

- Layers - Muster
- Pipes - and - Filters - Muster
- Blackboard - Muster

Das *Layers - Muster* beschreibt das am meisten verbreitete Prinzip, um auf der Architekturebene ein System zu unterteilen. Es lassen sich Anwendungen strukturieren, welche in Gruppen von Teilaufgaben zerlegbar sind. Die International Standardization Organization (ISO) definierte das OSI - 7 - Schicht Modell (Open System Interconnection). Jede Schicht behandelt einen bestimmten Aspekt der Kommunikation und greift dabei auf die Dienste der darunter liegenden Schicht zurück.

Das *Pipes - and - Filters - Muster* bietet eine Struktur für Systeme, die Datenströme verarbeiten. Jeder Verarbeitungsschritt wird in einer eigenen Filterkomponente gekapselt und die Daten werden durch Kanäle weitergegeben. Durch die Neuanordnung der Filter, können Familien verwandter Systeme gebildet werden.

Das *Blackboard - Muster* kommt aus dem Bereich der künstlichen Intelligenz und hilft bei Problemen, für welche keine deterministischen Lösungsstrategien bekannt sind.

Verteilung
Diese Kategorie beinhaltet die Realisierung einer Infrastruktur für verteilte Anwendungen.

- Broker - Muster

Das *Broker - Muster* ist für die Strukturierung verteilter Softwaresysteme mit entkoppelten Komponenten nutzbar, welche durch das Aufrufen entfernter Dienste interagieren. Der Vermittler (Broker) trägt die Verantwortung für die Koordination der Kommunikation, sowie für die Übertragung von Ergebnissen und Fehlermeldungen.

Interaktion
Diese Kategorie beinhaltet die Strukturierung von interaktiven Systemen. Die Benutzerfreundlichkeit von Anwendungen soll verbessert werden. Der funktionale Kern ist unabhängig von der Benutzerschnittstelle zu halten. Dieser bleibt in der Regel stabil, während Benutzerschnittstellen häufig angepasst und verändert werden.

- Model - View - Controller - Muster (MVC)
- Presentation - Abstraction - Control - Muster

Das *Model - View - Controller - Muster (MVC)* ist die bekannteste Organisation einer Architektur für interaktive Softwaresysteme und wird in Abschnitt 2.4 näher erläutert.

Das *Presentation - Abstraction - Control - Muster* definiert eine Struktur für interaktive Softwaresysteme in Form einer Hierarchie kooperierender Agenten (informations-verarbeitende Komponente). Jeder Agent trägt für einen bestimmten Aspekt der Funktionalität der Anwendung Verantwortung. Alle Agenten zusammen, stellen die Systemfunktionalität zur Verfügung.

Adaption
Diese Kategorie beinhaltet die Erweiterung von Anwendungen und ihre Anpassung an sich weiterentwickelnde Technologien. Ein für Veränderung offener Entwurf ist eine wichtige Aufgabe bei der Entwicklung der Architektur eines Softwaresystems.

- Microkernel - Muster
- Reflection - Muster

Bei Softwaresystemen, welche sich an veränderte Systemanforderungen anpassen müssen, findet das *Microkernel - Muster* Anwendung. Ein minimal funktioneller Kern wird von der erweiterten Funktionalität getrennt. Dieses Muster dient als Basisarchitektur für mehrere moderne Betriebsysteme, wie zum Beispiel Windows NT und stellt eine Plug - & - Play - Software - Umgebung zur Verfügung.

Das *Reflection - Muster* stellt einen Mechanismus für die dynamische Veränderung der Struktur und des Verhaltens von Softwaresystemen bereit. Das System besitzt Informationen über sich selbst und benutzt diese, um veränderbar und erweiterbar zu bleiben. Die Prinzipien der Reflexion werden von verschiedenen Programmiersprachen wie Smalltalk und Betriebssystemen oder industriellen Großapplikationen unterstützt.

2.2.2 Entwurfsmuster

Subsysteme einer Softwarearchitektur bestehen in der Regel aus mehreren kleinen Einheiten. Diese werden mit Entwurfsmustern beschrieben. Entwurfsmuster beschreiben ein Schema zur Verfeinerung von Subsystemen oder Komponenten eines Softwaresystems oder den Beziehungen zwischen diesen. Sie sind auf der mittleren Abstraktionsebene anzusiedeln.

Folgende Gruppierung kann vorgenommen werden:

Strukturelle Zerlegung
Diese Kategorie beinhaltet Muster, welche Subsysteme und komplexe Komponenten in geeignete miteinander kooperierende Teile zerlegen. Subsysteme und komplexe Komponenten lassen sich leichter bearbeiten, wenn sie in kleinere, voneinander unabhängige Komponenten aufgeteilt sind.

- Whole - Part - Muster

Das *Whole - Part - Muster* unterstützt die Zusammenfassung von Komponenten, welche zusammen eine semantische Einheit bilden. Diese Einheit kapselt die Teilobjekte und steuert deren Kooperation. Die Teilobjekte sind nicht direkt ansprechbar. Fast jedes Softwaresystem enthält Komponenten oder Subsysteme, die mit Hilfe dieses Musters organisiert werden können.

Kooperation
In dieser Kategorie zusammengefassten Muster definieren, wie Komponenten zusammenarbeiten können. Komplexe Dienste werden sehr oft mittels einer Menge miteinander kooperierender Komponenten realisiert. Jede Komponente muss einen klar definierten Aufgabenbereich besitzen.

- Mater - Slave - Muster

Das *Mater - Slave - Muster* arbeitet nach dem Prinzip „teile und herrsche“. Eine Aufgabe wird von dem Master in verschiedene Teilaufgaben zerlegt, welche die Slaves zu bearbeiten haben. Das Endergebnis wird aus den Teilergebnissen berechnet. Dieses Muster findet in parallel- und verteilt arbeitenden Systemen Anwendung.

Zugriffskontrolle
Muster, welche dieser Kategorie zugeordnet sind, bewachen und kontrollieren den Zugriff auf Dienste und Komponenten. Clients sollten nicht immer direkt auf Komponenten oder Subsysteme zugreifen können.

- Proxy - Muster

Das *Proxy - Muster* führt einen Stellvertreter für eine Komponente ein. Somit kommunizieren Clients nicht mit der Komponente direkt, sondern mit deren Stellvertreter. In fast allen verteilten Systemen findet dieses Muster Verwendung.

Management
Mit Objekten, Diensten und Komponenten als Ganzes können Muster umgehen, welche in dieser Kategorie enthalten sind. Softwaresysteme enthalten viele Objekte, Dienste und komplexe Komponenten, welche einander ähnlich sind. Zum Beispiel die von Benutzern oder Systemen ausgelösten Ereignisse. Manager - Komponenten bearbeiten diese Objektmengen.

- Command - Processor - Muster
- View - Handler - Muster

Das *Command - Processor - Muster* trennt die Anforderung eines Dienstes von seiner Ausführung. Ein Befehlsverwalter verwaltet Anforderungen als eigene Objekte, steuert ihre Ausführung und stellt weitere Dienste zu Verfügung.

Das *View - Handler - Muster* verwaltet mehrere Ansichten auf den Datenbestand eines Softwaresystems. Clients können mittels eines Ansichtenverwalters Ansichten offnen, verändern und schließen, koordinieren Abhängigkeiten verschiedener Ansichten und realisieren ihre Aktualisierung.

Kommunikation
Muster dieser Kategorie unterstützen bei der Organisation von Kommunikationsaufgaben zwischen verschiedenen Komponenten eines Softwaresystems. Die meisten Softwaresysteme sind auf ein Netz von Computern verteilt.

- Forwarder - Receiver - Muster
- Client - Dispatcher - Muster
- Publisher - Subscriber - Muster (Observer - Muster)

Das *Publisher - Subscriber - Muster* realisiert die Synchronisation miteinander arbeitender Komponenten. Dafür wird ein Mechanismus zur unidirektionalen Weiterleitung von Änderungen benutzt. Ein Herausgeber benachrichtigt eine beliebige Anzahl von Abonnenten. Dieses Muster ist auch unter dem Namen *Observer - Muster* bekannt.

2.2.3 Idiome

Ein Idiom ist ein Muster, welches auf der niedrigsten Abstraktionsebene anzusiedeln ist. Es beschreibt, wie man bestimmte Aspekte von Komponenten oder Beziehungen zwischen ihnen mit Mitteln einer Programmiersprache implementieren kann. Ein Beispiel für ein solches Problem ist die Speicherverwaltung in C++. Idiome können aber auch direkt die Implementierung eines Entwurfsmusters beschreiben.

Das *Counted - Pointer - Idiom* erleichtert die Speicherverwaltung von dynamisch erzeugten, mehrfach referenzierten Objekten in C++.

2.3 Entwurfsmuster - Katalog von Erich Gamma

Im Buch von Erich Gamma unterscheidet man Erzeugungsmuster, Strukturmuster und Verhaltensmuster. *Klassenbasierte Muster* befassen sich mit Klassen und ihren Unterklassen. Diese Beziehungen werden mittels Vererbung zur Übersetzungszeit festgelegt. Die Objekterzeugung wird in die Unterklassen verlagert. *Objektbasierte Muster* befassen sich mit Objektbeziehungen, welche zur Laufzeit geändert werden können. Die Objekterzeugung wird an ein anderes Objekt delegiert. Die meisten Muster sind objektbasiert. Objektbasierte Muster beschreiben Mittel und Wege, Objekte zusammenzuführen, um neue Funktionalität zu gewinnen.

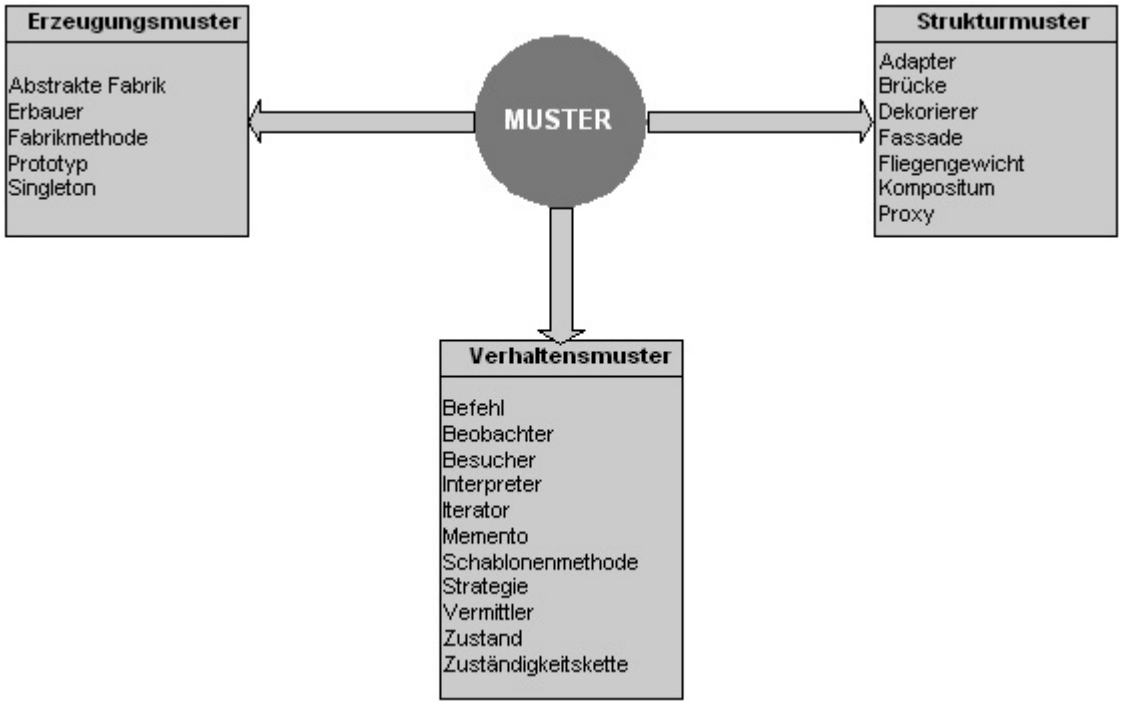

Abbildung 2.2: Entwurfsmuster - Katalog von Erich Gamma

Im Übersetzungsverzeichnis sind die entsprechenden englischen Musternamen aufgeführt.

2.3.1 Erzeugungsmuster

Entwurfsmuster sind von Bedeutung, wenn Systeme beginnen, mehr von Objektkomposition als von Vererbung abzuhängen. Die Erzeugungsmuster hängen eng zusammen. Außer der Abstrakten Fabrik sind alle Erzeugungsmuster in Abbildung 2.2 objektbasiert.

Die *Abstrakte Fabrik* bietet eine Schnittstelle zum Erzeugen von Familien verwandter oder voneinander abhängiger Objekte, ohne ihre konkreten Klassen zu benennen.
Beim *Erbauer - Konzept* wird die Konstruktion eines komplexen Objektes von seiner Repräsentation getrennt. Somit erzeugt derselbe Konstruktionsprozess unterschiedliche Repräsentationen.

2.3.2 Strukturmuster

Durch die Komposition von Klassen und Objekten werden größere Strukturen gebildet. Außer dem Adapter Musters sind alle erwähnten Strukturmuster in Abbildung 2.2 objektbasiert.

Das *Kompositionsmuster* beschreibt die Erstellung einer Klassenhierarchie, welche aus primitiven und zusammengesetzten Objekten besteht. Es lassen sich beliebig komplexe Strukturen zusammenfügen.

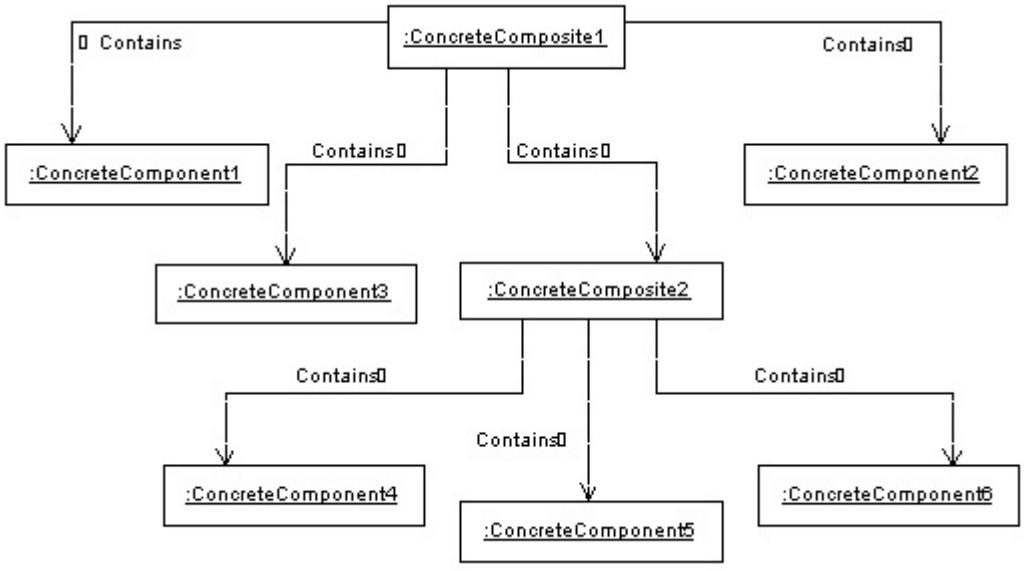

Abbildung 2.3: Das Kompositionsmuster [Gra98]

Das *Fliegengewichtmuster* definiert eine Struktur zur gemeinsamen Nutzung von Objekten aus Gründen der Speicherplatzeffizienz.

Das *Brückenmuster* trennt die Schnittstelle eines Objektes von seiner Implementierung, so dass man beide unabhängig voneinander variieren kann.

Mit dem *Dekorierermuster* kann man Objekte dynamisch um neue Funktionalitäten erweitern. Objekte werden rekursiv zusammengesetzt, um zusätzliche Funktionalität zu erlangen. Viele der Strukturmuster sind bis zu einem gewissen Grad miteinander verwandt.

2.3.3 Verhaltensmuster

Verhaltensmuster befassen sich mit Algorithmen und mit der Zuweisung von Zuständigkeiten zu Objekten. Die Muster beschreiben komplexe Kontrollflüsse. Die Konzentration wird auf die Art und Weise gelenkt, wie die Objekte miteinander interagieren. Das Interpretermuster und die Schablonenmethode sind klassenbasiert. Die restlichen Verhaltensmuster in Abbildung 2.2 sind objektbasiert.

Die *Schablonenmethode* ist eine abstrakte Definition eines Algorithmus. Einzelne Schritte werden an Unterklassen delegiert.

Die *Zuständigkeitskette (Chain of Responsibility)* ermöglicht Anfragen an ein Objekt entlang einer Kette möglicher Kandidaten zu schicken. Jedes Kandidatenobjekt kann in der Lage sein, die Anfrage zu erfüllen.

Das *Beobachtungsmuster* definiert und verwaltet die Abhängigkeiten zwischen Objekten. Zum Einsatz kommt dieses Muster im Model - View - Controller - Konzept.

Das *Strategiemuster* definiert eine Familie gekapselter und austauschbarer Algorithmen. Somit kann der Algorithmus unabhängig von den nutzenden Klienten variiert werden.

2.4 Model - View - Controller - Muster (MVC)

Dieses Konzept wurde als erstes in der Programmiersprache Smalltalk-80 eingeführt.
Das MVC - Paradigma wird durch drei Objekte umgesetzt.

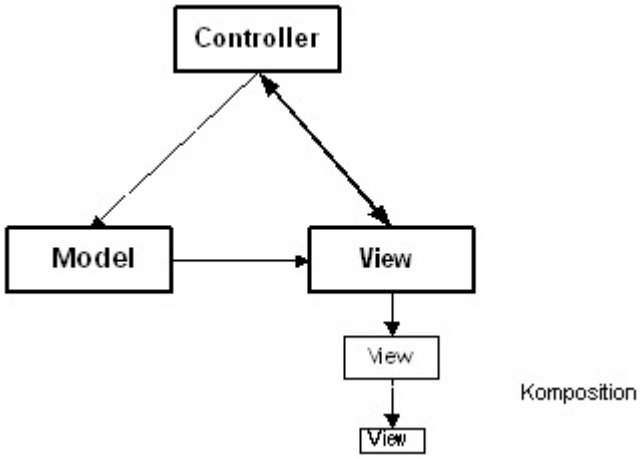

Abbildung 2.4: Das MVC - Konzept

Model - Objekt
- stellt das Anwendungsobjekt dar
- enthält die Kernfunktionalität und die Daten

View - Objekt
- stellt Bildschirmrepräsentation dar
- Objekt erhält die Daten vom Model

Controller - Objekt
- Reaktion auf Benutzereingaben

In den Zeiten vor MVC wurden diese Objekte in einem zusammengefasst. Die Entkopplung erhöht die Flexibilität und Widerverwendbarkeit. Ein View - Objekt muss sicherstellen, dass seine Darstellung den Zustand des Model - Objektes wiedergibt. Das Model benachrichtigt die von ihm abhängigen Views, wenn sich seine Daten ändern. Daraufhin holen sich die Ansichten die neuen Daten und passen sich an. Somit kann man mehrere Views an ein Model binden, um verschiedene Präsentationen zu erlangen. Neue Views sind entwickelbar, ohne das Model selbst ändern zu müssen. Das MVC - Paradigma verwendet Entwurfsmuster, wie zum Beispiel eine Fabrikmethode und einen Dekorierer. Die Fabrikmethode kann dazu genutzt werden, die Klasse der Controller- oder View - Instanzen zur Laufzeit festzulegen. Dekorierer

erweitern das View - Objekt um zusätzliche Funktionalitäten. View und Controller umfassen die Benutzerschnittstelle. Der Anwender interagiert mit der Anwendung über den Controller. Der Controller empfängt Eingaben in Form von Ereignissen, wie zum Beispiel das Klicken eines Buttons. Wenn der Anwender ein Model mittels des Controllers ändert, so müssen alle Ansichten des Models diese Änderung anzeigen. Dieser Mechanismus ist im Publisher - Subscriber - Muster wieder zu finden. Der Controller entspricht dem Strategiemuster. Funktionalität unterliegt einer Kapselung. Das Model enthält den funktionalen Kern der Anwendung. Die Daten werden gekapselt und es werden Operationen angeboten, welche für die Anwendung spezifische Verarbeitung vornehmen. Der Controller ruft diese Operationen im Auftrag des Anwenders auf. Das Model stellt Operationen zum Zugriff auf seine Daten zu Verfügung. Views präsentieren dem Anwender Informationen. Jede Ansicht definiert eine Aktualisierungsoperation, welche vom Mechanismus zur Benachrichtigung über Änderungen aktiviert wird. Weiterhin ist die Schachtelung von Views möglich. Ein Dialog kann eine komplexe View repräsentieren, welche aus weiteren Views besteht. Dies entspricht dem Kompositionsmuster. Während der Initialisierung werden alle Ansichten mit dem Model assoziiert und beim Mechanismus zur Benachrichtigung über Änderungen registriert. View und Controller sind während der Laufzeit ein fest verdrahtetes Paar. Ihnen ist genau ein Modell zugeordnet. Einem Modell können mehrere View/Controller - Paare zugeordnet sein.

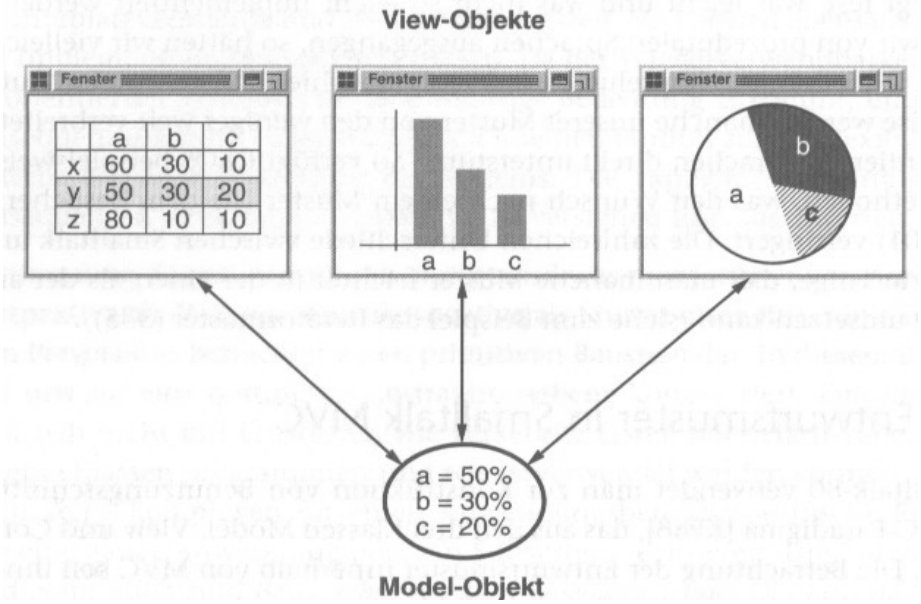

Abbildung 2.5: Unterschiedliche Repräsentation der Model Daten [Gam96]

2.4.1 Hierarchisches Model - View - Controller - Muster (HMVC)

Die Benutzeroberfläche wird in mehrere Schichten unterteilt. Dies entspricht dem Layers - Konzept. Innerhalb dieser Schichten existieren mehrere MVC - Triaden, eine Gruppe bestehend aus drei Objekten. Es werden Eltern - Kind - Beziehungen zwischen den einzelnen Komponenten aufgebaut. Man kann sich vorstellen, dass auf einem Dialog Panels platziert sind, welche wiederum grafische Elemente enthalten. Der Dialog, das Panel und die grafischen Elemente entsprechen jeweils einer MVC - Triade. Der Controller verarbeitet die Ereignisse direkt oder leitet sie an das Controller - Objekt der darunter liegenden Schicht weiter. Dies geschieht solange, bis ein Controller in der Lage ist, das Ereignis zu verarbeiten. Somit entsteht eine Zuständigkeitskette. Mit HMVC verspricht man sich eine Verringerung der Abhängigkeiten zwischen Programmelementen sowie eine einfache Erweiterbarkeit und Wartung.

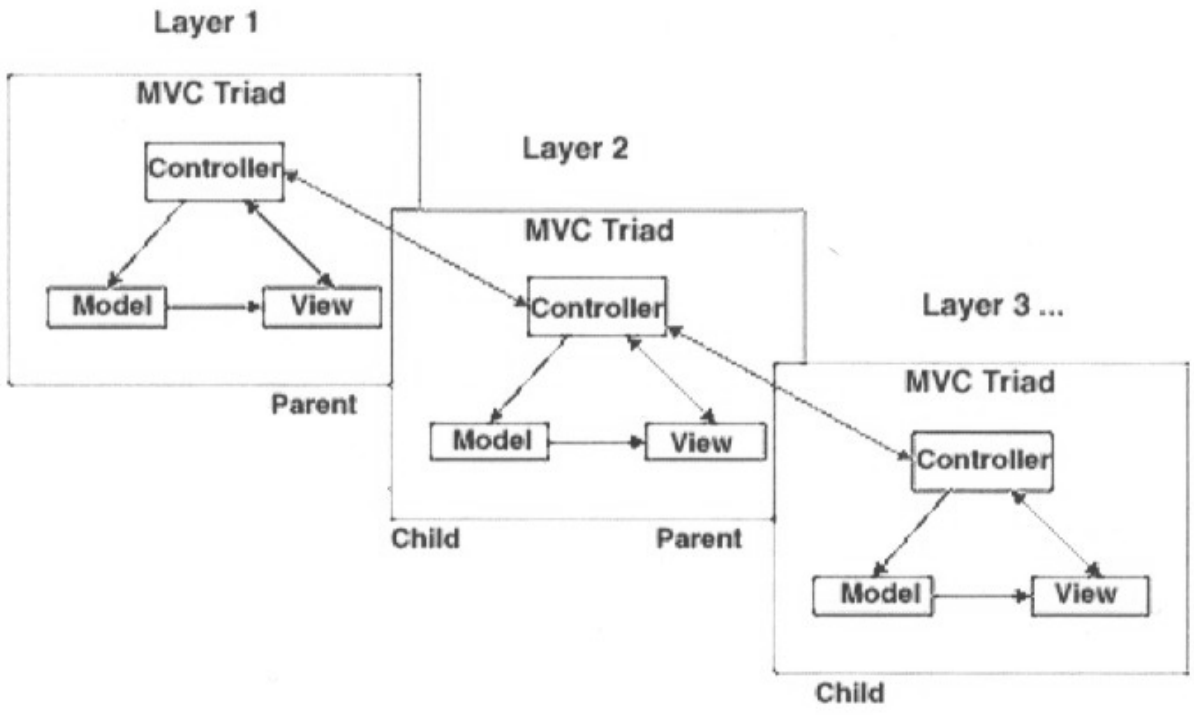

Abbildung 2.6: Das HMVC - Konzept [Jav00]

Hier sind noch einmal die Muster aufgeführt, welche im Model - View - Controller Konzept Verwendung finden.

	Musterart	Mustername	Kapitel
MVC	Entwurfsmuster	Publisher - Subscriber - Muster	2.2.2
	Erzeugungsmuster	Fabrikmethode	2.3.1
	Strukturmuster	Komposition	2.3.2
	Strukturmuster	Dekorierer	2.3.2
	Verhaltensmuster	Verhaltensmuster	2.3.3
HMVC	Architekturmuster	Layers	2.2.1
	Verhaltensmuster	Zuständigkeitskette	2.3.3

Abbildung 2.7: Muster im MVC und HMVC

2.5 Anwendungsbereiche von Mustern

Bei der Besprechung des MVC - Konzeptes wurde bereist aufgezeigt, dass darin gängige Muster Anwendung gefunden haben.

In [Uni02] sind folgende Anwendungsbereiche aufgeführt.

- In der Steuerungs- und Automatisierungstechnik z. B. *"Singleton"*, *"Strategie"*, *"Zustand"*.
- Zur Implementierung von grafischen Oberflächen z. B. *"Beobachter"*, *"Dekorierer"*, *"Zuständigkeitskette"*.
- Für die plattformunabhängige Programmierung bieten sich *"Brücke"*, *"Fassade"*, *"Erbauer"* an.
- Frameworks verwenden häufig:
  "Schablonenmethode", *"Fabrikmethode"*, *"Abstrakte Fabrik"*. Als Anwender eines Frameworks versteht man durch das Studium dieser Entwurfsmuster die Funktionsweise des Frameworks besser.

Kapitel 3

Das Projekt Res Medicinae

Viele Arztpraxen nutzen heutzutage veraltete Software und sind von einem bestimmten Anbieter abhängig. Die Anschaffung neuer Technologien und Anwendungen ist unabdingbar, um konkurrenzfähig zu bleiben, zieht jedoch enorme Kosten nach sich. Aus diesem Missstand heraus entstand eine Vision einer Software. Diese soll stabil, plattformunabhängig, erweiterbar, frei zugänglich für jedermann sein und dabei die neuesten Technologien nutzen können. Ebenso sollen gängige Standards Verwendung finden. Es sollte nicht länger eine Vision bleiben. Das Projekt Res Medicinae wurde ins Leben gerufen, um diese Ziele umzusetzen. Die Software dient der Verwaltung und Darstellung medizinischer Daten, welche zum Beispiel in Arztpraxen anfallen. Res Medicinae ist ein Open - Source - Projekt. Die Software ist also kostenfrei erhältlich. Es wird darauf geachtet, für das Gesamtsystem ausschließlich Open - Source - Programme zu verwenden, um den späteren Anwendern keine Kosten zu verursachen.

3.1 Das ResMedLib Framework

Ein *Framework* ist ein unvollständiges Software-(Sub)System, welches noch instanziiert werden muss. In einer objektorientierten Umgebung besteht es aus einer Reihe von abstrakten und konkreten Klassen. Ein Framework zu instanziieren heißt, die darin enthaltenen Klassen zu kombinieren und Unterklassen abzuleiten. Durch dessen Einsatz, kann man einen sehr hohen Grad an Widerverwendbarkeit erreichen. Langwierige Entwurfsentscheidungen können vermieden werden, was den Softwareentwicklungsprozess wesentlich beschleunigen kann. Jedoch ist der verursachte Einarbeitungsaufwand nicht zu unterschätzen.

Res Medicinae baut auf dem ResMedLib Framework auf. Eine *Ontologie* bildet die Grundlage für dieses Frameworks, welches einer Strukturierung unterliegt und zwischen den Komponenten eine strikte „Teil - Ganzes" - Beziehung vorsieht. Somit steigt die Granularität der Komponenten von Ebene zu Ebene. Solche Strukturen und Darstellung komplexer Zusammenhänge werden als *Ontologien* bezeichnet.

Von der Basisklasse *Item* erben alle anderen Klassen. Elementare Operationen und Eigenschaften werden durch Vererbung an jede andere Klasse des Frameworks weitergegeben. Im Framework erben momentan noch viele Klassen von *Object*. *Item* soll zukünftig die einzig erlaubte direkte Unterklasse von Object sein und die Wurzel in einer hierarchischen Klassenstruktur. Jedes Objekt von *Item* kann Bestandteil eines Baumes, bestehend aus einer beliebigen Anzahl von Instanzen der Klasse Item werden. Dies entspricht der Umsetzung des Kompositionsmusters (Abschnitt 2.3.2, Abbildung 2.3).

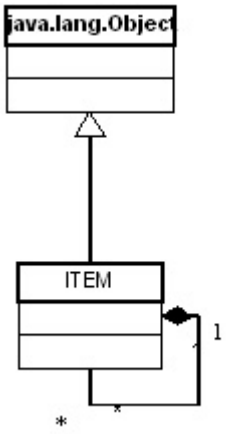

Abbildung 3.1: Die Klasse Item [Boh03]

Entlang jedem Pfad des Baumes sind die Operationen des *Lebenszyklus* einzuhalten. Angelehnt an den natürlichen Lebenszyklus von organischen Zellen, kann man Softwarekomponenten einen ähnlichen Verlauf unterwerfen. Ein Kindobjekt wird immer durch ein Elternobjekt erzeugt und nimmt so am Lebenszyklus der Komponenten teil. Nicht mehr benötigte Komponenten treten aus dem Lebenszyklus aus, indem sie von ihren Elternkomponenten zerstört werden. In Abbildung 3.2 ist die geordnete Erzeugung und Zerstörung einer Komponente dargestellt.

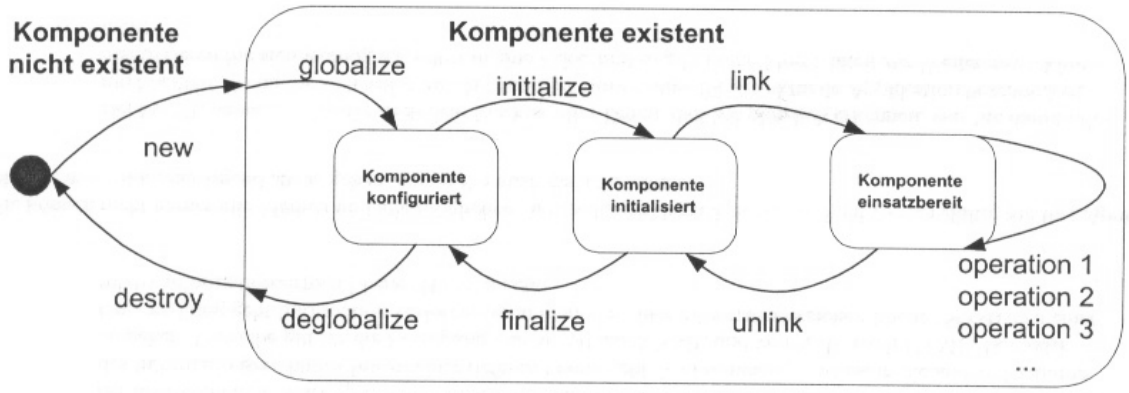

Abbildung 3.2: Der Komponenten Lebenszyklus [Boh03]

Im ResMedLib Framework unterscheidet man zwischen folgenden Ontologien.

Bei der *System - Ontologie* wurde sich an der Organisation des natürlichen Lebens ein Beispiel genommen. Die Festlegung von Unterklassen der Klasse Item, bildet die Grundlage einer System - Ontologie. Ausgehend von einem System (Software - Applikation) unterteilt sich die Struktur in Block, Regien, Komponente, Part und Kette, wobei Systeme zu Familien gruppierbar sind. Klassen einer Ebene dürfen die Klassen einer niederen Schicht benutzen, jedoch nicht umgekehrt.

Die *Model - Ontologie* folgt einer hierarchischen Struktur. Die oberste Ebene bildet *Record*. Objekte höherer Schichten referenzieren stets nur Objekte der gleichen oder niederen Ebene.

In Abbildung 3.3 ist ersichtlich, wie sich bestehende und noch zu implementierende Module in die Gesamtarchitektur eingliedern.

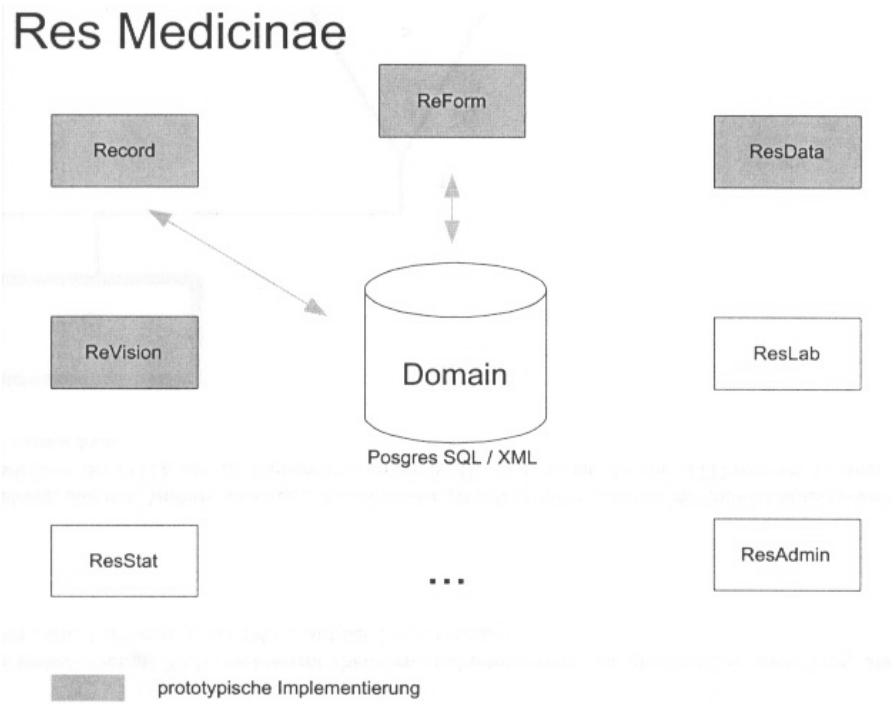

Abbildung 3.3: Module von Res Medicinae [Boh03]

Die Domain - Architektur gliedert sich in drei Ebenen.

- Domain Model (Datenverwaltung)
- Data Mapper (Umschalten zwischen Speicherformaten)
- Persistenz - Schicht (langzeitige Speicherung der Daten)

Jedes Modul ist nach dem MVC - Konzept zu implementieren. Sämtliche Patientendaten werden im Domain Model gehalten und persistent entweder in einer XML - Datei oder in einer Datenbank gespeichert. Der Anwender hat die Möglichkeit, zwischen beiden Speicheformaten zu wählen. Die Daten des Domain Models werden an das Model des hinzugefügten Moduls transferiert. Das Backup Modul, welches als praktische Umsetzung des MVC - Konzeptes erstellt wurde, braucht jedoch nicht auf das Domain Modell zuzugreifen, denn es dient dem Sichern der XML - Dateien, welche der Anwender auf dem Speicher-medium auswählen muss. Die Einordnung des Backup - Moduls in die Klassenhierarchie ist in Abbildung 4.4 vorgenommen worden.

3.2 Open Source Software

Res Medicinae ist ein Open Source Projekt, deshalb wird im Folgenden auf die Vorteile von Open Source Software näher eingegangen.

Open Source Software kann unverändert oder verändert kopiert und verbreitet werden. Open Source beinhaltet also die Möglichkeit der Modifikation und Erweiterung der Quelltexte eines Programms. Folgende Vorteile können angeführt werden.

(1) Mehr Kontrolle
Da jeder Einsicht in den Quellcode nehmen kann, kann das Programm auf Fehlerfreiheit und seine Funktionsweise getestet werden.

(2) Mehr Unterstützung
Große Open Source Projekte, wie der Apache Webserver werden weltweit von hunderttausenden von Anwendern genutzt. Die Vielzahl von Anwendern helfen das Produkt zu testen und zu verbessern, durch die Programmierung von Erweiterungen und die Entdeckung von Fehlern.

(3) Mehr Individualität und Unabhängigkeit
Open Source Software ist auf individuelle Bedürfnisse durch Modifikationen und Erweiterungen des bereits existierenden Quellcodes anpassbar. Mit dem Kauf einer großen Softwareanwendung macht man sich vom Hersteller des Produktes abhängig, da nur er Zugang zum Quellcode hat und auftretende Fehler beseitigen kann. Bei Open Source Software ist man nicht auf den Hersteller angewiesen.

(4) Weniger Kosten
Da die Grundfunktionen im Programm schon vorhanden sind, ist die Erweiterung und Modifikation von Open Source Software schneller und billiger, als die komplette Software selbst zu erstellen. Die eigentliche Software kostet in der Regel nichts.

Kapitel 4

Das Backup Modul

4.1 Verwendungszweck

Die Aufgabe der Studienarbeit bestand in der Implementierung eines Backup - Moduls. Dieses Modul kann sowohl Bestandteil des Res Medicinae Projektes werden, als auch unabhängig davon Benutzung finden. Momentan werden die zu einem Patienten gehörenden Daten in XML - Dateien abgespeichert. Dem Nutzer bietet sich außerdem die Möglichkeit, für die Speicherung der Daten eine Datenbank zu wählen. Eine Datenbank enthält bereits Backup - Funktionalitäten. Der Anwender soll jedoch die Speicherungsform der Patienten-daten frei wählen können. Somit kann er XML - Dateien als Speicherformat wählen, braucht jedoch durch die Verwendung des Backup - Moduls auf Sicherungsmöglichkeiten nicht zu verzichten. Mit dem Backup - Tool können patientenrelevante oder beliebige Dateien gesichert und gegebenenfalls zurück gesichert werden.

4.2 Funktionalität des Moduls

Mit Hilfe des Backup - Moduls bietet sich die Möglichkeit, eine Menge von Dateien, Ordern oder rekursiven Ordern an eine andere Stelle der Festplatte oder auf ein anderes Speichermedium zu sichern. Dieses Repository oder eine Untermenge davon kann rückgesichert werden. Um nicht bei jedem Programmstart die Auswahl neu treffen zu müssen, kann das Repository dauerhaft gespeichert werden. Das Hauptfenster erscheint, nachdem man die Applikation gestartet hat. Man kann den Backup- oder Restore - Prozess starten, nachdem man alle nötigen Einstellungen im Backup Option Fenster oder Restore Option Fenster getroffen hat. Wenn man kein existierendes Repository laden möchte, so muss man den *Storing Path* im Storing Option Fenster wählen. Das ist der Pfad, unter dem die Daten dauerhaft abgespeichert werden.

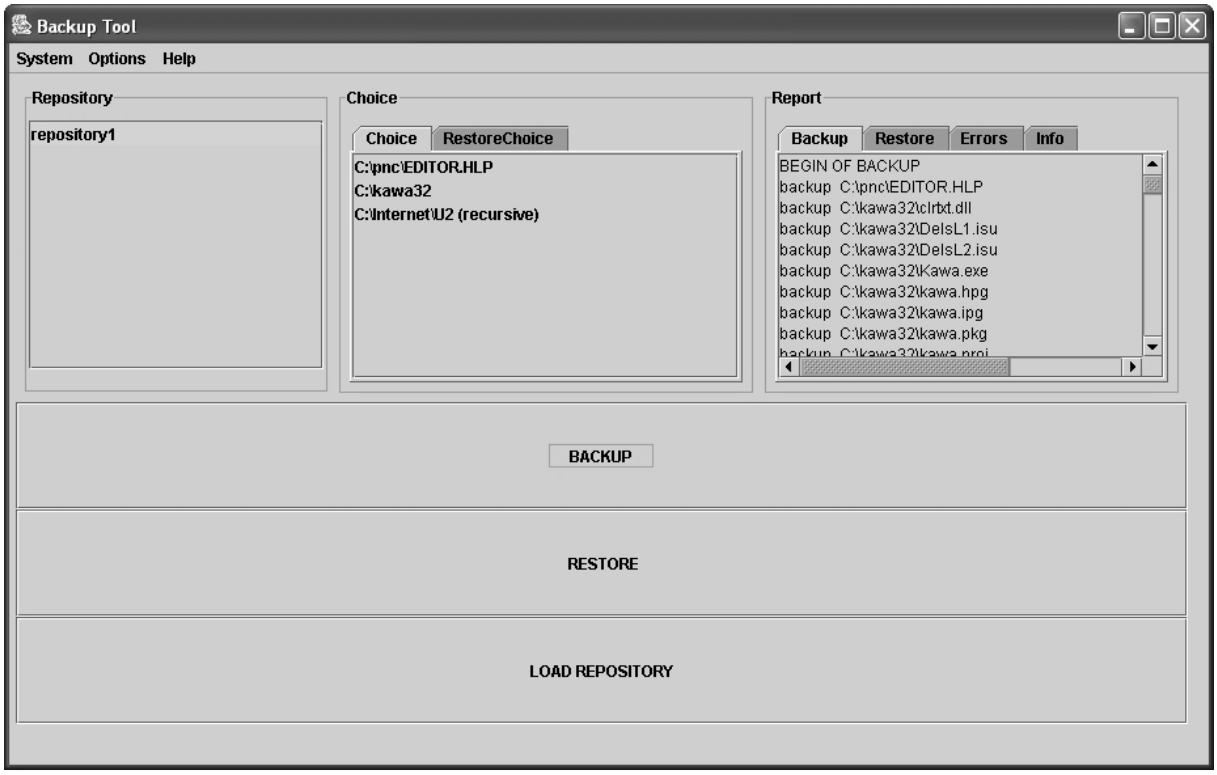

Abbildung 4.1: Das Hauptfenster

Im Backup Option Fenster kann man Repository - Files erstellen und diesen eine Menge von Dateien, Ordnern oder rekursiven Ordnern zuordnen.

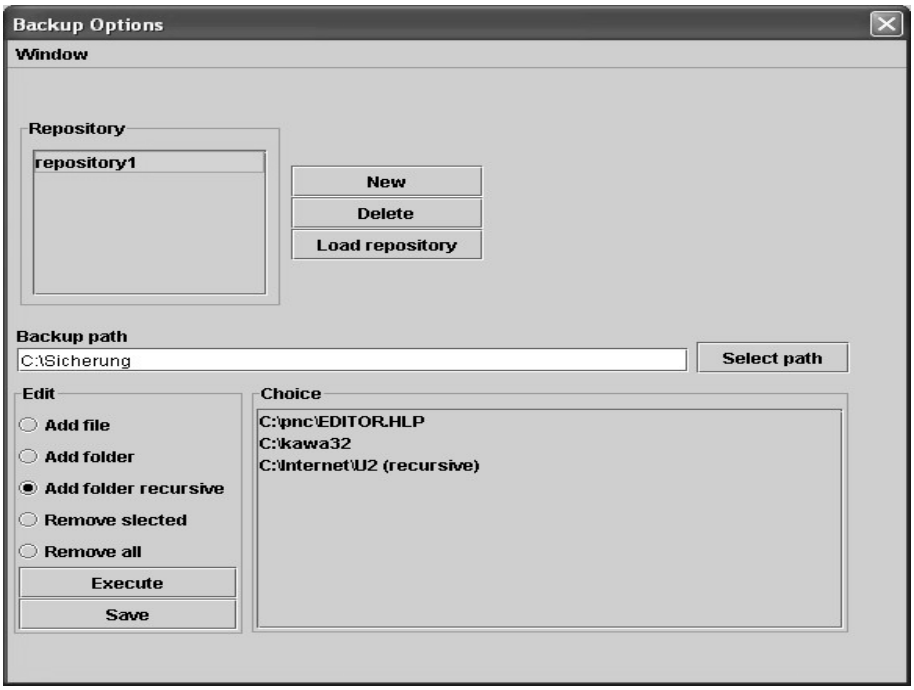

Abbildung 4.2: Das Backup Option Fenster

Im Restore Option Fenster kann man den Inhalt eines kompletten Repository - Files für das Rücksichern auswählen oder eine Untermenge davon.

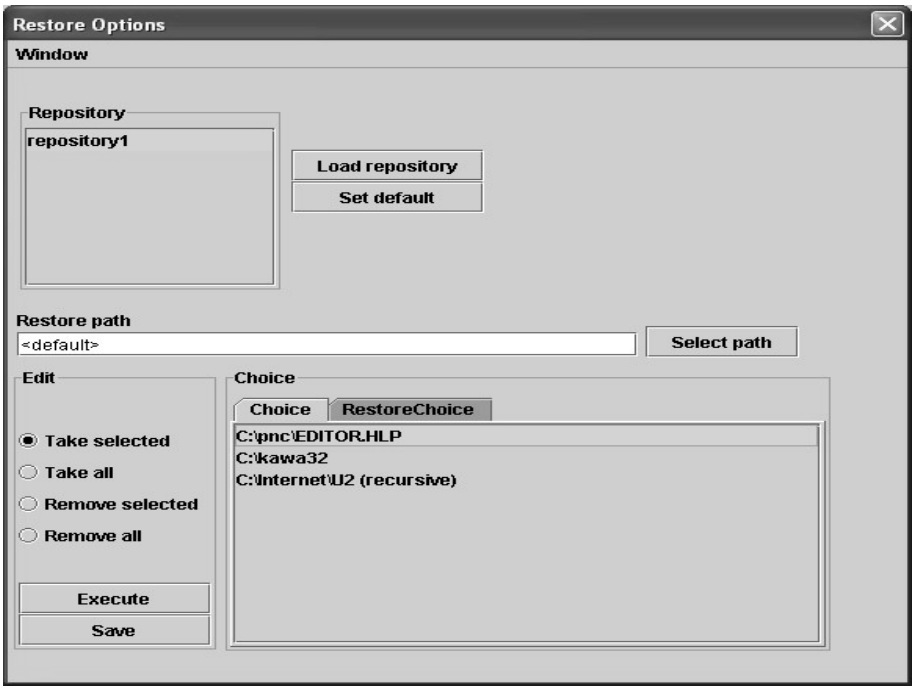

Abbildung 4.3: Das Restore Option Fenster

4.3 Umsetzung und Implementierung

In Res Medicinae wurde auf das Observermuster verzichtet, wodurch der Implementierungsaufwand für den Observer entfällt. Aktualisierungsoperationen werden vom Controller gesteuert.

In der folgenden Abbildung ist die Einordnung der erstellten Klassen in die Vererbungshierarchie des ResMedLib Frameworks ersichtlich. Zu beachten ist, dass sich das Framework in einem ständigen Entwicklungsprozess befindet, so dass diese Übersicht nicht den aktuellen Entwicklungsstand widerspiegelt. Um die Implementierungsarbeit zu erleichtern, wurde auf die Anpassung auf neuere Versionen des Frameworks verzichtet.

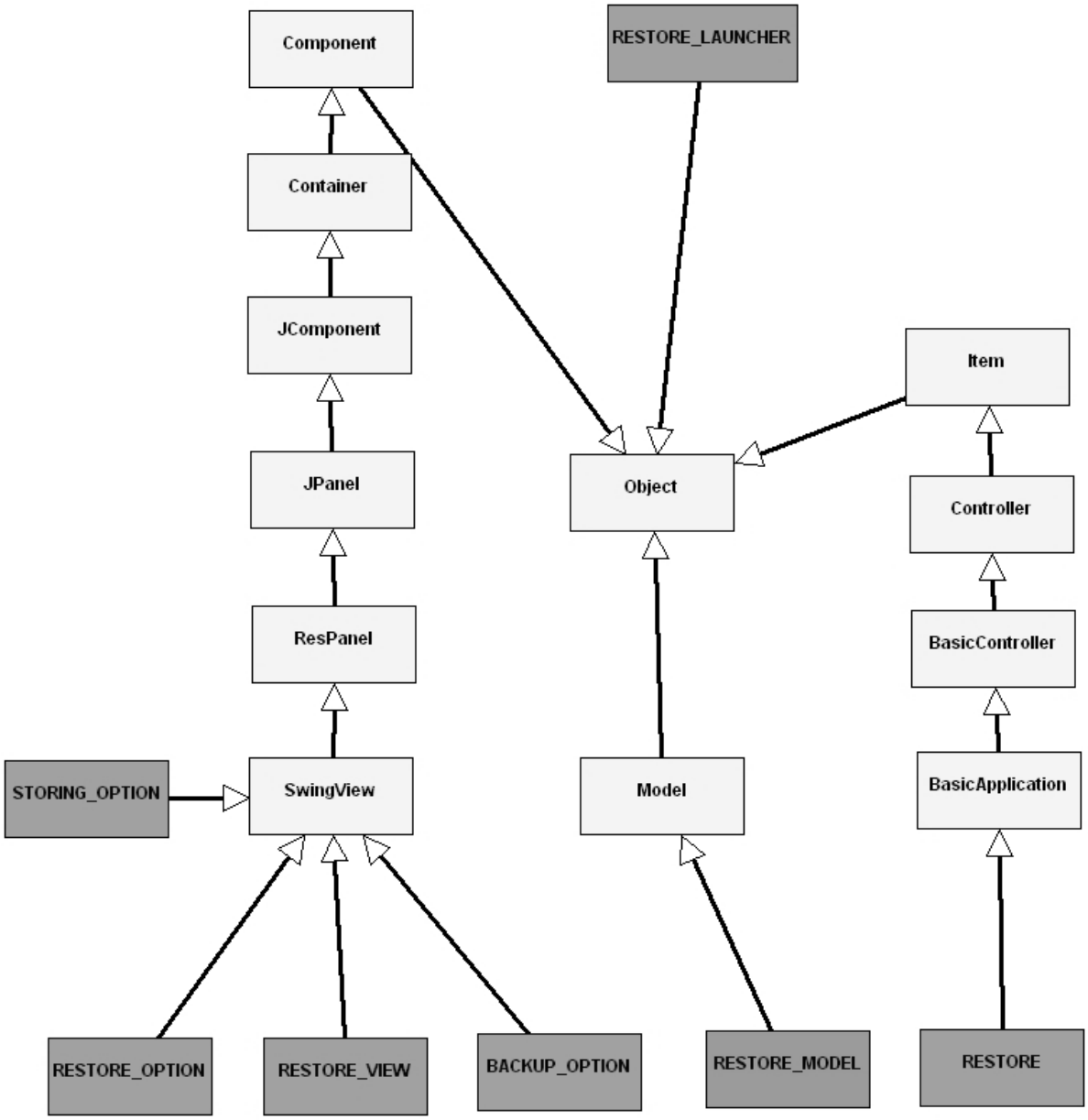

Abbildung 4.4: Einordnung der Klassen in die Vererbungshierarchie

Verwendete Klassen:
RestoreLauncher → Startet die Applikation

Restore → Controller (erbt von BasicApplication)

- Abfangen der Button - Events (Methode *actionPerformed()*)
- Reaktion auf Listenereignisse (Methode *valueChanged()*)
- Bekanntmachung mit dem Model (Methode *createModel()*)
- Bekanntmachung mit der View (Methode *createView()*)

RestoreModel → Model

- Verwaltung der Daten in Vektoren
- Methoden zur Datenmanipulation

- Methoden für den Kopierprozess
- Methoden zur Aktualisierung der Views

RestoreView → View

- Modellierung der Oberfläche mittels Swing
- Methoden zur Aktualisierung der Oberflächenkonstrukte

BackupOption → Sub - View (siehe RestoreView)

RestoreOption → Sub - View (siehe RestoreView)

StoringOption → Sub - View (siehe RestoreView)

RestoreFileFilter → stellt Filter für den FileChooser Dialog bereit

FileChooser → enthält verschiedene FileChooser Dialoge

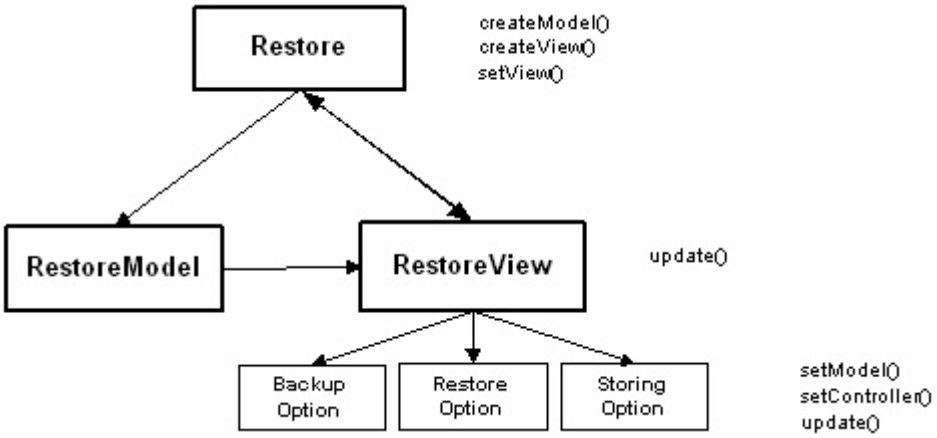

Abbildung 4.5: Einordnung der Klassen in das MVC - Konzept

Mit den Methoden *createModel(), createView()* und *setView(,* wird der Controller mit seinem Model und seiner View bekannt gemacht. Der Controller verarbeitet sämtliche Nutzereingaben und veranlasst die entsprechenden Reaktionen. So erkennt er zum Beispiel, dass zu einer Liste ein Element hinzugefügt wurde und ruft die *update()* - Methode in der entsprechenden View auf, um den Listeninhalt zu aktualisieren. In allen Views existieren update() - Methoden, um Aktualisierungen der Nutzeroberflächen vornehmen zu können. Die Sub - Views BackupOption, RestoreOption und StoringOption müssen mit den Methoden *setModel()* und *setController()* mit dem Model und mit dem Controller bekannt gemacht werden.

Persistente Speicherung der Daten

Das Repository kann dauerhaft gespeichert werden. Dies geschieht mit Hilfe von Textfiles. In dem folgenden Textfile werden die Namen aller Repository - Files abgespeichert. Ebenso der Pfad, unter welchem diese Files zu finden sind.

```
BEGIN_OF_INFORMATION_FILE
BEGIN_OF_REPOSITORY_FILE_INFORMATION
repository1
END_OF_REPOSITORY_FILE_INFORMATION
BEGIN_OF_STORING_PATH
C:\Sicherung
END_OF_STORING_PATH
END_OF_INFORMATION_FILE
```

Abbildung 4.6: Die Datei RestoreInfo.res

Im folgendem ist zu sehen, wie ein Repository - File abgespeichert wird. Dauerhaft wird gesichert:

- Backup Path
- Restore Path
- Choice (=Menge zu sichernder Dateien/Ordner/Rekursiven Ordner)
- Art des Choice - Eintrages (file/folder/recursive folder)
- RestoreChoice (=Menge rückzusichernder Dateien/Ordner/Rekursiven Ordner)
- Art des RestoreChoice - Eintrages (file/folder/recursive folder)

```
BEGIN_OF_FILE
BEGIN_OF_BACKUP_PATH
C:\Sicherung
BEGIN_OF_RESTORE_PATH
<default>
BEGIN_OF_DATA_CHOICE_VECTOR
C:\pnc\EDITOR.HLP
C:\kawa32
C:\Internet\U2
END_OF_DATA_CHOICE_VECTOR
BEGIN_OF_RECURSIVE_CHOICE_BOOLEAN_VECTOR
file
folder
recFolder
END_OF_RECURSIVE_CHOICE_BOOLEAN_VECTOR
BEGIN_OF_DATA_RESTORE_CHOICE_VECTOR
C:\pnc\EDITOR.HLP
END_OF_DATA_RESTORE_CHOICE_VECTOR
BEGIN_OF_RECURSIVE_RESTORE_CHOICE_BOOLEAN_VECTOR
file
END_OF_RECURSIVE_RESTORE_CHOICE_BOOLEAN_VECTOR
END_OF_FILE
```

Abbildung 4.7: Ein abgespeichertes Repository - File

Kapitel 5

Zusammenfassung und Ausblick

Nach der Beschreibung der wichtigen Entwurfsmuster - Kataloge von Gamma und Buschmann, wurde das MVC - Konzept näher betrachtet. Dabei wurde ersichtlich, dass sich dieses Konzept zahlreicher Muster aus den besprochenen Katalogen bedient.

Im praktischen Teil dieser Studienarbeit wurde ein Backup Tool unter Nutzung des Model - View - Controller - Konzeptes im Rahmen des Res Medicinae Projektes erfolgreich implementiert. Jedoch kann das Modul auch unabhängig davon benutzt werden. Bei einer möglichen Weiterentwicklung dieses Moduls können einige Verbesserungen und Erweiterungen vorgenommen werden. Für die dauerhafte Speicherung der Daten könnte auf das XML - Format zurückgegriffen werden. Weiterhin wäre eine Statusanzeige für den Kopierprozess sinnvoll. Außerdem könnte eine History der getätigten Kopierprozesse angefertigt werden. Eine automatische Sicherung zu einem vom Anwender vorgegebenen Zeitpunkt ist ebenso denkbar.

Das Projekt Res Medicinae steht noch am Anfang, beinhaltet jedoch schon einige Modul - Prototypen. Es haben bereits einige Studenten im Rahmen einer Studien- oder Diplomarbeit an diesem Projekt mitgearbeitet. Da der Quellcode frei zugänglich ist, können Entwickler und Nutzer die Module ihren Bedürfnissen anpassen. Durch die Verwendung neuester Technologien und die Plattformunabhängigkeit, wird die Konkurrenzfähigkeit zu bestehenden Produkten gewährleistet. Bis dahin haben noch viele Studenten die Möglichkeit, an diesem Projekt mitzuwirken und sich praxisrelevantes Wissen anzueignen. Wenn sich weiterhin motivierte Studenten für dieses Projekt finden lassen und der finanzielle Rahmen erhalten bleibt, so kann bereits in 2 Jahren eine einsatzfähige Software zur Verfügung stehen. Langfristig gesehen ist die Etablierung von Res Medicinae in verschiedenen Einrichtungen wie Arztpraxen, pharmazeutischen Institutionen, Krankenhäusern, Laboren oder Krankenkassen geplant. Dies sind zwar im Moment nur Visionen, aber warum soll kostenfreie Software schlechter sein als kommerzielle? Das entstehende Produkt soll eine echte Alternative zu kommerziellen Angeboten werden. Im Zeitalter zunehmender Rationalisierung im Gesundheitswesen sind Visionen nötig, um zukünftigen Problemen entgegentreten zu können.

Begriffsverzeichnis

API	Application Programming Interface
AWT	Abstract Window Toolkit
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
ISO	International Standardization Organization
JDBC	Java Database Connectivity
JFC	Java Foundation Classes
JSP	Java Server Pages
JVM	Virtuelle Java - Maschine
HMVC	Hierarchisches Model - View - Controller - Muster
MVC	Model - View - Controller - Muster
OSI	Open System Interconnection

Übersetzungsverzeichnis

Abstract Factory	Abstrakte Fabrik
Builder	Erbauer
Factory Method	Fabrikmethode
Prototype	Prototyp
Singleton	Singelton
Adapter	Adapter
Bridge	Brücke
Decorator	Dekorierer
Facade	Fassade
Flyweight	Fliegengewicht
Composite	Kompositum
Command	Befehl
Observer	Beobachter
Visitor	Besucher
Template Method	Schablonenmethode
Strategy	Strategie
Mediator	Vermittler
State	Zustand
Chain of Responsibility	Zuständigkeitskette
Design Patterns	Entwurfsmuster

Quellenverzeichnis

[Act03] Active Web Software Entwicklungs- und Vertriebsgesellschaft m.b.H.:
Open Source Software
http://www.activeweb.at/open_source.html

[Boh03] Bohl, Jens:
Möglichkeiten der Gestaltung flexibler Software - Architekturen für Präsentationsschichten, dargestellt anhand episodenbasierter, medizinischer Dokumentationen unter Einbeziehung topologischer Befundung
Diplomarbeit, TU Ilmenau, 2003.

[Bus98] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal:
Pattern - orientierte Softwarearchitektur
Addison - Wesley, 1998.

[Gam96] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides:
Entwurfsmuster
Addison - Wesley, 1998.

[Gra98] Mark Grand:
Patterns in Java 1
Catalogue of Reusable Designs Patterns Illustrated with UML
John Wiley & Sons 1998.

[Gra99] Mark Grand:
Patterns in Java 2
John Wiley & Sons 1999.

[Gra01] Mark Grand:
Patterns in Java 3
Java Enterprise Design Patterns.
John Wiley & Sons 2001.

[Hel02] Heller, Christian:
"Cybernetics Oriented Programming (CYBOP)"
http://resmedicinae.sourceforge.net/model/design/cybop/index.html

[Hor99] Cay S. Horstmann, Gary Cornell:
Core Java, Band 1 Grundlagen
Prentice Hall, 1999
Addison - Wesley, 1998.

[Jav00] JavaWorld:
HMVC The layered pattern for developing strong client tiers
http://www.javaworld.com/javaworld/jw-07-200/jw-0721-hmvc_p.html

[Krü00] Krüger, Guido:
GoTo Java 2, Handbuch der Java Programmierung
Addison - Wesley, 2000.

[Kun03] Kunze, Torsten:
Untersuchung zur Realisierbarkeit einer technologieneutralen Mapping - Schicht für den Datenaustausch am Beispiel einer Anwendung zum medizinischen Formulardruck als integrativer Bestandteil eines Electronic Health Record (EHR)
Diplomarbeit, TU Ilmenau, 2003.

[Net03] NETWAYS GmbH:
Open Source Software

http://www.netways.de/Open_Source.oss.0.html

[Res03] Res Medicinae:
The Res Medicinae Homepage
http://www.resmedicinae.org

[Uni02] Unilog Integrata Training AG:
Was sind Design Patterns?
http://www.integrata.de/training/news/pdfnews/trainingnews0204.pdf

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work

under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute.

However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the

Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warrany Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.