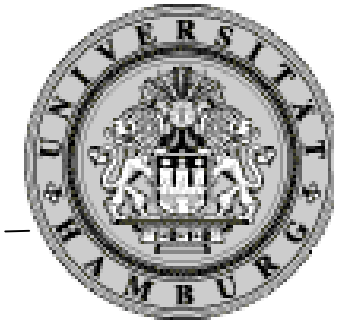


Universität Hamburg
FB Informatik
Datenbanken und
Informationssysteme

STUDIENARBEIT

GENERISCHE VERWALTUNG UND REALISIERUNG SEMANTISCH STRUKTURIERTER BILDARCHIVE IN HETEROGENEN RECHNERNETZEN

Dezember 1998

Rolf Jestrimski
Habichtsweg 2
22307 Hamburg

Marc Vollmann
Am Ochsenzoll 147
22851 Norderstedt

Betreuer:

Prof. Dr. Joachim W. Schmidt

Inhaltsverzeichnis

ABBILDUNGSVERZEICHNIS

I Einführung

„In the future there will be only two kinds of companies: the quick and the dead“.

J. Bodenkamp [INTEL]

Bei einer Betrachtung des Handelsverkehrs ist die Höhe der Gewinnspannen in starkem Maße abhängig vom zeitlichen Aufwand des Transports von einem Lieferanten zu einem Kunden. Je schneller ein Lieferant seine Ware liefern kann, desto besser stehen die Aussichten, die Marktposition zu halten und zu verbessern. Dabei ist von Fall zu Fall abzuwägen, ob aus Kostengründen längere Transportzeiten mit einem günstigen Transportmittel in Frage kommen können oder eine schnelle und damit teurere Lieferung nötig ist. Es setzen sich diejenigen Lieferanten durch, deren Transportlogistik sich von anderen abhebt.

Bei der Betrachtung der Ware *Information*[1] zeichnen sich vergleichbare Gesetzmäßigkeiten ab. Eine effiziente Kommunikation mit bestehenden Kunden und potentiellen Käufern ist für jedes Unternehmen fundamental wichtig. Die globale Struktur des Internet bietet die Voraussetzungen dafür, Informationen unterschiedlichster Art sowohl kostengünstig als auch effizient zu verbreiten. Grundlage dafür sind Systeme, die qualitativ hochwertige Informationen effektiv und gemäß den Anforderungen der Kunden verfügbar machen. Dies erfordert eine effiziente Erstellung flexibler Informationssysteme.

1 Aufgabenstellung und Motivation

Eine wichtige Zielsetzung bei der Entwicklung von Informationssystemen ist es, die in starkem Maße wachsende Flut an digitaler Information effizient zu verwalten und verfügbar zu machen. Der flexible Umgang mit digitalen multimedialen Archiven, die Informationen unterschiedlicher Struktur beinhalten, ist ein wichtiger Aspekt des technischen Fortschritts der letzten Jahre. Hohe Anforderungen sind an die Datenspeicherung, Suche und Wiedergewinnung abgelegter Informationen zu stellen. Durch die zunehmende Bedeutung des Internet spielt die Effizienz der Datenübertragung ebenfalls eine wichtige Rolle. Mit der Migration unternehmensinterner Informationssysteme in das Internet müssen neue Anforderungsprofile beachtet werden. Ziel dieser Arbeit ist es, generische Ansätze für die Verwaltung multimedialer Archive im Internet zu untersuchen und dabei allgemeingültige Aussagen zu treffen, die im weitesten Sinne für eine große Klasse von Archivierungssystemen gelten und verwendet werden können.

Die Projektierung der *Warburg Electronic Library*, einem multimedialen Archivierungssystem mit konkretem Anforderungsprofil für den Bereich der Kunstgeschichte (siehe Kapitel 2 und Kapitel 3), stellt eine exemplarische Ausprägung dar, anhand derer die generischen Ansätze der Arbeit ausgearbeitet und evaluiert werden.

[1] Information: Wissen über Sachverhalte und Vorgänge bzw. zweckorientiertes Wissen

1.1 Verwaltung von multimedialen Archiven

Multimediale Archive sind Sammlungen von Dokumenten mit verschiedenartigen Informationen. Im Vordergrund stehen die Daten, die mit Hilfe verschiedener Dienste in ein Archiv hineingestellt, modifiziert, gesucht und gelesen werden. In digitalisierter Form können Dokumente in unterschiedlichen Formen abgelegt sein, so beispielsweise Bild-, Video-, Text- oder auch Audiodokumente. Bei strukturellen oder semantischen Änderungen an einem Archiv sind die Architekturen gängiger Informationssysteme, die für spezielle Archive entwickelt oder angepaßt wurden, häufig inflexibel. Diese Informationssysteme müssen mit einem hohen Aufwand an diese Änderungen angepaßt werden.

Der Bedarf an mehr Flexibilität der Systeme hinsichtlich dieser Problematik ist sehr groß, da Archive und deren Inhalte von kontextabhängiger und damit von dynamischer Natur sind. Die technische Entwicklung bei der Archivierung von Bild-, Ton- und Videoinformationen zeigt diese Dynamik sehr deutlich. Auch in Zukunft werden sich die Formate dem Stand der Technik und neu entstehenden Komprimierungsalgorithmen anpassen. Die Architekturen der Archive und Anwendungen müssen also Konzepte enthalten, die dieser Dynamik gerecht werden.

1.2 Klassifikation von Bildarchiven

Die meisten Bildarchive sind einer der drei folgenden Kategorien zuzuordnen[1]:

- *Dateisystem (Flat-File):*
  Jedes Bild wird betrachtet als eine reine Sequenz von Bytes. Das Archiv besitzt keine Information über die Bildsemantik. Abfragen laufen linear durch den gesamten Bildbestand nur über den Dateinamen. Sie sind demnach sehr unzureichend für höhere Anforderungen.
- *Graphen-basiert (Graph-based):*
  Jedem Bild wird eine Menge von sogenannten Verständnisprozessen zugeordnet, die in Form eines semantischen Graphen repräsentiert sind. Der Zugriff erfolgt schnell und unabhängig von der Größe und der Anzahl der Bilddaten im Archiv.
- *Merkmal-basiert (Feature-based):*
  Merkmale von Bildern, wie Strukturen, Formen, Farben, etc. werden als Attribute abgelegt. Diese ein Bild begleitenden Informationen werden relational, räumlich oder objektorientiert mit Hilfe von Indexen verwaltet.

Aufgrund der mannigfaltigen Möglichkeiten durch verschiedenartige, kontextabhängige Indexe stellt die auf Merkmale basierende Kategorie die wahrscheinlich mächtigste Form der oben genannten Klassifikationen dar. Trotzdem gilt es, die interessante Frage zu beantworten, ob sich ein Modell entwickeln ließe, um eine geeignete Fusion der Vorteile aller Kategorien anzustreben. Indexe, die verschiedene semantische Sichtweisen auf ein Archiv darstellen, basieren auf einer Hierarchisierung und Kategorisierung von Schlagworten[2], die den Archivdaten (Bilder, Texte, etc.) zugeordnet werden können.

---

[1] search and retrieval in large image archives, J. Turek, 1995. 18 S.,(IBM: research reports; RC 20214)

[2] An dieser Stelle sei ein Schlagwort ein Suchbegriff zur Kategorisierung und ein Index eine Assoziation von

Schlagworte können in einen bestehenden Index aufgenommen werden und decken somit eine Sicht auf die auf Merkmale basierende Kategorie ab, da z.B. Merkmale als Schlagworte in einem gewünschten Kontext angelegt werden können.

Durch die Möglichkeit, Schlagworte in einem hierarchischen Index weiter zu verknüpfen, um verwandtschaftliche Assoziationen herstellen zu können, nähert man sich der Struktur eines semantischen Graphen und damit der auf Graphen basierenden Kategorie an. Es gilt, ein möglichst allgemeingültiges Spektrum für viele Einsatzmöglichkeiten abzudecken.

1.3 Erweiterbarkeit, Wiederverwendbarkeit und Wartung

Informationssysteme bilden wichtige Ressourcen eines Unternehmens. Ist es im Unternehmen erst einmal zum Einsatz gekommen, so läßt es sich schon nach kurzer Zeit nicht mehr ohne großen Aufwand durch ein anderes ersetzen. Dieser Aufwand wird größer, je länger das System im Einsatz ist. Informationssysteme können nicht als statische, in sich abgeschlossene Anwendungen betrachtet werden. Informationssysteme als langlebige Ressourcen sind stets neuen und wachsenden Anforderungen ausgesetzt. Um auf diese adäquat und mit vertretbarem Kostenaufwand reagieren zu können, sollten während des gesamten Entwurfs einer Archivanwendung die Aspekte der Erweiterbarkeit vordergründig bleiben. Analyse, Modellierung und Systementwurf müssen flexible Ansätze aufweisen, die Anpassungen des Systems unkompliziert gestalten und sie nachvollziehbar machen.

Wurde beim Entwurf nicht auf die Erweiterbarkeit, die Wiederverwendbarkeit sowie auf die Wartungsfreundlichkeit geachtet, werden Anpassungen mit der Zeit immer aufwendiger und können leicht durch unbeabsichtigte Nebenwirkungen zu gravierenden Systemfehlern führen, die, wenn sie erst spät entdeckt werden, enorme Kosten verursachen können.

1.4 Vorgehensweise und Struktur der Arbeit

Von dem Anforderungsprofil ausgehend, welches mit Hilfe der Mitarbeiter aus dem Warburg-Haus erstellt wird, wird der Versuch unternommen, schrittweise, unter Verwendung objektorientierter Modellierungskonzepte, ein wiederverwendbares Modell für äquivalente Archivsysteme zu entwickeln. Die Entwicklung eines Modells für eine Problemklasse, anstatt für ein spezielles Problem, erfordert zwar einen höheren Aufwand bei der konzeptionellen Entwicklung, erweist sich aber bei der Erstellung ähnlicher Systeme aufgrund der Wiederverwendbarkeit langfristig als nutzbringend.

Die Entwicklung gliedert sich in verschiedene Phasen, die mit Hilfe geeigneter Werkzeuge unterstützt werden. Die *Object Modelling Technique* als Hilfsmittel für den objektorientierten Entwurf hat sich in vielen Bereichen bewährt und wird in dieser Arbeit als Instrument zur Modellierung eingesetzt.

Begleitend zu den Modellierungsphasen werden konzeptionelle Grundstrukturen, die als Entwurfmuster bezeichnet werden, verwendet. Sie sind einfache und präzise Lösungen für wiederkehrende Entwurfsprobleme in der objektorientierten Softwareentwicklung.

Die Konzepte der OMT und der Entwurfsmuster werden im zweiten Abschnitt erläutert. Der dritte und der vierte Abschnitt behandeln die Anforderungsdefinition sowie die Phasen

---

Schlagworten, die einer Struktur im Sinne der Merkmal-basierten Kategorie entspricht.

der Analyse und des Entwurfs des Projektes.

Die erarbeiteten Modelle finden ihre Implementierung in dem Prototypen, der im sechsten Abschnitt vorgestellt wird. Die Realisierung geschieht mit Hilfe der Programmiersprache Java. Die Vorstellung der Sprache und ihrer Einsatzmöglichkeiten findet im fünften Abschnitt „Realisierung in einer speziellen Programmiersprache“ statt.

2 Der *Bildindex zur politischen Ikonographie* im Warburg-Haus

Das Warburg-Haus ist eine Stiftung des Kunst- und Kulturhistorikers Aby Warburg (1866-1929). Durch die finanziellen Mittel der Stiftung werden diverse wissenschaftliche Projekte unterstützt. Eine Arbeitsstelle befaßt sich mit der politischen Ikonographie, d.h. der Deutung von Darstellungen mit politischem Inhalt oder Hintergrund[1]. Zur Klassifikation der betrachteten Werke wurde der *Bildindex zur Politischen Ikonographie* von Prof. Dr. M. Warnke aufgebaut. Dieser Index wurde mit dem Leibnizpreis dotiert. Er besteht aus ca. 300.000 Photokarten, die nach etwa 100 Schlagworten geordnet sind. Die Schlagworte sind alphanumerisch geordnet und durch Unterbegriffe strukturiert. Die Arbeitsstelle „Politische Ikonographie“ arbeitet mit dem Arbeitsbereich Softwaresysteme unter der Leitung von Prof. Dr. J. W. Schmidt im Projekt *Warburg Electronic Library*[2] (WEL) zusammen. Die kunsthistorischen Anforderungen an den bestehenden politischen Index in Kartenform bilden den interdisziplinären roten Faden während der Entwicklung der WEL.

3 Die *Warburg Electronic Library*

Ziel des WEL-Projektes ist eine digitale Bibliothek, die eine personalisierte Arbeitsumgebung für die kunstgeschichtliche Forschung zur Verfügung stellt. Kern dieser digitalen Bibliothek ist ein digitales multimediales Archiv, das eine erweiterte Repräsentation des heutigen Papierindexes bildet. Ein kontextabhängiger Zugriff auf die unstrukturierten Archivinformationen ist in Form von Schlagworten individuell zu realisieren. Wichtig ist dabei, daß individuelle Sichten des Informationsbestandes semantisch unterschiedlich sein können. Dieser Aspekt ist bei der Konzepterstellung fundamental. Der politische Index verkörpert eine ausschließlich kunsthistorische Sichtweise. Es ist wichtig, einem Benutzer des Archivsystems eine Präsentation der Informationen adäquat anzubieten, um die kunsthistorische Bedeutung in den Vordergrund zu stellen. Unabhängig davon könnte eine bibliothekarische Sichtweise existieren, die wenig mit dem politischen Index gemein hat. Ein weiterer zentraler Gedanke der WEL besteht in der Integration verschiedener Medien. Informationen unterschiedlicher Typen können zu virtuellen kontextabhängigen Karteikarten zusammengefaßt werden.

Bei der Recherche nach Informationen bietet die WEL als digitales Archiv performantere und vielseitigere Zugriffsmöglichkeiten als der Bestand an Bildkarten des Indexes für

---

[1] "Der kulturwissenschaftliche Blick auf die Geschichte beobachtet die jahrhundertealte Erfahrung der visuellen Inszenierung der politischen Macht der Herrschenden:...", Zeitschrift: Hamburg forscht, 1997

[2] C. Niederée, C. Hattendorff, S. Müßig, Warburg Electronic Library- Eine digitale Bibliothek für die politische Ikonographie, in: "uni hh - Forschung", Ausgabe XXXI / 1997, Pressestelle der Universität Hamburg

politische Ikonographie im Warburg-Haus. Die geographische Situation bei der Akquisition von Informationen verschwimmt durch die Möglichkeit der Kommunikation globaler Rechnernetze. Interessenten holen sich mit Hilfe der Internet-Technologie ihren gewünschten Index der WEL auf den Bildschirm. Es wird erwartet, daß die Publizität des mit dem Leibnizpreis dotierten Indexes für politische Ikonographie durch die Digitalisierung zunimmt und sich daraus neue interessante Kooperationen ergeben.

II Analyse- und Entwurfskonzepte

Bei der Erstellung von Informationssystemen werden Instrumente benötigt, um Anforderungen methodisch zu analysieren und um schrittweise Lösungen zu erarbeiten. Für verschiedene Teilanforderungen existieren bereits allgemeine Ansätze, deren Potential für die Lösung zukünftiger Problemstellungen in Form von Vorlagen zur Verfügung stehen.

In dieser Arbeit kommen die Object Modelling Technique (OMT) und Entwurfsmuster zum Einsatz. Aus diesem Grund folgt in diesem Abschnitt ein entsprechender Überblick dieser Konzepte.

1 Object Modelling Technique

OMT ist eine Technik, die den gesamten Zyklus der Softwareentwicklung begleitet. Dieser Zyklus wird in drei Abschnitte aufgeteilt, die, jeweils für unterschiedliche Sichtweisen auf das zu erstellende System, ein Modell zum Ziel haben. Das *Objektmodell* repräsentiert die statischen, strukturellen und auf Daten bezogenen Aspekte des Systems. Das *dynamische Modell* verkörpert die zeitlichen und die auf das Verhalten und die Steuerung bezogenen Systemaspekte. Die Übergangs- und Funktionsaspekte eines Systems formieren das *funktionale Modell*[1].

In der Objektorientierung werden Systeme als eine Kollektion diskreter, kooperierender Objekte betrachtet, die sowohl Datenstruktur als auch Verhalten in sich vereinen. Bei der OMT werden die Konzepte objektorientierter Systeme, wie Vererbung[2], Überschreibung[3] von Methoden, etc., auf die Modellierung von Systemen übertragen.

1.1 Phasen der OMT-Methodologie

Bei der Entwicklung umfangreicherer Systeme gliedert sich der Prozeß der Entwicklung in unterschiedliche und meistens aufeinander aufbauende Phasen. Auch die Umsetzung einer hohen Transparenz vorangegangener Entwicklungsabschnitte bildet eine wichtige Anforderung an Instrumente der Modellierung. Dies heißt, daß bereits erarbeitete Ergebnisse für neue Ansätze zur Verfügungen stehen müssen und sich erweitern beziehungsweise anpassen lassen. Die OMT besteht aus drei Phasen, der Anforderungsanalyse, dem Systementwurf und dem Objektentwurf.

---

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen, Hanser–Verlag / Dt. Übers. 1993 Doris Märtin

[2] Unter Vererbung versteht man, das Subklassen von ihrer Superklasse Attribute und Methoden übernehmen. Zum Beispiel sind die Klassen PKW und LKW Subklassen von Kraftfahrzeug. Sie erben die Eigenschaften „hat Motor“ und „zulässiges Gesamtgewicht“.

[3] Mit Überschreibung beschreibt man die Möglichkeit, Methoden, die von einer Superklasse geerbt wurden, neu zu implementieren. Zum Beispiel könnte es eine Methode zur Berechnung des nächsten TÜV-Termins bei Kraftfahrzeugen geben. Wenn es einen Kraftfahrzeugtyp gibt, bei dem die Regelung von der allgemeinen Regel abweicht, so überschreibt er diese durch seine eigene Methode.

1.1.1 Die Anforderungsanalyse

„Bei der Anforderungsanalyse, dem ersten Schritt der OMT-Methodologie, geht es darum, ein präzises, kompaktes, verständliches und korrektes Modell der realen Welt zu entwickeln“, [J. Rumbaugh[1]].

Die Analyse beginnt mit einer Problemstellung, die in der Regel unvollständig ist und durch die Analyse präzisiert wird. Es gilt, das reale System zu verstehen, es zu beschreiben und von irrelevanten Details zu abstrahieren. Unklarheiten, z.B. durch unzureichende verbale Beschreibungen, werden ausgeräumt. Die Analyse trägt keine Aussagen über Implementation mit sich. Es wird auf drei Aspekte von Objekten eingegangen: Ihre statische Struktur (*Objektmodell*), die Reihenfolge von Interaktionen (*dynamisches Modell*) und Datentransformationen (*funktionales Modell*).

Fachlich versierte Benutzer mit entsprechenden interdisziplinären Anforderungen an ein System formulieren zusammen mit den Systementwicklern ein Anforderungsprofil. Die wirtschaftlichen Aspekte fließen durch das Management ein, so zum Beispiel Kostenrestriktionen aufgrund eines vorgegebenen Etats für die Systementwicklung. Auf Grundlage der entstandenen Problembeschreibung werden Modelle gefertigt. Vorhandenes Fachwissen, praktische Erfahrungen und Kenntnisse der Domäne gehen in die Modellentwicklung ein. Im Rahmen der Entwicklung mit OMT wird ein Objektmodell, ein dynamisches Modell und ein funktionales Modell erstellt.

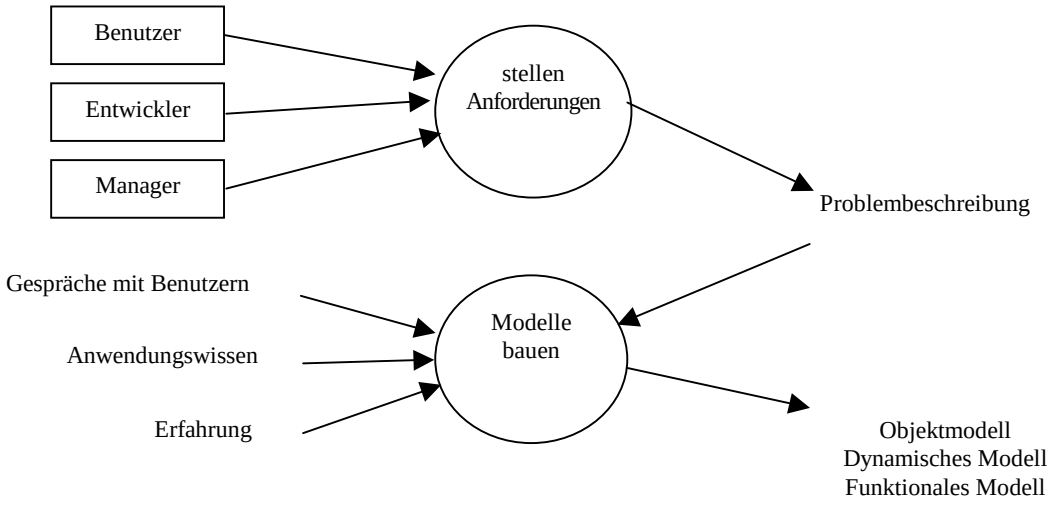


Abbildung II-1: Vorgehensweise bei der Anforderungsanalyse

1.1.2 Der Systementwurf

„Die Analyse befaßt sich schwerpunktmäßig damit, was zu tun ist, unabhängig davon, wie

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen, Hanser–Verlag / Dt. Übers. 1993 Doris Märtin, S. 181ff

dies realisiert wird. Beim Entwurf werden Entscheidungen darüber getroffen, wie das Problem gelöst wird, zunächst auf abstrakter Ebene und später auf immer detaillierteren Ebenen“ [J. Rumbaugh[1]]. Der Systemdesigner trifft grundlegende Entscheidungen über die Gesamtarchitektur des Systems. Das Zielsystem wird auf Grundlage der Ergebnisse der Analyse organisiert. Entscheidungen bzgl. Optimierungen bestimmter Leistungseigenschaften werden getroffen. Versuchsweise findet eine Zuweisung von Ressourcen statt.

1.1.3 Der Objektentwurf

Beim Objektentwurf wird die beim Systementwurf gewählte Strategie umgesetzt und die Details ausgearbeitet. Die bei der Analyse ermittelten Objekte dienen als Gerüst für den Entwurf. Beim Objektentwurf geht es hauptsächlich um Datenstrukturen und Algorithmen, die zur Implementierung jeder Klasse benötigt werden. Die Klassen, Attribute und Assoziationen aus der Analyse müssen als spezifische Datenstrukturen implementiert werden. Das Objektmodell aus der Analysephase muß für die technische Umgebung angepaßt und optimiert werden. Die Optimierung des Entwurfs sollte nicht exzessiv betrieben werden, weil einfache Implementierung, Wartbarkeit und Erweiterbarkeit ebenfalls wichtige Anliegen darstellen.

1.2 Modelle der OMT-Methodologie

Modelle sind orthogonale Teile der Beschreibung eines vollständigen Systems und durch Querverbindungen vernetzt. Das *Objektmodell* spielt eine tragende Rolle, da beschrieben wird, *was* sich verändert (also die Objekte), bevor die Frage geklärt wird, *wann* und *wie* sie verändert werden.

1.2.1 Das Objektmodell

Der Bau eines *Objektmodells* erfordert folgende Schritte:

- Zu Beginn werden Objekte der realen Welt, die für die Problembetrachtung von Interesse sind bzw. sein können, identifiziert und klassifiziert (Darstellung, Bild, Text, Schlagwort, etc.).
- Ist dieser Schritt vollzogen, ist es sinnvoll, jede Klasse verbal zu erläutern. Diese Aufstellung dient als Glossar (*Data Dictionary)*. Hier wird die jeweilige Bedeutung einer Klasse bezogen auf das aktuelle Problem herausgearbeitet („Darstellung: Abbildung in einem bestimmten Format“).
- Anschließend ist es notwendig, die Assoziationen einschließlich der Aggregationen zwischen Objekten zu identifizieren. Dieser Vorgang sollte ebenfalls erst verbal geschehen. (Beispiel: „Jedes Bild besitzt einen nur ihm zugeordneten Text“), wobei im Anschluß eine Überarbeitung die Assoziationen auf eine Menge von geeigneten Beziehungen reduziert.

---

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen, Hanser–Verlag / Dt. Übers. 1993 Doris Märtin, S. 241ff

- Um gemeinsame Strukturen zu nutzen, wird das Konzept der Vererbung verwendet. Die Klassen werden auf Ähnlichkeiten ihrer Attribute, Assoziationen oder Operationen untersucht (Beispiel: Die Klasse „Mitarbeiter“ erbt die Eigenschaften der Klasse „Person“, die damit nicht explizit in der Klasse „Mitarbeiter“ erneut implementiert werden müssen).
- Der letzte Schritt der Objektmodellierung besteht in der Gruppierung von Klassen zu Modulen. Ein Modul ist eine Menge von Klassen, die eine logische Untermenge des Gesamtmodells beschreibt (Beispiel: Das Modell eines Betriebsystems könnte sich aus Modulen für die Prozeßsteuerung, die Gerätesteuerung, die Dateiverwaltung und die Speicherverwaltung zusammensetzen).
- Durch die Tatsache, daß jeder Software-Entwicklungsprozeß immer neue Iterationen erfordert, sind Revisionen des Objektmodells in der Regel unvermeidbar.

1.2.2 Das dynamische Modell

Die folgende Aufstellung faßt zusammen, welche Schritte bei der Konstruktion eines *dynamischen Modells* durchgeführt werden. Das *dynamische Modell* zeigt das zeitabhängige Verhalten des Systems und seiner Objekte und ist für interaktive Systeme sehr wichtig. Die Zusammenfassung typischer Dialoge des Benutzers mit dem System wird verbal in Form von Szenarios[1] beschrieben. Die Szenarios dienen primär dazu, ein Gefühl für das zu erwartende Systemverhalten zu bekommen. Wichtige Interaktionen, externe Anzeigeformate und die Art des Informationsaustausches gehen aus ihnen hervor. Im Anschluß werden die Ereignisse[2] zwischen Objekten identifiziert. Jedes Szenario wird mit Hilfe der zugehörigen Ereignisse als Ereignispfad[3] dargestellt. Für die Objektklassen wird ein Zustandsdiagramm erstellt, welches die Ereignisse zeigt, die das Objekt sendet oder empfängt. Jedes Szenario bzw. jeder Ereignispfad entspricht einem Pfad durch das Zustandsdiagramm[4].

1.2.3 Das funktionale Modell

Das *funktionale Modell* zeigt, wie Werte berechnet werden, ohne Reihenfolgen, Entscheidungen oder Objektstrukturen zu berücksichtigen. Es zeigt die Abhängigkeit zwischen Werten und beschreibt Funktionen, die sie verbinden. Zur Darstellung eignen sich Datenflußdiagramme, wobei die Flüsse den Objekten oder Attributwerten in einem Objektdiagramm entsprechen.

- Der erste Schritt bei der Erstellung des funktionalen Modells ist die Identifikation der Ein- und Ausgabewerte. Sie bilden die Parameter von Ereignissen zwischen dem

---

[1] Ein Szenario ist eine Folge von Ereignissen, die bei einer bestimmten Ausführung des Systems auftreten.

[2] Ereignis: Ein Geschehen in einem bestimmten Augenblick. Ereignisse sind unter anderem Signale, Eingaben, Entscheidungen, Unterbrechungen, Transitionen und Aktionen an oder durch Benutzer oder externe Geräte.

[3] Ereignispfad: Diagramm, das den Sender und Empfänger von Ereignissen und Ereignisfolgen zeigt.

[4] Zustandsdiagramm: Gerichteter Graph, in dem Knoten Systemzustände und Kanten Transitionen zwischen Zuständen repräsentieren.

System und der Außenwelt.

- Im Anschluß erfolgt die Entwicklung von Datenflußdiagrammen, die ausweisen, wie aus Eingabewerten die Ausgabewerte berechnet werden. Die Konstruktion erfolgt schichtweise. Innerhalb jeder Schicht wird ausgehend vom Ausgabewert die Funktion festgelegt, mit der dieser berechnet wird. Datenflußdiagramme spezifizieren nur Abhängigkeiten zwischen Operationen. Entscheidungen oder Operationsreihenfolgen werden nicht aufgezeigt, da einige Operationen optional sein können bzw. sich gegenseitig ausschließen können. Ist das Datenflußdiagramm ausreichend detailliert, wird für jede Funktion eine geeignete Beschreibung[1] erstellt.

Im Vordergrund steht die Aufgabe der Funktion und nicht ihre Implementation. Zu beachten sind eventuelle Einschränkungen. Sie stellen funktionale Abhängigkeiten zwischen Objekten dar, die unabhängig von den Ein- und Ausgabewerten sind. Abschließend werden Optimierungskriterien spezifiziert. Sie legen fest, welche Eigenschaften, zum Beispiel die Netzwerkbelastung oder Antwortzeiten, optimiert werden sollen.

Diese Modelle werden in allen weiteren Entwurfsphasen verwendet.

2 Entwurfsmuster

Während der Entwicklung von Software treten zuweilen Probleme auf, für die bereits bewährte Lösungen existieren. Innerhalb der einzelnen Entwurfsphasen ist es daher nützlich, auf vorhandene Methoden und Teillösungen zurückgreifen und aufbauen zu können, um Zeit und Entwicklungskosten zu vermindern. In der Regel haben sich Entwickler, die viele Jahre mit dem Entwurf von Programmen zugebracht haben, wiederkehrende Lösungen erarbeitet. Eigene Implementationen von flexibel gehaltenen Modulen oder Klassenbibliotheken finden immer wieder erneuten Einsatz. Diese Form spiegelt eine konkrete Möglichkeit der Wiederverwendbarkeit von bestehenden Systemkomponenten wider.

Entwurfsmuster sind eine systematische Beschreibung wiederkehrender Strukturen. Sie dienen als konzeptuelle Vorlagen für den Entwurf und stellen eine taxonomische Klassifikation wiederkehrender Lösungsschemata dar. Durch sie gewinnt der Aspekt der Wiederverwendbarkeit schon in der Entwurfsphase an Bedeutung. Die Verwendbarkeit von Entwurfsmustern ist unabhängig von dem jeweiligen sprachlichen Paradigma und dem betrachteten Gesamtsystem.

2.1 Was sind Entwurfsmuster ?

„Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so daß Sie diese Lösung beliebig oft anwenden können, ohne sie jemals ein zweites Mal gleich auszuführen“ [Christopher Alexander[2]].

---

[1] Diese Beschreibung kann in natürlicher Sprache, mathematische Formeln, Pseudocode oder einer anderen geeigneten Form erfolgen.

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Entwurfsmuster, Addison-Wesley (Deutschland) GmbH, 1.

Ein Entwurfsmuster benennt, abstrahiert und identifiziert die relevanten Aspekte einer allgemeinen Entwurfsstruktur. Ein Muster besitzt vier grundlegende Elemente:

- Mustername:
  Bezeichnung für das Entwurfproblem und seine Lösungen
- Problemabschnitt:
  Beschreibung der Einsatzmöglichkeit bei einem spezifischen Problem oder einer Problemklasse
- Lösungsabschnitt:
  Beschreibung der Elemente des Entwurfs mit ihren Beziehungen, Zuständigkeiten und Interaktionen, wobei nicht auf eine spezielle Implementation eingegangen wird
- Konsequenzabschnitt:
  Auflistung der Vor- und Nachteile des resultierenden Entwurfs, um den Einsatzspielraum besser überblicken zu können und um Alternativen besser bewerten zu können

Die Beschreibung von Entwurfsmustern erfolgt nicht nur in grafischer, sondern auch in Textform, da die grafische Form nicht ausreicht, um die oben erwähnten Alternativen und deren Vor- und Nachteile aufzuzählen.

2.2 Vorteile von Entwurfsmustern in der Anwendungsentwicklung

Die Verwendung von Entwurfsmustern integriert bekanntes Wissen in das jeweils benötigte Lösungsspektrum. Die Zeitersparnis wirkt sich direkt kostensenkend auf die Entwicklung aus. Durch die einheitliche Aufmachung der Entwurfsmuster lassen sich diese immer wieder leicht nachvollziehen und schnell als Vorlage heranziehen. Das kritische Studieren solcher Muster verhilft zu einer vorteilhafteren Denkweise im Bereich des objektorientierten Softwareentwurfs[1]. Überflüssige und zum Teil kostenintensive Einarbeitungszeiten werden somit rationalisiert.

2.3 Vorstellung eines verwendeten Entwurfsmusters

Ein konkretes und häufig verwendetes Entwurfsmuster wird in diesem Abschnitt zur Erläuterung betrachtet[2].

Das Entwurfsmuster *Befehl* (*Command*)

Die Aufgabe des Musters *Befehl* besteht in der Kapselung eines Befehls als eigenständiges Objekt. Dies ermöglicht es zum Beispiel, Klienten mit verschiedenen Anfragen zu parametrisieren, Operationen in eine Queue zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen. Dreh- und Angelpunkt dieses Musters ist eine

---

Auflage 1996 / Dt. Übers. D. Riehle

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Entwurfsmuster, Addison-Wesley (Deutschland) GmbH, 1. Auflage 1996/ Dt. Übers. D. Riehle

[2] Das vorgestellte Entwurfsmuster und weitere finden sich ausführlicher in: E. Gamma, R. Helm, R. Johnson, J. Vlissides, Entwurfsmuster, Addison-Wesley (Deutschland) GmbH, 1. Auflage 1996/ Dt. Übers. D. Riehle

abstrakte Klasse *Befehl*, die eine Schnittstelle zum Ausführen von Methoden deklariert, so zum Beispiel enthält diese Schnittstelle im einfachsten Fall eine abstrakte Methode *FuehreAus*. Konkrete Unterklassen von *Befehl* bestimmen ein Empfänger/Methoden-Paar, indem sie den Empfänger als eine Exemplarvariable speichern und die Methode *FuehreAus* so implementieren, daß sie die Anfrage ausführt. Der Empfänger verfügt über das entsprechende Wissen, das benötigt wird, um die Anfrage umzusetzen.

Teilnehmer am Entwurfsmuster:

Befehl
deklariert eine Schnittstelle zum Ausführen einer Methode.

KonkreterBefehl
definiert die Anbindung eines Empfängers an eine Aktion; implementiert *FuehreAus* durch Aufrufen der entsprechenden Methode(n) beim Empfänger

Klient
erzeugt ein *KonkreterBefehl*-Objekt und übergibt ihm den Empfänger.

Aufrufer
befiehlt dem Objekt *Befehl*, die Anfrage auszuführen.

Empfänger
weiß, wie die an die Ausführung einer Anfrage gebundenen Methoden auszuführen sind. Ein *Empfänger* kann von einer beliebigen Klasse sein.

Interaktionen der Teilnehmer:

- Der Klient erzeugt ein Befehlsobjekt einer konkreten Befehlsklasse und bestimmt ihren Empfänger.
- Ein Auslöser speichert das Befehlsobjekt der konkreten Klasse.
- Der Auslöser löst eine Anfrage aus, indem er die *FuehreAus*-Methode des Befehlsobjekts aufruft. Wenn Befehle rückgängig gemacht werden können, speichert das Befehlsobjekt vor dem Ausführen des Befehls den Zustand des Empfängers, um ihn später wiederherstellen zu können.
- Das konkrete Befehlsobjekt ruft Methoden auf seinem Empfängerobjekt auf und setzt die Anfrage um.

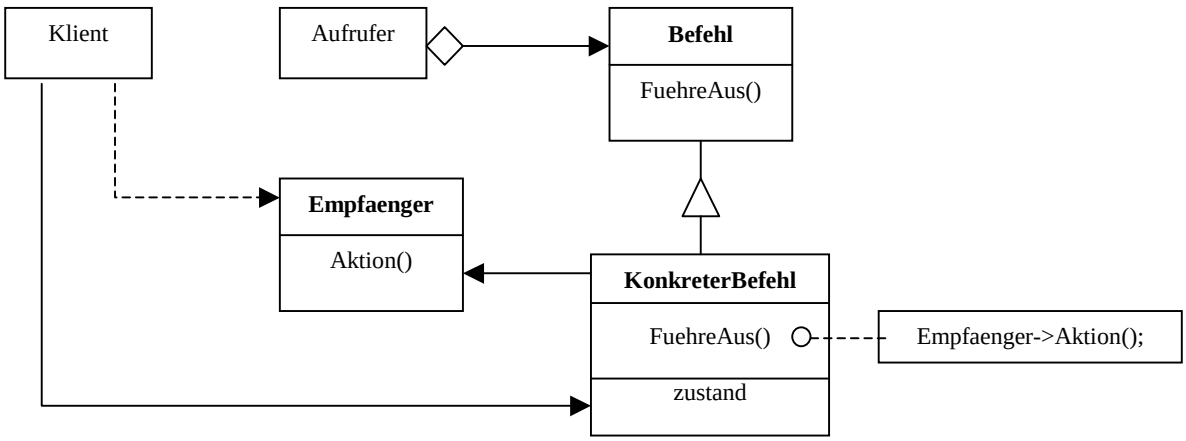


Abbildung II-2: Das Entwurfsmuster *Befehl*

In analoger Form werden Entwurfmuster definiert, um in der Planung und Realisierung die Entwicklung von objektorientierter Software bestimmte Problemstellungen zu klassifizieren und für diese Problemklassen Lösungsmuster aufzustellen.

III Anforderungsdefinition

Die Entwicklung von Informationssystemen gestaltet sich heutzutage immer noch kosten- und zeitintensiv. Bestehende Komponenten vorhandener Informationssysteme finden bei Neuentwicklungen häufig keine Wiederverwendung. Diese Tatsache ist eine der Ursachen der Entstehung der Softwarekrise[1]. Um diesem Mißstand entgegenzuwirken, sollten Entwicklungsaufgaben nicht stets fallweise neu gelöst werden. Durch die Betrachtung allgemeinerer Problemklassen ist es möglich, für diese Klasse generische Lösungen zu entwickeln, die bei zukünftigen Informationssystemen wiederverwendet werden können. Das in dieser Arbeit zu modellierende Archivsystem soll als konkretes Beispiel diesen Anforderungen genügen.

1 Eigenschaften von Archivsystemen

In diesem Kapitel wird versucht, in Form von verschiedenen Sichtweisen auf ein Archivsystem und den daraus resultierenden Anforderungen, eine Problemklasse herauszuarbeiten.

1.1 Semantik und Struktur in Archivsystemen

Aufgrund der hohen Quantität an Informationen in einem Archiv ist es notwendig Ordnungskriterien einzuführen und es dadurch zu strukturieren. Strukturen stellen Sichtweisen auf die Informationen im Archiv dar. Der Kontext, in dem das Archiv genutzt wird, beeinflußt die Form des Zugriffs auf die Informationen. Beispielsweise in einem Buch befinden sich ein Inhaltsverzeichnis und ein Register, mit deren Hilfe auf Informationen zugegriffen werden kann. Im Gegensatz zum Register, welches eine lexikographische Ordnung auf die Informationen aufweist, stellt das Inhaltsverzeichnis eine semantische Struktur dar. Die Bedeutung eines angegebenen Kapitels (im Archiv vergleichbar mit einem Schlagwort) geht aus dem Inhaltsverzeichnis hervor. Der Leser (analog zum Benutzer des Archivs) findet durch die semantische Einbettung schon ausreichende Informationen für den Zugriff auf eine Auswahl gewünschter (Archiv-) Informationen entsprechend seinen Anforderungen. Während das Inhaltsverzeichnis eines Buches genau eine Indizierung des Buchinhaltes in gedruckter Form manifestiert, soll es möglich sein, in einem digitalen Archivsystem verschiedene Strukturen (Indexe) auf demselben Archivbestand darstellen zu können.

1.2 Allgemeine Anforderungen

Die im Folgenden aufgestellte Anforderungsdefinition ist mit der Prämisse erstellt, allgemeine Anforderungen zu fixieren, die prinzipiell für jedes semantisch strukturierte (Bild-) Archiv gelten. Erfolgt bei einem Archiv ein Zugriff über Schlagworte, stehen diese in der Regel in einem hierarchischen Kontext, um Klassen- oder Gruppenbildung zu realisieren. Neben dieser Klassifizierung durch Schlagworte liegt ansonsten oft eine flache,

[1] Während sich im Laufe der EDV-Entwicklung die Effizienz der Software nur um den Faktor 100 erhöht hat, erhöhte sich die Leistung der Hardware um den Faktor 10.000

nicht hierarchische und meistens nur alphabetische Ordnung vor. Es kann aber auch noch ein Zugriff nach anderen Kriterien unterstützt werden.

Die weitere Darstellung erfolgt unter verschiedenen Sichtweisen (Anwendersicht, administrative Sicht und technische Sicht). Sie wird konkret am Beispiel der WEL angelehnt sein, indem der politische Index seine Umsetzung in einen speziellen digitalen Index des Archivsystems der WEL findet.

1.3 Anwendersicht

Der Zugriff eines anwendungsorientierten Benutzers manifestiert sich in den meisten Fällen in Lesevorgängen, um Informationen abzurufen. Dabei soll ein möglichst großes Spektrum an Möglichkeiten offenstehen, um bestimmte Daten zu suchen.

- Ein Index besteht aus einer Menge von Schlagworten, die aufgrund einer gewünschten Semantik in eine hierarchische Form gebracht sein können (Bei der WEL werden die Schlagworte „Bismarck“, „Denkmalsturz“ und „Ereignisdenkmal“ dem Schlagwort „Denkmal“ hierarchisch untergeordnet, d.h. einer Rubrik zugeordnet.). Aus einer Menge von mehreren unabhängigen Indexen, die jeweils den Archivbestand nach unterschiedlichen Aspekten strukturieren, muß der Anwender den jeweiligen Index auswählen, mit dem er arbeiten möchte (Aus einer entsprechenden Liste wird z.B. der politische Index für die Nutzung der WEL gewählt.). Nur die Schlagworte und Rubriken dieses Indexes werden angezeigt.
- Ein Schlagwort kann, mit einer anderen Semantik, in verschiedenen Rubriken vorhanden sein (Bei der WEL ist das Schlagwort „Allegorisch“ ist u.a. in den Rubriken „Bündnisse“ und „Demokratie“ zu finden.). Die Semantik eines Schlagwortes ergibt sich aus dem jeweiligen Kontext.
- Alle Archivdokumente sind über Schlagworte erreichbar (Die Darstellung der Kaiserkrönung Napoleons könnte z.B. über die Schlagwörter „Napoleon“ oder „Kaiserkrönung“ geschehen.). Diese Zuweisung kann unabhängig einer hierarchischen Position des Schlagwortes erfolgen, d.h. jedes Schlagwort kann Dokumente referenzieren (Obwohl bei der WEL das Schlagwort „Bismarck“ hierarchisch unterhalb des Schlagwortes „Denkmal“ steht, können Archivdokumente beiden Schlagworten zugeordnet werden.). Dies dient zur Einordnung von Dokumenten, die nicht in eine der gegebenen Unterkategorien fallen. Zur Veranschaulichung sei die übliche Dateistruktur in Form von Ordnern genannt, in denen Dateien und wiederum weitere Ordner abgelegt sind.
- Archivdokumente repräsentieren Informationen für unterschiedliche Medien (Text, Bild, Film, Ton), die gruppiert dargestellt werden.
- Zu einem Schlagwort erhält man eine Menge von Archivobjekten, die diesem Schlagwort zugeordnet sind (Das Schlagwort „Napoleon“ würde verschiedene Bilder und Darstellungen bzw. Texte zu Napoleon als Menge zur Auswahl stellen.).
- Zu einem bestehenden Schlagwort können semantisch verwandte Schlagworte festgelegt werden. Diese Querverweise sind dabei unabhängig von der hierarchischen Relation der jeweils verknüpften Schlagworte.
- Jeder Benutzer kann sich zusätzliche persönliche Indexe zusammenstellen. Dies

ermöglicht, eine benutzereigene Verschlagwortung der Archivobjekte vorzunehmen.

- Gruppenindexe können angelegt werden. Die Funktionsweise erfolgt analog zu dem persönlichen Index, wobei eine Benutzergruppe darauf Zugriff hat.
- Ebenfalls ist eine gefilterte Suche nach beliebigen Attributen der Dokumente möglich, so auch die Zusammenstellung von Archivdokumenten (z.B. eines Jahres, Erstellers, Art des Kunstobjektes bei der WEL).
- Ein kontextsensitives Hilfesystem steht dem Benutzer zur Verfügung.

1.4 Administrative Sicht

Administratoren sind Benutzer, die bezüglich des Systems eine andere Rolle einnehmen, andere Zugriffsrechte haben und ein anderes Funktionsspektrum des System benötigen. Sie befassen sich mit der Verwaltung des Systems und sind fachlich versierte Benutzer.

- Es besteht die Möglichkeit, neue Indexe zu erstellen, also vorhandene Schlagworte in einem neuen Kontext zu strukturieren, ohne einen vorhandenen Index aufgeben zu müssen.
- Neue Schlagworte können erfaßt und in jeden Index des Archivs integriert werden, indem diese in den jeweiligen Indexkontext hierarchisch eingeordnet werden. Dabei wäre auch der Extremfall denkbar, das ein Schlagwort in dem einen Index ein Unterschlagwort eines anderen Schlagwortes ist, aber in einem weiteren Index die inverse Relation vorliegen könnte.
- Schlagworte eines Indexes können mit semantisch verwandten Schlagworten verknüpft werden. Zu jeder Verknüpfung können entsprechende Begleitinformationen für einen Querverweis abgelegt werden.
- Eine hierarchische Ordnung innerhalb der Schlagworte eines Indexes kann angelegt werden, wobei keine Restriktion bezüglich der Schachtelungstiefe an Unterschlagworten vorhanden ist. Zyklen innerhalb dieser Hierarchie sind nicht erlaubt, da eine Klasse sich in einem Zyklus wiederum als Unterklasse hätte. In Verbindung mit verwandtschaftlichen Beziehungen können zyklische Verknüpfungen entstehen.
- Zu jedem Schlagworteintrag besteht die Möglichkeit, zusätzlich Informationen zu erfassen. Auf diese Weise kann z.B. eine zusätzliche Klassifizierung eingeführt werden.
- Archivdokumente können erstellt, modifiziert oder gelöscht werden.
- Der Zugriff auf die Dokumente erfolgt über die Schlagworte, die auf verschiedene Archivinformationen verweisen können. Die Erstellung oder Aufhebung solcher Verknüpfungen müssen möglich sein.
- Im Rahmen einer Benutzerverwaltung erfolgt die Pflege, Neuanlage und das Entfernen von Benutzern. Benutzer können unterschiedliche Rechte haben.

1.5 Technische Sicht

Für die Konstruktion eines Systems ist es notwendig, die technischen Eigenschaften

verwendeter Komponenten zu sichten.

- Der Zugriff auf die Daten darf nicht auf einen Einzelplatzrechner oder einem LAN beschränkt sein. Deshalb sollte die Internet-/Intranettechnologie genutzt werden.
- Als Backend sollte ein leistungsfähiges Datenbanksystem (zum Beispiel ein relationales) zur Verfügung stehen.
- Die Schnittstellen zur Datenbasis müssen klar definiert sein, um nachfolgende Erweiterungen, die nicht unbedingt direkt die Software betreffen, zu ermöglichen.
- Spätere Erweiterungen, wie z.B. individuelle Archivanfragen mit Hilfe von SQL-Anweisungen, müssen leicht einzubinden sein.
- Die Netzbelastung ist möglichst gering zu halten, d.h. große Bilddaten sollten nur auf expliziten Wunsch in bester Qualität zur lokalen Darstellung übertragen werden.
- Jeder Benutzer besitzt ein Benutzerprofil. Hier ist definiert, welche Rechte einem Benutzer für bestimmte Systemkomponenten zugeordnet sind.
- Arbeiten, die an der Indexstruktur vorgenommen werden, sind nur entsprechend befugten Benutzern möglich.
- Menü-, Masken- und Fehlertexte müssen so modelliert werden, daß die Unterstützung mehrerer Sprachen (z.B. Deutsch und Englisch) möglich ist.
- Informationen aus anderen Archiven sollen integriert werden können.
- Schnittstellen von anderen Anwendungen sollen genutzt werden können.

Im Anschluß erfolgt ein Katalog von Anforderungskriterien, der speziell auf die WEL zugeschnitten ist und vorwiegend das graphische Layout betrifft. Auch hier sind sicherlich Aspekte allgemeinerer Natur, jedoch sind diese stark abhängig von der subjektiven Sicht- und Beurteilungsweise.

- Text- und Bildeinträge sollten aus ergonomischen Gründen gemeinsam angezeigt werden.
- Die graphische Benutzeroberfläche (Menü- und Maskenführung) soll per Maus und Tastatur zu bedienen sein.
- Das Anlegen der Verwandtschaftsrelationen geschieht durch spezielle Benutzer, wie z.B. dem Administrator.
- Werden Objekte, wie Bilder, Texte oder sonstige Informationen, auf dem Bildschirm dargestellt, so ist es möglich diese in einen persönlichen oder Gruppenindex aufzunehmen.
- Jeder Index kann mit weiteren Zugangsrestriktionen versehen werden.

IV Analyse und Entwurf

Am Beispiel der WEL erfolgt in diesem Abschnitt auf Grundlage der beschriebenen Anforderungen eine Umsetzung mit Hilfe der genannten Werkzeuge *OMT* und *Entwurfsmuster*. In der Analyse wird die Erstellung eines präzisen, kompakten und verständlichen Modells angestrebt. Dieses bildet in der sich anschließenden Entwurfsphase die Basis zur Entwicklung einer globalen Problemlösungsstrategie für die Implementation. Eine detaillierte Aufstellung der Modellierungsergebnisse befindet sich im Anhang.

1 Anforderungsanalyse

Die Anforderungsanalyse ist die erste Phase der Systementwicklung. In ihr wird festgestellt, welche Eigenschaften das System grundsätzlich haben muß, welche Aufgaben und wie diese zu erfüllen sind. Der Aufwand bei der Analyse kann je nach Komplexität des zu erstellenden Systems sehr unterschiedlich ausfallen. Auf jeden Fall sollten alle beteiligten Gruppen, die mit dem System in Berührung kommen, sei es bei der Entwicklung, Finanzierung oder Nutzung, die Möglichkeit haben, Anforderungen an das System zu formulieren. Diese unterschiedlichen Anforderungen führen nach Ihrer Auswertung zu einer ersten Problembeschreibung.

Mit Hilfe dieser Problembeschreibung wird dann bei Gesprächen, bei denen Manager, Entwickler und Benutzer vertreten sein sollten, die Modelle entwickelt, die das zukünftige System beschreiben[1].

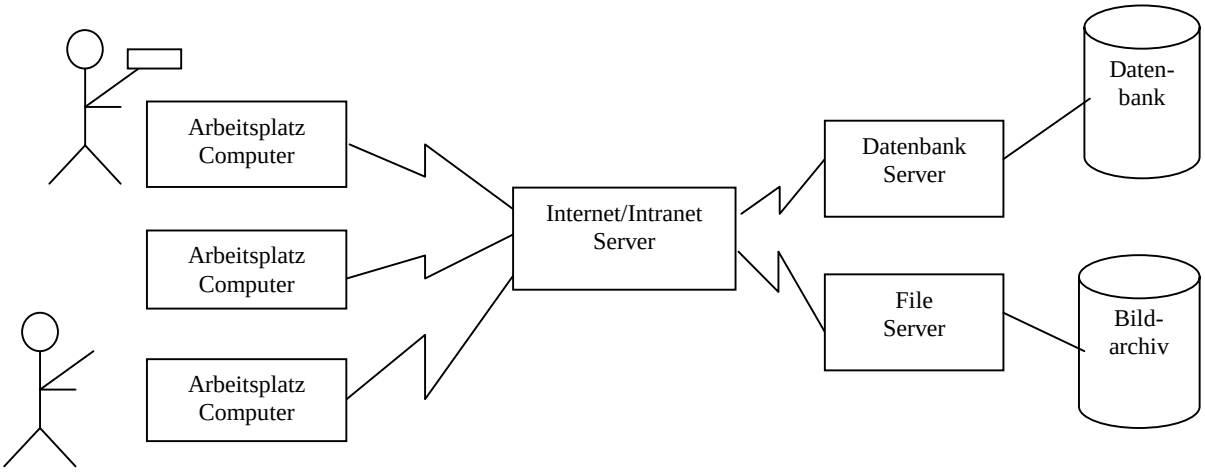


Abbildung IV-1: Vorgegebene Komponenten des Systems

Die Abbildung zeigt die vorgegebenen Komponenten für den verteilten Systemeinsatz. Diese Vorgaben sind Bestandteil der grundsätzlichen Systemanforderungen. Die

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen, Hanser–Verlag / Dt. Übers. 1993 D. Märtin, S. 181 ff

gleichzeitige Verwendung des Systems durch mehrere Benutzer von einem beliebigen Standort aus, prädestiniert das Internet für eine Nutzung. Eine weitere Prämisse stellt die Verwendung einer relationalen Datenbank zur persistenten Archivierung der Informationen, sowie die Nutzung von Fileservern zur Archivierung der Abbildungen in separaten Dateien dar. Die Verwendung von Fileservern hat gegenüber der Archivierung der Abbildungen in der Datenbank sowohl Vorteile als auch Nachteile. Der entscheidende Vorteil von Fileservern ist, im Fall der WEL, der einfache Zugriff auf die Abbildungen. Nachteilig ist die fehlende Zusicherung der referenziellen Integrität zwischen Datenbank und Fileservern.

1.1 Identifikation der Klassen und Assoziationen

Die Konstruktion eines Objektmodells beginnt mit der Identifikation relevanter Objektklassen des Anwendungsbereichs. Die Klassen werden aus der Problembeschreibung gewonnen. Es werden Substantive herausgearbeitet, die zum System gehören[1].

Objekte der Anwendung			**Objekte des Systems**
Archiv	Archivobjekt	Person	Anwendungsserver
Index	Bild	Ersteller	Datenbankserver
Persönlicher Index	Text	Benutzer	Fileserver
Gruppenindex	Audio	Benutzergruppe	Datenbank
Indexobjekt	Video		DBMS
Schlagwort			Anwendung
Karte			GUI

Abbildung IV-2: Objektklassen des Archivsystems aus der Problembeschreibung

Isolierte Begriffe lassen sich unterschiedlich interpretieren. Die aufgestellten Objektklassen werden aus diesem Grund in einem *Glossar* beschrieben, um die konkrete Bedeutung zu fixieren. Es ist hier exemplarisch nur ein Ausschnitt aus dem Glossar aufgeführt. Ein vollständiges Glossar findet sich im Anhang A.

Objekte der Anwendung:

- *Bild*

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen“, Hanser–Verlag / Dt. Übers. 1993 D. Märtin, S.186 f

Das Bild ist eine zweidimensionale Abbildung, die in einem bestimmten Dateiformat auf einem Fileserver abgelegt ist. In Form von mehreren Dateien können verschiedene Ausprägungen der Abbildung vorliegen.

- *Text*
  Informationen in Textform, die im Archiv liegen.
- *...(Anhang A)*

Objekte des Systems:

- *Anwendungsserver*
  Computer, auf dem die Anwendung abläuft, auf die ein Arbeitsplatzrechner zugreift. Der Anwendungsserver greift seinerseits über das Netzwerk auf einen Fileserver und einen Datenbankserver zu.
- *Fileserver*
  System, auf dem die Darstellungen in Dateien mit bestimmten Dateiformaten abgelegt sind.
- *Datenbankserver*
  Computer, auf dem die Datenbank installiert ist.
- *...(Anhang)*

1.1.1 Identifikation der Assoziationen zwischen Objekten

Jede Abhängigkeit zwischen zwei oder mehr Klassen ist eine *Assoziation*. Für die erarbeiteten Objektklassen werden die Assoziationen verbal aufgestellt[1,2.] Eine vollständige Auflistung befindet sich im Anhang A:

Anwendung *benutzt* Archiv(1)
Anwendung *enthält* GUI(1)
Datenbank *speichert* Archiv (1)
Datenbank *gehört zu* DBMS(2)
Fileserver *speichert* Bilddaten(1)
Fileserver *speichert* Textdaten(1)
Bilddaten *sind auf* Fileserver(2)
Textdaten *sind auf* Fileserver(2)
Karte *hat* Text(1)
Karte *hat* Bild(1)
Karte *gehört zu* Schlagwort(1)
Text *ist von* Ersteller(1) ...

---

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen“, Hanser–Verlag / Dt. Übers. 1993 D. Märtin, S. 191 f

[2] Die mit „(1)“ gekennzeichneten Assoziationen finden sich im Objektdiagramm wieder. Assoziationen, die mit „(2)“ signiert sind, sind im Objektdiagramm (Kap.: IV-IV1.1.2) implizit gegeben, da sie die Gegenrichtung einer anderen Assoziation widerspiegeln. Hat z.B. eine Darstellung mehrere Schlagworte, so hat jedes Schlagwort mindestens eine Darstellung.

1.1.2 Identifikation der Attribute von Objekten und Assoziationen

Attribute sind Eigenschaften eines Objektes. Sie sollten selbst keine Objekte sein und entsprechen normalerweise Substantiven, an die sich ein Genitiv anschließt, z.B. *Die Größe des Bildes*. Anders als Klassen und Assoziationen sind Attribute selten vollständig in der Problembeschreibung beschrieben[1].

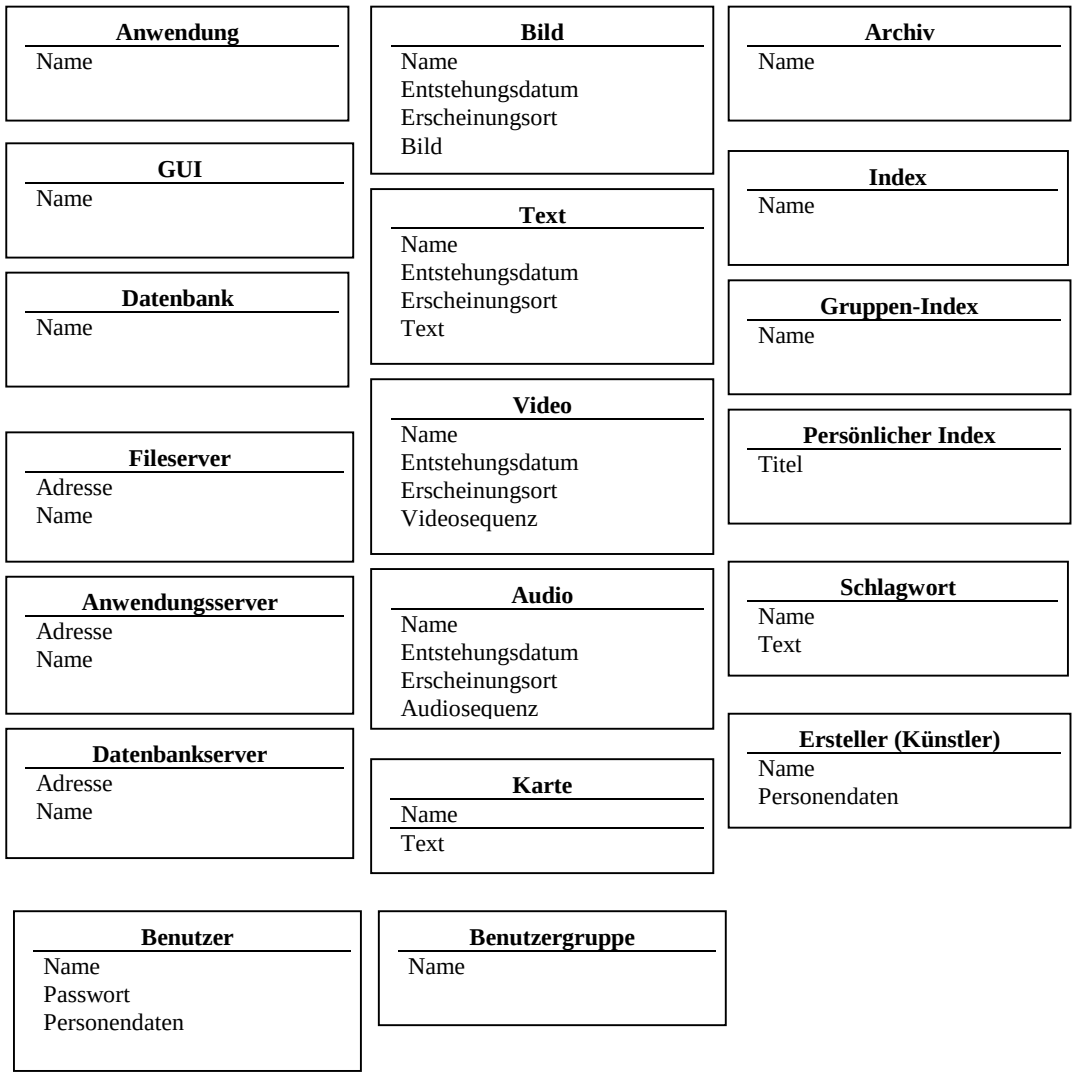


Abbildung IV-3: Einige Objekte mit Attributen

Das Objektdiagramm setzt sich aus den Objekten und ihren Assoziationen zusammen. An diesem Diagramm lassen sich Fehler leichter erkennen, als in den getrennten Auflistungen der Objekte und ihren Assoziationen untereinander.

---

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen“, Hanser–Verlag / Dt. Übers. 1993 D. Märtin, S. 196 f

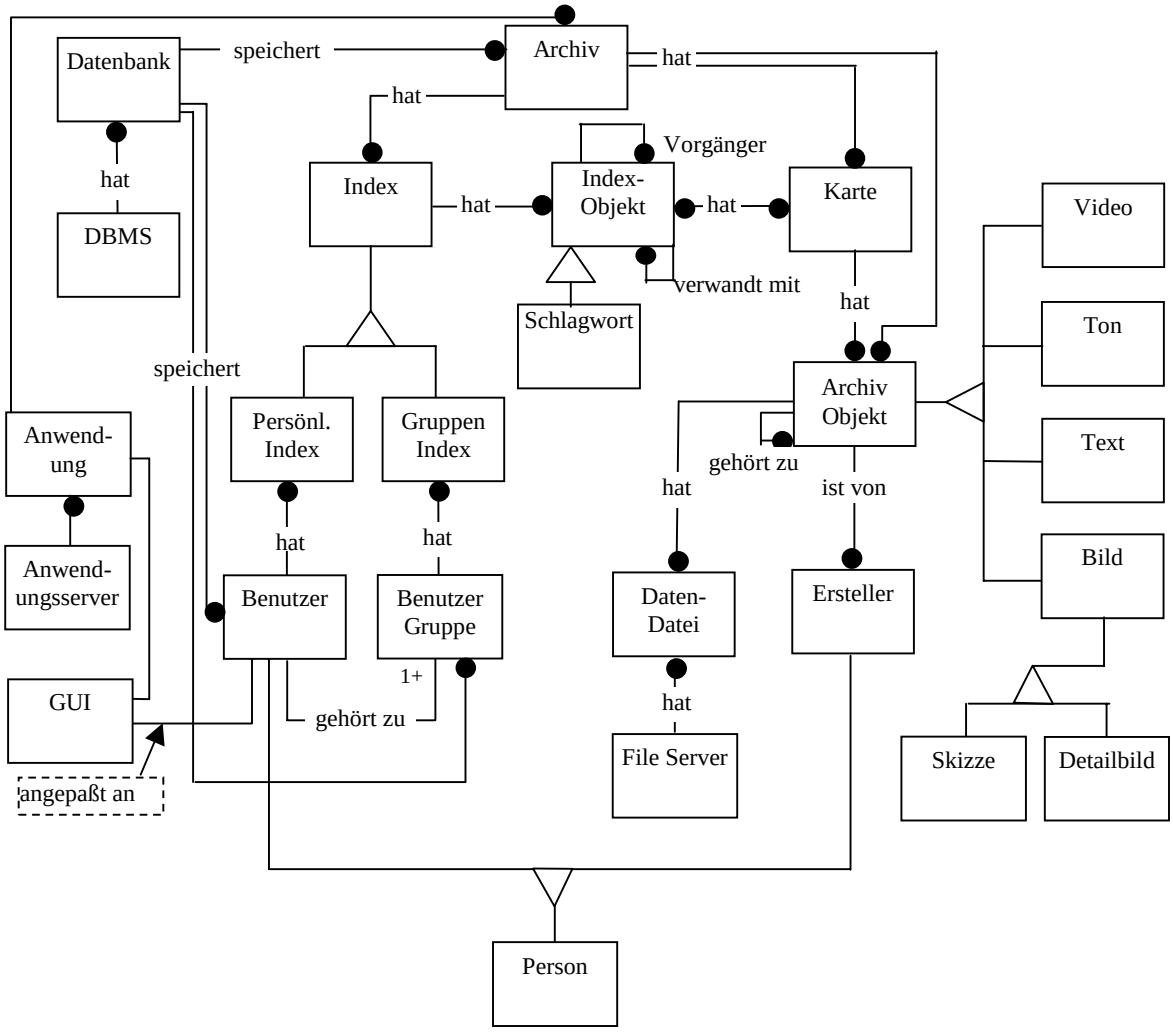


Abbildung IV-4: Objektdiagramm

1.2 Beispielszenarien typischer Interaktionssequenzen

In den beschriebenen Szenarien werden zwei Kategorien von Benutzern unterschieden. „B.“ ist ein anwendungsbezogener Benutzer, der in der Regel eingeschränkte Rechte bezüglich der Systemverwaltungsfunktionalität besitzt. „A.“ dagegen stellt einen Benutzer dar, der weiterreichende Rechte und Möglichkeiten bei der Nutzung und Verwaltung der Anwendung hat als „B.“. Die Beschreibungen für einen Benutzer „B.“ gelten implizit auch für „A.“

Szenario 1

B. möchte direkt einen der digitalisierten Bildindexe nutzen. Ein Bildindex ist in seiner Struktur unter Beachtung eines beabsichtigten semantischen Kontextes bestehender Informationen in das Datenmodell eingebettet. B. wird aufgefordert, den Benutzernamen und ein Paßwort einzugeben. B. wählt den Index aus, mit dem er arbeiten möchte. B. bekommt auf dem Bildschirm diesen Index hierarchisch dargestellt. Weiterhin erhält B. eine alphabetische Auswahl aller enthaltenen Schlagworte dieses Indexes, unabhängig von

der hierarchischen Einbettung. B. kann aus beiden Listen wählen. Für den selektierten Begriff kann B. verschiedene Operationen ausführen:

> *Szenario 1.1* : B. kann sich die verwandten Schlagworte dieses Begriffes anzeigen lassen. („Kaiser“, „König“, „Herrscher“, etc.). Wählt B. einen Begriff aus und übernimmt diesen, wird dieser in der alphabetischen Ausgangsauswahl markiert und steht für andere Operationen zur Verfügung.
>
> *Szenario 1.2* : B. kann sich den hierarchischen Kontext ausgeben lassen, in dem der Begriff steht. (Bsp.: Ist beim politischen Index der Begriff „Demokratie“ markiert und wird die Funktion für die hierarchische Darstellung angestoßen, werden sowohl die Oberbegriffe als auch die enthaltenen Stichwörter angezeigt). Ein beliebiges Schlagwort, unabhängig seiner hierarchischen Stellung, kann übernommen werden und wird in der alphabetischen Ausgangsliste markiert und steht für andere Operationen zur Verfügung.

Weitere Szenarien befinden sich im Anhang.

1.2.1 Identifikation der Ereignisse

Ein Szenario ist eine Folge von Ereignissen, die bei einer ganz bestimmten Ausführung eines Systems auftritt. Der jeweilige Umfang kann unterschiedlich sein, das heißt ein Szenario setzt sich aus einer Teilmenge der Gesamtereignisse des Systems zusammen. Die nachstehende Aufzählung enthält verschiedene Ereignisse, die in Form von Ereignispfaden[1] dargestellt werden können.

- Benutzer wählt ein Schlagwort aus
- Anwendung fordert Daten vom Datenbankserver an
- Anwendung fordert Daten vom Fileserver an
- Datenbankserver liefert Daten an die Anwendung
- Fileserver liefert Daten an die Anwendung
- Anwendung zeigt die zugehörigen Informationen (Darstellungen, Texte, etc.) zu dem Schlagwort
- ...(Anhang)

Im Folgenden wird ein Ereignispfad exemplarisch herausgegriffen, um einzelne Abläufe zu spezifizieren. Jeder Ereignispfad stellt eine in sich abgeschlossene Aktion dar, ähnlich einer Transaktion. Ereignispfade lassen sich zusammensetzen, wenn jeweils die Nach- und Vorbedingungen übereinstimmen.

<u>Verwandte Schlagworte:</u>

Für ein selektiertes Schlagwort fordert der Benutzer alle Schlagworte an, zu denen eine inhaltliche Verwandtschaft besteht. Die Anwendung holt die erforderlichen Schlagworte aus der Datenbank und zeigt sie an.

---

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen, Hanser–Verlag / Dt. Übers. 1993 D. Märtin, S. 105 f

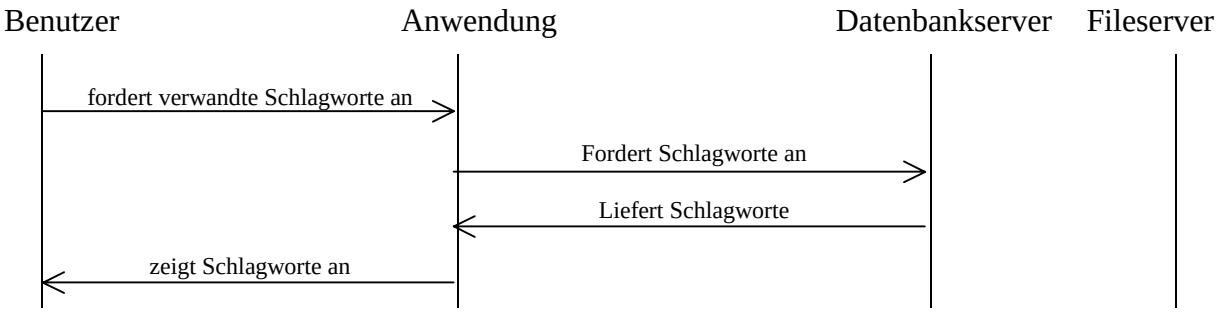


Abbildung IV-5: Anforderung verwandter Schlagworte

1.2.2 Ereignisflußdiagramm

Das Ereignisflußdiagramm faßt die Ereignisse zwischen Klassen ohne Berücksichtigung ihrer Reihenfolge zusammen. Es nimmt die Ereignisse aus allen Szenarios, auch

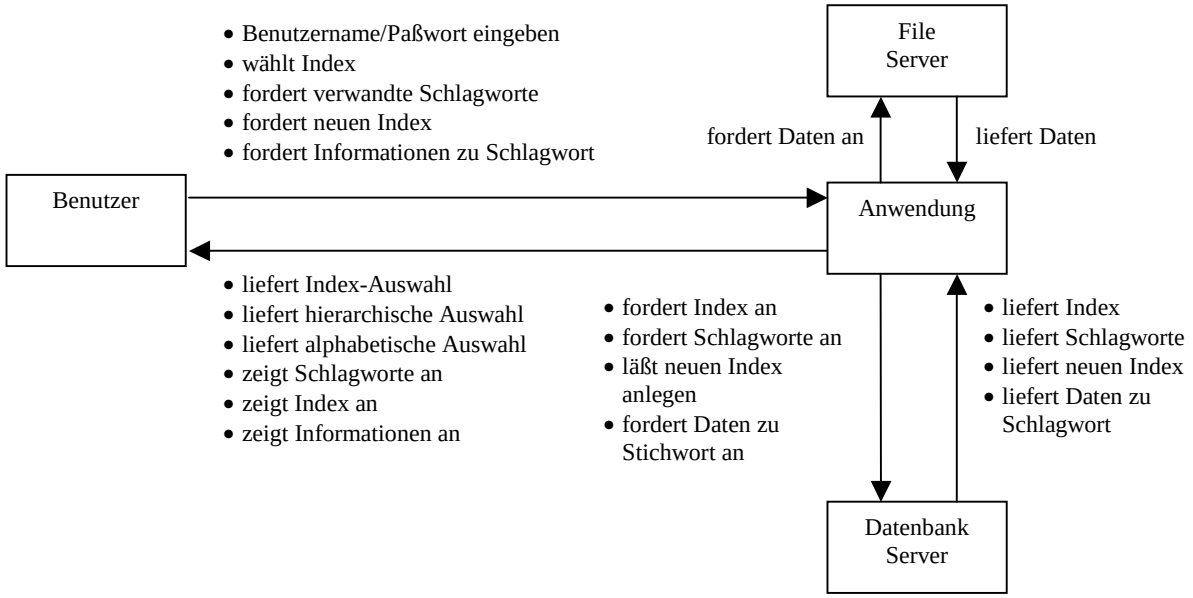


Abbildung IV-6: Ereignisflußdiagramm

Fehlerereignisse, auf. Es ist in der dynamischen Modellierung das Gegenstück zum Objektdiagramm. Pfade in Objektdiagrammen zeigen mögliche Informationsflüsse. Pfade in Ereignisdiagrammen zeigen mögliche Kontrollflüsse[1].

Die Ereignisflußdiagrammabbildung zeigt die Zuordnung der Ereignisse der WEL zu den Klassen. Es wird ersichtlich, wie Objekte bezüglich der Kontrollflüsse miteinander agieren.

---

[1] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen“

1.2.3 Zustandsdiagramm für die Klasse *Anwendung*

Zustandsdiagramme werden für alle Klassen mit einem nicht trivialen dynamischen Verhalten erstellt (als Beispiel hier nur für die Klasse *Anwendung*). Es zeigt die Ereignisse, die das Objekt sendet oder empfängt und die daraus resultierenden Zustandsübergänge. Jedes Szenario bzw. jeder Ereignispfad entspricht hier einem Pfad durch das Diagramm. Bei der Entwicklung dieser Diagramme beginnt man den Ereignispfaden, die sich auf die aktuelle Klasse auswirken. Das Zustandsdiagramm einer Klasse ist fertig, wenn es alle Szenarios abdeckt und alle Ereignisse berücksichtigt, die sich auf ein Objekt dieser Klasse in einer seiner Zustände auswirken können.

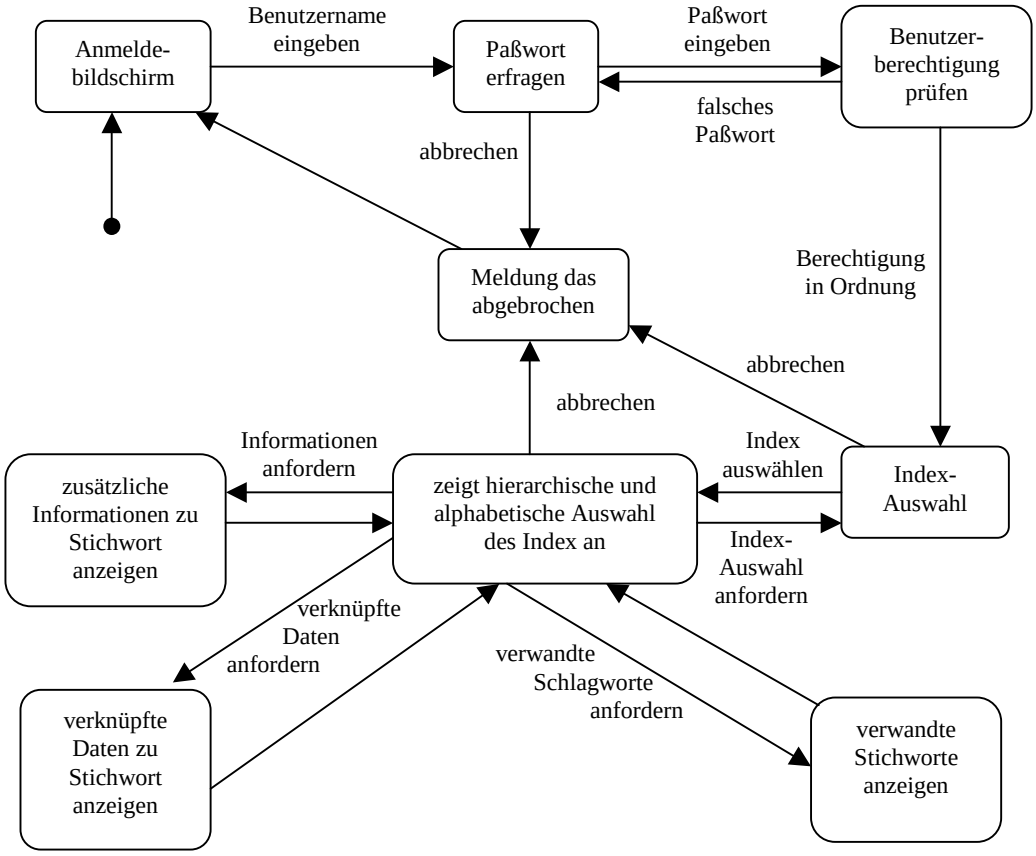


Abbildung IV-7: Zustandsdiagramm für die Klasse *Anwendung*

1.3 Partitionen des Systems

Die Partitionierung des Systems erfolgt durch die Aufteilung in Aufgabenbereiche, da sich diese in diesem Fall gut gegeneinander abgrenzen lassen. Dadurch ergibt sich eine Dreiteilung des Systems in Arbeitsplatz, Applikationsserver und Archivsystem. Das Archivsystem ist für den Zugriff auf die relationale Datenbank und auf das Dateisystem zuständig. Diese Partition übernimmt die Verwaltung, Speicherung und die Wiedergewinnung der persistenten Daten. Die Partition Arbeitsplatz stellt die Benutzerschnittstelle mit den zugehörigen Funktionen zur Verfügung und wird auf dem Arbeitsplatzrechner ausgeführt. Die letzte Partition des Systems, der Applikationsserver, übernimmt als Hauptaufgabe die Bereitstellung der Datenobjekte. Diese Partition enthält

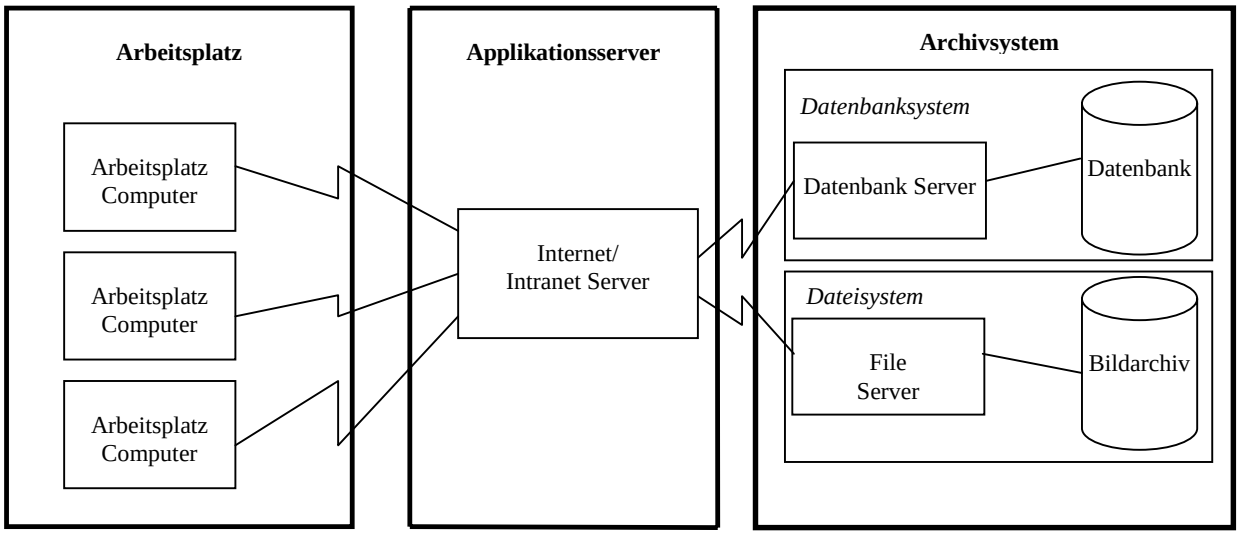


Abbildung IV-8: Partitionen des Systems nach Aufgabenbereichen

die Anwendungslogik, die unter anderem dafür verantwortlich ist, daß die Clients die angeforderten Objekte erhalten und die von den Clients vorgenommenen Änderungen an den Objekten zur persistenten Speicherung an die Datenbankpartition des Systems weitergereicht werden. Die Partitionierung nach Aufgaben der Systemkomponenten soll hier reichen. Bei großen Systemen sind sicherlich weitere Unterteilungen (z.B. Aufteilung des Archivsystems in Datenbank- und Dateisystem) notwendig, damit das System handhabbar wird.

V Realisierung in einer speziellen Programmiersprache

1 Verwendung von Java als objektorientierte Umgebung

Im Rahmen des vorliegenden Anforderungsprofils werden an die Programmiersprache hohe Anforderungen in Bezug auf die Anbindung von Datenbanken, die Unterstützung unterschiedlicher Plattformen und die Erstellung verteilter Anwendungen gestellt. Die Programmiersprache Java unterstützt mit ihren Standardklassen die Entwicklung eines solchen Systems entscheidend. In den folgenden Kapiteln wird auf Java und die verwendeten Konzepte näher eingegangen.

1.1 Die Grenzen von HTML

HTML[1] ist eine Sprache zur plattformunabhänigen Erstellung und Formatierung von Dokumenten (HTML-Seiten). Bei der Realisierung interaktiver Benutzerschnittstellen wird sehr oft HTML in Verbindung mit dem CGI[2], einer Schnittstelle zu Programmen, die dynamisch HTML-Seiten erzeugen, eingesetzt. Mit HTML lassen sich Texte in beliebiger Form darstellen und es ist möglich, interaktive Elemente[3] in diese Texte einzubetten. Komplexe Funktionen können in HTML aber nicht direkt lokal auf dem Arbeitsplatzrechner realisiert werden. Wenn man also mit Hilfe von HTML ein Informationssystem im Netz zur Verfügung stellt, so müssen alle Berechnungen, sowie das Speichern von Zwischenergebnissen auf dem Server geschehen und jede Eingabe des Benutzers, die eine Änderung der HTML-Seite zur Folge hat, erfordert einen Verbindungsaufbau und die wiederholte Generierung und Übertragung der kompletten Benutzerschnittstelle durch den Server. Auf dem Arbeitsplatzrechner befindet sich also ausschließlich die Benutzeroberfläche. Bei einer kleinen Zahl von Benutzern des Systems und einem schnellen Netzwerk ist die Entwicklung eines solchen Systems mit HTML-Seiten sinnvoll. Im Internet sind diese Voraussetzungen aber häufig nicht gegeben. Ein weiteres Problem stellt die Übertragung der Bildinformationen dar. Bilder werden bei HTML-Seiten mit Hilfe von Bilddateien, die über das Netz geladen werden, dargestellt. Das Bild wird also bei jedem Aufruf der HTML–Seite kopiert und kann vom Anwender sehr einfach kopiert und lokal gespeichert werden. Diese Möglichkeit ist aber im Rahmen des in dieser Arbeit vorgestellten Projektes aus urheberrechtlichen Gründen ungeeignet. Es können aber auch wirtschaftliche Interessen gegen diese Möglichkeit sprechen.

1.2 Vorstellung von Java

Die objektorientierte Programmiersprache Java ist von einem kleinen Team um James Gosling bei Sun Microsystems entwickelt worden[4]. Das ursprüngliche Java-Team arbeitete an der Software-Erstellung für Konsumelektronik. Dieses Team stellte fest, daß

---

[1] HTML - Hyper Text Markup Language.

[2] CGI – Common Gateway Interface

[3] Interaktive Elemente sind z.B. Querverweise zu anderen Dokumenten, Schaltflächen, Eingabefelder, etc.

[4] A Brief History of the Green Project, http://www.javasoft.com/people/jag/green/index.html

herkömmliche Sprachen wie C oder C++ dafür nicht geeignet waren. Programme, die mit Hilfe dieser plattformabhängigen Sprachen erstellt wurden, müssen für jeden speziellen Prozessortyp übersetzt werden. Sobald es einen neuen Prozessor gibt, muß die Software neu übersetzt werden, um die Eigenschaften des neuen Prozessors zu nutzen. Das heißt, daß für jeden Prozessortyp die entsprechend übersetzte Software vorhanden sein muß. 1990 wurde mit der Entwurfsphase einer neuen Programmiersprache begonnen, die auf die Bedürfnisse der Konsumelektronik besser zugeschnitten sein sollte als die traditionellen Programmiersprachen wie C und C++. Das Ergebnis war Java, eine sehr schnelle, kleine, verläßliche Sprache, die auf einer Vielzahl von Prozessoren laufen kann. Den Durchbruch schaffte Java, als das Java-Team feststellte, das die Sprache Ideal für die plattformunabhängige Entwicklung von Anwendungen für das Internet wäre und einen Webbrowser („WebRunner“) komplett in Java entwickelte. Kompilierte Java-Programme können nicht direkt ausgeführt werden, sie werden von einem Interpreter, einer virtuellen Maschine, ausgeführt. Die virtuelle Java Maschine (JVM – *Java virtuell machine*) ist in der Regel im Internet-Browser[1] integriert. Der Bytecode, das kompilierte Java-Programm, wird von der JVM ausgeführt und in die entsprechenden plattformabhängigen Anweisungen umgesetzt.

1.3 Java in Rechnernetzen

Java ist eine Programmiersprache, die ausdrücklich für verteilte Applikationen in Netzwerken entwickelt wurde. Zu diesem Zweck stellt Java dem Entwickler eine Vielzahl von Mitteln zur Verfügung, mit denen die Entwicklung von Netzwerkapplikationen erheblich erleichtert wird. Für die Dienste vom Netzwerkprotokoll über den Verbindungsaufbau bis hin zum Zugriff auf entfernte Objekte stellen die beim JDK 1.1 (Java Development Kit) mitgelieferten Klassenbibliotheken bereits eine weitgehende Unterstützung zur Verfügung, so daß der Programmierer nur die eigentliche Anwendungslogik entwickeln muß. Das Netzwerk bleibt für ihn nahezu transparent. Das Paket[2] *java.net* enthält die Klassen, die nötig sind, um eine Netzwerkverbindung zwischen zwei Rechnern herzustellen. Die Pakete *java.rmi* und *java.rmi.server*, die das Java eigene *Remote Method Invocation* Protokoll (siehe Kapitel 1.6.2: Remote Method Invocation) verwenden, werden für die Referenzierung entfernter Objekte benötigt. Das *Java Interface Description Language* Package (*JavaIDL*, siehe Kapitel V1.6.3: CORBA) ist für den entfernten Zugriff auf *CORBA*[3]-Objekte zuständig. Grundlage für *RMI* und *Java IDL* ist das *Object Serialization Protocol*, mit dessen Hilfe es möglich wird, Objekte zu serialisieren, das heißt, auf einen Bytestrom zu schreiben, bzw. von einem Bytestrom zu lesen. Von Entwicklungsbeginn an wurde Java für die Verwendung in heterogenen und offenen Netzen entwickelt. Sicherheitsmechanismen sind aus diesem Grund von großer Bedeutung und finden sich schon in den Konzepten der Sprache wieder[4].

---

[1] Ein Internet-Browser, Webbrowser oder einfach nur Browser ist eine Anwendung, mit der es möglich ist, im Internet, speziell dem World Wide Web (WWW), zu navigieren.

[2] Ein Paket (*package)* ist eine Sammlung von zusammengehörigen Klassen.

[3] CORBA – Common Object Request Broker Architecture

[4] Norman Hendrich, Java für Fortgeschrittene, Springer-Verlag Berlin Heidelberg 1997, S. 7 ff

1.4 Java und Datenbanken

Die Realisierung von Informationssystemen erfordert den Zugriff auf persistente und meist umfangreiche Datenbestände. Es ist also für Programmiersprachen zur Erstellung von Informationssystemen notwendig, daß sie mindestens eine Schnittstelle anbieten, mit der die Nutzung von Informationen aus vorhandenen Datenbeständen möglich ist. Java bietet gleich mehrere Möglichkeiten an.

Eine Möglichkeit ist der Zugriff auf Datenbanken über *native* Methoden. Java nutzt dabei plattformabhängige native Funktionsbibliotheken, die vom Betriebssystem oder dem Datenbanksystem zur Verfügung gestellt werden. Diese Funktionsbibliotheken enthalten die notwendige Funktionalität zur Bearbeitung der Daten. Die Vorteile dieser Vorgehensweise sind die Nutzung vorhandener Ressourcen, in diesem Fall der Funktionsbibliotheken und hohe Performanz, da die Bibliotheken in Maschinencode vorliegen und für die jeweilige Plattform optimiert sind. Der Nachteil dieser Vorgehensweise ist die geringe Flexibilität. Die Anwendung läuft nur auf einer Plattform und nur zusammen mit den verwendeten Funktionsbibliotheken. Benötigte Funktionen, die nicht in den Funktionsbibliotheken implementiert sind, müssen, wenn überhaupt möglich, unter relativ hohem Aufwand zusätzlich implementiert werden.

Eine weitere Möglichkeit der Datenbankanbindung besteht in der Verwendung von CORBA. Der Vorteil dieser Vorgehensweise gegenüber der nativen Methode ist die Beibehaltung des objektorientierten Programmierstils und die Möglichkeit, das CORBA-Objekte mit unterschiedlichen Programmiersprachen entwickelt werden können. Die Java-Objekte sind also auch CORBA konforme Objekte und können mit anderen CORBA-Objekten, welche auf die Datenbank zugreifen, Nachrichten austauschen.

JDBC (Java Database Connectivity) ist das Java API[1] zur Anbindung an relationale Datenbanken. JDBC stellt Methoden zur Verfügung, mit denen eine Verbindung zur Datenbank hergestellt werden kann, SQL Anweisungen zu dieser Datenbank geschickt und die Ergebnisse zurückgeliefert werden können. Auch JDBC hat einen Nachteil. Der liegt in der Verwendung von ausschließlich relationalen Datenbanken, auf die sich Objektsysteme nur schwer und unzulänglich abbilden lassen. Die Verwendung von objektorientierten Datenbanken ist mit JDBC nicht möglich.

Die Verbindung einer objektorientierten Datenbank mit einer objektorientierten Programmiersprache hat einige schöne Eigenschaften. Im Kern ist dies der Anschluß der Datenbank an die Programmiersprache ohne semantische Lücken, so wären z.B. persistente Objekte ohne Umwege realisierbar. Für den Einsatz von objektorientierten Datenbankmanagementsystemen (ODBMS) gibt es heute aber noch Hindernisse:

- Die wenigen Anbieter von ODBMS sind relativ junge und kleine Unternehmen mit zumeist ungewisser Zukunft. Die Pilotanwender begeben sich in die Abhängigkeit eines Herstellers.
- Es gibt noch wenig Erfahrung im Einsatz der ODBMS–Technologie. Insbesondere für Anwendungen mit großem Datenvolumen oder hoher Transaktionslast haben sich diese Systeme noch nicht bewährt.

---

[1] API – Applictation Programming Interface

- Relationale Datenbanksysteme (RDBMS) haben sich bewährt und sind zur Zeit den objektorientierten Datenbanken technisch überlegen, das heißt eine Ablösung durch eine neue Technologie nur wegen der leichteren Implementierung von Systemen ist unter wirtschaftlichen Aspekten noch nicht zu rechtfertigen. Von einem Wechsel wären zudem nahezu alle Komponenten vorhandener Informationssysteme betroffen.

Eine Ablösung der RDBMS durch ODBMS ist heute in den meistens Fällen also unwirtschaftlich und ein technisches Wagnis[1]. Eine Anbindung von Java an ODBMS ist aber möglich und wird von meisten kommerziellen Herstellern von ODBMS angeboten.

1.5 JDBC und JDBC/ODBC–Bridge

Eine Verbindung zwischen objektorientierter Software und relationalen Datenbanken ist aus den oben genannten Gründen also zur Zeit noch notwendig, sofern man auf Datenbanktechnologien angewiesen ist.

JDBC spezifiziert alle notwendigen Klassen und Methoden die für diese Verbindung implementiert werden müssen. Das Package *java.sql* enthält die Klassen, die den JDBC Treiber benutzen. Es gibt vier Arten von JDBC Treibern (siehe Abbildung V-1). Sie unterscheiden sich in der Art, wie sie die Verbindung zur Datenbank herstellen. Die Schnittstelle zur Anwendung ist bei allen Treibern identisch.

JDBC-Netzwerkprotokoll Treiber (in Java geschrieben): Dieser Treiber übersetzt JDBC Aufrufe in ein vom DBMS unabhängiges Netzwerkprotokoll welches vom Server in ein DBMS Protokoll übersetzt wird. Dieser Server Middleware ist es möglich Java Applets mit unterschiedlichen Datenbanken zu verbinden. Das verwendete Protokoll ist abhängig von der verwendeten Datenbank. Im Allgemeinen ist dieses aber die flexibelste JDBC Lösung, da alle großen Datenbankanbieter JDBC Treiber dieser Art anbieten werden, um die Verwendung Ihrer Datenbanktechnologie in Intranets und im Internet mit JDBC zu ermöglichen.

JDBC/ODBC Bridge und ODBC Treiber: Die JDBC-ODBC Bridge realisiert den JDBC Zugriff auf die Datenbank über vorhandene ODBC Treiber. Der ODBC Treiber muß auf jedem Client-Rechner installiert sein, damit auf die Datenbank zugegriffen werden kann. ODBC Treiber sind native Programme, das heißt, das Java Applets, die über das Netzwerk geladen werden, diesen Code, wegen der strengen Sicherheitsanforderungen, nicht direkt ausführen dürfen.

Nativer-API Treiber (teilweise in Java geschrieben): Diese Art von Treiber konvertiert JDBC Aufrufe in API Aufrufe für Oracle, Sybase, Informix, DB2, oder andere DBMS. Wie bei der JDBC-ODBC Bridge werden hier auch native Programme verwendet, was zu Restriktionen bei der Verwendung von Applets führt.

---

[1] Hans Dicken, JDBC–Internet-Datenbankanbindung mit Java, 1. Auflage, International Thomsom Publishing Bonn Albany 1997, S. 67ff

Nativer-Netzwerkprotokoll Treiber (in Java geschrieben): Dieser Treiber konvertiert JDBC Aufrufe in ein Netzwerkprotokoll das direkt vom DBMS verstanden wird. Dieses erlaubt die direkte Kommunikation des Clients mit dem Datenbankserver und ist eine praktikable Lösung für Intranets. Diese Treiber müssen vom Datenbankanbieter zur Verfügung gestellt werden, da die verwendeten Protokolle datenbankabhängig sind. Eine Verwendung im Internet ist hier nicht praktikabel, da auf allen Rechnern dieses Protokoll installiert sein müßte, und Applets wegen der Sicherheitsrestriktionen diesen nativen Code nicht verwenden dürfen.

ODBC Treiber sind für viele Datenbanken schon vorhanden. Die JDBC/ODBC Bridge

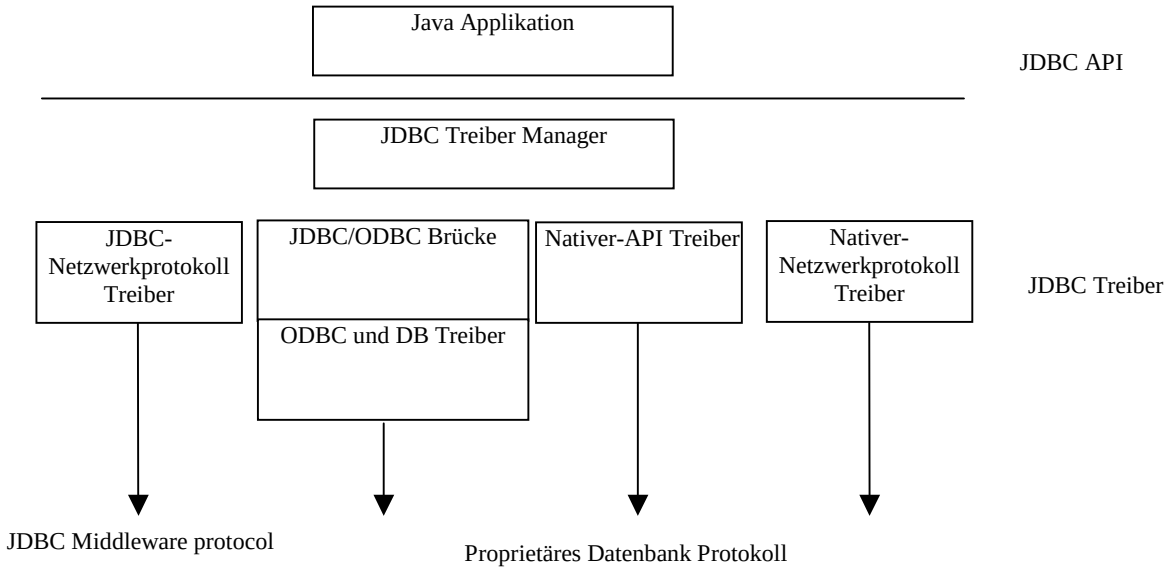


Abbildung V-1: Übersicht der JDBC Treiber

wird im JDK 1.1 mitgeliefert. Das heißt, daß keine weiteren Treiber für die Verwendung von Datenbanken zusammen mit Java benötigt werden. Die Einschränkung, daß Applets, die nicht lokal gestartet wurden, nicht auf lokale Ressourcen zugreifen dürfen, ist nur dann von Bedeutung, wenn das Applet direkt auf die Datenbank zugreifen soll. Kommuniziert das Applet mit der Datenbank aber nur indirekt über eine Serverapplikation, so ist diese Einschränkung nicht von Bedeutung. In dem hier vorgestellten Projekt kommuniziert nur die Serverapplikation mit der Datenbank (siehe Abbildung V-2). Das Applet nutzt die von der Serverapplikation mit RMI[1] zur Verfügung gestellten entfernten Objekte, die eine Verbindung zu Datenbank herstellen können.

[1] RMI: Remote Method Invocation

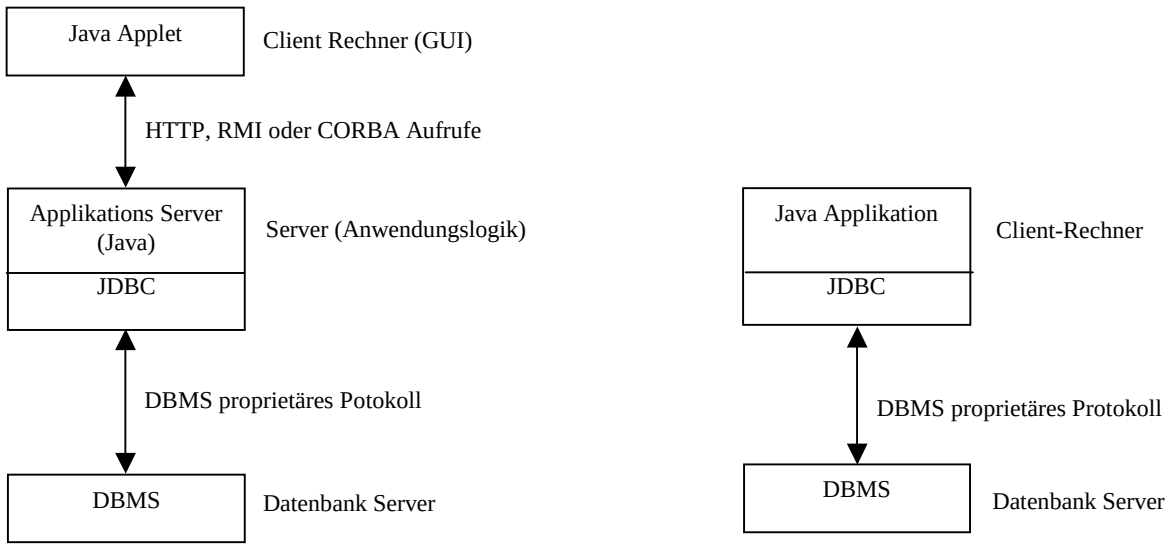


Abbildung V-2: Datenbankzugriff mit Serverapplikation oder direkt vom Client-Rechner

1.6 Verteilte Anwendungen in Java

Im folgenden wird auf die Kommunikationsmöglichkeiten von Anwendungen untereinander und mit anderen Systemen näher eingegangen. Java unterstützt dabei verschiedene Ebenen der Abstraktion vom Netzwerk. Diese reichen vom expliziten Verbindungsaufbau und Protokollvereinbarungen bis hinauf zum Austausch von entfernten Objekten.

1.6.1 Socket-Verbindungen

Socket-Verbindungen sind auf relativ niedrigem Abstraktionsniveau angesiedelt. Der Entwickler muß sich um den Verbindungsaufbau und -abbau explizit kümmern. Der Datentransfer muß von „Hand“ angestoßen werden. Daten werden über Datenströme (streams) ausgetauscht. Der Transfer von Objekten über Datenströme erfordert ihre vorherige Serialisierung. Java-Objekte, die serialisierbar sein sollen, müssen die Schnittstelle *Serializable* implementieren. Dasselbe gilt für alle Objekte, aus denen das zu serialisierende Objekt besteht. Ein weiterer Grund, der gegen Datenströme als Kommunikationsweg zwischen unabhängigen Anwendungen spricht ist, daß Datenströme nicht in die Konzepte der objektorientierten Softwareentwicklung passen. Objekte tauschen in der objektorientierten Welt Nachrichten aus, der Datenaustausch über Datenströme entspricht nicht diesem Konzept.

1.6.2 Remote Method Invocation

Verteilte Systeme im objektorientierten Umfeld erfordern ein verteiltes Objektmodell und einen verteilten Nachrichtenfluß, welcher einem entfernten Methodenaufruf entspricht. Die neben dem RMI derzeit in Java existierenden Möglichkeiten einer Kommunikation zwischen Clients und Servern sind die Basiskommunikation über Datenströme und die Datenbankschnittstelle JDBC. Zur Realisierung von objektorientierten verteilten Systemen

sind diese nicht ausreichend. Die Kommunikation von verteilten Objekten wird vom Java Remote Method Invocation (RMI) weitgehend automatisiert und ist damit ein geeigneter Ansatz ein Server/Client Modell in der Java-Umgebung, d.h. bei miteinander kommunizierenden Objekten in verschiedenen Java Virtual Machines (JVM), zu realisieren (Ausführliche Beschreibung im Anhang).

Die beiden wichtigsten Punkte bilden die Möglichkeit, komplexe Objekte zwischen Client und Servern auszutauschen und transparent Nachrichten an entfernte Objekte zu senden, d.h. Methoden entfernter Objekte aufzurufen.

RMI Systemarchitektur

Die Systemarchitektur des RMI ist in drei voneinander vollständig unabhängige, eigenständige Ebenen aufgeteilt. Die *Stub/Skeleton*-Schicht, die *Remote Reference*-Schicht und die *Transport*-Schicht.

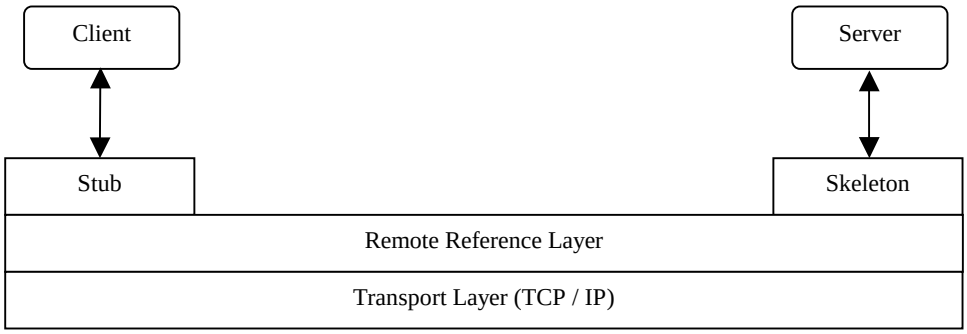


Abbildung V-3: Schichten des Java RMI-Protokolls

Die Zuordnung von Client- und Serverfunktion gilt stets nur für eine konkrete Kommunikation zweier Objekte. Somit kann ein Objekt Client bezüglich eines Zugriffs auf ein entferntes Serverobjekt sein und gleichzeitig einen Serverdienst für ein anderes Objekt zur Verfügung stellen, welches eventuell auf einem anderen, entfernten Rechner existiert.

Die Stub/Skeleton-Schicht

Wenn von einem Objekt im Client eine Methode eines Serverobjekts aufgerufen wird, das heißt dem Serverobjekt eine Nachricht gesendet werden soll, wird ein lokales Stellvertreterobjekt verwendet, welches als *Stub* bezeichnet wird. Dieses existiert im Adressraum der JVM, die das Clientobjekt ausführt, und stellt in diesem Adressraum ein reguläres Java-Objekt dar. Die *Stub*-Methode faßt die Parameter des Methodenaufrufs zusammen (*parameter marshalling*) und bildet mit der eindeutigen Referenz des Remote-Objekts und der Nummer der aufgerufenen Methode einen Datenblock, der über die *Remote Reference*-Schicht zum Server gesendet wird.

Auf der Serverseite empfängt das entfernte Stellvertreterobjekt des eigentlichen Remote-Objektes, bezeichnet als *Skeleton*, über die dortige *Remote Reference*-Schicht den Datenblock und ruft die gewünschte Methode des Serverobjekts auf. Rückgabewerte, wozu auch ausgelöste Ausnahme- beziehungsweise Fehlerobjekte (*Exceptions*) gehören, werden auf dem analogen Weg zum Clientobjekt zurückgesendet.

Die Remote Reference-Schicht

In dieser Schicht wird die Kommunikation mit der Transportschicht geregelt und im wesentlichen das Protokoll bestimmt, nach dem die entfernten Referenzen zugeordnet werden. Dies ist unabhängig von den *Stub*- und *Skeleton*-Objekten.

Die *Remote Reference*-Schicht überträgt die Daten über abstrakte Verbindungskanäle, die die darunterliegende Transport-Schicht implementiert.

Die Transport-Schicht

In der Transport-Schicht werden die folgenden Aufgaben wahrgenommen:

- Aufbau, Verwaltung und Überwachung von Verbindungskanälen zu entfernten Adressräumen
- Verwaltung einer Referenztabelle von aktuellen entfernten Objekten, die im lokalen Adressraum verwendet werden
- Annahme von Verbindungswünschen anderer Objekte
- Aufbau der Verbindungskanäle

Beispiel:

RMI Interface:

```
public interface RMIObjectInterface extends Remote

/*  die Methode execute(Object obj) wird für den
    entfernten Zugriff deklariert  */
public abstract void execute(Object obj)
  throws RemoteException;
```

Klassen Implementierung:

```
public class RMIObject implements RMIBeispielInterface
    public static void main(String argv[]){
    /*  das Objekt wird für den entfernten Zugriff
        zur Verfügung gestellt und bekannt gemacht  */
      UnicastRemoteObject.exportObject(this);
      Naming.rebind(this.toString(),"rmiServer");
    }
    /* Implementierung der "RMI"-Methode  */
    public void execute(Object obj) throws RemoteException {
      System.out.println( obj.toString() );
    }
```

Programmfragment:

```
rmiObject =
  (RMIObject) Naming.lookup("rmi://localhost/rmiServer");
rmiObject.execute(this);
```

1.6.3 CORBA

RMI ist ein proprietärer ORB (*Object Request Broker*). Mit ihm können ausschließlich Java-Objekte untereinander Nachrichten austauschen. Der Austausch von Nachrichten zwischen Java-Objekten und Objekten aus der C++, Eiffel oder Smalltalk Welt ist mit RMI nicht möglich. Abhilfe kann hier die von der OMG[1] entworfene *Interface Definition Language,* IDL, zusammen mit dem *Internet Inter Object Request Broker Protocol,* IIOP, schaffen. Beide sind Teil der CORBA[2] 2.0 Spezifikation[3]. Der Austausch von Nachrichten zwischen den Objekten geschieht ähnlich wie bei RMI über Stubs und Skeletons, die die Nachrichten in das benötigte Format transformieren. Auch die Deklaration im IDL-Modul ist dem Interface von RMI sehr ähnlich:

IDL Interface:

```
module IDLObjectExample {
       interface IDLObjectInterface {
              void execute(in object obj);
       };
};
```

Die Ähnlichkeit der Notation in IDL und in Java kommt von den Ursprüngen beider Sprachen. Beide sind an die Notation von C angelehnt. Dieser Umstand kommt der Entwicklung von Systemen, die Java Programme, C Programme und CORBA als „Object Broker“ verwenden, sehr entgegen. Sind die Entwickler mit einer der Sprachen vertraut, so finden sie sich in den anderen Sprachen sehr schnell zurecht.

---

[1] OMG: Die Object Management Group ist ein herstellerübergreifendes Konsortium von über 500 Unternehmen, darunter auch IBM, DEC, Hewlett-Packard und Microsoft. Sie wurde 1989 mit dem Ziel gegründet, eine geeignete Architektur für die Verteilung und Zusammenarbeit objektorientierter Softwarebausteine in vernetzten, heterogenen Systemen bereitzustellen.

[2] CORBA: Common Object Request Broker Architecture: Von OMG entwickelte Architektur zur Verteilung und Zusammenarbeit von objektorientierten Softwaresystemen

[3] The Common Object Request Broker: Architecture and Specification, Revision 2.0, Juli 1995, aktualisiert: Juli 1996, The Object Management Group (OMG)

VI Architektur eines Prototypen

Dieser Abschnitt befaßt sich mit der Beschreibung der Implementation eines Prototypen für die WEL. Er stellt den ersten Schritt auf dem Weg zur Erstellung einer einsatzfähigen Anwendung dar und versucht, die im Laufe dieser Arbeit gestellten Anforderungen zu berücksichtigen. Die in den vorangegangenen Kapiteln erzielten Ergebnisse bilden die Grundlage für die hier beschriebenen Implementationsansätze.

Der näheren Erläuterung des Begriffes *Prototyp* widmen sich die nächsten Zeilen, da sich unterschiedliche Ansätze für die Prototyp-Erstellung etabliert haben.

1 Prototypen und ihre Varianten

Der Begriff *Prototyp* wird in der Systementwicklung schon seit Mitte der 70er Jahre verwendet. In der Systementwicklung wird beim Prototyping auf schriftliche Dokumente und eine umfangreiche Analyse der Problemstellung und ihrer möglichen Lösungen weitgehend verzichtet. Auf der Grundlage von Gesprächen der Entwickler mit den Benutzern soll mit fortgeschrittenen Softwarewerkzeugen rasch ein Prototyp erstellt werden, der als Grundlage weiterer Diskussionen dient. In wenigen Prototyp-Zyklen soll so eine tragfähige Lösung entwickelt werden[1]. Die Aufteilung des Prototyping in mehrere Zyklen birgt den Vorteil eines hohen Grades an Interaktion zwischen Entwickler und Benutzer. Die Erstellung vollzieht sich in kleineren Schritten und ist sowohl von den benutzerseitigen Anforderungen als auch von der technischen Umsetzung für alle Teilnehmer eines Projektes am einfachsten nachzuvollziehen.

Prototypen lassen sich grob in zwei Arten unterscheiden[2]:

- *Exploratives Prototyping:* Diese Variante dient hauptsächlich als Diskussionsgrundlage zwischen Entwicklern und Benutzern. Es werden die Anforderungen an ein System geklärt. Unterschiedliche Prototypen verdeutlichen verschiedene denkbare Alternativen. Die Benutzer des zukünftigen Systems können mit dem Prototypen „spielen“ und so Schwächen und Stärken des Prototypen erkennen. Diese Ergebnisse werden im nächsten Prototyp berücksichtigt.
- *Experimentelles Prototyping:* Im Vordergrund steht die Frage nach der technischen Realisierung einer Problemlösung. Untersucht wird zum einen die Realisierbarkeit, zum anderen aber auch die Angemessenheit für das zu entwickelnde System.

Die Verwendung des endgültigen Prototypen läßt eine weitere Trennung zu:

- Der Prototyp dient als lauffähige Spezifikation des Zielsystems. Diese Art von Prototyp spezifiziert nur einige Aspekte des Zielsystems, z.B. kann bei dieser Art die Fehlerbehandlung des Systems noch völlig fehlen. Das Zielsystem stellt eine betriebsfähige Anwendung dar.

---

[1] Budde, R., Kautz, K.-H., Kuhlenkamp,K., Züllighoven,H.:Prototyping – An Approach to Evolutionary System Developement. Heidelberg u.a.: Springer 1990

[2] Floyd, C.: A Systematic Look at Prototyping. In: Budde, R., Kuhlenkamp, K., Mathiassen, L., Züllighoven, H. (Hrsg.): Approaches to Prototyping. Berlin Heidelberg, New York, Tokyo: Springer 1984.

- Das Zielsystem ist der letzte Prototyp. In diesem Fall wird der Prototyp so lange weiterentwickelt, bis er den Anforderungen an das Zielsystem entspricht.

2 Der erste Prototyp für die WEL

Im Rahmen des WEL-Projektes wurde parallel zu den Gesprächen zwischen Entwicklern und Benutzern der WEL ein erster Prototyp entwickelt. Dieser Prototyp wird sowohl explorativ als auch experimentell eingesetzt. Mit seiner Unterstützung sollen die Anforderungen an den Funktionsumfang sowie an die Oberfläche der WEL spezifiziert werden. Der explorative Teil seiner Aufgaben richtet sich eher an die Benutzer als an die Entwickler der WEL. Die eingesetzte Programmiersprache, sowie die Art der Verteilung und der Kommunikation der einzelnen Komponenten dient als Diskussionsgrundlage der am Projekt beteiligten Entwickler untereinander.

2.1 Die Architektur

Der Prototyp wurde als Client-/Server-System entwickelt. Er besteht aus einer Client- und einer Serverapplikation, bei der sowohl die Clientapplikation teilweise auch Dienste anbietet und damit im Rollenverhalten als Server arbeitet und umgekehrt.

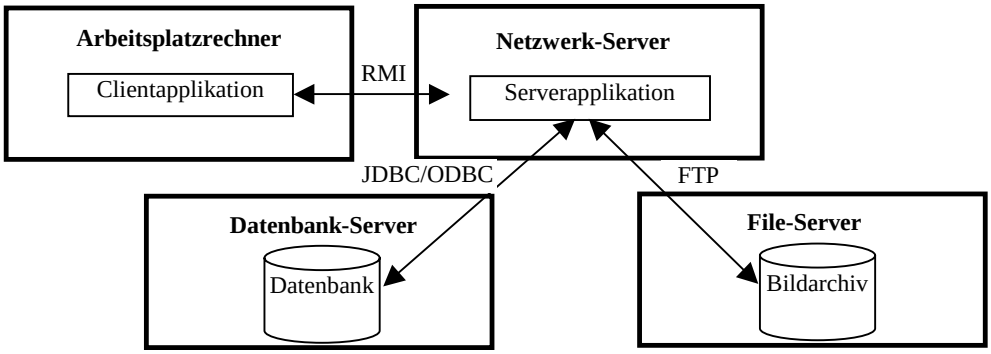


Abbildung VI-1 Komponenten und verwendete Kommunikationsprotokolle

Die Kommunikation zwischen den Komponenten findet durch den Austausch von Nachrichten über das RMI-Protokoll von Java statt. Die Serverapplikation kommuniziert mit dem Datenbankserver über JDBC/ODBC-Brigde und mit dem Fileserver über das FTP-Protokoll.

Die Architektur wurde so entworfen, daß die Protokolle zwischen den Komponenten, sowie auch die Komponenten selbst durch andere ohne großen zusätzlichen Aufwand ersetzt werden können. So kann das RMI-Protokoll durch das CORBA IIOP (siehe Kap:V1.6.3 CORBA) ersetzt werden und so als Serverapplikation eine native Anwendung eingesetzt werden.

2.1.1 Clientapplikation

Die Benutzer des Systems arbeiten an Arbeitsplatzrechnern mit der Clientapplikation. Sie ist in diesem Prototypen als Java-Applet realisiert, d.h. die Anwendung läuft in einem

Internet-Browser, der Applets ausführen kann. Die Clientapplikation besteht aus der Benutzerschnittstelle und den notwendigen Funktionen zur Kommunikation mit der Serverapplikation. Zwischen diesen Teilen der Anwendung fungiert der *Listener*[1], der auf Eingaben des Benutzers wartet und diese weiterleitet.

Die Clientapplikation läßt sich also in drei Aufgabenbereiche gliedern:

- *Klassen der Benutzerschnittstelle:* Diese Klassen enthalten alle Komponenten zur visuellen Darstellung der vom Benutzer angeforderten Informationen, sowie die interaktiven Elemente wie Schaltknöpfe, Listen, Menüs, etc., mit denen der Benutzer die zur Verfügung gestellte Funktionalität des Systems nutzen kann.
- *Klassen des Listeners:* Diese Klassen reagieren (*lauschen*) auf Benutzereingaben, sowie systeminterne Ereignisse und senden die entsprechenden Nachrichten an Objekte, die die *Anwendungslogik* realisieren.
- *Klassen der Anwendungslogik:* Diese Klassen enthalten die Methoden zur Kommunikation mit der Serverapplikation und zur Aufbereitung der Informationen für die Benutzerschnittstelle. In der Anwendungslogik ist die gesamte anwendungsspezifische Funktionalität der Clientappliktation implementiert.

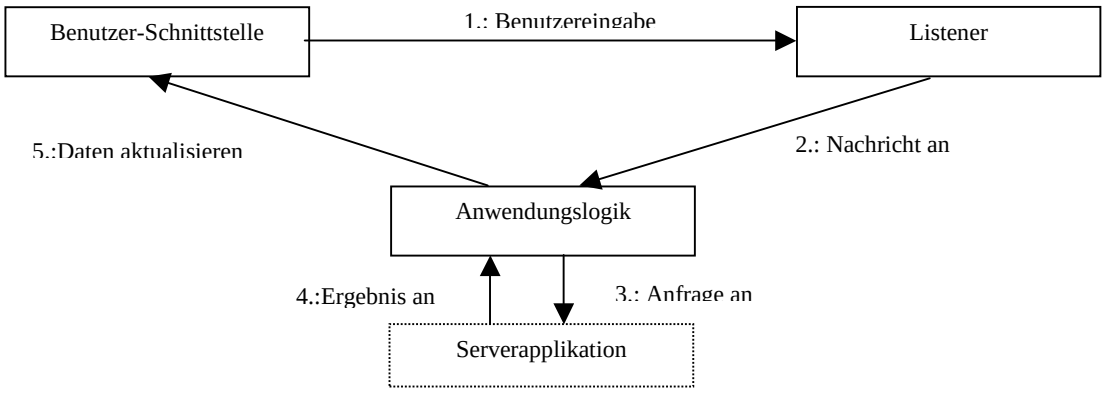


Abbildung VI-2: Interaktion der Komponenten der Clientapplikation

2.1.2 Serverapplikation

Die Serverapplikation ist das Bindeglied zwischen der Clientapplikation, den Datenbankservern und den Fileservern. Sie erhält von der Clientapplikation Befehlsobjekte (siehe Kap.: II2.3 Vorstellung eines verwendeten Entwurfsmusters) und führt die in den

[1] *Listener* heißt im Deutschen soviel wie Lauscher. Hier sind damit Klassen gemeint, die auf bestimmte Ereignisse, z.B. Benutzereingaben, warten und dann entsprechende Aktionen durchführen

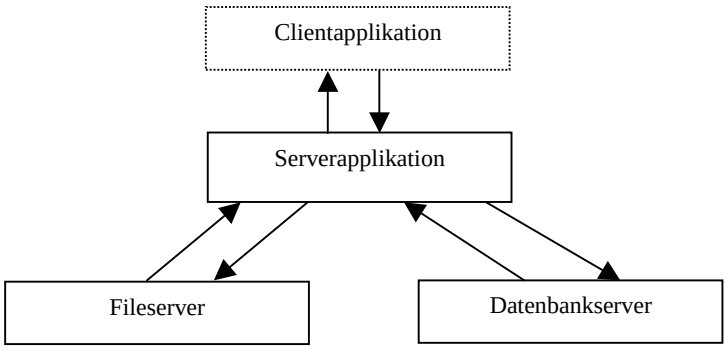


Abbildung VI-3: Komponenten der Serverapplikation

Objekten enthaltenen Befehle aus. Diese Befehle sind in der Regel Datenbankanfragen und werden an den Datenbankserver weitergeleitet, um dort ausgeführt zu werden.

Das folgende Beispiel soll die Verwendung der Befehlsobjekte illustrieren[1]:

Befehlsobjekt von der Clientapplikation:

Ein Objekt der Klasse *AuftragBefehl* wird als Parameter einer Nachricht an die Serverapplikation geschickt.

```
public class AuftragBefehl implements ServerBefehl{
	...
	}
	public ClientBefehl execute(InfoBaseServer ibServer){
		HashMap ergebnis = new HashMap();
		try {

			...
			// von der Serverapplikation auszuführende Befehle
			// z.B. Datenbankanfragen an einen Datenbankserver oder
			// Dateizugriffe auf einem Fileserver
			...
		}
		catch (SQLException ex) {
			// eventuelle Datenbankfehler abfangen
			System.out.println("Datenbankfehler:\n "+ ex.toString());
		}
		// Befehl für Clientapplikation erzeugen
		return new LieferungBefehl(ergebnis);
	}
}
```

Methode der Serverapplikation die das Befehlsobjekt annimmt:

```
public void execute(ServerBefehl befehl) throws RemoteException
{
	// In der Klammer wird die Methode aufgerufen, die die Befehle
	// für die Serverapplikation enthält. Als Ergebnis liefert die
	// Methode ein Objekt der Klasse ClientBefehl das wieder an die
	// Clientapplikation geschickt wird.
	client.execute( befehl.execute( this ) );
}
```

[1] Für die Erläuterung unwesentliche Abschnitte wurden aus den folgenden Programmfragmenten entfernt.

Der Vorteil dieser Architektur liegt in der Flexibilität, die durch die Verwendung von Befehlsobjekten erreicht wird. Änderungen und Erweiterungen in der Funktionalität oder der Struktur der Datenbasis wirken sich hauptsächlich auf die Klassen von Befehlsobjekten aus, d.h. nur diese Klassen müssen erweitert oder geändert werden.

2.2 Kommunikation zwischen Client- und Serverapplikation

Die Kommunikation zwischen den Applikationen basiert auf dem Entwurfsmuster *Befehl*. Die involvierten Schnittstellen und Klassen sowie der detaillierte Ablauf werden im Folgenden näher erläutert.

2.2.1 Die Klasse InfoBaseServer und die Schnittstelle InfoBaseRMI

Die Klasse *InfoBaseServer* übernimmt alle zentralen Aufgaben auf der Serverseite. Jeder Client erhält ein eigenes *InfoBaseServer*-Objekt mit dem er kommuniziert. Die Clients können an ihr Serverobjekt Nachrichten schicken, die die Schnittstelle *ServerBefehl* implementiert haben. Die Clients schicken diese Nachrichten an die Methode *execute* ihres Server-Objektes.

```
public interface InfoBaseRMI extends java.rmi.Remote {
   // Diese Methode wird vom Client aufgerufen
   // Hier wird dem Server der Client bekanntgemacht
   public abstract void register(InfoBaseClientRMI client)
   throws java.rmi.RemoteException;
   // Diese Methode wird vom Client aufgerufen
   public abstract void execute(ServerBefehl befehl)
   throws java.rmi.RemoteException;
}
public class InfoBaseServer implements InfoBaseRMI {
   private InfoBaseClientRMI client;
   private String rmiKennung;
   private static final int DEFAULT_PORT= 4242;
   public static void main(String argv[]) {
      try {
         java.net.ServerSocket serverSocket=
            new java.net.ServerSocket(DEFAULT_PORT);
         System.setSecurityManager( new java.rmi.RMISecurityManager());
         // Endlosschleife, die auf neue Clients wartet und
         // jedem Client einen neuen Server zuteilt
         for(;;) {
            // auf Socketverbindung warten
            java.net.Socket clientSocket = serverSocket.accept();
            // Neues Server-Objekt erzeugen und in der RMI-Registry eintragen
            java.rmi.server.ObjID ibServerKennung = new java.rmi.server.ObjID();
            InfoBaseServer ibServer =
               new InfoBaseServer(ibServerKennung.toString());
            java.rmi.server.UnicastRemoteObject.exportObject(ibServer);
            java.rmi.Naming.rebind(ibServerKennung.toString(),ibServer);
            // Dem Client den Namen seines Servers mitteilen
            java.io.PrintWriter out =    new java.io.PrintWriter(
               new java.io.BufferedOutputStream(clientSocket.getOutputStream()));
            out.println( ibServerKennung.toString() );
            out.flush();
         }
      }catch( java.lang.Exception e) {e.printStackTrace();}
   }
   public InfoBaseServer(String rmiKennung) throws java.rmi.RemoteException {
      super();
      this.rmiKennung = rmiKennung;
   }
   public void register(InfoBaseClientRMI client)
```

```
   throws java.rmi.RemoteException {
      this.client = client;
   }
   public void execute(ServerBefehl befehl) throws java.rmi.RemoteException {
      try {
         client.execute(befehl.execute());
      }  catch(Exception e) {e.printStackTrace();}
   }
}
```

2.2.2 Die Klasse InfoBaseLogik und die Schnittstelle InfoBaseClientRMI

Die Klasse *InfoBaseLogik* und die Schnittstelle *InfoBaseClientRMI* stellen die notwendige Funktionalität auf der Clientseite bereit. Die Funktionalität und der Ablauf sind analog zur Serverseite. Der Client agiert in diesem Fall als Server, der diese Dienste zur Verfügung stellt.

```
public interface InfoBaseClientRMI extends java.rmi.Remote {
   // Wird vom zugehörigen Server aufgerufen
   public abstract void execute(ClientBefehl befehl)
   throws java.rmi.RemoteException;
}

public class InfoBaseLogik implements InfoBaseClientRMI, java.io.Serializable {
...
   protected InfoBaseRMI server;
...
   public InfoBaseLogik(InfoBaseListener listener,
      java.applet.Applet applet,  String kennung) {
      // Den zugeordneten Server suchen und sich bei ihm bekanntmachen
      try {
         java.rmi.server.UnicastRemoteObject.exportObject(this);
         server = (InfoBaseRMI) java.rmi.Naming.lookup("rmi://"
            + applet.getCodeBase().getHost()+"/"+kennung);
         server.register(this);
      }  catch(Exception e) {e.printStackTrace();}
   }
   // Befehl an die Server-Methode execute schicken
   public void schickeBefehl(ServerBefehl befehl) {
      try {
         ClientBefehl clientBefehl;
         // nur an den Server schicken, wenn noch nie geschehen.
         // Pro: Reduzierung der Netzbelastung
         // Contra: Updateanomalie
         // => Nur geeignet bei Lesevorgängen
         if ((clientBefehl = befehl.executeLocal())== null)
            server.execute(befehl);
         else
            this.execute(clientBefehl);
      } catch(java.rmi.RemoteException e) {e.printStackTrace();}
   }
   public void execute(ClientBefehl befehl) throws java.rmi.RemoteException {
      befehl.execute(this);
   }
   ...
}
```

2.2.3 Die Klassen AuftragBefehl und LieferungBefehl, die Schnittstellen Befehl, ServerBefehl und ClientBefehl

Die Klassen *AuftragBefehl* und *LieferungBefehl* sind Containerklassen, die Objekte der Klasse *IndexObjekt* enthalten. Diese Objekte werden auf der Clientseite erzeugt. Auf der Serverseite werden die Attribute der Objekte aktualisiert und dann an den Client

zurückgegeben. Für den Transport werden diese Objekte in ein entsprechendes Objekt der Klasse *AuftragBefehl* (Objekt kommt vom Client), beziehungsweise *LieferungBefehl* (Objekt kommt vom Server) eingebettet.

```
public interface ServerBefehl extends java.io.Serializable {
   public ClientBefehl execute();
   public ClientBefehl executeLocal();
}

public interface ClientBefehl extends java.io.Serializable {
   public abstract void execute(InfoBaseLogik ibLogik);
}

public class AuftragBefehl implements ServerBefehl {
   private Object obj;
   public AuftragBefehl(Object obj)     {
      this.obj = obj;
   }
   public ClientBefehl execute(){
      IndexObjekt indObj = (IndexObjekt) this.obj;
      try {
         indObj.update();
      } catch( Exception e) {e.printStackTrace();}
      return new LieferungBefehl(indObj);
   }
   public ClientBefehl executeLocal(){
      IndexObjekt indObj = (IndexObjekt) this.obj;
      try {
         if (indObj.isComplete()) return new LieferungBefehl(indObj);
         else return null;
      } catch( Exception e) {e.printStackTrace();}
      return null;
   }
}

public class LieferungBefehl implements ClientBefehl{
   private IndexObjekt lieferung;
   public LieferungBefehl(IndexObjekt lieferung){
      this.lieferung = lieferung;
   }
   public void execute(InfoBaseLogik ibLogik) {
      lieferung.update(ibLogik);
   }
}
```

2.2.4 Ein exemplarischer Ablauf

Der Ablauf von der Erzeugung eines Auftrages (*AuftragBefehl*) bis zur Auslieferung des Ergebnisses (*LieferungBefehl*) soll hier detailliert dargestellt werden. Durch eine Benutzeraktion, zum Beispiel die Auswahl eines Schlagwortes, wird vom entsprechenden *Actionlistener* ein neues Objekt der Klasse *AuftragBefehl* mit dem ausgewählten Schlagwort als Parameter erzeugt und an die Methode *schickeBefehl* gesendet.

```
public class InfoBaseListener ... {
   ...
   protected void buttonAction(InfoBaseButton button){
      ...
      // neuen Auftrag erzeugen und an die Methode schickeBefehl
      // der Instanz logik der Klasse InfoBaseLogik senden
      if (command.equals("getWEL")) {
      ...
      }else if (command.equals("getSchlagwort")) {
         auftrag = new AuftragBefehl(logik.getSelectedSchlagwort());
         logik.schickeBefehl(auftrag);
      }else if (command.equals("getIndex")) {
      ...
```

```
      }
   }
   ...
}
```

Die Methode *schickeBefehl* der Klasse *InfoBaseLogik* übergibt diesen Auftrag an die Methode der entfernten Instanz *server* der Klasse *InfoBaseServer*.

```
public class InfoBaseLogik ... {
   ...
   public void schickeBefehl(ServerBefehl befehl) {
      try {
         ClientBefehl clientBefehl;
         if ((clientBefehl = befehl.executeLocal())== null)
            server.execute(befehl);
         else
            this.execute(clientBefehl);
      } catch(java.rmi.RemoteException e) {e.printStackTrace();}
   }
   ...
}
```

Auf der Serverseite wird die Methode *execute* des Auftrages aufgerufen.

```
public class InfoBaseServer ... {
   ...
   public void execute(ServerBefehl befehl) throws java.rmi.RemoteException {
      try {
         client.execute(befehl.execute());
      }  catch( Exception e) {e.printStackTrace();}
   }
   ...
}
```

Die Methode *execute* des Auftrages veranlaßt das Schlagwort, durch den Aufruf der Methode *update*, sich zu aktualisieren. Danach wird ein neues Objekt der Klasse *LieferungBefehl* erzeugt und an das aufrufende Objekt zurückgegeben.

```
public class AuftragBefehl ... {
   ...
   public ClientBefehl execute(){
      IndexObjekt indObj = (IndexObjekt) this.obj;
      try {
         indObj.update();
      } catch(Exception e) {e.printStackTrace();}
      return new LieferungBefehl(indObj);
   }
   ...
}
```

Das zurückgelieferte Objekt wird an die Methode *execute* des entfernten Objektes *client* der Klasse InfoBaseLogik übergeben.

```
public class InfoBaseServer ... {
   ...
   public void execute(ServerBefehl befehl) throws java.rmi.RemoteException {
      try {
         client.execute(befehl.execute());
      }  catch( Exception e) {e.printStackTrace();}
   }
   ...
}
```

Das Objekt *client* ruft die Methode *execute* des Objektes der Klasse *LieferungBefehl* auf.

```
public class InfoBaseLogik ... {
   ...
   public void execute(ClientBefehl befehl) throws java.rmi.RemoteException {
```

```
      befehl.execute(this);
   }
   ...
}
```

Diese Methode ruft dann die Methode *update* auf, welche für die Aktualisierung des anfangs ausgewählten Schlagwortes und dessen Visualisierung sorgt.

```
public class LieferungBefehl ... {
   ...
   public void execute(InfoBaseLogik ibLogik) {
      lieferung.update(ibLogik);
   }
   ...
}
```

Die Methoden *update()* und *update(InfoBaseLogik logik)* werden durch die Schnittstelle *IndexObjekt* deklariert und müssen von allen Klassen implementiert werden, welche die Schnittstelle verwenden.

```
public interface IndexObjekt extends java.io.Serializable {
   ...
   public void update();
   public void update(InfoBaseLogik logik);
   ...
}
```

```
public class Schlagwort implements IndexObjekt{
   ...
   // Wird auf der Serverseite aufgerufen
   public void update(){
      try {
         this.setKarten();
         this.setVerwandteSchlagworte();
         this.setUntergeordneteSchlagworte();
      } catch( Exception e) {e.printStackTrace();}
   }
   // Wird auf der Clientseite aufgerufen
   public void update(InfoBaseLogik logik) {
      logik.set(this);
   }
   ...
}
```

```
public class Schlagwort implements IndexObjekt{
   ...
   // Wird auf der Serverseite aufgerufen
   public void update(){
      try {
         this.setKarten();
         ...
      } catch(Exception e) {e.printStackTrace();}
   }
   // Wird auf der Clientseite aufgerufen
   public void update(InfoBaseLogik logik) {
      logik.set(this);
   }
   public void setKarten() {
      this.karten.clear();
      try {
         // Dem Schlagwort zugeordnete Karten abfragen
         String query = new String(
              "SELECT KarteID AS inID
            + "FROM SchlagwortZuKarte WHERE SchlagwortID="+ this.id );
         SQLRequest anfrage = new SQLRequest(query,this.databaseURL);
         java.sql.ResultSet rs = anfrage.execute();
         // Abfrageergebnis abarbeiten
         while (rs.next()) {
            long id;
```

```
            id = rs.getLong("inID");
            // Relevante Informationen zur aktuellen Karte abfragen
            query = new String("SELECT KarteID AS ID, Titel AS TIT, "
               + "Kommentar AS KOM, RegisterNummer AS REGNR "
               + "FROM Karte WHERE KarteID ="+ id );
            anfrage = new SQLRequest(query,this.databaseURL);
            java.sql.ResultSet rs1 = anfrage.execute();
            ...
            Karte karte = new Karte(rs1.getLong("ID"), this.databaseURL);
            karte.setTitel(rs1.getString("TIT"));
            karte.setKommentar(rs1.getString("KOM"));
            karte.setRegisterNummer(rs1.getString("REGNR"));
         ...
         rs1.close();
         }
         rs.close();
      } catch (java.sql.SQLException ex) {ex.printStackTrace();}
   }
   ...
}
```

VII Kommerzielle Relevanz

1 Nutzen des Internet für Institutionen und Firmen

Die Selbstdarstellung einer Firma oder die Repräsentation eines firmeneigenen Produktes ist für das Marketing von großer Bedeutung. Das Internet als Medium für die Verbreitung von Informationen bietet die Möglichkeit, dies effektiv und kostengünstig zu realisieren.

Die Veröffentlichung von Markt- und Geschäftsberichten gestaltet sich problemlos und kostenminimal. Multimediale Dokumente eignen sich für Werbung, Produktinformationen und Bestellkataloge. Sie sind, im Gegensatz zu gedruckten Dokumenten, sofort verfügbar und aktualisierbar. Die Möglichkeit, Informationen einem breiten Anwenderspektrum zur Verfügung zu stellen, verhilft zu mehr Kunden- und Öffentlichkeitsnähe. Universitäten, Forschungsinstitute und Firmen nutzen das Internet zur Präsentation ihrer Arbeiten und Projekte. Die globale Zusammenarbeit nahezu auf allen Sektoren und Fachrichtungen wird deutlich gefördert. Der kostengünstigere und schnelle Nachrichtenaustausch per elektronischer Post, im Vergleich zum Fax oder des herkömmlichen Schriftverkehrs, gestaltet den Informationsaustausch effizienter.

Für den Privatkundenbereich gelten diese Voraussetzungen zur Zeit nur eingeschränkt. Die privaten Internetzugänge sind relativ langsam und kostenintensiv. Diese Eigenschaften beschränken zur Zeit die technischen Möglichkeiten, zum Beispiel bei aufwendig gestalteten Produktinformationen, da hier auch immer auf die zu übertragenden Datenmengen geachtet werden muß.

Die Angebote an kostenpflichtigen Informationsdiensten im Internet stellen weitere Einnahmequellen dar. Jedes Unternehmen muß eine derartige Entscheidung für eine Kostenerhebung treffen. Je höher allerdings die Nutzungskosten sind, desto interessanter müssen die angebotenen Informationen für den Kunden sein.

2 Die WEL im Internet

Durch das Internet wird eine globale Informationsakquisition möglich. Ein potentieller Benutzer ist nicht mehr gezwungen, in seiner unmittelbaren Umgebung zu recherchieren, um seinen Informationsbedarf zu decken. Ein Kunsthistoriker aus Amerika, der z.B. die Einrichtungen des Warburg-Hauses und ebenso den Index für politische Ikonographie nicht kennt, würde bei seinen Arbeiten nicht auf die Idee kommen, daß eine derartige Quelle in Deutschland existent ist. Durch die WEL und ihre Präsenz im Internet kann das Warburg-Haus Wissen zur Verfügung stellen.

Eine Kostenerhebung für die Nutzung von Informationen aus der WEL ist grundsätzlich möglich. Die WEL ist ein universitäres Projekt und aus diesem Grund wird der kommerzielle Nutzen nicht weiter beleuchtet. Ziel der WEL ist es vielmehr, die vorhandenen Informationen einem größeren Publikum anzubieten, als dies bisher möglich ist.

3 Kostenbetrachtungen bei der Systemerstellung

Systementwicklungen unterliegen grundsätzlich den Budgetvorgaben der jeweiligen

Hersteller. Sowohl die Arbeiten der Experten aus der Software- und Datenverarbeitungsbranche als auch Fachleute der Anwendungsdomäne, die das System primär nutzen werden, stellen einen großen Kostenfaktor dar. Dabei nimmt die Planung einen nicht unerheblichen Teil der gesamten Projektzeit ein. Anforderungsdefinitionen müssen erstellt und analysiert werden, bis mit der Implementierung begonnen werden kann.

Im Rahmen von Kostenminimierungen ist es für jede Entwicklungsfirma sinnvoll, vorhandenes und wiederverwendbares Material zu nutzen, unabhängig davon, in welcher Entwicklungsphase dieses Material benötigt wird. Bei Entwürfen von Systemen, die einer äquivalenten Problemklasse zuzuordnen sind, ist eine hohe Wiederverwertbarkeit konzeptioneller Ausarbeitungen besonders in den Planungsphasen folgender Projekte von kostenminimierender Wirkung. Allgemeine Anforderungen, die ebenso durch allgemein gehaltene, strukturierte Konzepte Lösungsvorschläge präsentieren, vermitteln den Beteiligten der Entwicklung ein nutzbares Spektrum an vorhandenem Wissen und erweitern gegebenenfalls die Sichtweise einiger Problemstellungen. Ein erhebliches Maß an Analysezeit kann auf der Basis wiederverwendbaren Materials eingespart werden.

Bei der Systemerstellung ist auf verschiedene Aspekte einzugehen, wie zum Beispiel die Portabilität des erstellten Systems bezüglich seiner Migrierbarkeit auf andere Plattformen.

4 Ausblick

Das Angebot von kostengünstigen oder kostenlosen, Dienstleistungen im Internet wird expandieren. Der Grund hierfür liegt in der wachsenden Bedeutung des Internet als Werbeträger. Es findet eine Verlagerung der Finanzierung vom Endkunden zum Werbetreibenden statt. Es besteht dabei ein kausaler Zusammenhang zwischen Exklusivität und Qualität der Dienstleistung, Anzahl der Nutzer und den Werbeeinnahmen.

Ein besonders großes Wachstum wird dabei dem Vertrieb über das Medium Internet vorhergesagt. Dafür werden unter anderem Kataloge benötigt, welche die Produkte ausführlich und vorteilhaft beschreiben. Die dafür benötigten Technologien sind die gleichen, die auch bei der WEL zum Einsatz kommen. Der Kunde soll problemlos zu den gewünschten Produkten gelangen. Auf dem Weg zu dem gesuchten Produkt sollen ihm auch andere Artikel angeboten werden, die im Zusammenhang mit den gesuchten stehen.

Die Entwicklung der Bestellsysteme für das Internet steht noch am Anfang. Dies gilt für die Verbreitung genauso, wie für die Ausnutzung der technischen Möglichkeiten. Die WEL zeigt hier neue und bisher kaum genutzte Möglichkeiten auf.

Anhang A: Modell des Prototypen

Im Folgenden werden die Ergebnisse der Modellierungsschritte des Prototypen der Warburg Electronic Library detailliert dargestellt.

Anforderungsanalyse

Objektklassen

Objekte der Anwendung			Objekte des Systems
Archiv	Archivobjekt	Person	Anwendungsserver
Index	Bild	Ersteller	Datenbankserver
Persönlicher Index	Text	Benutzer	Fileserver
Gruppenindex	Audio	Benutzergruppe	Datenbank
Indexobjekt	Video		DBMS
Schlagwort			Anwendung
Karte			GUI

Glossar der aufgestellten Objektklassen

Objekte der Anwendung:

- *Bild*
  Das Bild ist eine zweidimensionale Abbildung, die in einem bestimmten Dateiformat auf einem Fileserver abgelegt ist. In Form von mehreren Dateien können verschiedene Ausprägungen der Abbildung vorliegen.
- *Text*
  Informationen in Textform, die im Archiv liegen.
- *Audio*
  Tonaufzeichnungen sind archivierte Informationen.
- *Video*
  Filmaufzeichnungen sind archivierte Informationen.
- *Archiv*
  Strukturierte Sammlung von Dokumenten (Darstellungen und Text).
- *Archivobjekt*

Dokumente des Archivs.

- *Index*
  Eine spezielle (hierarchische) Verknüpfung von Schlagworten nach subjektiven Ordnungskriterien bezogen auf die Dokumente im Archiv.
- *Persönlicher Index*
  Vom Benutzer erstellte Teilmenge eines Indexes.
- *Gruppenindex*
  Von Benutzergruppe erstellte Teilmenge eines Indexes; mindestens ein Benutzer hat das vollständige Verwaltungsrecht bezüglich dieses Gruppenindexes.
- *Person*
  Personen, die einen bestimmten Benutzungsbezug des Systems aufweisen.
- *Ersteller*
  Verfasser eines Textes oder Ersteller eines Kunstobjektes, das auf den Darstellungen abgebildet ist.
- *Benutzer*
  Benutzer des Systems.
- *Benutzergruppe*
  Eine Gruppe von Benutzern.
- *Schlagwort*
  Bestandteil des Index.
- *Karte*
  Zusammenfassung verschiedener Archivobjekte, die mindestens einem Schlagwort zugeordnet werden kann.

Objekte des Systems:

- *Anwendungsserver*
  Computer, auf dem die Anwendung abläuft, auf den ein Arbeitsplatzrechner zugreift. Der Anwendungsserver greift seinerseits über das Netzwerk auf einen Fileserver und einen Datenbankserver zu.
- *Fileserver*
  System, auf dem die Darstellungen in Dateien mit bestimmten Dateiformaten abgelegt sind.
- *Datenbankserver*
  Computer, auf dem die Datenbank installiert ist.
- *DBMS (Datenbankmanagementsystem)*
  Parallel zur Anwendung existierende Umgebung zur Verwaltung der Archivdaten.
- *Datenbank*
  Persistente Speicherung sämtlicher Informationen in Tabellen (mit Ausnahme von Darstellungen).
- *Anwendung*

Die Anwendung umfaßt das Anwendungsprogramm.

- *GUI*
  Interaktive graphische Benutzerschnittstelle.

Assoziationen zwischen Objektklassen

Anwendungsserver *hat* Anwendung(1)

Anwendung *läuft auf* Anwendungsserver(2)
Anwendung *benutzt* Archiv(1)
Anwendung *enthält* GUI(1)

Datenbank *speichert* Archiv (1)
Datenbank *gehört zu* DBMS(2)

Fileserver *speichert* Bilddaten(1)
Fileserver *speichert* Videodaten(1)
Fileserver *speichert* Audiodaten(1)
Fileserver *speichert* Textdaten(1)

Bilddaten *sind auf* Fileserver(2)
Textdaten *sind auf* Fileserver(2)
Audiodaten *sind auf* Fileserver(2)
Videodaten *sind auf* Fileserver(2)

Karte *hat* Text(1)
Karte *hat* Bild(1)
Karte *hat* Audio (1)
Karte *hat* Video(1)
Karte *gehört zu* Schlagwort(1)

Text *ist von* Ersteller(1)
Text *ist im* Archiv(1)
Text *gehört zu* Karte(1)
Text *hat* Textdaten

Bild *ist von* Ersteller(1)
Bild *ist im* Archiv(1)
Bild *gehört zu* Karte(1)
Bild *hat* Bilddaten

Audio *ist von* Ersteller(1)
Audio *ist im* Archiv(1)
Audio *gehört zu* Karte(1)
Audio *hat* Textdaten

Video *ist von* Ersteller(1)
Video *ist im* Archiv(1)
Video *gehört zu* Karte(1)
Video *hat* Videodaten

Archiv *hat* Text(1)
Archiv *hat* Bild(1)
Archiv *hat* Audio (1)
Archiv *hat* Video(1)
Archiv *hat* Schlagwort(1)
Archiv *hat* Index(1)
Archiv *ist in* Datenbank

Karte *hat* Bild (2)
Karte *hat* Text(2)
Karte *hat* Audio (2)
Karte *hat* Video(2)

Schlagwort *referenziert* Karte(2)
Schlagwort *gehört zu* Gruppenindex (2)
Schlagwort *gehört zu* persönlichen Index(2)
Schlagwort *gehört zu* Archiv(2)
Schlagwort *ist parent von* Schlagwort
Schlagwort *ist verwandt mit* Schlagwort

persönlicher Index *ist Teil von* Index(1)
persönlicher Index *gehört* Benutzer(1)
persönlicher Index *gehört* zu Archiv(1)
persönlicher Index *hat* Schlagwort(1)

Gruppenindex *ist Teil von* Index(1)
Gruppenindex *gehört* Benutzergruppe(1)
Gruppenindex *gehört* zu Archiv(1)
Gruppenindex *hat* Schlagwort(1)

Ersteller *erstellte* Bild(1)
Ersteller *erstellte* Text(1)
Ersteller *erstellte* Audio (1)
Ersteller *erstellte* Video(1)

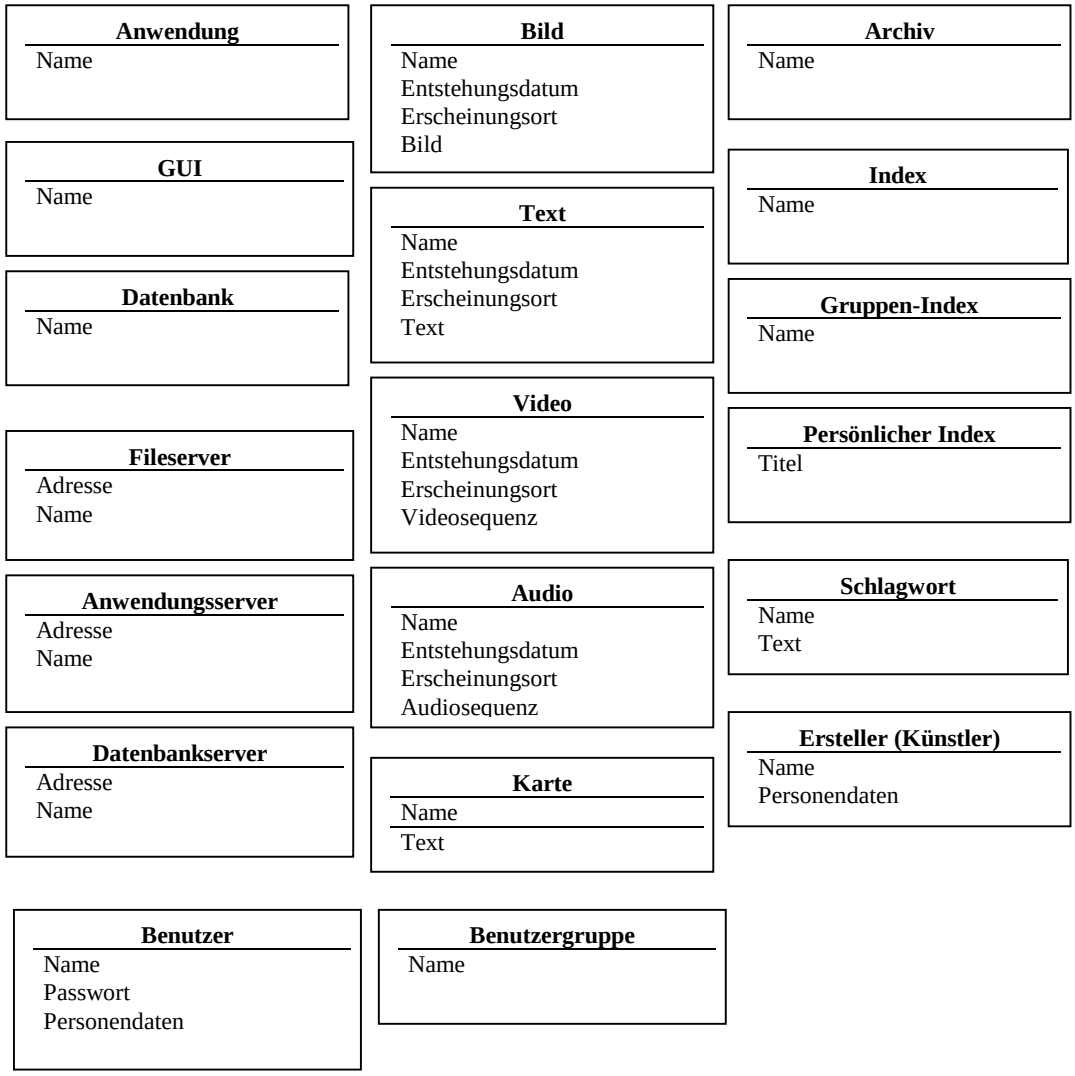
GUI *ist Teil von* Anwendung(2)

Benutzer *interagiert mit* GUI(1)
Benutzer *gehört zu* Benutzergruppe(1)

Benutzergruppe *hat* Benutzer(1)

DBMS *hat* Datenbank(1)

Objektklassen mit Attributen



Szenarien typischer Interaktionssequenzen

In den beschriebenen Szenarien werden zwei Kategorien von Benutzern unterschieden. „B.“ ist ein anwendungsbezogener Benutzer. „A.“ dagegen stellt einen Benutzer dar, der Möglichkeiten bei der Nutzung und Verwaltung der Anwendung hat.

Szenario 1

B. möchte direkt einen der digitalisierten Bildindexe nutzen. Ein Bildindex ist in seiner Struktur unter Beachtung eines beabsichtigten semantischen Kontextes bestehender Informationen in das Datenmodell eingebettet. B. wird aufgefordert, den Benutzernamen und ein Paßwort einzugeben. B. wählt den Index aus, mit dem er arbeiten möchte. B. bekommt auf dem Bildschirm diesen Index hierarchisch dargestellt. Weiterhin erhält B. eine alphabetische Auswahl aller enthaltenen Schlagworte dieses Indexes, unabhängig von der hierarchischen Einbettung. B. kann aus beiden Listen wählen. Für den selektierten

Begriff kann B. verschiedene Operationen ausführen:

Szenario 1.1 : B. kann sich eine Auswahl der verwandten Schlagworte dieses Begriffes anzeigen lassen. („Kaiser“, „König“, „Herrscher“, etc.). Wählt B. einen Begriff aus und übernimmt diesen, wird dieser in der alphabetischen Ausgangsauswahl markiert und steht für andere Operationen zur Verfügung.

Szenario 1.2 : B. kann sich den hierarchischen Kontext ausgeben lassen, in dem der Begriff steht. (Bsp.: Ist beim politischen Index der Begriff „Demokratie“ markiert und wird die Funktion für die hierarchische Darstellung angestoßen, werden sowohl die Oberbegriffe als auch die enthaltenen Stichwörter angezeigt). Ein beliebiges Schlagwort, unabhängig seiner hierarchischen Stellung, kann übernommen werden und wird in der alphabetischen Ausgangsliste markiert und steht für andere Operationen zur Verfügung.

Szenario 1.3: Zu jedem Schlagwort - unabhängig seiner hierarchischen Position - können Darstellungen, Texte und Informationen anderer Art abgelegt sein. Wählt B. einen Begriff aus und aktiviert die Suchfunktion, wird festgestellt, welche Darstellungen und welche Texte für diesen Begriff referenziert sind. Es erfolgt eine Ausgabe der entsprechenden Informationen. B. hat nun verschiedene Möglichkeiten:

> *Szenario 1.3.1*: Durch Auswahl einer konkreten bildlichen Darstellung wird diese dargestellt. Alle direkt mit diesem Werk verbundenen Informationen werden angezeigt. B. kann sich alle Schlagworte anzeigen lassen, die mit dem selektierte Objekt verknüpft sind.
>
> *Szenario 1.3.2*: Durch Auswahl einer konkreten Textdarstellung werden alle verknüpften Darstellungen und Verfasserinformationen angezeigt. Der Verlauf ist analog zu Szenario 1.3.1..
>
> *Szenario 1.3.3*: Durch Auswahl eines konkreten Erstellers oder Verfassers werden alle verknüpften Darstellungen und Texte aufgelistet. Der weitere Verlauf ist analog zu Szenario 1.3.1 bzw. 1.3.2.
>
> *Szenario 1.3.4*: Beliebige Darstellungen, Texte und andere Informationen können in ein neues oder bestehendes persönliches Archiv oder Gruppenarchiv, das benutzerspezifisch angelegt wird, eingetragen werden. Für sein eigenes Archiv hat B. die vollen Rechte und kann sich Daten, für deren Zugriff B. ebenfalls die Rechte haben muß, in seinem Archiv referenziell zusammenstellen.
>
> *Szenario 1.3.5*: Mit den durch die Auswahlmöglichkeiten der Szenarien 1.3.1. bis 1.3.4. erhaltenen Objekte lassen sich bei erneuter Auswahl wiederum neue Informationsmengen bezüglich jedes selektierten Objektes zusammenstellen. Die Szenarien 1.3.1 bis 1.3.4 können iteriert werden.

Szenario 1.4: B. möchte ein bestehendes persönliches Archiv bearbeiten oder löschen. Per Auswahl erscheint eine Auswahl seiner Archive, aus der er eines selektiert. B. kann verschiedenen eingetragenen Benutzern des Systems die Einsicht oder gar die Mitbearbeitung gewähren. Sind einem persönlichen Archiv mehrere Benutzer zugeordnet, wird das Archiv implizit zum Gruppenarchiv.

Szenario 1.5: B. möchte sein Paßwort ändern. Er wird aufgefordert, das alte Paßwort anzugeben.

Szenario 1.5.1: B. hat sein altes Paßwort korrekt eingegeben und wird aufgefordert, das neue Paßwort einzugeben.

Szenario 1.5.2: B. hat sein altes Paßwort nicht korrekt eingegeben und wird wiederholt aufgefordert, das alte Paßwort einzugeben.

Szenario 2

Szenario 2.1: Zur Verwaltung von Benutzern stehen A. verschiedene Funktionen zur Verfügung wie „Erstellen“, „Löschen“ bzw. „Bearbeiten“. A. kann einem Benutzer eine Benutzerpriorität zuordnen und ihm ein Paßwort geben. Nur hier angelegte Benutzer haben Zugang zum System.

Szenario 2.2: A. möchte zu bestehenden Klassen (Bsp.: „180 Frieden“, „200 Gesten“, etc.) weitere Klassen anlegen[1]. Eine Klasse stellt eine Menge von Schlagworten innerhalb eines Indexes dar, die unter einem Oberbegriff zusammengefaßt sind. Eine weitere Klasse würde durch A. eingegeben werden, indem der Begriff und optional eine Beschreibung (z.B. die Nummer des Klassenbeschreibung des Indexes zur politischen Ikonographie) eingegeben wird. Anschließend muß die Einbettung in die bestehende Hierarchie vorgenommen werden. Danach können Schlagworte für eine Verwandtschaftsrelation ausgewählt werden. Die neue Klasse erscheint in Zukunft in der alphabetischen Auswahl und steht für Benutzerfunktionen zur Verfügung.

Szenario 2.3: Zu einem bestehenden Schlagwort können (siehe Szenario 1) verwandte Schlagworte und der hierarchische Kontext ausgegeben werden. Als Administrator hat A. die Möglichkeit, bestehende Referenzen zu löschen oder neue hinzuzufügen. Soll ein Schlagwort selbst aus der (alphabetischen Haupt-) Auswahl gelöscht werden, ist dies nur möglich, wenn „unterhalb“ in der Hierarchie keine Einträge verzeichnet sind, d.h. der zu löschende Eintrag wird nicht durch ein anderes Schlagwort referenziert. Jedes Schlagwort enthält eine eindeutige Nummer, die vom System automatisch verwaltet wird und in anderen Bereichen Einsatz findet.

Szenario 2.4: A. kann jedem Schlagwort, unabhängig von dem hierarchischen Kontext, beliebige Archivobjekte zuordnen. Die zuzuordnenden Objekte werden markiert und das gewünschte Schlagwort selektiert.

---

[1] Da die Numerierung des *Indexes zur politischen Ikonographie* kein gutes Kriterium hinsichtlich flexibler Erweiterungsmöglichkeiten darstellt, wird diese Nummer eine Zusatzinformation zu den Schlagworten darstellen. Dies ist keine Restriktion für die WEL, da auch nach diesen Nummern gesucht werden kann. Zu jedem Schlagworteintrag besteht grundsätzlich die Möglichkeit, zusätzlich Informationen zu erfassen. Der Vorteil ist zum einen, daß diese Nummern durch A. im laufenden Betrieb geändert werden können, falls der Bedarf besteht (dient nicht als Schlüssel) oder zum anderen diese Numerierung als zukünftiges Relikt rationalisiert werden kann.

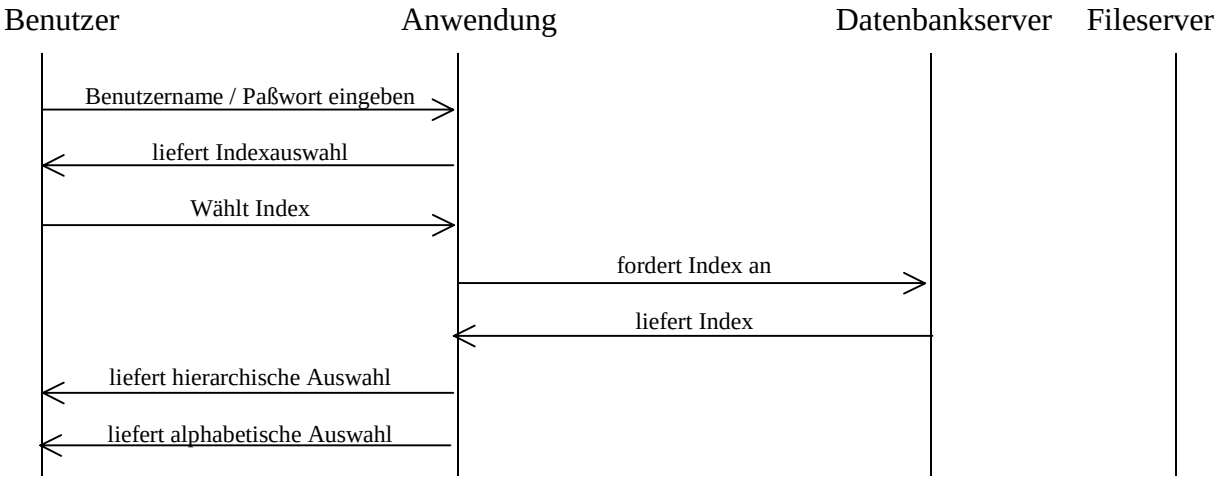
Identifikation der Ereignisse

- Benutzer gibt den Benutzernamen und das Paßwort ein
- Anwendung evaluiert die Benutzerdaten
- Anwendung liefert eine Indexauswahl
- Benutzer wählt einen Index aus
- Anwendung liefert hierarchische und alphabetische Auswahl der Schlagworte
- Benutzer wählt ein Schlagwort aus
- Anwendung fordert Daten vom Datenbankserver an
- Anwendung fordert Daten vom Fileserver an
- Datenbankserver liefert Daten an die Anwendung
- Fileserver liefert Daten an die Anwendung
- Anwendung zeigt die zugehörigen Informationen (Darstellungen, Texte, etc.) zu dem Schlagwort
- Benutzer fordert verwandte Schlagworte zu einem selektierten Schlagwort an
- Anwendung zeigt verwandte Schlagworte
- Benutzer gibt Auftrag zur Erstellung eines Archivs
- Anwendung erstellt Archiv
- Benutzer fordert Darstellung aller Informationen zu einem selektierten Schlagwort
- Anwendung zeigt Informationen
- Benutzer stellt Anforderung, selektierte Objekte einem Archiv zuzuweisen
- Anwendung weist selektierte Informationen dem Archiv zu
- Benutzer wählt konkretes Objekt aus und fordert verknüpfte Informationen an
- Benutzer fordert Paßwortänderung an
- Anwendung fordert zur Paßworteingabe auf
- Benutzer selektiert Schlagwort und will dieses mit vorhandenen selektierten Schlagworten des Indexes verknüpfen
- Anwendung trägt Verknüpfungen in die Datenbank ein
- Benutzer selektiert Schlagwort und will dieses einem anderen Oberbegriff zuweisen
- Anwendung trägt Hierarchie in die Datenbank ein

Im Folgenden werden vier Ereignispfade exemplarisch herausgegriffen, um einzelne Abläufe zu spezifizieren. Jeder Ereignispfad stellt hier eine in sich abgeschlossene Aktion dar, ähnlich einer Transaktion. Ereignispfade lassen sich zusammensetzen, wenn jeweils die Nach- und Vorbedingungen übereinstimmen. Es kann sich eine baumartige Struktur ergeben.

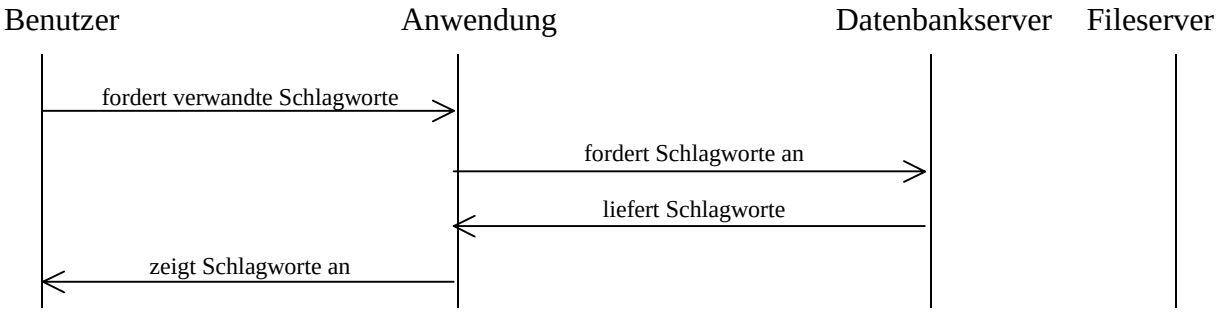
Anmeldung:

Nach dem Starten der Anwendung wird jeder Benutzer aufgefordert, sich am System anzumelden. Nach erfolgreicher Anmeldung liefert das System die Indexliste, aus der ein konkreter Index vom Benutzer selektiert werden muß. In Abhängigkeit der Indexwahl erhält der Benutzer die jeweiligen Schlagwortlisten.



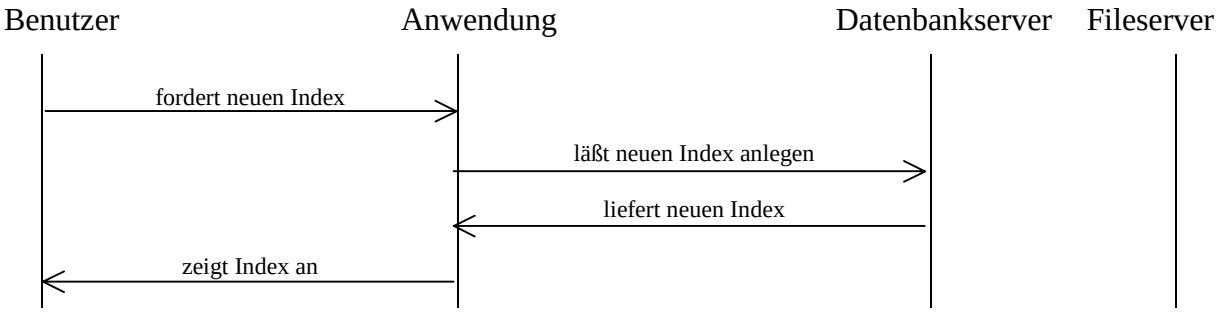
Verwandte Schlagworte:

Für ein selektiertes Schlagwort fordert der Benutzer alle Schlagworte an, zu denen eine inhaltliche Verwandtschaft besteht. Die Anwendung holt die erforderlichen Schlagworte aus der Datenbank und zeigt sie an.



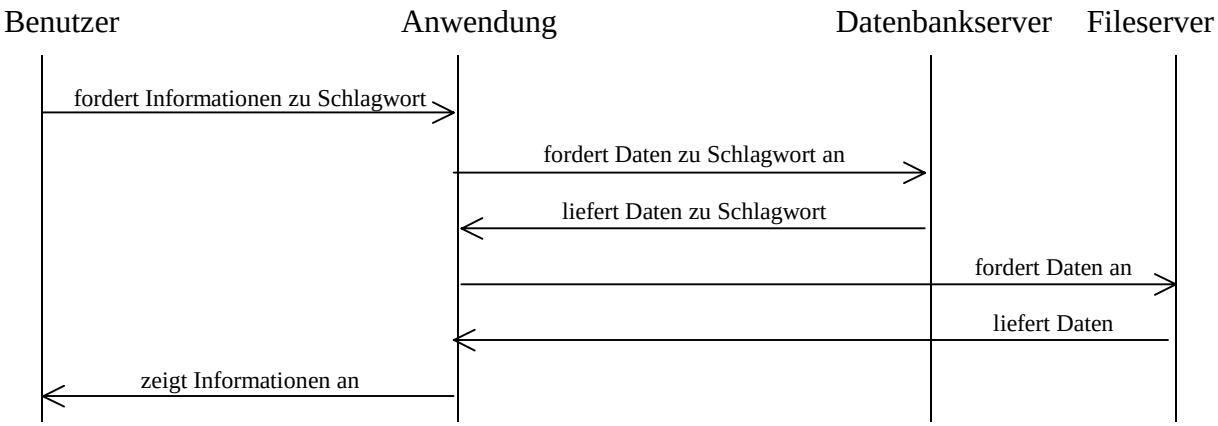
Neuerstellung eines Indexes:

Es wird ein neuer Index erstellt.

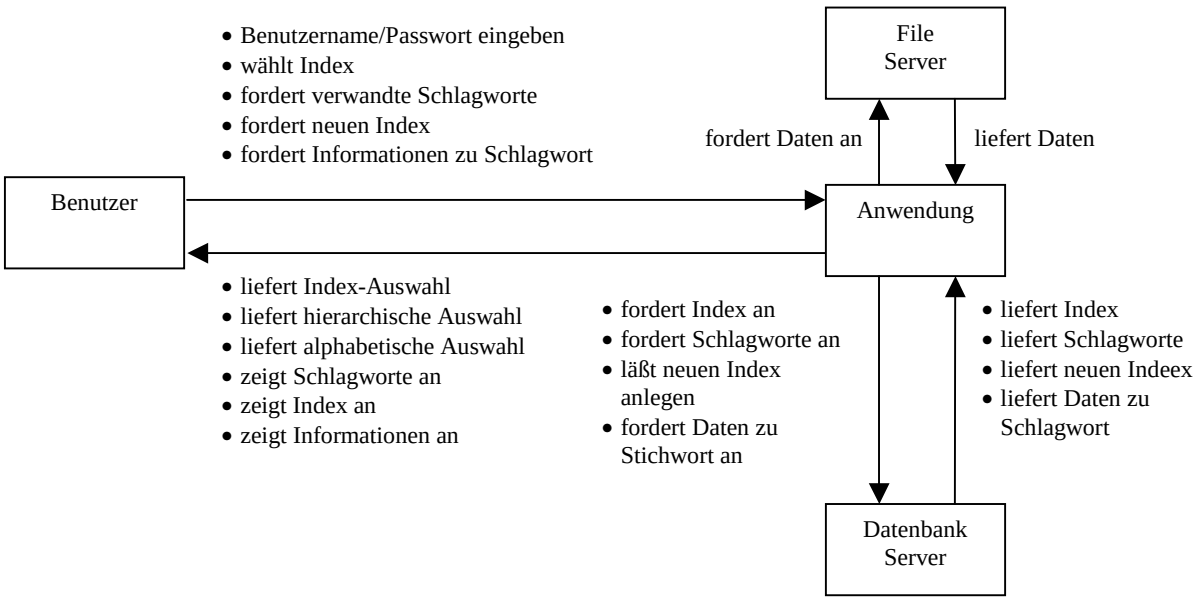


Informationsanfrage zu Schlagwort:

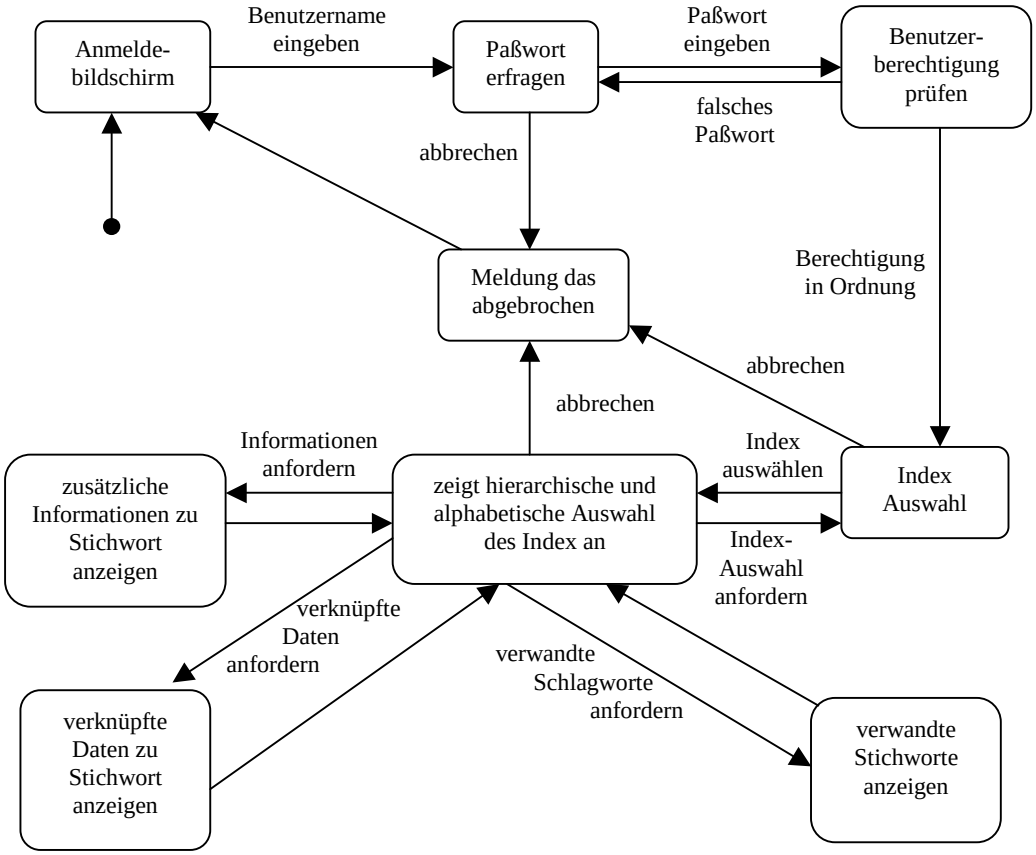
Zu einem ausgewählten Schlagwort werden die zugehörigen Informationen zu diesem Schlagwort angefordert.



Ereignisflußdiagramm



Zustandsdiagramm für die Klasse Anwendung



Anhang B:
Java Remote Method Invocation

Verteilte Systeme im objektorientierten Umfeld erfordern ein verteiltes Objektmodell und einen verteilten Nachrichtenfluß, welcher einem entfernten Methodenaufruf entspricht. Da der Remote Procedure Call (RPC) in objektorientierten Sprachen nicht, oder nur eingeschränkt verwendbar ist, steht mit dem RMI ein Transfer der RPC-Konzepte zur Verfügung.

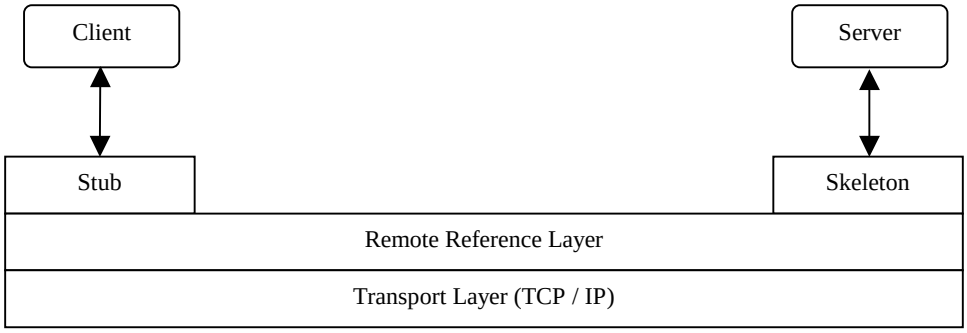
Die neben dem RMI derzeit in Java existierenden Möglichkeiten einer Kommunikation zwischen Clients und Servern sind die Basiskommunikation über Sockets und die Datenbankschnittstelle JDBC. Die Kommunikation von verteilten Objekten wird vom Java RMI weitgehend automatisiert. Damit ist sie, neben dem CORBA-Ansatz (Java IDL / IIOP vgl. Kap. V1.6.3: CORBA) für sprachübergreifende Kommunikation heterogener Systeme, ein vernünftiger Ansatz für ein objektorientiertes Client/Server Modell in der Java-Umgebung.

Die beiden wichtigsten Punkte sind die Möglichkeit, komplexe Objekte zwischen Client und Servern auszutauschen und transparent Nachrichten an entfernte Objekte zu senden, d.h. Methoden entfernter Objekte aufzurufen.

Von Vorteil für das RMI ist, daß die Behandlung der entfernten Objekte völlig analog zu den lokalen ist. Damit ist für die Entwicklung eine einheitliche SW-Plattform gegeben. Zudem kann durch den Verzicht auf die Kommunikation mit Objektinstanzen anderer Entwicklungsumgebungen, wie z.B. C++ Objekten, auf die Implementierung gemeinsamer Klassen in unterschiedlichen Sprachen verzichtet werden. Das heißt, auch Klassen, die vom Client und vom Server benutzt werden, müssen nur in Java implementiert werden und nicht zusätzlich in z.B. C++. Nachteil der Verwendung von RMI, im Gegensatz zu CORBA, ist die Festlegung auf eine Sprache. Eine Ankopplung von anderen Clients, die z.B. in C++ entwickelt wurden, ist mit RMI nicht möglich.

RMI Systemarchitektur

Die Systemarchitektur des RMI ist in drei voneinander vollständig unabhängige, eigenständige Ebenen aufgeteilt. Die *Stub/Skeleton* Schicht, *Remote Reference* Schicht und die *Transport* Schicht.



Schichten des Java RMI-Protokolls

Die Zuordnung von Client- und Serverfunktion gilt stets nur für eine konkrete Kommunikation zweier Objekte. Somit kann ein Objekt Client bzgl. eines Zugriffs auf ein entferntes Serverobjekt sein und kann wiederum einen Serverdienst für ein anderes Objekt erbringen, welches evtl. auf einem anderen entfernten Rechner existiert.

Die Stub/Skeleton Schicht

Wenn von einem Objekt im Client eine Methode eines Serverobjekts aufgerufen werden soll, d.h. dem Serverobjekt eine Nachricht gesendet werden soll, wird ein lokales Stellvertreterobjekt verwendet, welches als *Stub* bezeichnet wird. Dieses existiert im Adressraum der JVM, die das Clientobjekt ausführt und stellt somit für die restliche Applikation ein reguläres Java-Objekt dar. Die *Stub*-Methode faßt die Parameter des Methodenaufrufs zusammen (*parameter marshalling*) und gemeinsam mit der eindeutigen Referenz des Remote-Objekts und der Nummer der aufgerufenen Methode wird ein Datenblock gebildet, der über die *Remote Reference* Schicht zum Server gesendet wird.

Auf der Serverseite empfängt das entfernte Stellvertreterobjekt des eigentlichen Remote-Objekts, bezeichnet als *Skeleton*, über die dortige *Remote Reference* Schicht den Datenblock und ruft die gewünschte Methode des Serverobjekts auf. Rückgabewerte, wozu auch ausgelöste Ausnahme- bzw. Fehlerobjekte (*Exceptions*) gehören, werden auf dem analogen Weg zum Clientobjekt zurückgesendet.

Da ein generelles Designziel im RMI eine nahtlose Eingliederung des APIs[1] in die reguläre Sprache ist, kann der Entwickler die Remote-Objekte nahezu genauso verwenden wie originäre Java-Objekte. Damit ist auch klar, daß der oben beschriebene komplexe Kommunikationsweg vollständig transparent und vom RMI automatisiert ist.

[1] API: application programming interface.

Die Remote Reference Schicht

In dieser Schicht wird die Kommunikation mit der Transportschicht geregelt und im wesentlichen das Protokoll bestimmt, nach dem die entfernten Referenzen zugeordnet werden. Dies ist unabhängig von den *Stub*- und *Skeleton*-Objekten.

Jede Implementierung eines entfernten Objekts bestimmt die Art der Referenzierung. Es sind verschiedene Protokolle in dieser Schicht zu realisieren:

- Einfacher Methodenaufruf bei einer Punkt-zu-Punkt Zuordnung
- Methodenaufruf von replizierten Objekten
- Unterstützung für spezielle Replikationsstrategien
- Strategien zur Wiederherstellung bei einer unterbrochenen Verbindung zum Objekt

Zwei Komponenten bzgl. der Client- und Serverseite kooperieren bei der Methodenausführung. Die Client-Komponente enthält die relevanten Informationen über die Serverimplementierung des Objekts und kann z.B. bei einem replizierten Serverobjekt den Methodenaufruf an alle Replikate weiterleiten. Die Komponente der *Remote Reference* Schicht auf der Serverseite dient der Weiterleitung des Aufrufs an das *Skeleton*-Objekt.

Die *Remote Reference* Schicht überträgt die Daten über abstrakte Verbindungskanäle, die sich am Datenstrom orientieren. Die unterhalb liegende Transportschicht implementiert diese transparent.

Die Transportschicht

In der Transportschicht werden die folgenden Aufgaben wahrgenommen:

- Aufbau, Verwaltung und Überwachung von Verbindungskanälen zu entfernten Adressräumen
- Verwaltung einer Referenztabelle von aktuellen entfernten Objekten, die im lokalen Adressraum verwendet werden
- Annahme von Verbindungswünschen anderer und Aufbau der
- Verbindungskanäle

Zur Verwaltung von entfernten Objekten werden vier begriffliche Verallgemeinerungen (Abstraktionen) definiert:

- Ein Endpunkt (*endpoint)* ist die Abstraktion, um eine JVM bzw. einen Adressraum zuzuordnen.
- Ein Kanal (*channel)* ist die begriffliche Verallgemeinerung für eine virtuelle, leitungsgebundene Verbindung zweier Adressräume. Der Kanal ist für die Verbindung des lokalen und entfernten Adressraumes verantwortlich.
- Eine Verbindung (*connection)* ist die Abstraktion für eine bidirektionale Datenübertragung.
- Ein Transport (*transport)* ist die Abstraktion für die Verwaltungsinstanz der Kanäle. Bei jedem Transport kann nur ein Kanal für zwei Adressräume real aufgebaut werden.

Eine Zuordnung eines Endpunktes zu einem entfernten Adressraum bewirkt einen Verbindungsaufbau (Kanal). Weitere Aufgaben sind die Annahme von eingehenden Aufrufen, der Einrichtung einer Verbindung und der Weiterleitung zu der höheren Systemebene.

Die konkrete Repräsentation eines entfernten Objektes, bezeichnet als *live reference*, besteht aus einem Endpunkt und einer eindeutigen Identifikation (Schlüssel) des Objektes. Mit einer solchen *live reference* kann in der Transportschicht im Client per Endpunkt die Verbindung zum entfernten Adressraum aufgebaut, sowie auf der Serverseite das konkrete Objekt per Schlüssel bestimmt werden.

Dynamisches Laden von Klassen

Ein Client im RPC-Umfeld benötigt kompilierten Code für *Stubs*, der statisch oder dynamisch (über ein Netzwerk-Filesystem) an das aktuelle Programm gebunden wird, bevor die Routine ausgeführt werden kann.

RMI generalisiert diese Technik dahingehend, daß

- neutraler Bytecode verwendet wird, da dies ein wesentliches Merkmal der Sprache ist. Dies ermöglicht bzw. vereinfacht den heterogenen Mix von Client- und Serverplattformen.
- wahlfreies Laden von Klassen von lokalen oder entfernten Quellen möglich ist, indem ein spezieller *RMIClassLoader* zur Verfügung gestellt wird. Neben den anderen Möglichkeiten in Java (*AppletLoader, Classpath*), ist damit eine Kontrolle und genaue Steuerung der Klassenherkunft möglich. Er unterstützt das automatische Laden von:
  - fehlenden *Class-Files* zur Laufzeit,
  - entfernten Objektklassen und Interfaces,
  - *Stub*- und *Skeleton*-Klassen,
  - anderen Klassen, die in der RMI-Anwendung benötigt werden, wie z.B. für Parameter von Methoden oder Rückgabeobjekten.

Sicherheitsaspekte

Im Bereich der Netzanwendungen ist stets die Möglichkeit gegeben, daß Mißbrauch mit den Zugriffsmöglichkeiten betrieben wird. In dem konkreten Fall der verteilten Anwendungen ist eine Steuerung und Kontrolle der entfernten Methodenaufrufe notwendig. Das RMI-System bietet ein Standardsicherheitssystem an, den *RMISecurityManager*. Dieser muß in Applikationen als erstes gestartet werden, um die Kontrolle über die über Netz gehenden Methodenaufrufe zu erhalten und entscheiden zu können, welche Stub-Klassen von welchen Servern geladen werden dürfen. Auch kann dadurch kontrolliert werden, was *Stub*-Objekte, die im Adressraum des Clients ausgeführt werden, dürfen, z.B. Zugriff auf lokale Ressourcen (Festplatte, etc.) und Zugriff auf andere Server.

Werden *RemoteObjects* über Applets geladen, so übernimmt standardmäßig der

AppletSecurityManager die Kontrolle.

Es ist weiterhin möglich, eigene *Security-Manager* zu implementieren, die spezielle applikationsspezifische Sicherheitsbelange unterstützen.

Zudem ist seitens des Designs und der Entwicklung steuerbar, welche (entfernten) Objekte über das Netz transportiert werden können. Dies kann nur mit Objekten geschehen, die *serialisierbar* sind, also das Interface *Serializable* implementieren.

Verwaltung von entfernten Referenzen

Die Verwaltung von nicht mehr benötigten Objekten ist in Java automatisiert (Garbage Collection). Dies gilt auch für *entfernte* Objekte. Damit ist auch im verteilten Fall der Entwickler nicht mit dieser Verwaltung belastet. Natürlich ist es im Netzzugriff immer möglich, daß eine Referenz auf ein entferntes Objekt nicht aufgelöst werden kann (z.B. Rechnerausfall). In diesem Fall wird eine *RemoteException* ausgelöst.

Schnittstellen und Klassen des RMI-Package

Die Schnittstelle Remote

Die Basis der Applikationstypen bzw. -klassen ist die Schnittstelle *Remote*. Ein Client benötigt zur Benutzung von Objekten, die auf dem Server existieren, eine Beschreibung über die Schnittstelle eines entfernten Objektes. Diese Schnittstelle definiert die Methoden, die eine konkrete Implementierung und davon abgeleitete Instanzen entfernten Objekten zur Verfügung stellen. Somit wird die Logik der Applikation über Typen (analog zu abstrakten Datentypen) definiert. Alle Schnittstellen für entfernte Objekte müssen die Schnittstelle *Remote* direkt oder indirekt erweitern, d.h. von diesem abgeleitet sein.

Die Klasse RemoteObject und Subklassen

Die Implementierung der Applikationstypen erfolgt auf Basis der Klasse *RemoteObject*, von der die Instanzen die allgemeine Netzfähigkeit für Objekte erben, und die das Pendant für entfernte Objekte für die allgemeine Oberklasse *Object* in Java ist. Die Klassen der Applikation implementieren die Schnittstelle *Remote* mit Methoden und Datenfeldern, d.h. sie realisieren die konkreten Typen.

Die Klasse *RemoteServer* dient als Superklasse für die einzelnen Referenzierungsklassen wie *UnicastRemoteObject* und *MulticastRemoteObject*. Eine Zuordnung eines Stub-Objekts zu einer einzelnen Server-Implementierung oder zu Replikaten ist vom Design des RMI her möglich. Dies wird gesteuert über die Ableitung der Serverobjektimplementierung von der entsprechenden Subklasse von *RemoteServer*. Die Klasse *RemoteServer* vererbt die Semantik der entfernten Referenzierung und des Managements von Objekten an die abgeleiteten Objektklassen.

Damit wird also eine Applikationsklasse *X* im allgemeinen abgeleitet von *UnicastRemoteObject* und somit indirekt von *RemoteServer* und *RemoteObject*. Auch ist eine Ableitung von einer anderen Implementierung einer *Remote*-Klasse möglich. Die Klasse *X* kann eine oder mehrere *Remote*-Schnittstellen implementieren. Zudem können in *X* auch Methoden definiert werden, die in keiner *Remote*-Schnittstelle deklariert werden.

Dies sind Methoden die dann nicht entfernt verwendet werden können, aber der lokalen Serverinstanz zur Verfügung stehen.

Parameterübergabe

Zur Übertragung von Parametern, die einfachen Datentyps oder Objektinstanzen sein können, ist eine Serialisierung notwendig. Hierzu ist ein spezieller Mechanismus in Java vorhanden, genannt *Java Object Serialization*. Dieser wird durch die Schnittstelle *Serializable* angeboten, die von den Parameterklassen implementiert werden muß. Dies ist bei den Standardklassen des JDK fast stets gegeben, insbesondere bei den *Remote*-Klassen.

Prinzipiell kann jede Art von Objekt übertragen werden. Die Übergabe eines Objekts kann in Form eines Parameters beim Aufruf einer Methode oder als Rückgabewert einer entfernten Methode geschehen.

Zu unterscheiden sind die Übergabe von lokalen Objekten und entfernten Objekten, welche die Schnittstelle *Remote* implementieren.

- Für lokale Objekte existieren naturgemäß keine Stellvertreterobjekte und keine Registrierung in einem entfernten Server. Lokale Objekte werden bei einer Übertragung kopiert, damit auf dem entfernten System eine eigenständige Objektkopie verfügbar ist.
- Im Fall von entfernten Objekten existieren Stellvertreterobjekte (*Stub* und *Skeleton*) und die Registrierung(en) in dem/den entfernten Server(n). Damit ist eine Referenzierung dieser Objekte möglich. Wird ein *RemoteObject* als Parameter eines Methodenaufrufs angegeben, so wird sein Stellvertreter-Objekt, der *Stub*, übergeben.

Die Besonderheiten beim Kopieren von Objekten, wie Verweise auf andere Objekte, die in dem entfernten Adressraum nicht existieren, usw., werden von dem Mechanismus *Java Object Serialization* dem Entwickler gegenüber transparent geregelt. Damit können lokale und entfernte Objekte gleich behandelt werden.

Lokalisierung entfernter Objekte

Eine Kommunikation zwischen Client und Server erfordert eine Referenz (*Stub*) des entfernten Objekts. Man erhält diese im Allgemeinen als Rückgabewert eines Methodenaufrufs für ein entferntes Objekt. Dafür braucht man aber nun wieder den *Stub* für dieses Serverobjekt. Um dieses „Henne/Ei-Problem“ zu lösen gibt es einen Mechanismus im RMI (*bootstrap registry service*), um erstmalig eine Referenz (*Stub*) auf ein entferntes Objekt zu erhalten.

Dabei handelt es sich um einen einfachen Namensdienst, der für Serverobjekte Bezeichnungen, Namen und Referenzen verwaltet, über den die Clients eine Referenz zu einem Serverobjekt erhalten können. Dieser Namensdienst läuft als Dämonprozeß auf dem Serversystem und wacht an einem Port auf eine Dienstanforderung. Um diesen *Nameserver* anzusprechen, wird die *Uniform Ressource Locator* (URL) Syntax, wie z.B. *rmi://java.sun.com/rmiserv*, verwendet. Der Name wird vergeben durch die Implementierung auf Serverseite. Diese bindet die frei gewählte, eindeutige Bezeichnung an eine Instanz eines *RemoteObjects*. Ein Client erhält über den Namen (*lookup*) des entfernten Objekts eine Referenz (*Stub*-Objekt) und kann somit das Serverobjekt

ansprechen.

Ausnahmebehandlung

Im Bereich von verteilten Anwendungen spielt die Ausnahmebehandlung eine besondere Rolle, da die verschiedensten Fehlersituationen entstehen können.

Zum Beispiel:

- Fehlerhafte Ausführung von entfernten Methoden (Methodenaufrufe entfernter Objekte sind fehleranfällig)
- Ausfall einzelner Server bzw. Serververbindungen
- Ausfall bzw. Störung der Netzverbindung
- Geringe Performance auf dem Netz und daraus evtl. resultierende Fehler

Damit muß also die Ausnahmebehandlung (*Exception Handling*) stets beim Umgang mit entfernten Ressourcen berücksichtigt werden. Die Ausnahmebehandlung wird in Java über spezielle Objekte und Sprachkonstrukte geregelt. Tritt ein Fehler auf, so wird ein spezielles Objekt, eine *Exception*, ausgelöst. Diese Objekte werden im lokalen Fall von der Superklasse *Exception* und im entfernten (*remote*) Fall von der Superklasse *RemoteException* abgeleitet. Damit kann die Fehlerbehandlung in beiden Fällen analog vorgenommen werden, in dem die Ausnahmen abgefangen und von speziellen Programmteilen (*Exception-Handler*) verarbeitet werden.

Daher ist die Möglichkeit von Ausnahmen stets zu deklarieren und geschieht in den Klassendefinitionen über die Angabe der möglichen Ausnahmen.

Entwicklungsmodell / -vorgehen

Um eine verteilte Applikation zur Verfügung zu stellen, müssen die nachstehend beschriebenen Schritte vorgenommen werden:

1. Design der Applikation auf Basis der Schnittstelle *Remote.*
2. Realisierung der Applikation auf Basis von *RemoteObject* bzw. den Subklassen von *RemoteServer*.
3. Kompilierung der Java Quelldateien zu Java Klassendateien.
4. Generierung der *Stub-/Skeleton*-Klassen vollautomatisch aus den Applikationsklassen mit Hilfe des *rmic Bytecode-Compiler.*
5. Verteilung der Klassendateien (Schnittstellen und *Stub-/Skeleton*-Klassen) im Netz (Intranet/WWW).
    - Im statischen Fall werden die Klassendateien auf bestimmte Rechner (Server und Clients) verteilt.
    - Im dynamischen Fall können über Java-Applets auf Web-Seiten die entsprechenden Klassendateien bei der konkreten Benutzung geladen werden. Hierzu müssen in den HTML-Dokumenten entsprechende Informationen aufgenommen werden.

6. Starten des RMI *Registry Service* (*Nameserver*) auf dem/den Server(n).
7. Starten der Applikation auf dem/den Server(n), in deren Initialisierungsteil auch die Instanzen der Serverobjekte in der *Registry* eingetragen werden.
8. Starten der/des Clients und Lesen der ersten Referenzen von entfernten Serverobjekten vom Namensdienst.

Die Punkte (1), (2) und (3) sind allgemein bei einer Entwicklung durchzuführen und damit nicht spezifisch für RMI.

Die Punkte (4) und (5) sind vollständig automatisiert und somit mit sehr geringem Aufwand auszuführen. Der *rmic* Bytecode-Compiler erzeugt automatisch aus den angegebenen Java Klassendateien die Stellvertreter-Klassen. Der Namensdienst ist mit einem Kommando gestartet.

Ebenso sind die Punkte (7) und (8) nicht besonders RMI spezifisch, da dies allgemein bei Client-/Serveranwendungen der Fall ist. Damit ist nur im Client das *Objekt-Lookup* (8) und auf Serverseite das registrieren der Objekte besonders zu beachten. Bleibt Punkt (5), der naturgemäß das Thema *Verteilte Anwendungen* betrifft. Die Möglichkeit der Applets läßt hier eine zentrale Verwaltung zu.

Somit ist zusammenfassend zu bemerken, daß die Realisierung einer verteilten Anwendung mit Hilfe Java RMI nicht besonders aufwendig ist. Die Anforderungen liegen eher in der Analyse und im Design der Applikation und ist genereller Natur.

Zusammenfassung

Das Java-RMI-System bietet die Möglichkeit, eng gekoppelte und verteilte Systeme (Server-/Client-Applikationsstrukturen) innerhalb der Java-Umgebung zu realisieren.

Das verteilte Objektmodell und die APIs des Java-RMI fügen sich nahtlos in die Java-Umgebung ein. Das Java-Objektmodell wird durch RMI nur erweitert, und die Benutzung der entfernten Objekte ist analog zu lokalen Java-Objekten. Damit ist natürlich die Einführung in einem Java-Umfeld vereinfacht möglich und mit geringeren Kosten behaftet.

Somit ist zusammenfassend zu bemerken, daß sich die Realisierung einer verteilten Anwendung mit RMI durch Java anbietet. Aufgrund des formulierten Ziels seitens der Java-Herstellerfirma SUN, das RMI-Konzept möglichst nahtlos in die Sprache Java einzufügen, ist es für Entwickler nicht besonders aufwendig, eine verteilte Anwendung zu realisieren.

Literatur

T. Belzner, F. Toenniessen, Objektorientierung in großen Informationssystemen: Die Zeit ist reif, in: Computerwoche Extra, Ausgabe: 14. Februar 1997.

R. Budde, K.-H. Kautz, K. Kuhlenkamp, H. Züllighoven, Prototyping – An Approach to Evolutionary System Developement, Springer 1990 Heidelberg u.a.

C. Floyd, A Systematic Look at Prototyping. In: R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Züllighoven (Hrsg.), Approaches to Prototyping. Springer 1984, Berlin, Heidelberg, New York, Tokyo

J. Coldewey, W. Keller, Objektorientierte Datenintegration, in: Objekt Spektrum, Ausgabe: Juli / August 1996, Nr. 4.

F. F. Debatin, Java und Datenbanken, Java Spektrum, März/April 1997, SIGS Conferences GmbH Bergisch Gladbach

H. Dicken, JDBC – Internet-Datenbankanbindung mit Java, 1. Auflage, International Thomsom Publishing Bonn Albany 1997

B. Eckel, Thinking in Java, Revision 9, 16. August 1997, Bruce Eckel 1997, elektronische Version im WWW: http://www.EckelObjects.com

S. R. Farley, Mobile Agent System Architecture – A flexible alternative to moving data and code to complete a given task, in: Java Report, SIGS Publications, Inc., New York 1997.

D. Flanagan, JAVA IN A NUTSHELL, O´Reilly/International Thomson Verlag GmbH & Co KG, Dt. Ausgabe 1996

E. Gamma, R. Helm, R. Johnson, J. Vlissides, Entwurfsmuster, Addison-Wesley (Deutschland) GmbH, 1. Auflage 1996/ Dt. Übers. D. Riehle

W. Hahn, F. Toenniessen, A. Wittowski, Eine objektorientierte Zugriffsschicht zu relationalen Datenbanken, in: Informatik Spektrum, Ausgabe: Nr. 18, 1995

N. Hendrich, Java für Fortgeschrittene, Springer-Verlag Berlin Heidelberg 1997

R. Klute, Datenbankentwicklung, iX Magazin für professionelle Informationstechnik, Ausgabe: Juni 1997

O. Kyas, Internet professionell, Thomson Publishing 1996

B. Merkle, Flugzeug-Buchungssystem mit Java und CORBA, Java Spektrum, Ausgabe: Juli/August 1997, SIGS Conferences GmbH Bergisch Gladbach

C. Niederée, C. Hattendorff, S. Müßig, Warburg Electronic Library- Eine digitale Bibliothek für die politische Ikonographie, in: "uni hh-Forschung", Ausgabe: XXXI / 1997, Pressestelle der Universität Hamburg

M. Ortak, Die Kooperation von Objekten im Internet, OBJEKTspektrum, September/Oktober 1997, SIGS Conferences GmbH Bergisch Gladbach

J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Objektorientiertes Modellieren und Entwerfen“, Hanser–Verlag / Dt. Übers. 1993 D. Märtin

M. Schader, Objektorientierte Datenbanken, Springer-Verlag Berlin Heidelberg 1997

J. W. Schmidt, G. Scröder, C. Niederéee, F. Matthes, Linguistic and Architectural Requirements for Personalized Digital Libraries, Universität Hamburg, Fachbereich Informatik, Lehrstuhl Datenbanken und Informationssysteme, 1. Oktober 1996

A. Simon, Datenbanken im Internet, OBJEKTspektrum, September/Oktober 1997, SIGS Conferences GmbH Bergisch Gladbach

J. Turek, Search and retrieval in large image archives, IBM: Research reports; RC 20214, 1995

S. Weiland, Java-Applikationen im Intranet, Java Spektrum, Mai/Juni 1997, SIGS Conferences GmbH Bergisch Gladbach

CORBA/IIOP 2.0 Specification, http://www.omg.org/corba/corbiiop.htm, Juli 1997, The Object Management Group (OMG)

The Common Object Request Broker: Architecture and Specification,
Revision 2.0, Juli 1995, aktualisiert: Juli 1996, The Object Management Group (OMG)

Java Object Serialization Specification,
Revision 1.3, JDK 1.1 FCS, 10. Februar 1997, Sun Microsystems Inc.

Java Remote Method Invokation Specification,
Revision 1.4, JDK 1.1 FCS, 10. Februar 1997, Sun Microsystems Inc.

Java RMI Tutorial,
Revision 1.3, JDK 1.1 FCS, 10. Februar 1997, Sun Microsystems Inc.

Forschungsstelle Politische Ikonographie (Warburg-Haus), Bildindex zur politischen Ikonographie, ‚Privatdruck Hamburg 1996 (2. Auflage)

Voyager and RMI Comparsion, Mai 1997, ObjectSpace 1997

Voyager User Guide, Juli 1997, ObjectSpace 1997