

Studienarbeiten zum Thema

**Partizipative Anpassung einer CSCW-Kooperationsplattform
über die Bereitstellung expliziter Gestaltungsfunktionen**

Verantwortlicher Betreuer: Prof. Dr. S. Abeck
Prof. Dr. N. Kohler

Betreuender Mitarbeiter: Dipl.-Inform. C. Mayerl
Dipl.-Ing. C. Müller

Bearbeiter: Artur Hecker
Robert Rodewald

Inhaltsverzeichnis

1	EINLEITUNG	4
1.1	AUSGANGSLAGE	4
1.2	AUFGABEN DER STUDIENARBEITEN	4
1.2.1	<i>Gemeinsam zu lösende Aufgaben der Studienarbeiten</i>	5
1.2.2	<i>Studienarbeit von Artur Hecker (Entwicklung des Clients)</i>	5
1.2.3	<i>Studienarbeit von Robert Rodewald (Entwicklung der Serveranwendung)</i>	6
1.3	AUFBAU DER DOKUMENTATION	6
2	SYSTEMANALYSE	7
2.1	PROBLEMANALYSE	7
2.2	ZIELANALYSE	8
2.3	AUFGABENANALYSE	9
3	GRUNDLEGENDE KONZEPTE	10
3.1	ÜBERLEGUNGEN ZUR VERWENDETEN TECHNIK	10
3.1.1	<i>Überlegungen zur Struktur des Modellierungswerkzeugs</i>	10
3.2	ÜBERLEGUNGEN ZUR GESTALTUNG DER EINZELNEN SYSTEMKOMPONENTEN UND DEREN KOMMUNIKATION	11
3.2.1	<i>Gestaltung des Clients – ein Applet versus mehrere Applet-gestützte Seiten</i>	11
3.2.2	<i>Kommunikation der Komponenten - RMI/Sockets versus HTTP</i>	11
3.3	GROBE ÜBERSICHT ÜBER DIE NEUE SYSTEMSTRUKTUR	13
3.4	METHODE FÜR DEN DATENAUSTAUSCH	13
3.5	DEKODIERUNG UND KODIERUNG VON POST-DATEN	14
3.6	REQUESTEDINFO-SCHNITTSTELLE	14
3.7	MODELLIERUNG UND ROLLEN	17
3.8	REQUESTEDFCT-SCHNITTSTELLE FÜR DIE ÜBERGABE VON MODELLIERUNGSANFORDERUNGEN	18
4	CLIENT	20
4.1	EINFÜHRUNG	20
4.1.1	<i>Aufbau der Beschreibung</i>	20
4.1.2	<i>Verwendete Abkürzungen und Bezeichnungen</i>	20
4.2	KOMMUNIKATION	21
4.2.1	<i>StreamDecoder-Klasse</i>	21
4.2.2	<i>JavaPOST-Klasse</i>	22
4.2.3	<i>Cache-Konzept und Cache-Klasse</i>	23
4.3	DARSTELLUNGSMODUS	25
4.3.1	<i>Analyse des Prototyps</i>	25
4.3.2	<i>Zustandsaufteilung und zusätzlicher Fehlerzustand</i>	26
4.3.3	<i>Implementierungsproblem, Zustandsvaterkonzept und –klasse</i>	31
4.3.4	<i>Zustandssicherungsproblem und implementierte Lösung</i>	35
4.3.5	<i>Grundsätzliches Vorgehen beim Zustandsaufbau</i>	36
4.4	MODELLIERUNGSMODUS	36
4.4.1	<i>Einbindung ins Gesamtkonzept</i>	36
4.4.2	<i>Modellierungsfunktionen mit Rechten</i>	37
4.4.3	<i>Standardablauf einer Modellierungsfunktion</i>	38
4.4.4	<i>Funktionsvorlage</i>	38
4.4.5	<i>Rechteidentifizierung und Durchsetzung</i>	39
4.5	HAUPTAPPLET UND KURZE VORSTELLUNG ALLGEMEINER KLASSEN	39
4.5.1	<i>Hauptapplet</i>	39
4.5.2	<i>Allgemeine Hilfsklassen</i>	41

5	SERVERSEITIGE IMPLEMENTIERUNG	42
5.1	ÜBERBLICK.....	42
5.2	DIE STRUKTURINFORMATIONEN	43
5.3	DIE AGENTEN	44
5.3.1	<i>Der Agent „AnfrageAgent“.....</i>	<i>44</i>
5.3.2	<i>Der Agent „AktAntragBearbeitung“.....</i>	<i>45</i>
5.3.3	<i>Der Agent „AktAntragZeitkontrolle“.....</i>	<i>47</i>
5.3.4	<i>Weitere Agenten</i>	<i>47</i>
	<i>Fehlerbehandlung.....</i>	<i>48</i>
5.4	DIE MODELLIERUNGSFUNKTIONEN	48
	<i>Aufbau und Rechtesystem.....</i>	<i>48</i>
5.4.1	<i>Akteur hinzufügen</i>	<i>50</i>
5.4.2	<i>Akteur entfernen</i>	<i>50</i>
5.4.3	<i>Akteur zu Kontextbereich hinzufügen</i>	<i>50</i>
5.4.4	<i>Akteur aus Kontextbereich entfernen</i>	<i>50</i>
5.4.5	<i>Kontextbereich erzeugen</i>	<i>50</i>
5.4.6	<i>Wechselwirkung erzeugen</i>	<i>51</i>
5.4.7	<i>Kontextbereichsmoderator ändern</i>	<i>51</i>
5.4.8	<i>Wechselwirkungsmoderator ändern</i>	<i>51</i>
5.5	DER AKTIVIERUNGSANTRAG	51
5.5.1	<i>Der Aktivierungszyklus.....</i>	<i>52</i>
5.5.2	<i>Die Maske „Aktivierungsantrag“</i>	<i>53</i>
5.6	DIE ALLGEMEINEN HILFSKLASSEN	56
5.6.1	<i>Die Klasse „StringDecoder“.....</i>	<i>56</i>
5.6.2	<i>Die Klasse „RequestReply“</i>	<i>57</i>
5.6.3	<i>Die Klasse „UserInfo“.....</i>	<i>57</i>
5.6.4	<i>Die Klasse „ContextInfo“</i>	<i>57</i>
5.6.5	<i>Die Klasse „ForseenException“</i>	<i>58</i>
5.6.6	<i>Die Klasse „Assert“</i>	<i>58</i>
5.6.7	<i>Die Klassen „DatabaseRes“, „ErrorMessagesRes“ und „UserMessagesRes“</i>	<i>59</i>
5.7	BEMERKUNGEN.....	59
5.7.1	<i>Replizierung.....</i>	<i>59</i>
5.7.2	<i>Integration des Applets</i>	<i>59</i>
5.8	ZUSAMMENFASSUNG	60
6	LITERATURVERZEICHNIS.....	61

1 Einleitung

1.1 Ausgangslage

Wissenschaftlicher Hintergrund der beiden vorliegenden Studienarbeiten ist eine Dissertation am Institut für Industrielle Bauproduktion, in deren Rahmen eine prototypische Umsetzung eines "Virtuellen Projektraumes" erfolgen soll.

Aus den steigenden Anforderungen an Gebäude und der Notwendigkeit einer Betrachtung ihres gesamten Lebenszyklus' erwachsen für die Planer im Baubereich immer komplexere Problemstellungen. Deren innovative Lösungen können am besten von teamorientiert und integral arbeitenden interdisziplinären Planungsteams erbracht werden.

Im Gegensatz zu anderen Branchen mit ähnlicher Produktkomplexität wird die Anwendung der dazu notwendigen Kooperations- und Managementmethoden durch die schwierigen Rahmenbedingungen bei Bauprojekten jedoch erschwert. Diese sind z.B. projektweise wechselnde Konsortien mit einer Vielzahl von Fachplanern, die Überlappung von Planung und Ausführung oder die Ziel- und Anforderungsanpassung während des Planungszeitraumes.

Weitere Hindernisse sind formale phasen- oder gewerkeorientierte Organisations- und Vertragsmodelle und fehlende informationstechnologische Unterstützung der Kooperationsvorgänge bei räumlich und zeitlich verteiltem Arbeiten.

Der in der Dissertation vorgeschlagene Lösungsweg sieht die konsequente Anwendung organisatorischer Virtualisierungsstrategien zur Bildung projektbezogener virtueller Planungsunternehmen vor. Solche virtuellen Organisationen können sich Flexibilitäts- und Synergiepotentiale durch eine aufgabenbezogene und weitgehend selbstorganisierte Vernetzung intellektueller, technischer, informationeller oder finanzieller Ressourcen erschließen. Allerdings sind diese innovativen Organisationskonzepte nicht ohne eine geeignete Systemunterstützung umsetzbar.

Ziel ist es deshalb, eine Kooperationsplattform zu entwickeln, die sowohl die anforderungsorientierte Projektorganisation als auch die eigentliche verteilte Kooperation unterstützt. Dabei soll die netzwerkartige Organisationsstruktur so mit der des informations- und kommunikationstechnischen Systems verbunden werden, daß dynamische Projektorganisation und die Konfiguration der Systemumgebung zu einem Vorgang verschmelzen. Die so gestalteten Organisationsstrukturen werden durch eine geeignete Verankerung im CSCW-System ("awareness", Lese-, Schreib- und Ausführungsrechte) zu einem leistungsfähigen Koordinationsinstrument.

Das System verlagert somit seinen Schwerpunkt deutlich von einer reinen Informations- auf eine Organisationstechnologie. Der virtuelle Projektraum wird also zum Organisationsraum.

Die Bedeutung des Prinzips der Selbstorganisation in virtuellen Organisationen macht es erforderlich, diese Gestaltung der Projektorganisation, d.h. damit auch die Konfiguration des Systems, je nach Wirkungsbereich und Rolle partizipativ zu ermöglichen.

1.2 Aufgaben der Studienarbeiten

Eine erste Version der Kooperationsplattform ist schon seit 8/1998 in einem Anwendungsprojekt erfolgreich im Einsatz. Dieses CSCW-System (Abbildung 1-1) enthält schon eine Reihe von Unterstützungsfunktionen und bildet bereits statisch die Projektstruktur im System und im GUI ab. Allerdings stehen den Planern noch keine expliziten Gestaltungsfunktionen zur dynamischen Anpassung der Organisationsstruktur zur Verfügung. Jede Form organisatorischer Gestaltung muß durch direkten Eingriff des Entwicklers(!) in das System vorgenommen werden.

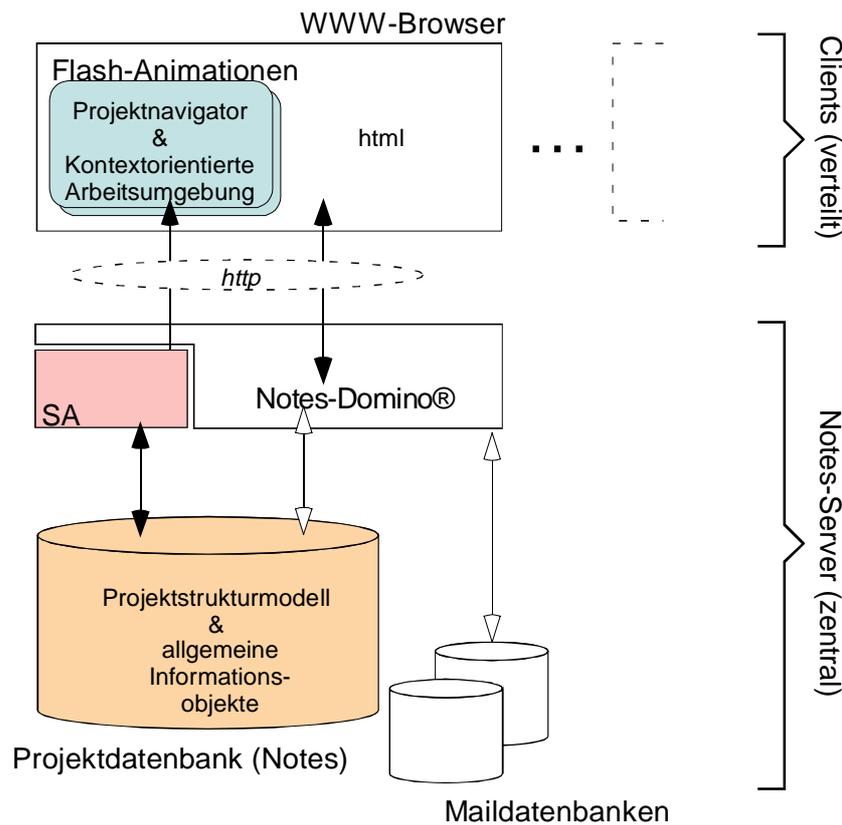


Abbildung 1-1

Die Erweiterung des bestehenden Prototyps um explizite organisatorische Gestaltungsfunktionen sollte im Rahmen von Studienarbeiten der Fachrichtung Informatik erfolgen. Die Architektur des Systems (Client-Server) sowie der zu erwartende Umfang machten eine Aufteilung in zwei in enger Kooperation zu erarbeitenden Studienarbeiten erforderlich. Interessant war diese Aufteilung sicherlich auch unter dem Aspekt der gemeinsamen Entwicklung eines einzigen Softwareprojekts. Aus dieser Vorgehensweise resultierten daher gemeinsam wie auch einzeln zu bearbeitende Aufgabenpakete, welche nachfolgend kurz skizziert werden.

1.2.1 *Gemeinsam zu lösende Aufgaben der Studienarbeiten*

Die Formulierung der Problemstellung ließ uns relativ viel Freiraum im Bezug auf den Lösungsweg. Vor der eigentlichen Implementierung mußten deshalb im Rahmen der Studienarbeiten noch folgende Vorarbeiten geleistet werden:

- Analyse des existierenden Prototyps und Präzisierung der Aufgabenstellung.
- Erstellung einer Problem-, Ziel- und Aufgabenanalyse
- Klärung konzeptueller und technischer Fragen (z.B. Aufteilung in Programmteile, verwendete Sprachen, Kommunikationswege, Protokolle)
- Definition einer offenen Kommunikationsschnittstelle zwischen Client und Server

1.2.2 *Studienarbeit von Artur Hecker (Entwicklung des Clients)*

Aufgrund der geforderten maximalen Plattformunabhängigkeit wird eine WWW-basierte Clientimplementierung gefordert. Der derzeitige Prototyp benutzt zur Darstellung interaktiver Elemente der Benutzerschnittstelle die Flash-Technologie von Macromedia, deren Plug-Ins in allen gängigen Browsern integriert sind. Diese ermöglichen multimediale Darstellungen durch vordefinierte Animationseffekte innerhalb des Browserfensters. Vor dem Hintergrund einer statischen Projektstruktur ist dieses Vorgehen ausreichend.

Im Hinblick auf die jetzt angestrebte dynamische Anpassung der Projektstruktur über die zu implementierenden Gestaltungsfunktionen des Clients ist dies jedoch nicht mehr möglich. Vielmehr müssen die vom Server bereitgestellten und durch den Client abgefragten

Strukturinformationen und Daten jedesmal - gemäß den Gestaltungsvorgaben des Prototyp-GUI – graphisch neu aufbereitet werden. Zusätzlich dazu ist es erforderlich, die GUI-Dialoge der Gestaltungsfunktionen und die hier notwendige Kommunikation mit der Serveranwendung zu realisieren.

1.2.3 Studienarbeit von Robert Rodewald (Entwicklung der Serveranwendung)

Die serverseitige Implementierung des Prototyps basiert auf einer Datenbank der Groupware-Entwicklungsplattform Lotus Notes/Domino, die um statische Flash-gestützte HTML-Seiten ergänzt ist. Die Grundfunktionalitäten des Domino-Servers bestehen dabei im wesentlichen aus einem verteilten Datenbanksystem und Kommunikationskomponenten, erweitert um die Funktionalität eines Web-Servers. In diesem Zusammenhang ist es auch möglich, in bestimmtem Umfang den Zugriff auf die Datenbanken über WWW-Browser zu ermöglichen (Web-Formulare). Die Anpassung der Struktur des in der Datenbank abgebildeten Projekts kann derzeit nur direkt (ohne Konsistenzprüfung) durch den Entwickler selbst erfolgen.

Die benutzerseitige partizipative Anpassung der Projektstruktur erfordert aber eine modellierbare Aufbereitung der Strukturinformation. Die serverseitige Anwendung muß weiterhin dahingehend erweitert werden, daß auf Client-Anfrage geeignet aufbereitete Strukturinformationen geliefert werden und andererseits die Durchführung der Gestaltungsfunktionen unter Anwendung von Konsistenzsicherungsmaßnahmen realisiert werden. Dies erfordert den Entwurf und die Implementierung eines entsprechenden "Verbinders". Da dieser „Verbinder“ für das GUI gewissermaßen als Zugriffsschicht für die Datenbank und die Strukturinformationen fungiert, wird für ihn im Folgenden der Begriff „DBAL“ (database abstraction layer) verwendet.

1.3 Aufbau der Dokumentation

Um die Konsistenz der Dokumentation sicherzustellen, wird für die beiden Studienarbeiten ein gemeinsames Dokument erstellt. Dabei werden die einzelnen Arbeitsleistungen jedoch in voneinander getrennten Kapiteln dokumentiert. Daraus ergibt sich die folgende Gliederung:

- Beschreibung der Systemanalyse
- Alternativen und Entscheidungen grundlegender Art
- Implementierung des Client
- Serverseitige Implementierung

2 Systemanalyse

2.1 Problemanalyse

Mit Hilfe einer Problemanalyse wird versucht das Kernproblem zu isolieren. Dazu werden alle Punkte gesammelt, die problematisch erscheinen und negativ formuliert. Diese werden dann in einen kausalen Zusammenhang gebracht, so daß das Kernproblem in der Mitte steht, die Folgen darüber und die Gründe darunter. Die Verbindungspfeile können gewissermaßen als Folgerungspfeile aufgefaßt werden:

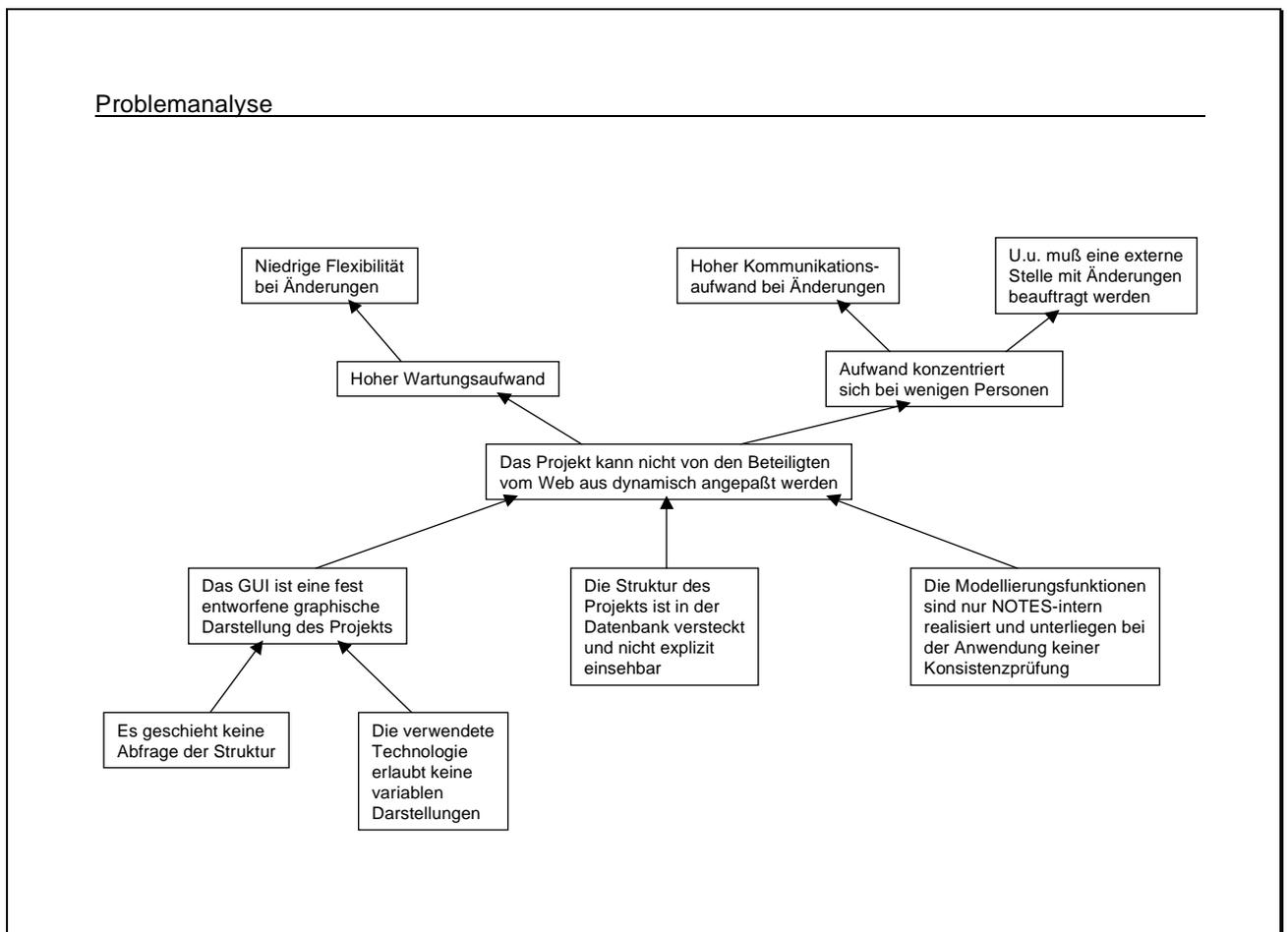


Abbildung 2-1

2.2 Zielanalyse

Aus der Problemanalyse läßt sich unmittelbar die Zielanalyse ableiten, indem die negative Formulierung in eine positive umgekehrt wird. Daraus ergibt sich das Kernziel in der Mitte, die dadurch erreichten positiven Effekte darüber und die zu erfüllenden Teilziele darunter. Die aufgeführten Ziele können dann noch in KANN- und MUß-Ziele eingeteilt werden.

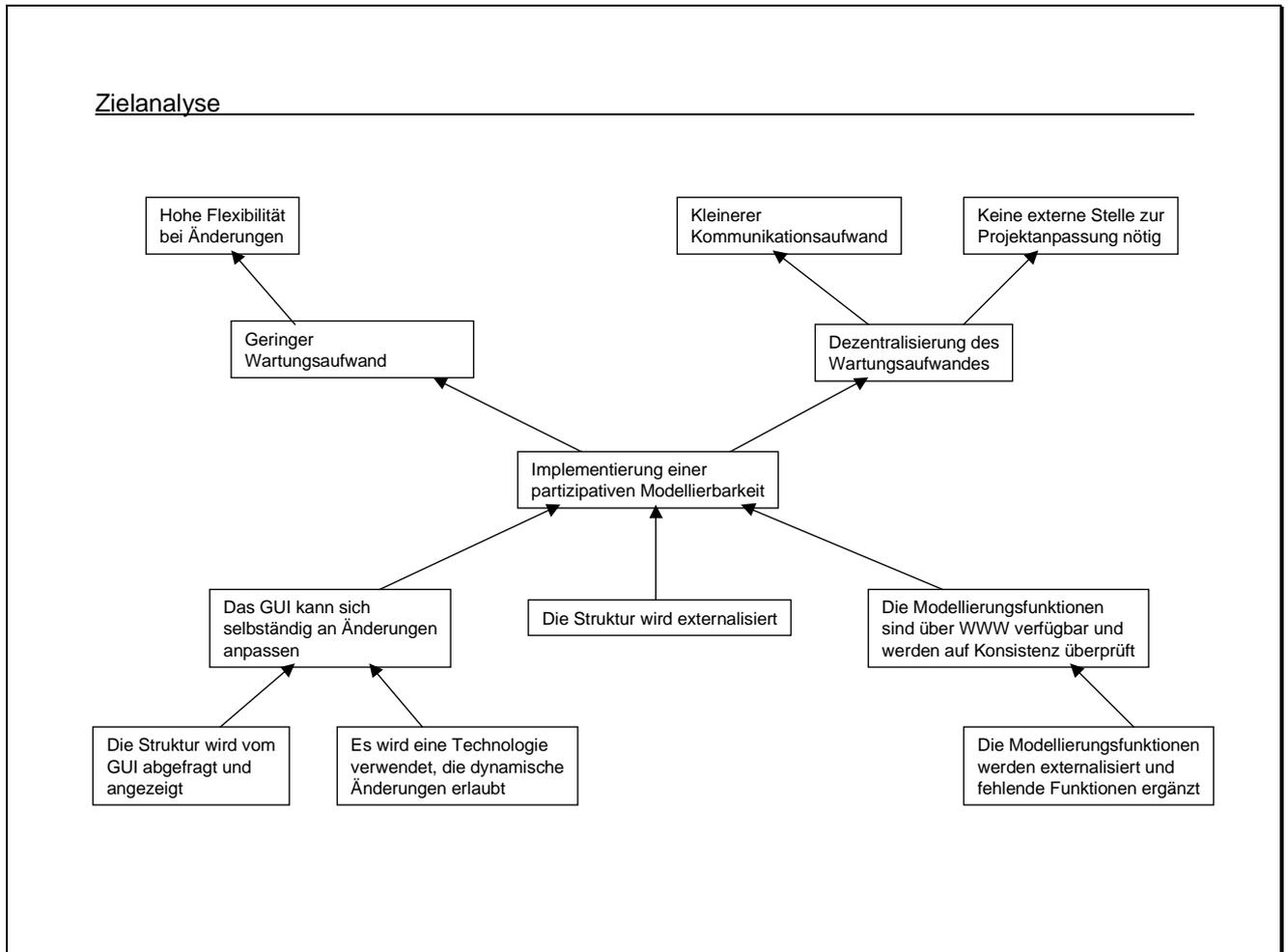


Abbildung 2-2

2.3 Aufgabenanalyse

Aus den MUß-Zielen und deren Teilzielen der Zielanalyse müssen im Zuge der Aufgabenanalyse Aufgaben und Teilaufgaben formuliert werden. Diese Aufgaben bilden zugleich die Grundlage für das Pflichtenheft und müssen so formuliert sein, daß deren Erfüllung einfach und eindeutig nachgeprüft werden kann.

Aufgaben

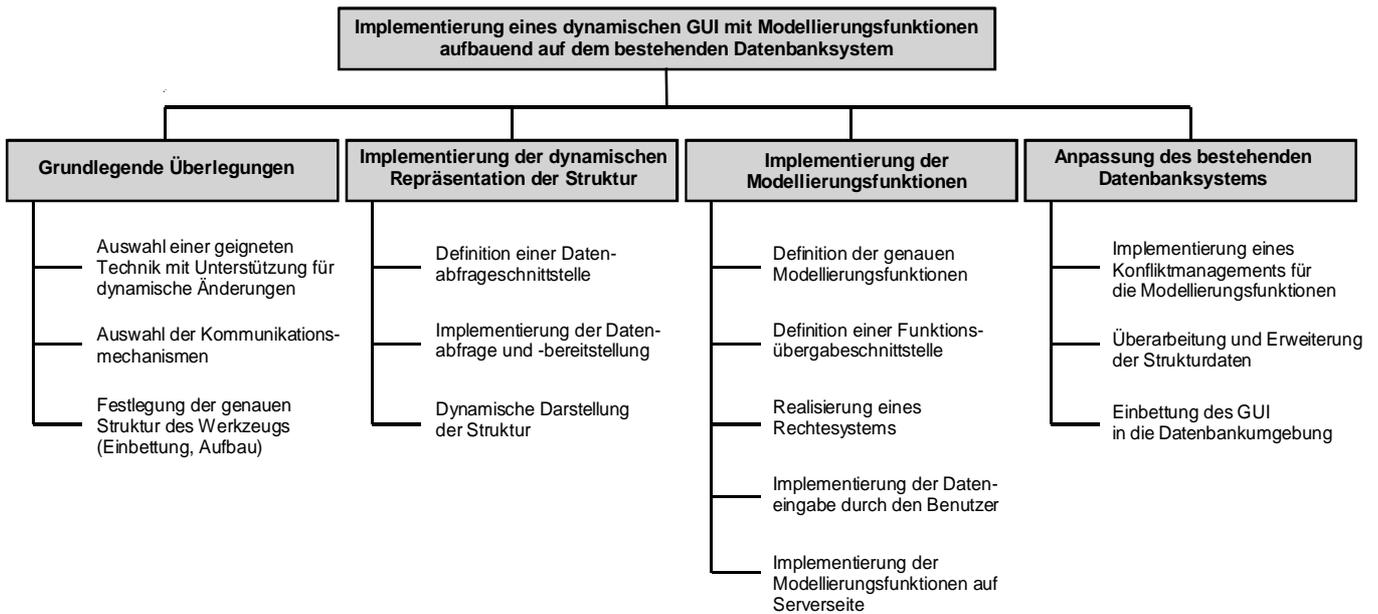


Abbildung 2-3

Es ist zu beachten, daß die daraus resultierenden Anforderungen nicht als die Beschreibung der Aufgabenstellung der Studienarbeiten anzusehen sind, da schon die vorangegangenen Abschnitte wie z.B. die Systemanalyse und auch die Ermittlung der daraus resultierenden Anforderungen selbst Bestandteil der Arbeit waren.

3 Grundlegende Konzepte

3.1 Überlegungen zur verwendeten Technik

3.1.1 Überlegungen zur Struktur des Modellierungswerkzeugs

Es wurden die Vor- und Nachteile zweier alternativer Strukturen untersucht. Eine davon mit und die andere ohne eine separate SQL-Datenbank für die Strukturinformationen des Kooperationsmodells. Die Abbildung 3-1 und Abbildung 3-2 skizzieren eben diese Alternativen. Die folgende Tabelle beschreibt die Vor- und Nachteile der ersten Variante.

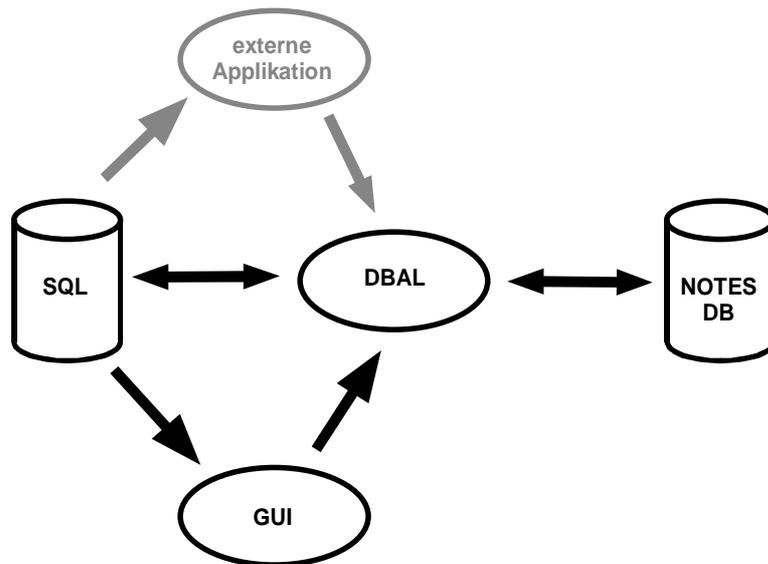


Abbildung 3-1

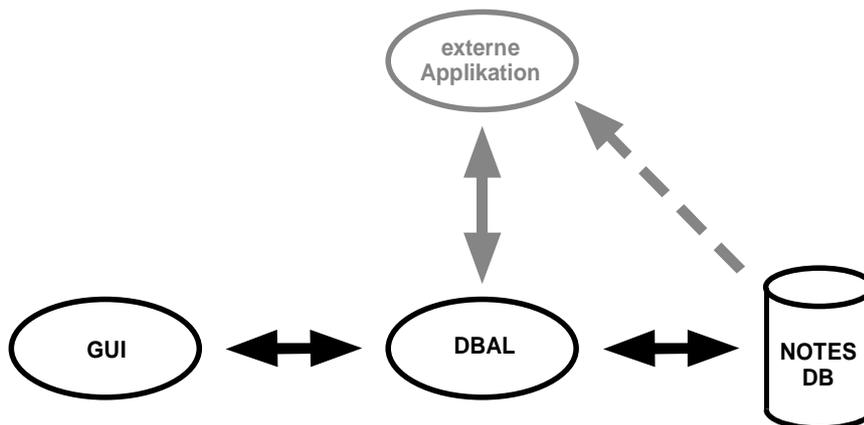


Abbildung 3-2

Vor- und Nachteile der Alternative mit SQL-Strukturdatenbank

PRO	KONTRA
<ul style="list-style-type: none"> • Die Abfrage durch das GUI bzw. externe Applikationen ist besonders einfach. • Die GUI/DBAL-Schnittstelle ist sehr einfach. • Die Strukturdaten werden durch die relationale Struktur besser abgebildet. • Komplizierte Abhängigkeiten von verschiedenen Objekten werden bei Änderungen automatisch durch SQL berücksichtigt, wenn die Relationen korrekt vorgegeben sind. 	<ul style="list-style-type: none"> • Die Strukturdaten werden z.T. doppelt gehalten • Die Synchronisation der beiden Datenbestände birgt ein hohes Fehlerpotential. • Der Datenfluß ist bei der zweiten Alternative deutlich übersichtlicher (Schichtenarchitektur). • Es gibt unterschiedliche Schnittstellen für Abfrage bzw. Änderung der Daten. • Auch die Alternative ohne SQL-Datenbank erlaubt externen Applikationen beschränkten Zugriff über SQL-Abfragen (über Notes SQL). • Die Programmierung erfordert mehr Aufwand, weil zwei verschiedene Systeme und eine weitere Schnittstelle zu implementieren sind.

Fazit: Aufgrund der oben stehenden Überlegungen wird eine Realisierung *ohne* getrennte SQL-Strukturdatenbank gewählt.

3.2 Überlegungen zur Gestaltung der einzelnen Systemkomponenten und deren Kommunikation

3.2.1 Gestaltung des Clients – ein Applet versus mehrere Applet-gestützte Seiten

Für die Gestaltung des GUI wurden zwei Möglichkeiten untersucht. Die eine Variante besteht darin, ein einzelnes umfangreiches Applet auf eine WWW-Seite zu legen, die andere darin, mehrere WWW-Seiten mit deutlich kleineren Applets (und unter Umständen Java Script Elementen) zu generieren.

Vor- und Nachteile der Alternative mit einem einzelnen Applet

PRO	KONTRA
<ul style="list-style-type: none"> • Es gibt keine Kommunikation über die Grenzen der WWW-Seite hinweg • Die Struktur ist deutlich kompakter. • Der Aufruf geschieht immer auf die selbe Art und Weise. • Es gibt keine Schnittstellen zwischen JavaScript und dem Applet. 	<ul style="list-style-type: none"> • Das Applet muß mit Informationen über den Einsprung versorgt werden und sich in verschiedene Zustände versetzen können. • Erweiterungen der Funktionalität müssen in das Applet eincompiliert werden.

Fazit: Das GUI wird als ein einziges Applet implementiert.

3.2.2 Kommunikation der Komponenten - RMI/Sockets versus HTTP

Für die Kommunikation von GUI und DBAL wurden zwei prinzipiell verschiedene Möglichkeiten erörtert. Zum einen besteht die Möglichkeit, das DBAL als getrenntes Programm auf dem Server laufen zu lassen und für die Kommunikation RMI oder Sockets zu nutzen. Die andere Alternative besteht darin, das DBAL als Teil der Notes Datenbank zu implementieren und Anfragen sowie Rückgabedaten nur über das HTTP-Protokoll zu versenden. Im folgenden wurden diese beiden Möglichkeiten kurz skizziert und bewertet.

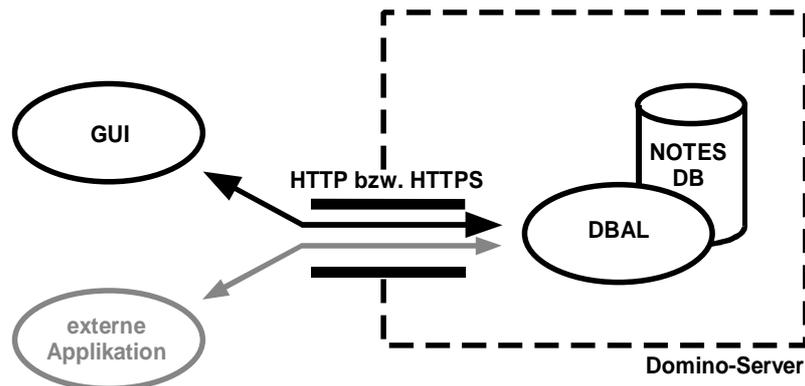


Abbildung 3-3

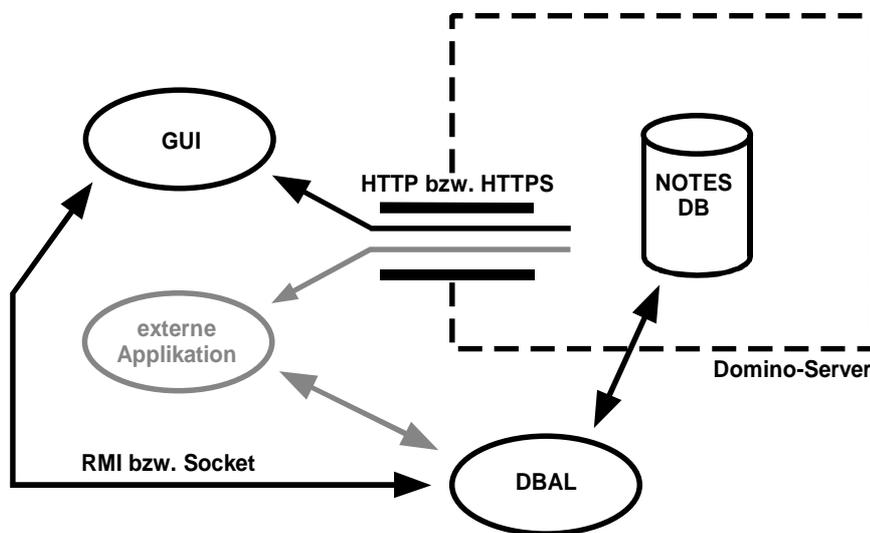


Abbildung 3-4

Vor- und Nachteile der Alternative mit reinen HTTP-Anfragen

PRO	KONTRA
<ul style="list-style-type: none"> • Ein einziges Protokoll wird für alle Anfragen und Rückgabedaten verwendet. • Die Authentifizierung wird vom Webserver (hier Domino) übernommen. • Eine Erweiterung der Sicherheit durch Übergang auf HTTPS ist problemlos möglich. • Das DBAL wird sauber in eine bestehende Plattform eingebunden, anstatt als eigenständiges Programm zu laufen. • RMI hat eine Beschränkung auf Java fähige externe Programme zur Folge. Die weitere Entwicklung von RCP ist noch nicht absehbar. 	<ul style="list-style-type: none"> • HTTP-Verbindungen sind ineffizienter als Sockets

Fazit: Die gesamte Kommunikation wird über HTTP-Anfragen realisiert. Das DBAL wird als Teil der Notes Datenbank (z.B. als Agent) implementiert.

3.3 Grobe Übersicht über die neue Systemstruktur

Aus den vorangegangenen Überlegungen leitet sich die in der Abbildung 3-5 dargestellte Struktur des Gesamtsystems ab.

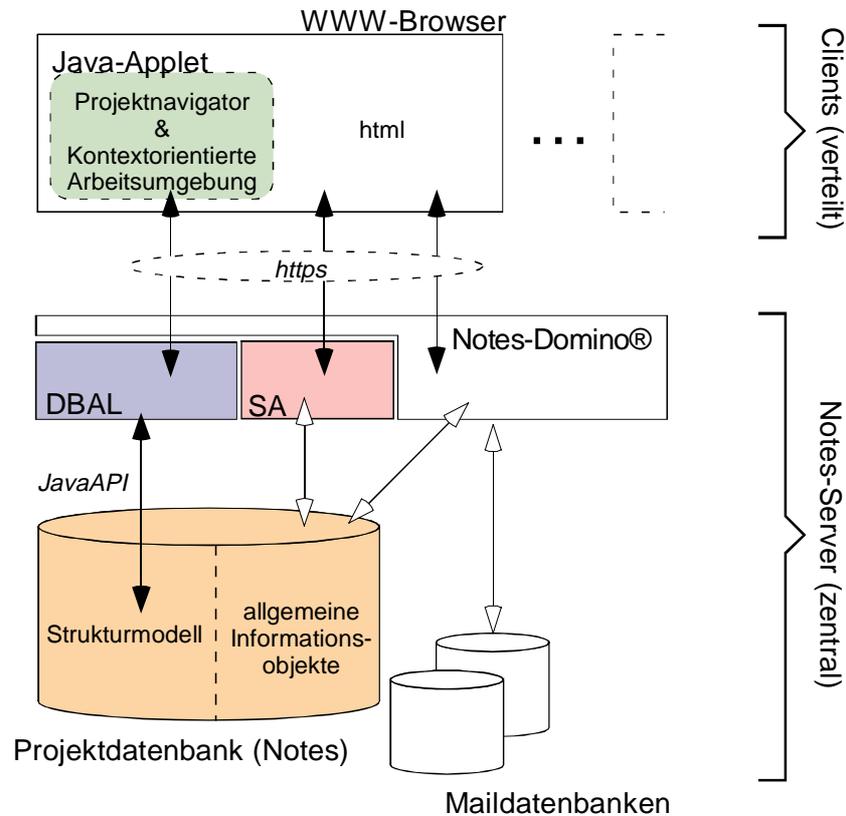


Abbildung 3-5

3.4 Methode für den Datenaustausch

Die Auswahl von HTTP als Kommunikationsprotokoll macht eine weitere Entscheidung notwendig: HTTP definiert verschiedene Methoden, wie ein HTTP-Client Daten an den HTTP-Server schicken kann. Die gängigen Methoden sind GET und POST. Bei GET werden die Daten an die URL-Adresse angehängt, während bei POST direkt auf die Verbindung zu einer URL-Adresse geschrieben wird. In beiden Fällen liefert der Server (bzw. das von diesem gestartete Programm) die Daten und schließt die Verbindung. Die Daten werden jeweils im folgenden Format an den Server übergeben:

```
param1=wert1[&param2=wert2 .. &paramN=wertN],
```

wobei alle Werte (wert1 .. wertN) nach „application/x-www-form-urlencoded“ kodiert werden.

Wir haben uns für die POST-Methode entschieden. Diese ist für den Endbenutzer unsichtbar, während man bei GET die URL-Adresse und die übergebenen Parameter beobachten kann. Andererseits können bei POST keine Fehler bei der zusätzlich anfallenden URL-Zusammensetzung gemacht werden, da immer auf die Verbindung zur Basis-URL geschrieben wird. Entstandene Fehler können vom DBAL erkannt werden. DBAL muß die übergebenen Daten dagegen in jedem Fall prüfen, so daß hier kein zusätzlicher Aufwand entsteht.

3.5 Dekodierung und Kodierung von POST-Daten

Da die Übergabe der Daten mittels POST-Anfragen eine Kodierung und Dekodierung voraussetzt, und zwar in beiden Kommunikationspartnern, mußte diese Aufgabe im allgemeinen Teil gelöst werden. Die Kodierung stellt dabei das geringere Problem dar, da sie von der Java-Standardklasse „URLEncoder“ übernommen wird. Für den Dekoder werden zwei Varianten benötigt, weshalb hier im allgemeinen Teil nur eine abstrakte Oberklasse definiert wird, die von den entsprechenden Modulen zu erweitern sind. In die Oberklasse „Decoder“ wurde als tatsächliche Funktionalität nur die Methode „decode“ aufgenommen, die eine Zeichenkette dekodiert. Desweiteren wurde die abstrakte Methode „getHashtable“ erzeugt, die den Benutzer der Klasse zwingt, eine solche Methode zu programmieren. Diese sollte aus einer beliebigen Basisdatenstruktur (Strom oder Zeichenkette) eine Hashtabelle generieren, die dekodierte Paare von Schlüssel und Wert besitzt.

3.6 RequestedInfo-Schnittstelle

Als erstes sollte die Funktionalität des bestehenden Flash/HTML-Prototypen komplett nachgeahmt werden. Die Daten mußten jedoch aus der Datenbank geholt werden. So ergab sich die Notwendigkeit der Definition einer (zunächst nur Lese-) Schnittstelle zwischen dem GUI (client-seitig) und dem DBAL (server-seitig). Diese Schnittstelle wurde der Aufgabenstellung wegen komplett an GUI-Bedürfnisse angepaßt: Der Aufbau der Schnittstelle und die dabei vom DBAL gelieferten Datentypen entsprechen im Wesentlichen den zeitlich und zustandsabhängig getrennten Datenanforderungen des GUI.

Die Schnittstelle wurde an die Standardsituation angepaßt. Unter der Standardsituation verstehen wir dabei das normale „Durchklicken“ von der obersten Ebene immer tiefer. In diesem Fall braucht das GUI nicht die Informationen zu holen, die es bereits in einem früheren Zustand geholt hat (s. auch Cache-Konzept im Client-Teil); das GUI ist also lediglich an den für diesen Zustand und diesen Benutzer spezifischen Informationen interessiert. Die Schnittstelle wurde daher logisch gesehen als ein Bitvektor organisiert. Das GUI hat so die Möglichkeit, abhängig von der aktuellen Situation durch Setzen von bestimmten Bits alle benötigten Informationen zu empfangen. Die Standardsituationen erfordern dabei das Setzen von maximal zwei Bits (ein weiteres wegen High- oder Low-Detail). Sollte eine Nichtstandardsituation auftreten (z.B. das GUI wird in einem tieferen Zustand gestartet), so kann das GUI dennoch *alle* benötigten Informationen in *einer* Anfrage geliefert bekommen, indem es mehrere Bits setzt. Ausnahme: Die erste Anfrage bestimmt, welche Anfragen noch gemacht werden müssen. DBAL untersucht die Anfragezeichenkette, sucht alle benötigten Informationen aus der Datenbank heraus und schreibt diese auf den Antwortstrom. Die Tatsache, daß in einer Anfrage alle Informationen geliefert werden können, ist sehr wichtig, um die Anzahl der HTTP-POST-Verbindungen zum DBAL möglichst klein zu halten, da der benötigte Aufwand für den Verbindungsaufbau im Vergleich zur Menge der gelieferten Daten in unserem Fall einen großen Overhead darstellt. Außerdem sinkt so die Belastung des WWW-Servers (Domino), was bei einer großen Anzahl der Clients durchaus eine Rolle spielen kann (theoretisch mögliche Anzahl der Projektmitglieder - 15 Kontextbereiche je acht Personen plus den zwanzig möglichen Gästen – liefert denkbare 140 Clients).

Im folgenden werden die gelieferten Datentypen und deren Bedeutung erläutert. Alle Daten werden beim Versand als Zeichenketten `x-www-form-urlencoded` übermittelt. Die jeweilige Instanz interpretiert diese jedoch immer als einen bestimmten Datentyp und führt entsprechende Umwandlungen nach dem Empfang durch. Der hier jeweils angeführte Datentyp entspricht also der logischen Sicht auf die Daten nach der Dekodierung und Konvertierung (s. auch die Beschreibung zu Stream/String-Decoder).

Bit 0: DatabaseInfo

- DatabaseDescription (String)

Der Name der Projektdatenbank

Bit 1: UserInfo

- UserName (String)

Der interne Name des eingeloggten Benutzers in der Projektdatenbank

Bit 2: KInfo

Durchs Setzen von Bit 3 liefert das DBAL die Informationen über alle verfügbaren Kontextbereiche. Die Anzahl der Bereiche wird in NumberOfKBs übermittelt.

- NumberOfKBs (integer)

Anzahl der Kontextbereiche ($\max(N) = \text{NumberOfKBs}$)

- NameKB[N] (String)

Name des N-ten Kontextbereichs

- PosKB[N] (String)

Position des N-ten Kontextbereichs

- ConKB[N] (15bit-Bitvektor)

Adjazenzvektor mit Wechselwirkungen des Kontextbereichs N:

$\text{ConKB}[N][j] = 1 \Leftrightarrow$ Es gibt eine Wechselwirkung zwischen Kontextbereichen N und j

Bit 3: High/Low Detail

- HighDetailOn (boolean)

Schalter für die Anzahl der benötigten Informationen in

PersonInfo (1 = HighDetail, 0 = Low Detail)

Bit 4..18 (= N+3): PersonInfo[K][N]

Durchs Setzen eines dieser Bits bekommt das GUI die Informationen über die Mitglieder eines der Kontextbereiche. Die Anzahl der Mitglieder wird in NumberOfPersons festgehalten.

- NumberOfPersons[N] (integer)

Die Anzahl der Personen im Kontextbereich N ($\max(K) = \text{NumberOfPersons}$)

- UserNamePerson[K][N] (String)

Der interne Benutzername des Benutzers Nr. K aus dem Kontextbereich N

- PicPerson[K][N] (String)

URL-Adresse des Bildes der Person K aus N

- NamePerson[K][N] (String)

Angezeigter Name der Person K aus N. Wird nur bei HighDetail ON übermittelt.

- `PosPerson[K][N]` (`Integer`)

Position der `PersonalCard` in der graphischen Darstellung der Mitglieder. Wird nur bei `HighDetail ON` übermittelt.

- `FunctionPerson[K][N]` (`String`)

Die Funktion der entsprechenden Person im Projekt. Wird nur bei `HighDetail ON` übermittelt.

- `ModPerson[K][N]` (`Boolean`)

Schalter mit `ModPerson[K][N] = 1 <=>` Person `K` aus dem Kontextbereich `N` ist Kontextbereichsmoderator im KBereich `N`. Wird nur bei `HighDetail ON` übermittelt.

Bit 19: GuestInfo[J]

`GuestInfo` hält die Informationen über alle Gäste des Projekts bereit. Gäste sind in keinem Kontextbereich Mitglieder; deren Anzahl wird in `NumberOfGuests` festgehalten.

- `NumberOfGuests` (`integer`)

Anzahl der insgesamt eingetragenen Projektgäste

- `GuestName[J]` (`String`)

Interner Projektdatenbankname des Gastes

Bit 20: ModeratorInfo[N]

`ModeratorInfo` stellt die Informationen über alle Moderatoren zur Verfügung. Diese Informationen sind zwar in den einzelnen Kontextbereichspersoneninformationen enthalten. Es hat sich jedoch als unpraktisch erwiesen, diese Informationen auf diesem Wege zu sammeln. Deswegen wurde noch ein weiteres Feld mit diesem Datentyp in die Schnittstelle integriert. Diese Informationen werden v.a. wegen Darstellungsunterschieden je nach verfügbaren Rechten vom GUI benötigt.

- `NumberOfMods` (`integer`)

Anzahl der Moderatoren (`max(J) = NumberOfMods`)

- `ModName[J]` (`String`)

Interner Name des Moderators des Kontextbereichs `N`

RequestedInfo:

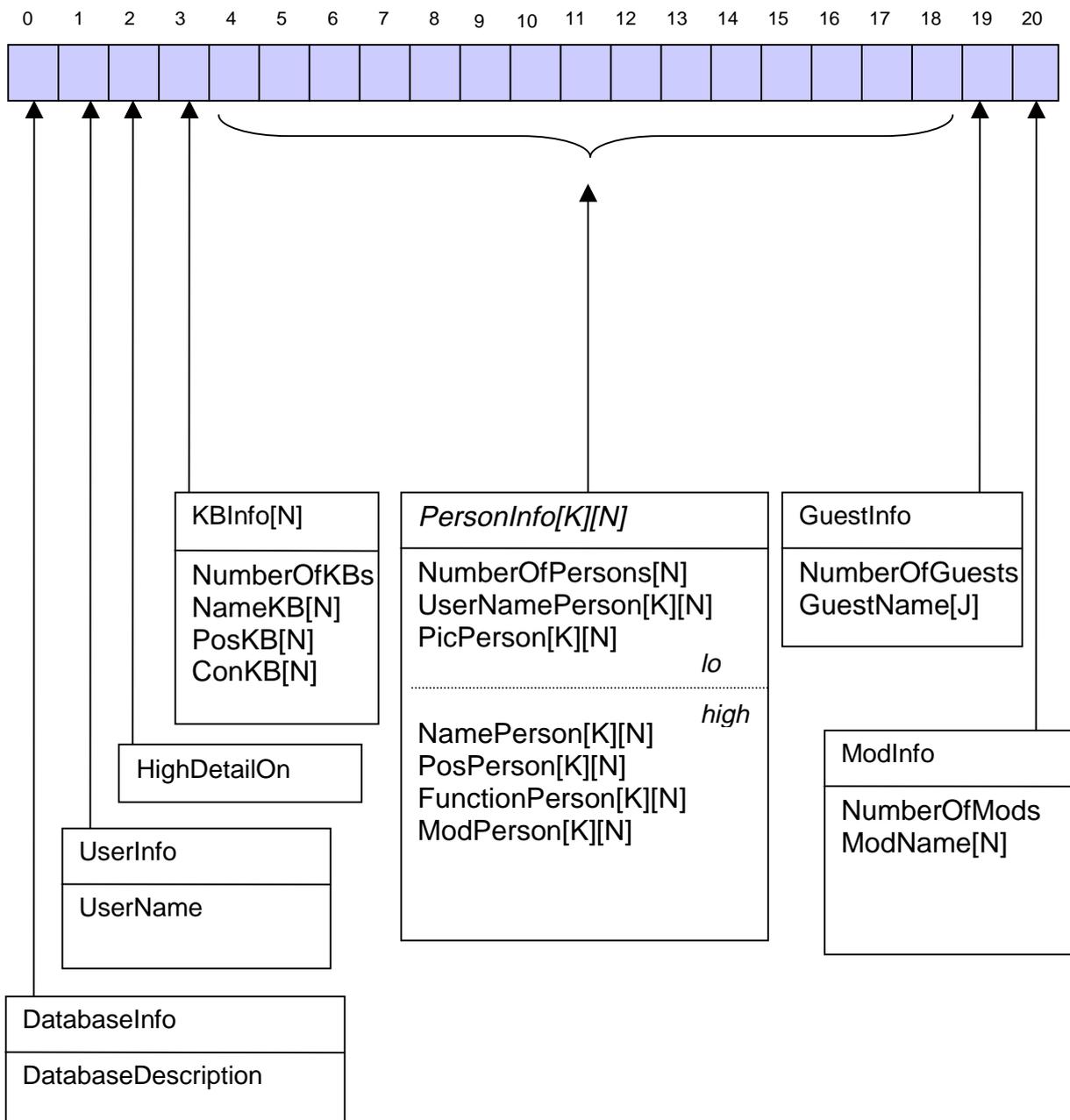


Abbildung 3-6

3.7 Modellierung und Rollen

Die bisherigen Überlegungen betrafen in erster Linie den lesenden Zugriff auf die Datenbank. Zwar kann dieselbe Kommunikationsgrundlage auch für den schreibenden / verändernden Zugriff benutzt werden. Jedoch muß ein ernstgemeintes verteiltes System aus Integritätsüberlegungen bestimmten Datenschutz v.a. bei Änderungen betreiben. Wir haben bereits ausgeführt (s. Abschnitt 3.2.2), daß bei dem verwendeten Konzept viele Sicherheitsfragen der zugrundeliegenden Projektdatenbank überlassen werden (u.a. die Authentifikation). Außerdem kann durch Übergang auf das Protokoll HTTPS eine Verschlüsselung für die Datenübertragung angewendet werden. Unter Annahme der Korrektheit der benutzten Projektdatenbank kann hier also von einer ausreichenden bzw. den Anforderungen entsprechender Sicherheit gesprochen werden.

Was bis jetzt allerdings nicht beachtet wurde, ist die Frage der systeminternen Sicherheit. Nicht jeder Benutzer eines Systems soll zwangsläufig die gleichen Möglichkeiten besitzen. In

unserem Fall, in dem eine virtuelle Organisation geschaffen wird, werden schon im Konzept verschiedene Benutzerrollen definiert und begründet. Diesen Rollen werden bestimmte Aufgaben und Rechte zugeordnet. Damit muß der Zugriff auf die Datenbank rollenabhängig eingeschränkt werden.

Die folgende Tabelle verdeutlicht die zu implementierenden Modellierungsfunktionen, sowie die zugehörigen betroffenen Rollen:

Nr	Aktion	Aktivator	Kommentar, Zustimmung	Initiator	Bedingung
1	Neuer Akteur	ProjM		Alle	noch nicht vorhanden
2	Akteur entfernen	ProjM		selbst, ProjM	kein KB mehr
3	Neuer KB + KBM ernennen	ProjM	KBM, KBM	Mod.	Platz vorhanden, Name neu
4	Neue WW + WWM ernennen	ProjM	WWM(neu), KBM, Planer	Mod. Der beiden KB	personelle Überlappung der betr. KBe
5	KBM ändern	ProjM	KBM, Planer, KBM(neu)	selbst, Mitgl.	ein Mitgl. des KBs wird neuer KBM
6	WWM ändern	ProjM	WWM(alt+neu), KBM, Planer	selbst, ProjM, KBMs	neuer ist gültig
7	Akteur nach KB	KBM	Planer, KBM, Planer(neu)	selbst, KBM, ProjM	noch nicht drin, #Personen(KB) < 8
8	Akteur aus KB	KBM	Planer, KBM, Planer(neu)	selbst, KBM, ProjM	kein WWM oder KBM

Legende:	
KB	Kontextbereich
WW	Wechselwirkung zwischen zwei KBs
KBM	Kontextbereichsmoderator
WWM	Wechselwirkungsmoderator
ProjM	Projektmoderator
Planer	Sonstiger Benutzer (keine Sonderrechte)
Selbst	die zu modifizierende Person

Diese acht Funktionen stellen eine Auswahl der möglichen Modellierungsaktivitäten dar, die durchaus erweiterbar wäre. Es gibt allerdings Funktionen, deren scheinbares Fehlen sofort ins Auge springt, wie z.B. Kontextbereich löschen oder Wechselwirkung löschen. Diese sind aber aus konzeptionellen Gründen nicht vorhanden. So müßte bei Wegfall eines Kontextbereichs über die darin vorhandenen Dokumente entschieden werden, was einen enormen Aufwand darstellen würde. Einzige denkbare aber nicht implementierte Möglichkeit wäre eine Deaktivierung des Kontextbereichs und Archivierung der Dokumente.

Auch die Liste der zu prüfenden Bedingungen wurde im Laufe des Projekts deutlich erweitert, zusätzliche Informationen diesbezüglich finden sich im Kapitel 5, Abschnitt „Die Modellierungsfunktionen“.

3.8 RequestedFct-Schnittstelle für die Übergabe von Modellierungsanforderungen

Bei der Übergabe der Modellierungsanfragen ergibt sich nicht das Problem, daß u.U. mehrere Anfragen zu einer gebündelt werden sollen. Daher wurde hier die gewünschte Funktion nicht in einem Bitvektor kodiert, sondern sie wird als Zahl direkt übermittelt. Je nach gewünschter Funktion ergeben sich dann weitere Felder, die in die Anfrage aufzunehmen sind.

Die Felder der Modellierungsanfrage im einzelnen:

- RequestedFct (Integer)

Die gewünschte Modellierungsfunktion (z.Z. 1-8).

- NameToAdd (String)

Name des neuen Akteurs, der dem Projekt oder einem Kontextbereich hinzugefügt werden soll (Funktion 1 und 7)..

- NameToDel (String)

Name des Akteurs, der aus dem Projekt oder aus einem Kontextbereich entfernt werden soll (Funktion 2 und 8).

- KBNameToAdd (String)

Der Name des neuen Kontextbereichs (nur Funktion 3).

- KBPos (Integer)

Die Position des Kontextbereichs. Diese Position entscheidet darüber, wo ein neuer Kontextbereich in Zukunft im Projektnavigator erscheint (nur Funktion 3)

- ModName (String)

Name des neuen Kontextbereichs- oder Wechselwirkungsmoderators, bei Erzeugung eines neuen Kontextbereichs oder einer Wechselwirkung. Dieser muß bei der Erstellung mitgeliefert werden, da kein Kontextbereich bzw. Wechselwirkung ohne Moderator existieren darf (Funktionen 3 und 4).

- LinkKB1, LinkKB2 (Integer)

Nummern der Kontextbereiche, die verknüpft werden sollen, bzw. deren Wechselwirkungsmoderator geändert werden soll (Funktionen 4 und 6)

- KBName (Integer)

Die Nummer des Kontextbereichs, dessen Moderator geändert werden soll (nur Funktion 5).

- NameToSet (String)

Name der Person, die neuer Kontextbereichs- oder Wechselwirkungsmoderator werden soll (Funktionen 5 und 6)

- AddToKB (Integer)

Nummer des Kontextbereichs, zu dem ein neues Mitglied hinzugefügt werden soll (nur Funktion 7)

- DelFromKB (Integer)

Nummer des Kontextbereichs, aus dem ein Mitglied entfernt werden soll (nur Funktion 8)

- Reason (String)

Bei allen Modellierungsanfragen kann dieser Parameter übergeben werden, um eine Begründung zu übermitteln, die in den Aktivierungsantrag aufgenommen wird.

4 Client

Dieses Kapitel beschreibt den im Rahmen der Studienarbeit entwickelten Client für den Zugriff auf eine Projektdatenbank über das Web im Sinne der vorausgegangenen Kapitel. Die Web-Fähigkeit bzw. das vorher beschriebene Konzept setzen Sun's Java™ als Programmiersprache voraus. Der am ifib/ Universität Karlsruhe(TH) entwickelte Client kann derzeit unter <http://ifib46.ifib.uni-karlsruhe.de:8080/studienarbeit.nsf> getestet werden. Vorausgesetzt wird ein Java™-fähiger Browser mit vollständig implementiertem JDK 1.1 – API (z.B. Netscape Communicator 4.5 (Win32 oder Unix-Version) oder MS Internet Explorer ab 4.0)

4.1 Einführung

Hier sollen sowohl die konzeptuellen Lösungen zu den entstandenen Problemen auf der Clientseite als auch die implementierungsnahen Details erklärt werden. Für das bessere Verständnis der Zusammenhänge und des zugrundeliegenden Systems der projektnahen partizipativen Modellierbarkeit von Herrn Dipl. Ing. Christian Müller sei sowohl auf die Testmöglichkeiten als auch auf die Doktorarbeit von C. Müller hingewiesen.

4.1.1 Aufbau der Beschreibung

Um dem Leser eine bessere Übersicht zu ermöglichen, ist dieses Kapitel in vier wichtige Abschnitte unterteilt. Als erstes wird die Kommunikationsschnittstelle des Clients im Abschnitt 4.2 erklärt. Diese bietet eine unabhängige Zugriffsmöglichkeit für alle übrigen Clientkomponenten und bildet auf diese Weise gewissermaßen einen notwendigen Unterbau. Als nächstes wird auf den reinen Darstellungsmodus näher eingegangen (4.3). Dabei werden die Probleme und Lösungen diskutiert, die bei der Nachahmung des starren Prototypen im Bezug auf die Darstellung und deren Implementierung aufgetreten sind. Dies entspricht außerdem der ersten großen Aufgabe dieser Studienarbeit: Die dynamische Datenrepräsentation der Strukturdaten aus der Projektdatenbank stand am Anfang im Vordergrund. Erst nach dem Abschluß dieses Teils der Arbeit wurden die Modellierungsfunktionen und die entsprechenden Benutzerdialoge in den Client-Teil eingebaut. Mit diesen Dialogen beschäftigt sich der Abschnitt 4.4. Zum Schluß werden im Abschnitt 4.5 das eigentliche Applet und die allgemeinen Hilfsklassen beschrieben.

4.1.2 Verwendete Abkürzungen und Bezeichnungen

- Client, GUI

Das hier beschriebene Java-Applet, `GUI.class`

- DBAL (Database abstraction layer), Server-Teil

Der Kommunikationspartner und das Datenbankfrontend des Clients. Alle Datenbankzugriffe d.h. auch die gesamte Kommunikation werden ausschließlich über das DBAL abgewickelt. Der Client benutzt niemals die tatsächliche Projektdatenbank, was dem im Kapitel 3. vorgestellten Konzept entspricht.

- RTS (Runtime System)

Die Laufzeitumgebung.

- Prototyp

Die starre, statische (datenbankunabhängige) Darstellung der Projektstruktur mit Macromedia's Flash© und HTML, die zum Anfang der Studienarbeit als Vorlage für das Design und die Konzeption diente.

4.2 Kommunikation

Wie im Kapitel 3 Abschnitt 3.2ff bereits erwähnt, wird die gesamte Kommunikation im Rahmen des HTTP-Protokolls durch POST-Anfragen erledigt. Der Client übermittelt dabei die gewünschte Anfrage - `RequestedInfo` oder `RequestedFct` - an `DBAL`, welches eine Antwort generiert und zurückliefert. Da solche HTTP-Anfragen ineffizient sind, v.a. wenn viele kleine Datenmengen übertragen werden, wurden verschiedene Maßnahmen getroffen, um die Anzahl der Anfragen zu minimieren. Diese sind in der `Cache`-Klasse zusammengefaßt. Der Anfragemechanismus an sich erfordert routinemäßige Vorgehensweise zum Senden und Empfangen von Daten. Da diese Routinen in sich abgeschlossen sind, wurden diese in eine separate `JavaPOST`-Klasse ausgelagert. Eine `Decoder`-Klasse (vgl. Abschnitte 3.4, 3.5) kümmert sich um die Rückgewinnung der Daten, so daß ein logisches Schichtenmodell erkennbar wird, welches beim Empfang greift (Abbildung 4-1):

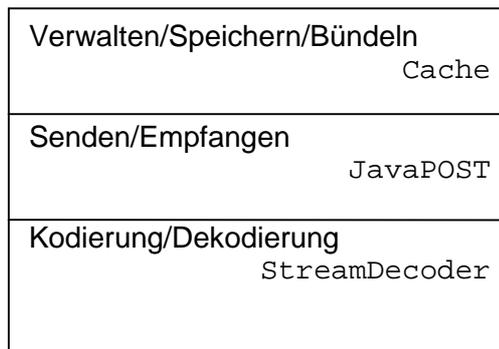


Abbildung 4-1

Der Datenstrom wird gleich im Dekoder in eine Hashtabelle umgewandelt, weil dieser Datentyp in Java™ bequem und schnell verwaltet werden kann. Diese Hash-Tabelle wird zwischen den Schichten ausgetauscht.

4.2.1 *StreamDecoder*-Klasse

Die POST-Formate erfordern eine Kodierung, um Steuerzeichen und Worttrennzeichen von den verwendeten Zeichen in den Werten der Variablen zu unterscheiden (vgl. auch Kap. 3.3). Dafür muß eine Kodierung/Dekodierung des Datenstromes durchgeführt werden.

Das Java™ 1.1 API steuert einen entsprechenden Kodierungsmechanismus bei: Die statische Methode `encode(String)` der `URLEncoder`-Klasse kodiert die Daten in `x-www-url-encoded` - Format, das auch von den WWW-Browsern verwendet wird. Leider fehlt jedoch eine korrespondierende Dekodieroutine. Diese wurde als Ableitung des allgemeinen `Decoder`-Interfaces von R. Rodewald (vgl. Kap. 5) programmiert.

`StreamDecoder`:

Benutzt:

- `Decoder::decode(String)`

Methoden:

- `StreamDecoder(InputStream)`:

Standardkonstruktor zum Erzeugen eines `StreamDecoder`-Objekts auf einem Eingabestrom.

- `StreamDecoder(InputStream, Hashtable)`:

Wie oben, es wird jedoch eine vom Benutzer übergebene `Hashtable` verwendet und keine neue erzeugt.

- `Hashtable getHashtable()`:
Dekodiert den Eingabestrom und schreibt die Daten in eine Hashtabelle. Die aus dem POST-Strom extrahierten Variablennamen fungieren dabei als Schlüssel und der Wert dieser Variablen als das gehashte Datum.

UML:

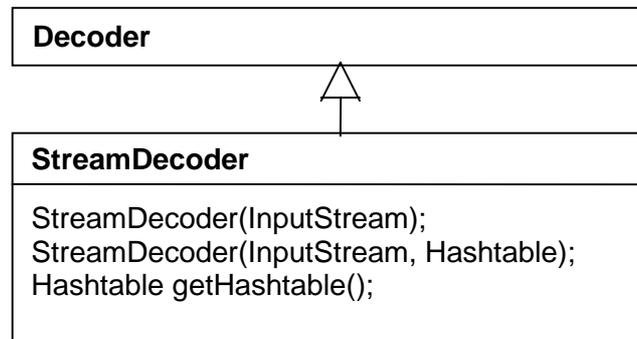


Abbildung 4-2

4.2.2 JavaPOST-Klasse

Diese Klasse kapselt aus der Appletsicht den kompletten Verbindungsaufbau für die POST-Übermittlung und die Methoden zum anschließenden Senden und Empfang von Daten. Die Klasse stellt allgemein gesehen eine unabhängige Schnittstelle bereit, mit der beliebige Java-Programme an beliebige URL-Adressen ihre Daten per POST übertragen können. Die Verbindung muß – dem HTTP-Protokoll und der POST-Methode entsprechend – nach jedem vollen Sende/Empfangszyklus reinitialisiert werden, da diese nach der Übermittlung des letzten angeforderten Datenzeichens vom Server geschlossen wird.

JavaPOST:

Benutzt:

- `StreamDecoder::StreamDecoder(InputStream, Hashtable);`
- `StreamDecoder::Hashtable getHashtable();`

Methoden:

- `JavaPOST(URL):`

Konstruktor: Erzeugt ein Objekt und vermerkt die übergebene URL-Adresse als POST-Kommunikationsadresse für spätere Verbindungen. Ruft anschließend `init()` auf.

- `JavaPOST(String):`

Konstruktor: Erzeugt ein Objekt, versucht aus der übergebenen Zeichenkette eine URL-Adresse zu extrahieren, mit der die später aufgerufenen Methoden kommunizieren. Macht eine Verbindung mit `init()` auf.

- `void init():`

Verbindet zu der im Konstruktor angegebenen Adresse und setzt die Eigenschaften der Verbindung wie folgt (Initialisierung):

- Unterstütze bei dieser Verbindung sowohl Datenausgabe als auch -eingabe (benötigt von Java-RTS)
- benutze keine (Browser-)Caches für diese Verbindung
- Setze den Inhaltstyp auf `application/x-www-urlencoded`

- `void send(String):`
Schreibt den übergebenen String auf die geöffnete Verbindung
- `Hashtable get(Hashtable):`
Liest von der offenen Verbindung die Daten ein, dekodiert und konvertiert diese in eine Hash-Tabelle mit Hilfe von `StreamDecoder`-Methoden, liefert die so veränderte Hash-Tabelle zurück. Ist die übergebene Tabelle ein `Null`-Zeiger, so wird eine neue durch die `StreamDecoder`-Klasse erzeugt. Anschließend wird die Verbindung (durch den Server) geschlossen.

UML:

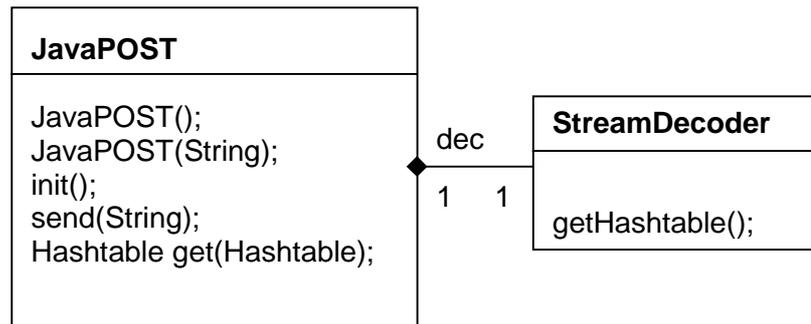


Abbildung 4-3

4.2.3 Cache-Konzept und Cache-Klasse

Die Cache-Klasse bildet die höchste Schicht des Kommunikationsteils des Applets und bietet dem Benutzer bequeme und transparente Zugriffsmöglichkeiten auf das DBAL-Modul. Die grundsätzliche Funktionalität eines Zwischenspeichers wird erweitert sowohl durch die Bündelungsmöglichkeiten, die ebenfalls zur Anfrageminimierung beitragen, als auch durch das Zusammenfassen und Auslagern der kompletten RequestedInfo-Schnittstelle. So waren die Erweiterungen der Schnittstelle während der Implementierung unproblematisch und übersichtlich durchzuführen. Z.B. wurde die Gästebuchauskunft und die Moderatoreninformationen erst später mit in diese Schnittstelle übernommen, weil einerseits eine zusätzliche, beim Prototypen nicht vorhandene Ansicht hinzukam, und andererseits eine separate Moderatorenliste den Implementierungsaufwand reduzierte. Außerdem ist es mit dieser Klasse grundsätzlich möglich geworden, einem Dritten alle Lesefunktionen auf dem DBAL zu ermöglichen, ohne daß die komplette RequestedInfo-Bitbelegung bekannt sein muß.

Die Bündelung der Anfragen geschieht durch zwei getrennte Methoden dieser Klasse. Mit der ersten läßt sich der gewünschte Schlüsselwert zu den abzufragenden Werten hinzufügen, ohne daß tatsächlich Informationen ausgetauscht werden. Bei aktiviertem Cache überprüft die Funktion zunächst, ob der Wert nicht bereits lokal vorliegt und fügt diesen in dem Fall nicht zu den zu holenden Schlüsseln hinzu. Die zweite Methode holt alle wartenden Schlüssel in einer Anfrage übers Netz.

Außer diesen Methoden wird noch eine universelle Methode zum Bereitstellen eines Schlüsselwertes angeboten. Diese Methode stellt sicher, daß nach deren Aufruf entweder die entsprechenden Fehler gesetzt sind, oder der benötigte Wert geliefert wird – ob dieser aus dem Cache stammt oder übers Netz geholt wird, bleibt dem Benutzer vorenthalten.

Die Fehlerbehandlung dieser Klasse wurde sehr passiv gestaltet: Die erkannten Fehler werden in zwei statischen Zustandsvariablen – einer Fehlererkennung und einer Fehlerbeschreibung – von jeder Methode nach außen sichtbar gemacht. Dabei werden auch die DBAL-Fehlermeldungen berücksichtigt und in derselben Form gemeldet. Die Methoden dieser Klasse selbst jedoch überprüfen diese Variablen nicht. Will der Benutzer wissen, ob ein

Fehler aufgetreten ist, so kann er diese öffentlichen Variablen herauslesen und anschließend mit beliebigen Werten füllen, die er als fehlerfrei ansieht. Dieses passive Fehlererkennungskonzept wurde gewählt, um dem Applet die Möglichkeit zu geben, bei Bedarf in einen Fehlerzustand zu wechseln.

Cache:

Benutzt:

- JavaPOST – alle Routinen

Variablen:

- `static int ERRORCODE, static String ERRORSTRING:`
Fehlervariablen, die im Fehlerfall wie folgt gesetzt werden:

ERRORCODE	ERRORSTRING	Bedeutung
< 10	„DBAL meldet Fehler Nr.“	DBAL meldet einen Fehler mit der gemeldeten Nummer und der angegebenen Bedeutung
10	„Unbekannter Schlüssel ...“	Der Schlüsselname wurden nicht in der RequestedInfo-Schnittstelle definiert
11	„DBAL liefert nicht den Inhalt des Schlüssels ...“	Obwohl der Schlüssel korrekt erscheint, liefert DBAL keine Informationen
12	„Leerer Schlüssel“	Die übergebene Zeichenkette war leer
13	„Keine Verbindung zu DBAL möglich“	JavaPOST meldet, daß die Verbindung zur angegebenen URL scheitert
99	„DBAL liefert eine ungültige Antwort (ErrorCode gesetzt aber nicht lesbar)“	DBAL setzte ErrorCode auf einen ungültigen Wert

Benutzer muß sich um die Rücksetzung der Variablen kümmern. Diese Werte werden intern nicht benötigt.

Methoden:

- `Cache(URL, int):`

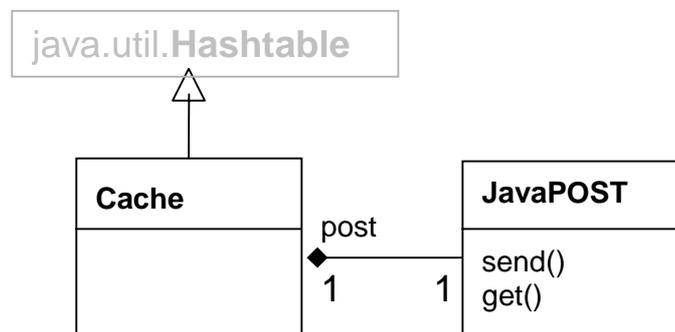
Konstruktor: Erzeugt einen Cache mit der angegebenen Größe, merkt die angegebene URL-Adresse als DBAL-Kommunikationsadresse bei späteren Verbindungsaufbauversuchen.

- `String getKey(String):`

Holt den in der Zeichenkette angegebenen Schlüsselwert vom DBAL, wenn dieser noch nicht im Cache ist. Im Fehlerfall (falscher Schlüssel, falsche DBAL - Adresse, Verbindungsfehler, etc.) werden `ERRORCODE` und `ERRORSTRING` mit der Fehlernummer und Fehlerbeschreibung gefüllt.

- `addRequestFor(String)`:
Fügt den angegebenen Schlüssel zu der zu startenden Anfrage zu, falls der Schlüssel nicht bereits lokal vorhanden ist, führt jedoch keine DBAL - Abfrage durch. Überprüft ebenfalls die Korrektheit des Schlüssels (`ERRORCODE`, `ERRORSTRING`)
- `getAllRequested()`:
Holt alle mit `addRequestFor(String)` in die Warteschlange gestellten Schlüssel vom DBAL.
- `void reload()`:
Alle im Cache vorhandenen Schlüsselwerte werden vom DBAL aufs Neue geholt.

UML:



4.3 Darstellungsmodus

4.3.1 Analyse des Prototyps

Die grundsätzliche Anforderung, den vorhandenen Prototypen dynamisch zu implementieren, erforderte einen zusätzlichen Aufwand für die Analyse des Prototypen im Bezug auf das Applet-Konzept. Von Anfang an war klar, daß bestimmte Prototypmöglichkeiten, die wegen dessen HTML-Seiten-Struktur natürlich gegeben waren, in irgendeiner Form in das bestehende Konzept integriert werden mußten. Dabei zeichneten sich zunächst folgende Schwierigkeiten ab:

- Der Benutzer hat die Möglichkeit, jede Prototypseite als Bookmark abzulegen und hat somit einen direkten Zugriff auf *jede* Projektebene, wenn gewünscht. Diese Funktion ist mit einem monolithischen Applet nicht vorhanden, da dieses nur interne Ansichten umschaltet.
- Für den Benutzer macht es keinen Unterschied, ob er die zum Prototyp gehörenden Flash®-Seiten oder beliebige externe Seiten anschaut. Ein Wechsel zwischen diesen wird vom Browser vollzogen und kann jederzeit rückgängig gemacht werden. Auch ein „Backlink“ von der externen Seite auf eine der Prototyp-Seiten ist möglich. Ein Applet kann externe Seiten zwar anzeigen, dabei wird dieses jedoch gestoppt. Ein „Backlink“ würde das Applet zwangsläufig in den Initialzustand versetzen. Die Betätigung des Browser-Backbuttons wäre denkbar; da jedoch z.T. die vom Domino®-Server generierten Ansichten angezeigt werden mußten, auf denen bereits „Backlinks“ vorhanden waren, war auch diese Funktionalität erwünscht. Außerdem war es geplant, die Symbolleiste des Browsers später komplett auszuschalten, um eine explizite Projektnavigation zu erzwingen.

Bei einer tiefergehenden Betrachtung des Teils des Prototyps, welcher mit Animationen und erwünschter Appletfunktionalität (Datenbankabhängigkeit!) nachgeahmt werden mußte, weil er nicht durch reine HTML-Seiten zufriedenstellend dargestellt werden konnte, konnten fünf

notwendige Einstiegspunkte identifiziert werden. Einstiegspunkte sind dabei im Sinne der angesprochenen Schwierigkeiten zu definieren – der Benutzer kann einen direkten Link zu einer dieser Seiten bilden und somit direkt auf dieser Ebene in das Projekt einsteigen. Der Rest wurde als externe (in diesem Fall Lotus Domino-generierte) Seiten aufgefaßt. Damit war klar, daß das zukünftige Applet auf irgendeine Weise mindestens fünf reproduzierbare Startzustände vereinen muß, deren Aussehen vom Zustand der Projektdatenbank und dem angemeldeten Benutzer abhängig sind.

Alle weiteren Prototypfunktionen, v.a. die reiner Darstellungsnatur, wie z.B. die Hover-Effekte der Buttons oder die Darstellung der JPEG- und GIF-Bilder konnten dagegen z.T. zwar mit nur mäßigem Erfolg aber dafür recht problemlos mit „Bordmitteln“ von Java nachgeahmt werden. Der mäßige Erfolg bezieht sich dabei auf die Qualität der Text- und Liniendarstellungen der Standardgrafikbibliothek von Java, die den speziell dafür entwickelten Werkzeugen bzw. Browser-PlugIns wie Macromedia's Flash® unterlegen ist (z.B. Antialiasing).

4.3.2 Zustandsaufteilung und zusätzlicher Fehlerzustand

Eine weitere Betrachtung des gesamten Prototypen (d.h. einschließlich der in diesem Fall von Lotus Domino® erzeugten HTML-Seiten) im Hinblick auf das Zusammenspiel der identifizierten Zustände, deren Eingrenzung und mögliche Zustandswechsel lieferte den folgenden Zustandsautomaten (Abbildung 4-4), wobei alle internen Zustände auch gleichzeitig Startzustände sind:

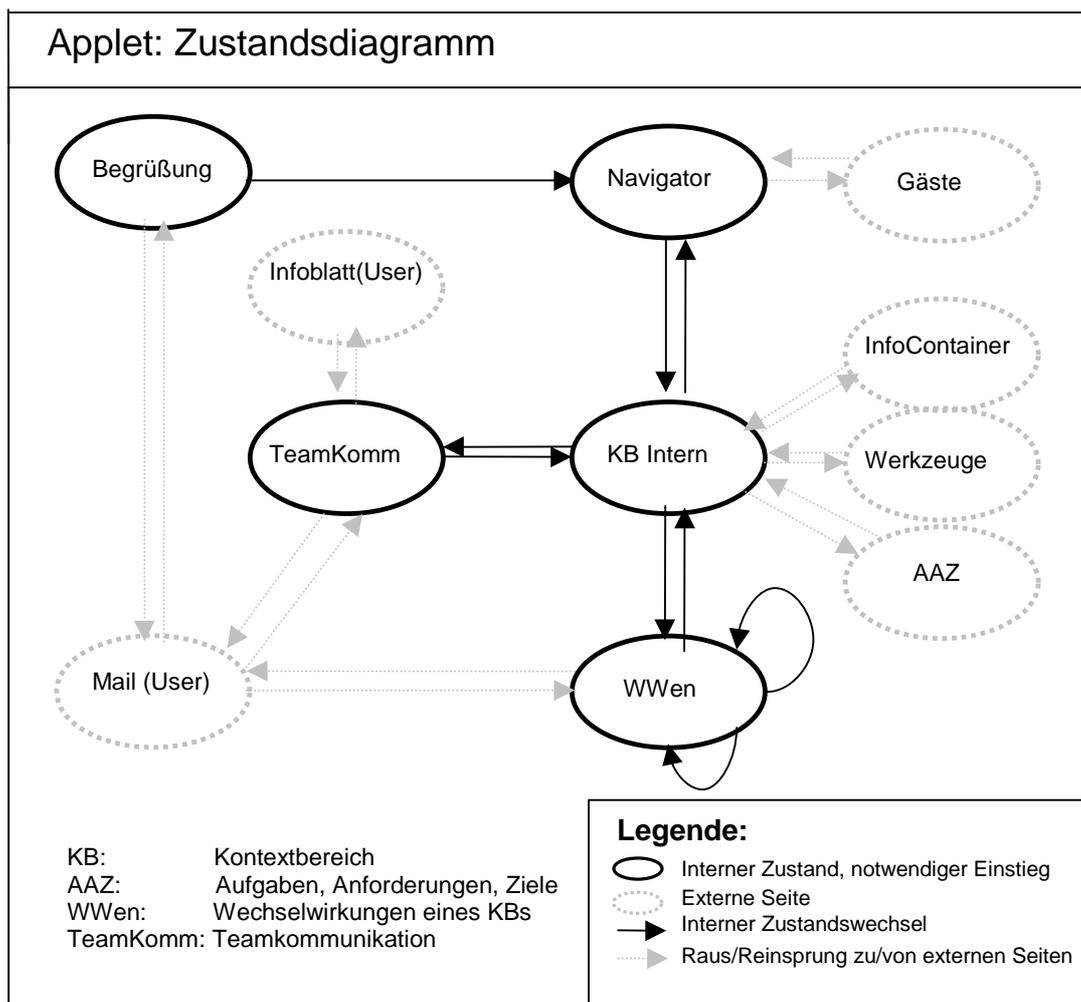


Abbildung 4-4

Da die Datendarstellung jedoch dynamisch im Zusammenspiel mit DBAL über das Netzwerk erfolgen sollte, wurde später noch ein Fehlerzustand hinzugefügt, um einerseits dem Benutzer die Fehlermeldungen bei Verbindungsproblemen und/oder fehlendem oder falschem DBAL anzeigen zu können und andererseits den Client in einen Endzustand zu versetzen: Es ist nämlich grundsätzlich nicht möglich, ein Applet im Browser von sich aus komplett zu beenden. Im Falle eines Fehlers würde der Browser die Appletausführung fortführen und auf weitere Benutzeraktionen auf einer nunmehr falschen Basis reagieren, was im schlimmsten Fall zum Browserabsturz (theoretisch unmöglich, praktisch jedoch oft vorgekommen) oder sogar Datenverlust führen könnte.

Damit mußte der zusätzliche Fehlerzustand v.a. folgende Eigenschaften aufweisen:

- Anzeige der Art und u.U. der Ursache des aufgetretenen Fehlers für den Benutzer
- Anzeige von *keinen* projektdatenbankabhängigen Informationen, da ja ein Verbindungsversuch zu dem Fehler geführt haben könnte
- Kein Rücksprung zu irgendeinem weiteren Appletzustand möglich (Fehlerzustand = logischer Endzustand)

Die konkrete Implementierung des Fehlerzustands erreicht diese Vorgaben dadurch, daß außer der Fehleranzeige keine weiteren Buttons auf dem Bildschirm dargestellt werden. Ein mailto-Link als einzig mögliche Benutzeraktion erstellt dabei mit Browsermitteln einen Fehlermailreport, in dem die aktuelle Fehlermeldung im Text eingeblendet wird und auf Knopfdruck per eMail an das Entwicklerteam geschickt werden kann.

Im folgenden sollen auch die weiteren Zustände und deren Funktionen kurz erläutert werden.

- **Begrüßung – initState** (Abbildung 4-5)

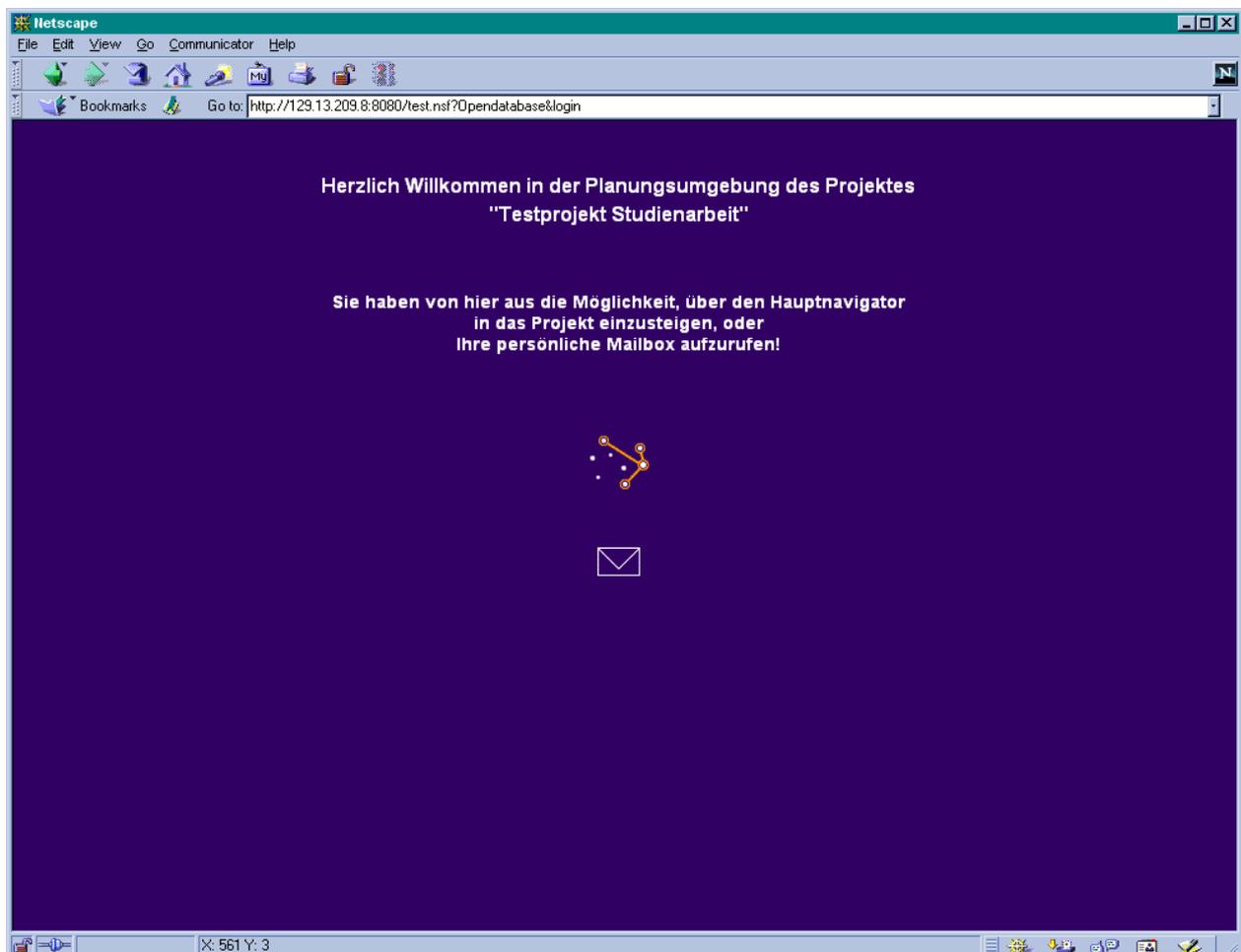


Abbildung 4-5

Dieser Zustand bietet gerade für neue Benutzer einen angenehmen Einstieg in die Projektumgebung. Die dynamischen Informationen sind dabei der eingeblendete Name der Projektdatenbank und der Maillink, der den DBAL-Mailagenten mit dem aktuellen Benutzer im Parameterstring als externe Seite aufruft. Das andere Bild ist ebenfalls anklickbar und bietet einen Zustandsübergang zum Projektnavigator, erzwingt also einen internen Zustandswechsel.

- Navigator – NavigatorState (Abbildung 4-6)

Der Projektnavigator ist die logische Abbildung der Projektorganisationsstruktur und seiner Gliederung in einzelne Kontextbereiche. Die verschiedenen Kontextbereiche werden durch weiße Punkte markiert, deren Größen von der Anzahl ihrer Wechselwirkungen mit anderen Kontextbereichen (dargestellt durch orange und rote Verbindungslinien zwischen den Punkten) abhängen. Jeder solche Punkt leuchtet mitsamt aller mit ihm verbundenen Kontextbereiche beim Anfahren mit der Maus auf (s. Beispiel im Bild) und führt beim Anklicken einen Sprung in das Innere des jeweiligen Kontextbereichs durch (interner Zustandswechsel). Alle Informationen, welche die Kontextbereiche betreffen, werden dynamisch über die ReqInfo-Schnittstelle abgefragt und vom DBAL aus der Datenbank geliefert. Dazu gehören die Namen, die Positionen und die Anzahl und Art der Verbindungen der einzelnen Elemente. Die Wechselwirkungen zum Projektmoderationsknoten (der erste, in jedem Projekt vorhandene Kontextbereich) sind dabei organisatorischer Natur und bestehen zwangsläufig, da jeder Kontextbereich moderiert werden kann. Diese Wechselwirkungen werden deswegen anders als eine normale Verbindung rot gezeichnet. Die oberen drei Symbole links im Bild sind die Schaltflächen verschiedener Modellierungsfunktionen, diese werden im Kapitel 4.3 ausführlich behandelt. Das Symbol unten links ist die Gästeansicht, also ein externer Link.

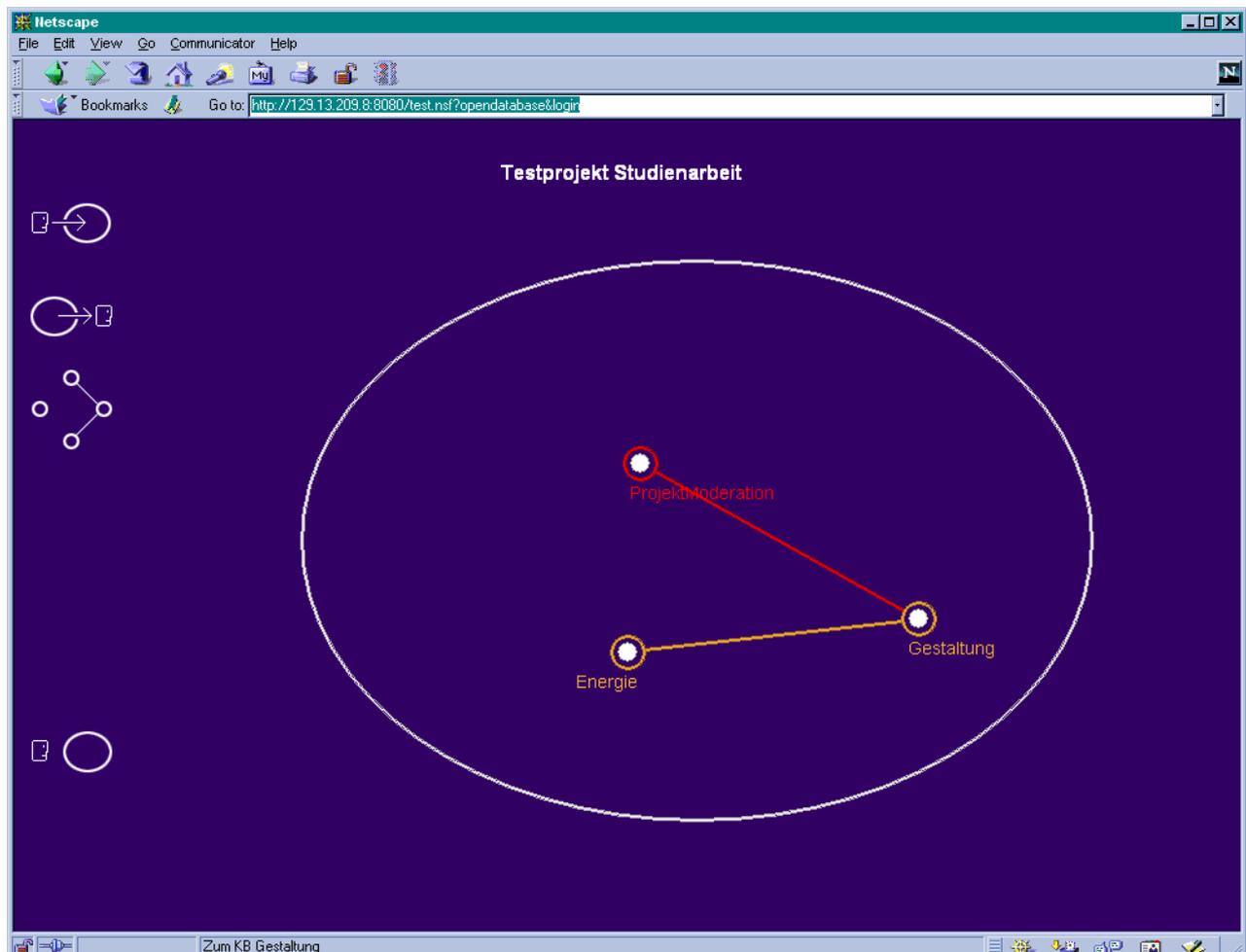


Abbildung 4-6

- KB intern – KBState (Abbildung 4-7)

Der Benutzer befindet sich hier im Inneren eines Kontextbereichs, dessen Name in der Mitte des Bildes angezeigt wird. Ihm stehen nun verschiedene Möglichkeiten zur Verfügung. Die Bilder leuchten beim Anfahren mit der Maus auf, und es wird ein Hilfstext in der Statuszeile eingeblendet. Das im Moment aktivierte Bild ganz oben führt beim Anklicken den Sprung zu den Kontextbereichsmitgliedern (TeamKommunikation, interner Zustandswechsel) durch. Das sternförmige Bild rechts davon ist der Sprung zu den Wechselwirkungen dieses Kontextbereichs (Wechselwirkungen, interner Zustandswechsel). Die Schaltfläche mit der Bezeichnung „Ressourcen“ wurde aus Kompatibilitätsgründen übernommen, hat aber zur Zeit keine Funktionalität. Die restlichen drei Bilder sind externe dynamisch generierte Links (Kontextbereichsnummer als Parameter) zu den verschiedenen Ansichten: „Werkzeuge“, „Aufgaben/Anforderungen/Ziele“ oder „Informationscontainer“ stellen Funktionen zur Verfügung, mit denen die Projektmitarbeit unterstützt werden soll. Schließlich hat der Benutzer die Möglichkeit, durch Betätigung des BACK-Feldes (Kreuz unten links), zum Projektnavigator zurückzukehren (interner Zustandswechsel).

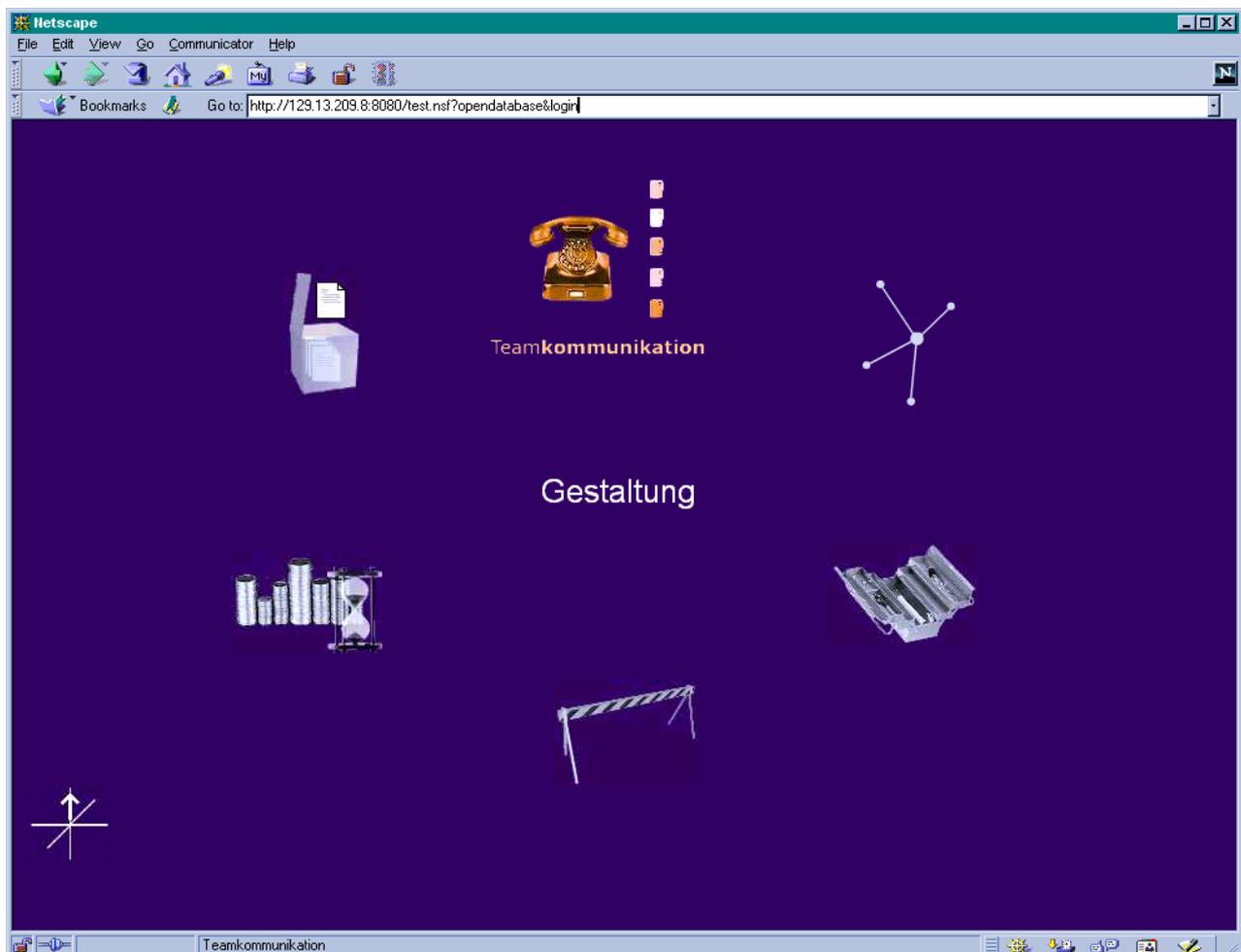


Abbildung 4-7

- TeamKommunikation – TeamCommState (Abbildung 4-8)

Die Teamkommunikation stellt alle Mitglieder eines bestimmten Kontextbereichs vor, indem hier alle verfügbaren Informationen in Form sogenannter „CommuniCards“ dargestellt werden. Die nach der Theorie maximal acht Mitgliederkarten enthalten die Informationen über den Namen, die Projektfunktion, ein Photo des jeweiligen Mitglieds und einen dynamischen Maillink (Name des vorgestellten Benutzers als Parameter) zu der Mailbox dieses Benutzers (externe Ansicht). Das orange hervorgehobene „M“ kennzeichnet den Kontextbereichsmode-

rator, der in jedem Kontextbereich existieren muß. Der Benutzer hat außerdem die Möglichkeit, über eine weitere externe Seite, deren Link ebenfalls dynamisch generiert wird, die weiterführenden Informationen über eine bestimmte Person einzusehen, bzw. seine eigenen zu verändern.

In der Mitte des Bildes erscheint der Name des aktuellen Kontextbereichs. Unten rechts ist der gewohnte Rücksprung, der diesmal zurück zum „KB Intern“ führt (interner Zustandswechsel). Die Bilder darüber sind wiederum verschiedene kontextbereichsspezifische Modellierungsfunktionen, deren Bedeutung später erklärt wird.

Beinah jede Information in diesem Zustand kommt über das Netzwerk, wird vom DBAL generiert und entspricht damit dem Inhalt der Datenbank.

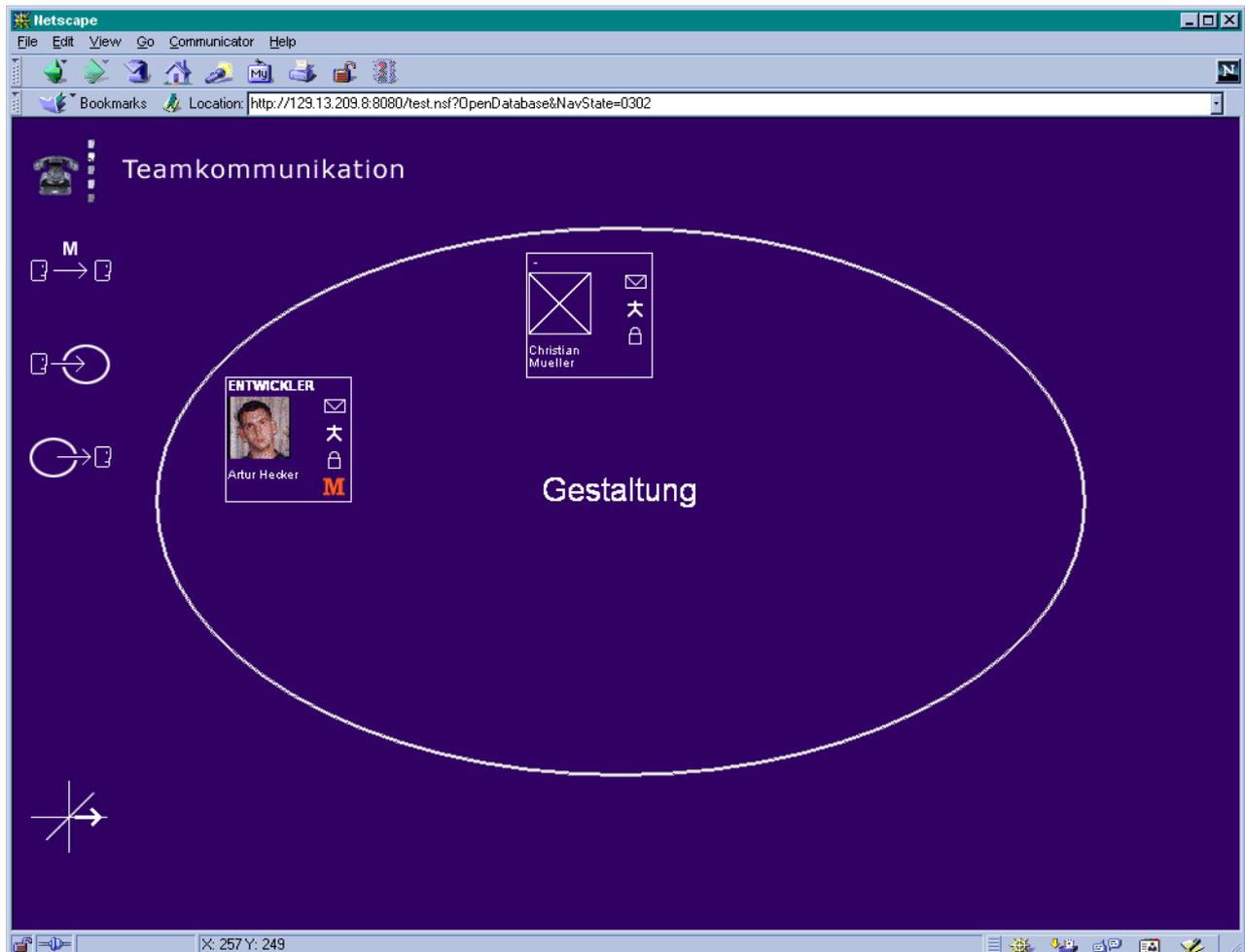


Abbildung 4-8

- Wechselwirkungen eines Kontextbereichs – `wwState` (Abbildung 4-9)

Dieser Zustand zeigt und verwaltet die Wechselwirkungen eines Kontextbereichs. Angezeigt werden dabei die Photos aller Mitglieder der verbundenen Kontextbereiche, die gleichzeitig als Links zu den Mailboxen der Mitglieder dienen, als auch interne Wechselmöglichkeiten in die entsprechenden Zustände der verbundenen Bereiche. Dabei können acht verbundene Kontextbereiche auf einmal angezeigt werden. Sind mehr Kontextbereiche verbunden, so werden entsprechende Links eingeblendet, die zu den weiteren Wechselwirkungen führen. Dieser Mechanismus ist durch einen Zustandswechsel zum selben Zustand (`wwState`) und Umschaltung der appletinternen Variablen implementiert. Außer den erwähnten Verbindungen werden noch der Name des momentanen Kontextbereichs, die Buttons zum Aufruf der Modellierungsfunktionen und der gewohnte Back-Button angezeigt.

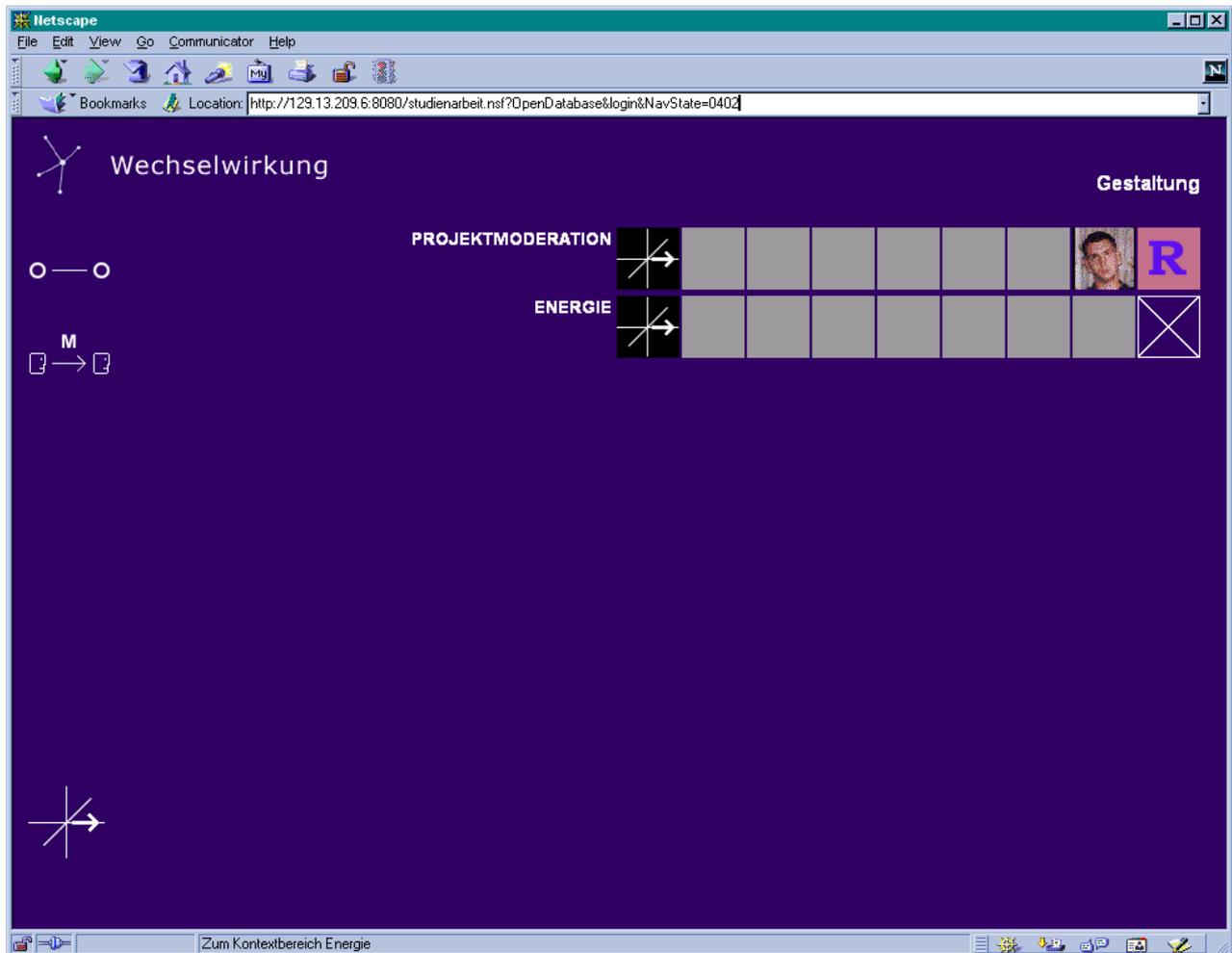


Abbildung 4-9

Auf diese Weise wurde die Zustandsaufteilung eindeutig festgelegt. Die Übergänge zwischen den Zuständen werden dabei durch die Benutzermausaktionen (`MouseClicked`-Methode des `MouseListener`-Interfaces bei Java) ausgelöst. Damit mußte jeder Zustand auf irgendeine Weise die Mausaktionen verfolgen können. Bedingt durch die Notwendigkeit weiterer mausgesteuerter Aktivitäten, wie z.B. die aufleuchtenden Verbindungen zwischen den Kontextbereichen und etc., benötigten manche Zustände hingegen weiterführende Mausbewegungsereignisse (`MouseMoved`-Methode des `MouseMotionListener`-Interfaces bei Java). Die Reaktionen auf letztere waren dagegen so unterschiedlich, daß es kaum möglich erschien, sie alle in einer Applet-Methode zu implementieren.

4.3.3 Implementierungsproblem, Zustandsvaterkonzept und -klasse

Das gerade erwähnte Problem mit der Implementierung der `MouseListener`-Methoden läßt sich noch verallgemeinern. Ein Java-Applet unterliegt bestimmten Gesetzmäßigkeiten und hat schon einen festgelegten Lebenszyklus. Das Applet wird durch den Aufruf seiner `init()` – Methode gestartet, wechselt nach der Ausführung automatisch in die `start()` – Methode und wird anschließend vom Browser so gesteuert, daß bei Minimierung des Browser-Fensters, Laden einer anderen Seite, usw. das Applet gestoppt wird. Es wird also die `stop()` – Methode des Applets aufgerufen. Auf diese Weise hat der Programmierer die Möglichkeit, etwaige Animationen oder rechenintensive Anwendungen kontrolliert anzuhalten. Der Browser führt bei der anschließenden Maximierung bzw. Neuladen der Seite oder Betätigung des BACK-Buttons den Aufruf der `start()` – Methode durch. So pendelt das Applet zwischen Zuständen, bis es endgültig beendet werden soll (Browserfenster wird geschlossen, etc.). In diesem Fall ruft der Browser die `destroy()` – Methode auf (Abbildung 4-10).

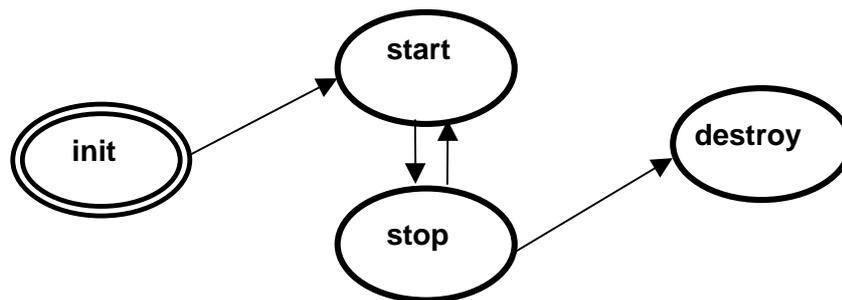


Abbildung 4-10

Damit mußte das recht komplexe Zustandsdiagramm des Applets aus dem vorigen Kapitel im Wesentlichen in einer Prozedur, nämlich in der `init()` – Methode, komplett nachgebildet werden. Dies würde den Code sehr unübersichtlich machen und eine nach Zuständen aufgeteilte Programmierung unmöglich machen. Man müßte sich für eine einheitliche Basis für alle Zustände im voraus entscheiden, welche natürlich sehr schwer im Vorfeld genau zu bestimmen ist, da dem Entwickler selbst oft nicht bewußt ist, welche Probleme im Bezug auf welche Teile des Programms auftreten werden, geschweige denn, welche Lösungen er dafür verwenden wird.

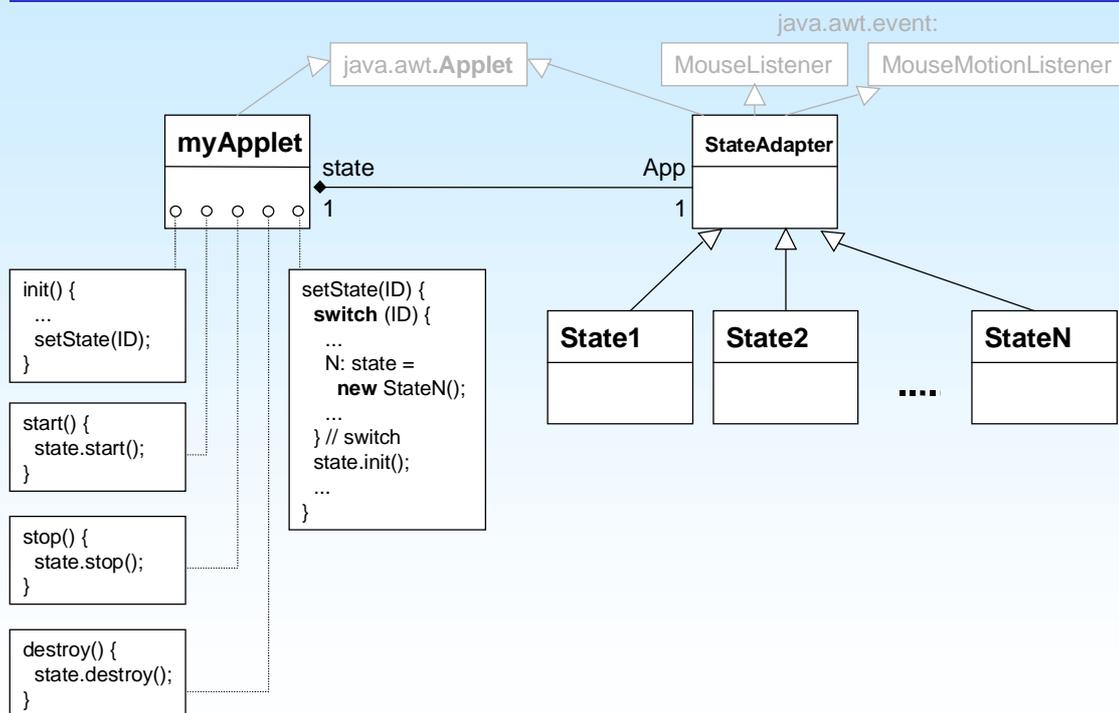
Die objektorientierte Programmierung erwies sich hier als eine große Abhilfe bei der Suche nach einem passenden Konzept zur Beseitigung dieses Problems. Die hier vorgeschlagene Lösung erledigt das angesprochene allgemeine Problem (viele unterschiedliche zustandsabhängige Funktionen in einer Methode) in seinem Kern, indem hier für jeden Zustand sozusagen ein „Unterapplet“ programmiert wird, welches von einem gemeinsamen `StateAdapter` abgeleitet wird, welches als „Vater“ für alle Zustände gilt (Zustandsvater-Konzept, Abbildung 4–11). Dieser `StateAdapter` zwingt die von ihm abgeleiteten Klassen dabei, das Applet-Interface zu implementieren, indem dieser selbst wie ein Applet organisiert wird und die typischen Applet-Methoden nach unten weitervererbt. Auch beliebige `Listener`-Interfaces werden auf dieselbe Art erzwungen. Umgekehrt lassen sich nach und nach die allen Zuständen gemeinsamen Methoden nach oben in den Zustandsvater selbst auslagern und vereinheitlichen. Bei Bedarf lassen sie sich bei entsprechender Programmierung sowieso beliebig überladen oder können sogar angereichert werden, da der Aufruf der Ursprungsmethode erlaubt ist. Das eigentliche Java-Applet `GUI` braucht dann nur noch einen Bezug zu einem solchen Zustandsvater zu haben und in einer Zustandsumschaltprozedur diesen Bezug mit der jeweils benötigten Unterklasse zu initialisieren (Polymorphie).

Die in der Abbildung 4–11 auftauchende „ID“, mit deren Hilfe die `setState()`–Methode des Hauptapplets (`myApplet` in der Zeichnung) den Zustand auswählt, repräsentiert dabei die gewählte Zustandskodierung. In unserem Fall muß zwischen fünf internen Grundzuständen umgeschaltet werden, von denen drei (`KBIntern`, `TeamComm` und `WWState`) selbst vom gerade gewählten Kontextbereich abhängen. Da in dem zugrundeliegenden Konzept eine maximale Anzahl von 15 Kontextbereichen definiert wird, muß für die eindeutige Identifizierung der Position, in der das Applet sich befand (Wiederkehr zum letzten verwendeten Punkt, etc.), die Zustandskodierung insgesamt mindestens:

$$2 + 3 \cdot 15 = 47$$

verschiedene Zustände repräsentieren können. Außerdem wird aus oben ausgeführten Gründen ein Fehlerzustand benötigt, der natürlich wie jeder andere Zustand programmiert wird und deshalb ebenfalls eine ID braucht, wenngleich ein Einstieg im Fehlerzustand natürlich sinnlos ist. Deswegen wird der Fehlerzustand zwar mit einer ID versehen (genauso wie der Modellierungsmodus, wie später erklärt wird), diese unterscheidet sich aber von der allgemeinen Kodierung und kann insbesondere nicht als Parameter ans Applet übergeben werden.

Zustandsvater - Konzept: Grundidee



Studienarbeit - Client / Artur Hecker

8

Abbildung 4-11

Da Veränderungen in der Anzahl der zulässigen Kontextbereiche bzw. verwendeter Zustände nicht auszuschließen waren, sieht das Format der grundsätzlichen gewählten Zustandskodierung so aus:

Grundzustände (als Zeichenkette):

- XXYY

wobei

XX: die Nummer des Grundzustandes, also $XX \in \{„01“ .. „05“\}$

YY: die Nummer des Kontextbereichs, also $YY \in \{„01“ .. „15“\}$. Die Nummer des Kontextbereichs wird dabei für $XX = „01“$ und $XX = „02“$ ignoriert.

Fehlerzustand:

- „F-“[Zeichenkette]

wobei [Zeichenkette] optional ist und die anzuzeigende Fehlermeldung enthalten sollte.

Damit können mit der gewählten Grundzustandskodierung $5 \cdot 99$ Zustände unterschieden werden. Dieses System ist damit ausreichend für mögliche Erweiterungen gewappnet und trotzdem übersichtlich.

Hier noch kurz die implementierte Zustandsvaterklasse im Überblick:

StateAdapter:

Benutzt:

- `GUI::showStatus()`;
- Diverse Prozeduren von `RectContainer`, `ClickableObject` und `Cache`;

Methoden:

- `public StateAdapter(GUI SomeApplet);`

Konstruktor: erzeugt einen Zustandsvater, übergibt diesem eine Referenz auf das eigentliche Applet (GUI).

- `public void init();`

Die zugrundeliegende Initialisierungsroutine. Sollte in jeder abgeleiteten `init()`-Methode als erstes aufgerufen werden.

Es folgen die typischen Appletmethoden:

- `abstract public void start();`
- `abstract public void stop();`
- `abstract public void update(Graphics g);`
- `public void destroy();`

Führt eine Standardauflösung der gemeinsam benutzten Datenstrukturen auf.

- `abstract public void paint(Graphics g);`

Hier die Methoden der `MouseListener` – Schnittstelle:

- `public void mouseClicked(MouseEvent e);`

Reagiert auf Mausaktivitäten über den eingecheckten Objekten mit dem Aufruf der in den Objekten definierten Ziele.

Die folgenden Methoden müssen vorhanden sein, die Funktionalität wird jedoch nicht gebraucht (Vorschrift des Interfaces), weswegen diese leergelassen wurden:

- `public void mouseEntered(MouseEvent e);`
- `public void mouseExited(MouseEvent e);`
- `public void mousePressed(MouseEvent e);`
- `public void mouseReleased(MouseEvent e);`

Hier die Methoden der `MouseMotionListener` – Schnittstelle:

- `public void mouseDragged(MouseEvent e);`

Leer.

- `public void mouseMoved(MouseEvent e);`

Reagiert auf Mausaktivitäten über den eingecheckten Objekten, wenn in diesen ein Überblendobjekt definiert ist (Hover-Effekt). Zeigt außerdem das in den Objekten definierte Ziel aller Sprünge in der Statuszeile an.

Benutzermethoden:

- `public boolean memberOfActKB();`

Liefert „true“ zurück, wenn der aktuelle Benutzer Mitglied im aktuellen Kontextbereich ist, „false“ sonst.

- `public boolean isProjMod();`

Liefert „true“, wenn der aktuelle Benutzer Projektmoderator ist, „false“ sonst.

Zustandsverwaltung:

- `public static String toKB(int u);`

Wandelt den übergebenen Ganzzahlwert der Zustandskodierung entsprechend (XXYY) in eine „YY“ Zeichenkette.

- `public static String getKB();`

Liefert die ID des aktuellen Kontextbereichs zurück.

- `public static boolean setKB(String KBid);`

Setzt den aktuellen Kontextbereich auf den übergebenen Wert, wobei eine Prüfung durchgeführt wird. Entspricht der Wert der Zustandskodierung („01“ .. „15“), so wird „true“ zurückgegeben, sonst „false“.

Grafikfunktionen:

- `public final void clrScr();`

Löscht den kompletten Appletbereich und deckt es mit der im Applet definierten Hintergrundfarbe.

- `public int textout(Graphics g, String text, Font f, int h);`

Gibt den übergebenen Text in der angegebenen Schrift mittig auf der übergebenen Grafik-Instanz aus und liefert die neue Höhenkoordinate zurück, indem zu der übergebenen Höhe die Schrifthöhe addiert wird.

Variablen:

- `protected GUI App;`

Das ausführende Applet, welches in allen Ableitungen benutzt wird.

- `protected static int KB;`

Der aktuelle Kontextbereich.

- `protected RectContainer rcont;`

Instanz des Rechteck-Containers, die die anklickbare Objekte speichert. Wird von `Mouse*Listeners` benutzt.

- `protected static Dimension mass;`

Appletbereichsmaße, Höhe und Breite.

- `protected static Cache cache;`

Die verwendete Cache-Instanz. Einmalig für alle Zustände.

4.3.4 Zustandssicherungsproblem und implementierte Lösung

Damit gab es nur noch ein Problem. Beim Sprung zu einer externen Seite mußte irgendwie die ZustandsID des Applets gesichert werden, damit das Applet anschließend (Rücksprung) im selben Zustand gestartet werden konnte. Leider unterliegen die Java-Applets starken Sicherheitsbeschränkungen, wodurch diese normalerweise nicht auf die lokalen Speichermedien wie Festplatten, Disketten usw. zugreifen dürfen. Die Standardbrowser befolgen diese Strategien sehr streng, um keine Angriffsmöglichkeiten für das Ausspähen oder gar Überschreiben der lokalen Daten aus dem Web zuzulassen. Deswegen wurde ein anderes Konzept gewählt, welches dem Wesen des Systems entspricht: Beim Aufruf einer externen Seite wird die aktuelle ZustandsID HTTP-GET-ähnlich an die URL angehängt und kann damit vom DBAL extrahiert werden. Wie genau das DBAL damit verfährt und wo die Kennung zwischengespeichert wird, ist Teil der serverseitigen Implementierung. Das DBAL ist ab dann dafür zuständig, den Rücksprung von der externen Seite so zu gestalten, daß die das Applet umgebende Seite derart verändert wird, daß das Applet einen entsprechenden Parameterwert erhält. Ab dann ist es nur noch eine Implementierungsfrage: das Applet untersucht beim Start den ihm übergebenen Zustandsparameter und bildet den entsprechenden Zustand durch den Aufruf seiner `setState()` – Methode. Wird kein Parameter gefunden oder ist dieses fehlerhaft, so wird stets im Initialzustand (Begrüßung) gestartet.

4.3.5 Grundsätzliches Vorgehen beim Zustandsaufbau

Alle Zustände folgen im Wesentlichen dem folgenden Ablaufplan:

Initialisierung:

- Aufruf der `init()` – Methode aus dem `StateAdapter`
- Holen aller benötigten Informationen über `Cache` – Methoden
- Prüfung auf Fehler, Sprung in den Fehlerzustand bei Bedarf
- Setzen interner Variablen usw. z.T. abhängig von den gewonnenen Werten
- Erzeugung benötigter `ClickableObjects` und setzen der Eigenschaften dieser (Art, Position, Hover, Maße, Ziel)
- Einchecken der erzeugten `ClickableObjects` in die im `StateAdapter` bestehende `RectContainer` – Klasse

Graphische Darstellung:

- Darstellung der benötigten Texte und starren Elemente wie normale Bilder, geometrischer Figuren, etc.
- Darstellung aller im `RectContainer` eingetragenen `ClickableObjects` an den in ihnen definierten Positionen.

Maus-Aktivitätsverfolgung:

- Wenn Maus bewegt wurde, so vergleiche die letzte Position mit den Polygonen der eingetragenen `ClickableObjects`. Ist eins davon von der Maus betroffen, so führe die benötigten Operationen durch (Darstellung der Hover-Effekte, Einblenden der definierten Hilfstexte, etc.) und speichere das betroffene Objekt. Befindet sich die Maus nicht über einem der Objekte, dann prüfe, ob die Maus gerade eins dieser Objekte verlassen hat, indem das gespeicherte Objekt überprüft wird. Wenn ja, dann stelle den inaktiven Zustand dieses Objekts (wenn nötig) wieder her.
- Wenn Maus geklickt wurde, dann prüfe, ob eins der eingetragenen Objekte angeklickt worden ist. Wenn ja, so führe den im Objekt definierten Sprung bzw. Zustandswechsel durch (`GUI::setState(Object_Target)`)

Zerstörung:

- Führe den bald folgenden `destroy()` – Befehl durch und löse die nur von diesem Zustand benötigten Datenstrukturen bei Bedarf auf (nichtstriktes Vorgehen wegen „garbage-collections“ von Java)

Der von Java-applets bereitgestellte und hier bereits beschriebene `start()` – `stop()` Mechanismus findet keine Anwendung, da keine fortlaufenden Aktivitäten vom Applet durchgeführt werden (etwa komplizierte Berechnungen oder aufwendige laufende Animationen).

4.4 Modellierungsmodus

4.4.1 Einbindung ins Gesamtkonzept

Die Tatsache, daß das Applet nur einen logischen Zustand zur gleichen Zeit haben konnte, wurde durch die Hinzunahme der Modellierungsfunktionen abgeschafft, weil diese in einem separaten Fenster angeboten werden sollten. Damit mußte der Aufruf der Modellierungsfunktionen entweder nach einem anderen Prinzip erfolgen, oder die schon vorgestellte Zustandsverwaltung etwas ausgedehnt werden. Gegen die erste Alternative sprach vor allem die Tatsache, daß alle bereits implementierten `MouseListener`-Aktivitätsfunktionen nur darauf vorbereitet waren, entweder externe Seiten anzuzeigen oder interne Zustandswechsel nach dem vorgestelltem Schema auszuführen. Eine Veränderung dieser Funktionen würde sich sehr mühsam gestalten, da sie z.T. in den abgeleiteten Zuständen überladen wurden. Dadurch würden sich die Änderungen nicht nur lokal, sondern über das ganze Programm verstreut, auswirken, wie im nächsten Unterabschnitt verständlich wird. Eine Erweiterung der

Zustandskodierung hingegen war erstens ziemlich einfach, zweitens erforderte sie eindeutige Änderungen in nur einer Methode: `setState()`.

Damit sah die Einbindung so aus, daß das Hauptapplet `GUI` gleichzeitig einen zweiten Zustand besitzen durfte. Es bekam also eine weitere Referenz auf die Zustandsvaterklasse, denn die verwendeten Modellierungsfunktionen sollten den Zugriff auf alle Zustandsverwaltungs- und Benutzerfunktionen bekommen, und der im Browser aktive Zustand nicht zerstört werden. Die Zustandskodierung wurde um einen Editierzustand erweitert, für den die beim Fehlerzustand angesprochenen Einschränkungen gelten:

Editierzustand (Zeichenkette):

- „E“ZYY

wobei

Z: ModellierungsfunktionsID, $Z \in \{„1“ \dots „8“\}$

YY: die ID des aktuellen KB wie gehabt

Die `setState()` – Routine im `GUI` wurde um entsprechende Erkennung der Zustandskodierung erweitert und überschreibt im Unterschied zu den sonstigen Zustandswechseln nicht mehr die vom `GUI` benutzte Hauptreferenz auf den Zustandsvater, sondern belegt die neu hinzugekommene zweite Referenz mit dem entsprechenden gebildeten Objekt.

Die Kommunikation mit dem DBAL geschieht über die im gemeinsamen Teil beschriebene `RequestedFunction` – Schnittstelle mit Hilfe der üblichen Kommunikationsmittel. Allerdings wird die oberste Schicht, der Cache, nicht für „Schreiboperationen“ benötigt, weswegen alle Modellierungsfunktionen für die eigenen Übertragungen unmittelbar mit der `JavaPOST`-Klasse arbeiten.

4.4.2 Modellierungsfunktionen mit Rechten

Es wurden insgesamt acht Modellierungsfunktionen benötigt, die im gemeinsamen Teil mit- samt der vorausgesetzten Rechte beschrieben werden. Im Client-Teil mußte eine sinnvolle Aufteilung der Modellierungsfunktionen auf die bestehenden Zustände gefunden werden, und die Zustände mußten um entsprechende Schaltflächen zur Aktivierung erweitert werden.

Die Aufteilung ist dabei wie folgt:

Grundzustand	FktID	Funktionsname	Bedeutung
Navigator	1.	Neuer Akteur	Neues Mitglied wird dem Projekt aus der umgebenden Datenbank hinzugefügt.
	2.	Akteur entfernen	Ein Projektmitglied soll aus dem Projekt gelöscht werden.
	3.	Neuer KB	Neuer Kontextbereich soll erzeugt werden
Teamkommunikation	5.	KBM ändern	KB-Moderator soll geändert werden.
	7.	Akteur nach KB	Ein Projektmitglied soll einem Kontextbereich hinzugefügt werden.
	8.	Akteur aus KB	Ein Projektmitglied soll aus einem Kontextbereich entfernt werden
Wechselwirkungen	4.	Neue WW	Neue Wechselwirkung zwischen zwei KBen soll erzeugt werden
	6.	WWM ändern	Der Moderator einer Wechselwirkung soll verändert werden

Diese Aufteilung wurde gewählt, um dem Prinzip des Kontextes Folge zu leisten, welches in dem den Studienarbeiten zugrundeliegenden System begründet wird: Alle Funktionen auf Objekten sollen da verfügbar gemacht werden, wo diese Objekte präsentiert werden.

4.4.3 Standardablauf einer Modellierungsfunktion

Der Standardablauf einer Modellierungsfunktion folgt stets dem gleichen Prinzip und ist primär in zwei Abschnitte unterteilt:

- Initialisieren einer Funktion (`<functionName>_init()`)
- Verarbeitung der Benutzereingaben (`<functionName>()`)

Dabei lassen sich diese zwei Punkte wie folgt aufschlüsseln (hier abstrakt):

Initialisierung:

- Öffne ein neues Desktop-Fenster auf der Basis der gemeinsamen Vorlage
- Fülle die Elemente dieses Fensters mit funktionsspezifischen Informationen. Dazu gehören die Gestaltung und Bildung eigener Elemente in einem zur Verfügung gestellten Bereich, Setzen aller benötigten Beschriftungen der Vorlage und der eigenen Elemente, etc.
- Binde die Maus an die gebildeten GUI-Elemente und warte auf Benutzeraktionen

Verarbeitung der Eingaben:

- Je nach durchgeführter Aktion: Setze die Eigenschaften von weiteren GUI-Elementen abhängig von den gemachten Benutzereingaben oder hole alle Werte aus den GUI-Elementen.
- Prüfe die Semantik und die Abhängigkeiten der Eingaben. Im Fehlerfall: informiere den Benutzer über die Fehler und warte auf die Veränderung der Eingaben. Übermittle die Daten im fehlerfreien Fall durch die `RequestedFct`-Schnittstelle an das DBAL und warte auf die Antwort. Prüfe die DBAL-Antwort auf mögliche Fehlermeldungen und informiere den Benutzer über diese Fehler bzw. melde den Erfolg der Aktion. Warte auf weitere Eingaben (Beendigung oder erneuter Modellierungswunsch).

Dabei verfolgen alle Funktionen schon während der Initialisierungsphase eine fehlervermeidende Strategie, indem diese dem Benutzer möglichst nur die Eingaben (z.B. durch zusammengestellte Auswahllisten) erlauben, die sinnvoll sind.

4.4.4 Funktionsvorlage

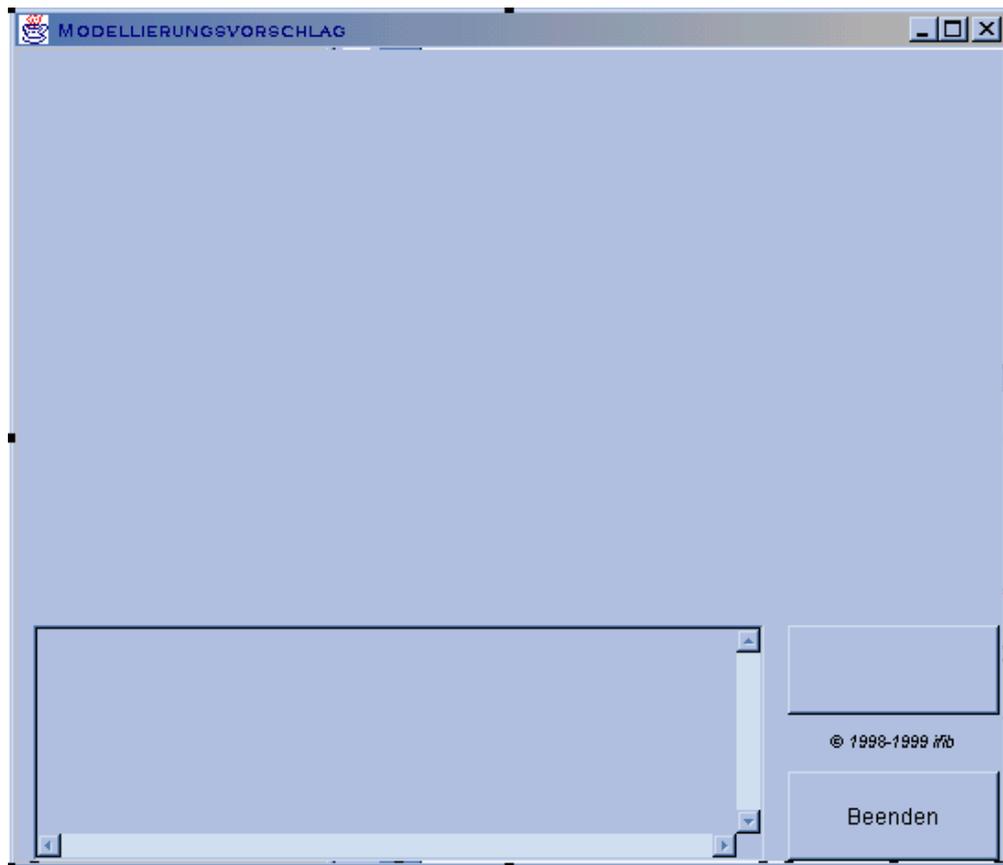
Da alle Modellierungsfunktionen einen gemeinsamen Teil haben, wurde zunächst eine Funktionsvorlage gebildet. Dazu gehört z.B. die komplette Funktionalität zum Öffnen eines Fensters auf dem Desktop mit bestimmten GUI-Elementen, die von allen Funktionen verwendet werden.

Dabei wurde im voraus Designarbeit geleistet und ein einheitliches Aussehen für alle Modellierungsfunktionen erarbeitet. Alle Funktionen sollten diesem Design entsprechend ihre eigenen GUI-Elemente wie oben beschrieben plazieren. Die Funktionsvorlage (`EditFrame`) bietet dabei:

- einen Textbereich, in dem die Funktionen ihre Meldungen dem Benutzer präsentieren können (Fehler, durchgeführte Aktion, Erfolg)
- einen beschrifteten „Beenden“-Button, dessen Funktion für alle Funktionen gleich ist
- einen unbeschrifteten Aktivierungsbutton, der in jedem Fall die entsprechende Funktion aktiviert, dessen Schriftzug aber dem Sinn der gerade durchgeführten Funktion entspricht
- einen untergeordneten gemeinsamen Bereich (`Panel`), auf den die Einzelfunktionen beliebig Einfluß nehmen können (für ihre GUI-Elemente)

In Abbildung 4-12 wird die hier beschriebene „leere“ Vorlage dargestellt.

Abbildung 4-12



4.4.5 Rechteidentifizierung und Durchsetzung

Die Durchsetzung der Rechte geschieht im Gesamtsystem beidseitig: Der Client überprüft die vom DBAL gemeldete Projektfunktion der angemeldeten Person und unterdrückt die Anzeige der Modellierungsschaltflächen, die für den jeweiligen Benutzer laut der Funktionstabelle im gemeinsamen Teil nicht zur Verfügung stehen, in den entsprechenden Zuständen. Für diese Zwecke stehen die bereits im `StateAdapter`-Abschnitt aufgeführten Funktionen `isProjMod()` und `memberOfActKB()` zur Verfügung. Auf der anderen Seite prüft das DBAL alle Eingaben (da diese nicht nur vom korrekten Client stammen könnten) und meldet Erfolg oder Fehler der Anfrage, was die jeweilige Modellierungsfunktion wiederum dem Benutzer mitteilt.

4.5 Hauptapplet und kurze Vorstellung allgemeiner Klassen

4.5.1 Hauptapplet

Nachdem die meisten Funktionen und das Prinzip vorgestellt worden sind, sollen hier die noch fehlenden Klassen kurz vorgestellt werden. Der Übersichtlichkeit wegen wird hier am ausführlichsten das Hauptapplet vorgestellt:

GUI :

Benutzt:

Alle Zustandskonstruktoren

Methoden:

Applet-typische Methoden:

- `void init():`

Setzt Umgebungsvariablen und interne Werte z.T. abhängig von den Parameterwerten. Z.B. Hintergrundfarbe, verschiedene Schriftarten, ruft den ersten benötigten Zustand auf.

Die folgenden Methoden benutzen entweder die Zustandsvariable und die darin deklarierte Prozedur (z.B. `state.destroy()`) oder bleiben leer (z.B. `stop()`)

- `void start()`
- `void stop()`
- `void paint(Graphics)`
- `void update(Graphics)`
- `void destroy()`

Die folgenden zwei Methoden gehören zum guten Ton bei der Appletprogrammierung und geben Auskunft über Appletparameter und Autor (soll später in die gängigen Browser integriert werden):

- `String[][] getParameterInfo()`

Liefert einen Vektor von Tripeln der Art „Param:Typ:Beschreibung“ für alle Appletparameter zurück

- `String getAppletInfo()`

Liefert eine kurze Info über die Art und den Autor des Applets

Zustandsmethoden:

- `StateAdapter getState():`

Liefert eine Referenz auf den zugrundeliegenden Grundzustandsvater

- `String getStateID():`

Liefert die volle Zustandskodierung des aktuellen Zustands

- `void setState(String):`

Vollführt den Zustandswechsel zu dem in der Zeichenkette kodierten internen Zustand, indem der alte Zustand zerstört, die entsprechende Referenz (s. unter Variablen) gesetzt und die `init()`-Routinen dieser Referenz aufgerufen werden. Bindet die Maus an die neuen Zustände. Verarbeitet auch Fehlerzustände und Editierzustände, wobei bei letzteren eine neue Referenz verwendet wird.

- `URL url(int Typ):`

Gibt eine URL zurück, die von dem als Ganzzahl übergebenen Typ abhängig ist, indem an der Basis-URL des Applets bestimmte weitere Zeichenketten, die wiederum aus der Parameterliste stammen, übergeben werden. Vordefinierte Typen: BASIS, DBAL, MAIL, VIEW, PIC, IBP, PHYS. (S. Appletparameterbeschreibung).

Variablen:

- `public Color backGround, foreGround;`

Appletfarben, werden überall verwendet

- `public Font mfont, sfont, lfont, xlfont;`

Allgemein verwendbare Fonts (klein, normal, groß, sehr groß)

- `protected static StateAdapter state, editstate;`

Die Referenzen auf die Zustände des Applets.

Um das Applet anpassungsfähiger zu machen und z.B. die theoretische Möglichkeiten einer Weiterverwendung in einer anderen Umgebung zu berücksichtigen, wurden verschiedene Parameter definiert, mit Hilfe welcher das Applet z.B. auch durch mehrere CGI-Skripte anstelle von DBAL bedient werden könnte.

Appletparameter:

Name	Typ	Beschreibung
DBALAddr	String	Adresse des Hauptkommunikationsskriptes (muß RequestedInfo und RequestedFunction entsprechend der jeweiligen Beschreibung per POST unterstützen)
MailAddr	String	Adresse des Mailskriptes (muß die jeweilige Benutzermaildatenbank als HTML-Seite anzeigen können, wobei das Applet das folgende Format verwendet: „http://basisadresse:port/<MailAddr>&To=<TargetUserName>&State=<State>&KBNr=<KBNr>“
ViewAddr	String	Adresse des Ansichtsskriptes. Ähnlich wie bei den Mails hängt das Applet hier Rücksprunginformationen an. Es muß also eine HTML-Seite generiert werden, die korrekte Rücksprünge definiert. &Ansicht=<Ansichtsname> wird statt &To verwendet.
PicAddr	String	Basisadresse für alle vom Applet verwendeten Bilder
IBPAddr	String	Adresse des Auswahlskriptes für persönliche Informationsblätter
State	String	Startzustand des Applets in Zustandskodierung

4.5.2 Allgemeine Hilfsklassen

Außer den bis jetzt ausführlich beschriebenen interessanten Klassen mit speziellen Aufgaben, mußten natürlich noch einige Hilfsklassen programmiert werden, die hier nur kurz vorgestellt werden sollen. Auf die Methodenbeschreibung der Klassen `ClickableObject`, `RectContainer`, `PersonalCard` und `SortedVector` wird somit verzichtet, da diese Klassen nur intern verwendet werden und nicht von konzeptionellem Interesse sind.

Die Funktionen von `RectContainer` und `ClickableObject` wurden bereits in der Zustandsbeschreibung erläutert. `RectContainer` ist eine Ableitung der Java-eigenen `Vector` Klasse, mit einer weiteren Methode, die einen Punkt einem eingecheckten Gebiet zuweisen kann und dieses Gebiet zurückliefert. `ClickableObject` ist eine Ableitung der Java-AWT-Klasse `Rectangle`, die aber als eigentliches Objekt auch Bilder aufnehmen und alle für einen Sprung benötigten Informationen speichert.

Jede Instanz der Klasse `PersonalCard` ist eine der „CommuniCards“, die im Teamkommunikationszustand auftauchen. Diese Klasse kümmert sich um die räumliche Anordnung aller Einzelbilder und setzt die Eigenschaften der in der Karte vorhandenen `ClickableObjects`.

`SortedVector` wird von den Modellierungsfunktionen gebraucht, um die Auswahllisten sortiert anzuzeigen. Diese Klasse ist ebenfalls eine Ableitung der Java-`Vector`-Klasse und kann die Elemente während des Füllvorganges in lexikographischer Ordnung einfügen, so daß beim anschließenden sequentiellen Lesen eine (lexikographisch) sortierte Liste ausgegeben wird. Entsprechende Ordnungsfunktionen sind ebenfalls darin definiert.

5 Serverseitige Implementierung

5.1 Überblick

Wie aus den vorherigen Kapiteln ersichtlich hat die serverseitige Implementierung drei wesentliche Aufgaben:

1. Rückgabe von Strukturdaten über die im allgemeinen Teil definierte Schnittstelle.
2. Bereitstellung von Modellierungsfunktionen über die zweite im allgemeinen Teil definierte Schnittstelle.
3. Durchführung eines Aktivierungszyklus zur Konfliktregelung.

Diese Aufteilung der Aufgaben entspricht allerdings nur teilweise der Systemstruktur, weshalb die Verteilung der Aufgaben auf die Module zunächst einmal in der folgenden Abbildung 5-1 kurz dargestellt werden soll.

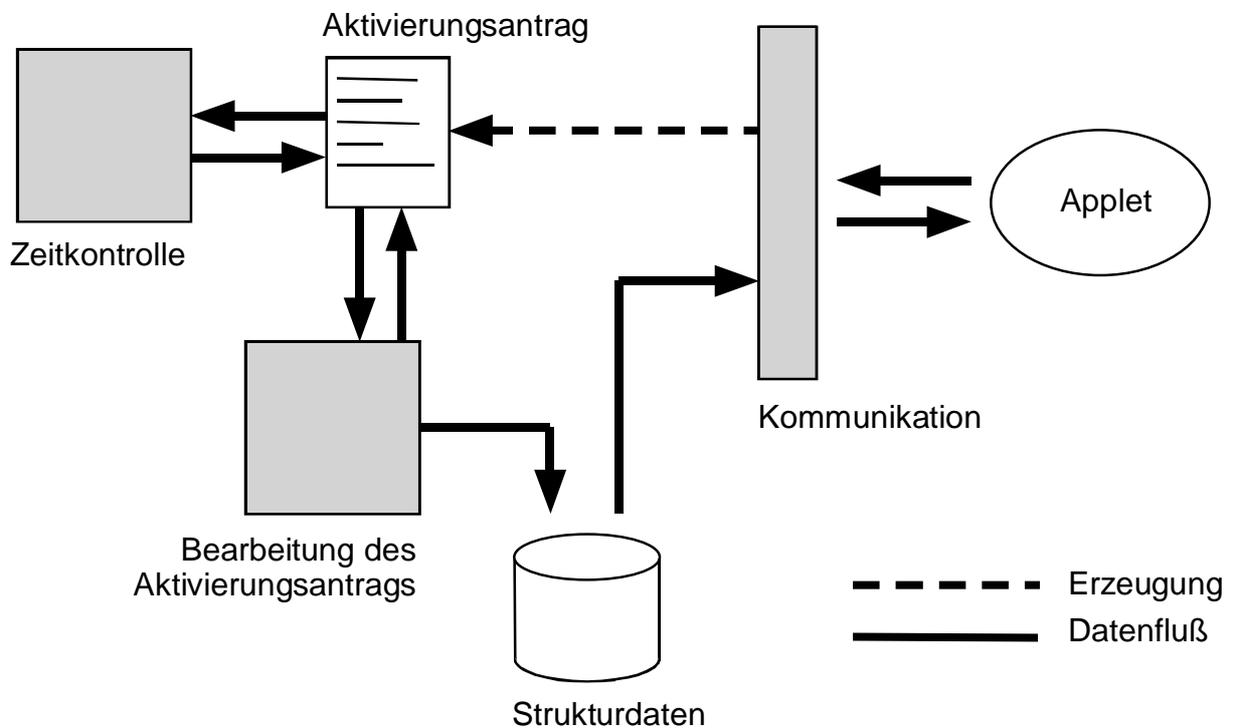


Abbildung 5-1

Die gesamte Kommunikation mit dem Applet geschieht über ein einziges Modul, das folglich sowohl Informations- als auch Modellierungsanfragen entgegennehmen muß. Hier kann also die Abfrage der internen Strukturinformationen, oder die Erzeugung eines Aktivierungsantrags nötig sein. Nach der Erzeugung des Aktivierungsantrags muß dieser bearbeitet werden. Diese Bearbeitung wird von einem weiteren Modul durchgeführt und besteht im wesentlichen darin, die vom Aktivator gewünschten Manipulationen an dem Antrag zu vollziehen, d.h. Start des Kommentierungszyklus, Aktivierung oder Ablehnung bzw. die daraus resultierenden Aufgaben wie z.B. Abbruch des Kommentierungszyklus. Der Vorgang der Zeitkontrolle innerhalb des Kommentierungszyklus ist offensichtlich zeitgesteuert und wird deshalb von einem periodisch ablaufenden Modul übernommen. Die Implementierung spiegelt die Aufteilung in die drei Module insofern wider, als es sich genau um die drei implementierten Agenten handelt, die im Abschnitt „Die Agenten“ beschrieben sind.

Die Implementierung der acht Modellierungsfunktionen ist die zweite zentrale Aufgabe der serverseitigen Implementierung, der deshalb ein eigener Abschnitt nämlich „Die

Modellierungsfunktionen“ gewidmet ist. Hier wird auf den generellen Aufbau und auch auf die Funktionen im einzelnen eingegangen.

Diese Programmteile werden wiederum durch weitere kleinere Module unterstützt, die gewisse elementare Funktionalitäten kapseln, wie z.B. die Dekodierung der Anfrage. Die implementierten Hilfsfunktionen befinden sich im Abschnitt „Die allgemeinen Hilfsklassen“.

Neben der Erstellung dieser ausführenden bzw. aktiven Teile ergibt sich als weitere Aufgabe die Planung des Aktivierungsantrags, der durch eine recht komplexe Lotus Notes® Maske realisiert ist und im Abschnitt „Der Aktivierungsantrag“ behandelt wird. Die Komplexität der Maske ergibt sich daraus, daß sie den Aktivierungsantrag in allen möglichen Zuständen des Lebenszyklus darstellen muß, d.h. Anzeige der benötigten Schaltflächen, Felder usw.

Grundlage für alle Aufgabenteile stellen dabei die Strukturinformationen des Projekts dar. Aufgrund der statischen Struktur des ursprünglichen Systems waren große Teile der Struktur nicht explizit verfügbar, und es mußte zunächst eine Neugestaltung der Datenstrukturen vorgenommen werden. Der daraus entstandene Aufbau der Strukturinformationen wird im folgenden Abschnitt dargelegt.

5.2 Die Strukturinformationen

Die benötigten Strukturinformationen lassen sich in drei Kategorien unterteilen:

- Allgemeine Daten des Projekts.
- Informationen zu einem Kontextbereich.
- Informationen zu einer Wechselwirkung zwischen zwei Kontextbereichen.

Diese natürliche Aufteilung der Daten wurde direkt in den Aufbau der Strukturinformationen übernommen, die in Abbildung 5-2 zu sehen ist.

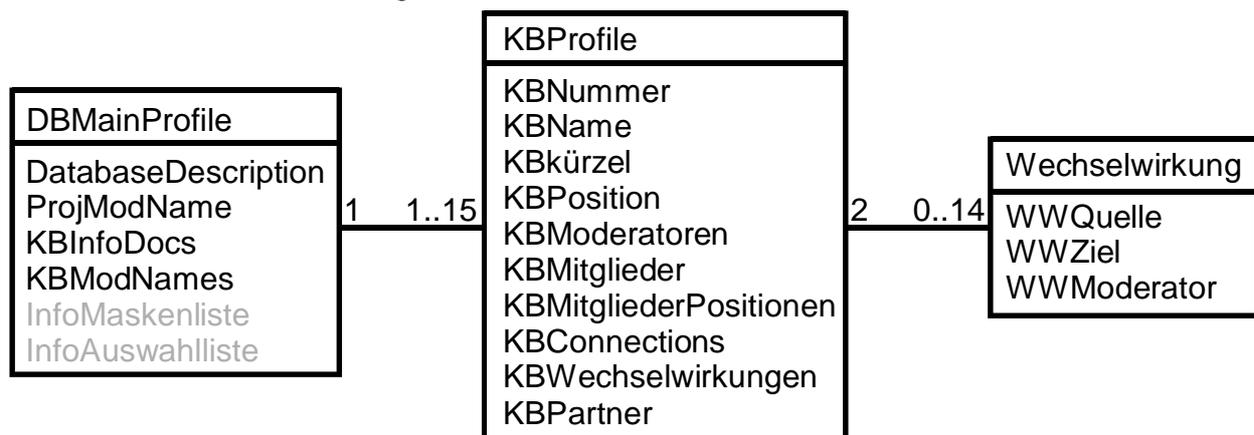


Abbildung 5-2

Die allgemeinen Daten werden in einem Profildokument der Datenbank mit dem Namen „DBMainProfile“ gespeichert. Ein solches Profildokument ist ein spezielles Dokument, das in Formeln und Agenten direkt über den Namen referenziert werden kann, für den Benutzer im allgemeinen aber unsichtbar ist. In diesem Dokument finden sich: eine prägnante Bezeichnung des Projekts (DatabaseDescription), der Name des Projektmoderators (ProjModName), eine Liste von Referenzen auf die vorhandenen Kontextbereiche (KBInfoDocs) und die Namen der entsprechenden Moderatoren (KBModNames), sowie eine Zuordnung von Maskennamen zu intuitiven Bezeichnungen, wie sie als Auswahl angeboten werden. Diese Zuordnung ist bereits Teil des Systems gewesen, wurde aber aus der tatsächlichen Abfragemaske in das Hauptdokument verlagert, um eine komfortablere Anpassung zu erlauben.

Zu den Informationen eines Kontextbereichs, die in einem normalen Dokument gespeichert werden, gehören: seine Nummer (KBNummer), sein Name (KBName), das Kürzel (KBkürzel), seine Position im Navigator (KBPosition), der Name des Moderators (KBModeratoren), eine Liste seiner Mitglieder (KBMitglieder) und ihrer jeweiligen Position im Bereich Teamkommunikation (KBMitgliederPositionen), ein Feld von Nullen und Einsen, das

die Verbindungen zu anderen Kontextbereichen beschreibt (KBConnections), eine Liste von Referenzen auf die Wechselwirkungsdokumente (KBWechselwirkungen), eine Liste von Referenzen auf die angeschlossenen Kontextbereiche (KBPartner).

Die Wechselwirkungsdokumente enthalten jeweils eine Referenz auf die beiden Kontextbereiche, die durch die Wechselwirkung verknüpft werden (WWQuelle, WWZiel), sowie den Namen des Wechselwirkungsmoderators (WWModerator).

Man merkt sofort, daß einige der Informationen doppelt gespeichert werden, wie z.B. die Namen der Kontextbereichsmoderatoren, die einmal im Hauptdokument und einmal einzeln in den Kontextbereichsdokumenten abgelegt sind. Dies liegt daran, daß durch die rein dokumentorientierte Struktur der Lotus Notes® Datenbank manche Informationsanfragen zu einem ungebührlich hohen Aufwand bei der Bereitstellung der Daten führen würden. Um dies zu vermeiden, werden solche Informationen doppelt abgelegt, auch wenn damit der Nachteil verbunden ist, daß bei der Manipulation der Daten an zwei oder mehr Stellen Daten angepaßt werden müssen.

5.3 Die Agenten

5.3.1 Der Agent „AnfrageAgent“

5.3.1.1 Überblick

Der Anfrageagent stellt die eigentliche Schnittstelle (in Abbildung 5-1 mit „Kommunikation“ bezeichnet) für externe Programme dar (insbesondere für das Applet) und hat zwei wesentliche Aufgaben. Zum einen wird über ihn die Struktur des Projekts zugänglich gemacht, was über die im gemeinsamen Teil beschriebene Schnittstelle für Informationsanfragen erfolgt. Dadurch kann eine Übergabe aller Projektstrukturen erfolgen, die für eine graphische Aufbereitung nötig sind. Zum anderen hat der *Agent* die Aufgabe, die Modellierungsfunktionen zur Verfügung zu stellen. Wie bei den Informationsanfragen kann über die im gemeinsamen Teil beschriebene Schnittstelle die *Initiierung* einer Modellierungsfunktion beantragt werden. In den folgenden Abschnitten soll nun zunächst die Einbettung des *Agenten* in die Datenbank und dann die Realisierung der beiden Hauptaufgaben erläutert werden.

5.3.1.2 Einbettung in die Datenbank und in die Objektstruktur

Der Anfrageagent ist als *Agent* in die Lotus Notes® Datenbank eingebettet. Dies bedeutet, daß dem Agenten durch den Server ein Umgebungsdokument zur Verfügung gestellt wird, in dem alle CGI-Variablen und andere Parameter der Verbindung übermittelt werden.

Die benötigte Klasse „ContextInfo“ nimmt eine Initialisierung der wichtigsten benötigten Variablen vor und macht insbesondere dieses Umgebungsdokument verfügbar, um die weiteren Aufgaben durchführen zu können.

Alle Aufgaben bezüglich der HTTP-Verbindung können dem Lotus Domino® Webserver überlassen werden. Der Inhalt einer POST-Anfrage steht dem *Agenten* wie ein Textfeld im Umgebungsdokument zur Verfügung. Um den Inhalt dieses Feldes nutzbar zu machen, wird die Klasse „StringDecoder“ benötigt, die eine Umwandlung der Übergabedaten aus dem MIME-Format „application/x-www-form-urlencoded“ in eine Java-Hashtabelle übernimmt. Für die Ausgabe steht dem Agent ein Java-PrintWriter Objekt zur Verfügung, das mit Hilfe der Klasse „RequestReply“ wiederum im MIME-Format „application/x-www-form-urlencoded“ beschrieben wird.

Ein weiterer Vorteil der Einbettung in die Datenbank ist die automatische Berücksichtigung der Berechtigungen des Benutzers. Diese werden gemäß der Zugriffskontrolliste der Datenbank vom Server abgefragt und das Login steht wiederum in einem Feld des Umgebungsdokuments zur Verfügung. Der Inhalt dieses Feldes wird von der Klasse „ContextInfo“ und der darin benötigten Klasse „UserInfo“ dazu benutzt, den offiziellen Benutzernamen des Benutzers zu ermitteln.

Desweiteren benötigt der Agent die Klassen mit den Modellierungsfunktionen, um diese bei Bedarf zu initiieren.

5.3.1.3 Bearbeitung einer Informationsanfrage

Nachdem der Inhalt der POST-Anfrage mit Hilfe der Klasse „StringDecoder“ in eine Hashtabelle umgewandelt wurde, wird zunächst geprüft, ob es sich um eine Informationsanfrage handelt. Dies ist dann der Fall, wenn der Schlüssel „InfoRequest“ in der Hashtabelle vorhanden ist. Da der Wert dieses Eintrags gemäß der Schnittstellenbeschreibung für jede Strukturinformation angibt, ob sie zurückzuliefern ist, wird dieser zunächst in ein Array von Wahrheitswerten umgewandelt, das dann in der Unterprozedur TreatInfoRequest Element für Element abgearbeitet wird. Für jeden als gewünscht markierten Wert werden die Strukturdokumente nach den entsprechenden Daten durchsucht und diese werden mit Hilfe der Klasse „RequestReply“ unter den in der Schnittstellenbeschreibung festgelegten Schlüsseln übergeben. Sinnvolle Erweiterungen der Schnittstelle können dadurch erfolgen, daß weitere Wahrheitswerte definiert werden, die in einem entsprechenden Block behandelt werden. Eine erfolgreiche Bearbeitung der Informationsanfrage wird mit dem Wert „0“ unter dem Schlüssel „ErrorCode“ bestätigt. Die Fehlerbehandlung wird gemäß den im Abschnitt Fehlerbehandlung beschriebenen Prinzipien durchgeführt.

5.3.1.4 Bearbeitung einer Modellierungsanfrage

Eine Modellierungsanfrage wird durch den Schlüssel „FctRequest“ gekennzeichnet, der die Nummer der gewünschten Modellierungsfunktion enthält. Der Wert wird an die Unterprozedur TreatFctRequest übergeben, die diesen in einer großen CASE-Anweisung weiterbehandelt. Je nach Funktion werden hier die in der Schnittstellenbeschreibung geforderten Werte überprüft und an das jeweilige Modellierungsobjekt weitergeleitet. Dazu wird zunächst das Modellierungsobjekt erzeugt, dann werden die jeweiligen Initialisierungsfunktionen aufgerufen, und schließlich wird der Aktivierungsantrag erzeugt und eine Benachrichtigung an den Verantwortlichen versandt. All diese Aufgaben sind für jede der acht Modellierungsfunktionen in einem getrennten Objekt gekapselt, sodaß im Anweisungsblock der CASE-Anweisung nur Methoden dieser Objekte verwendet werden. Das Hinzufügen weiterer Modellierungsfunktionen würde also durch Erweiterung des Wertebereichs und der CASE-Anweisung, sowie durch die Programmierung eines neuen Modellierungsobjekts, erfolgen. Auch hier wird der Wert „0“ unter dem Schlüssel „ErrorCode“ zurückgegeben, wenn die Bearbeitung erfolgreich war. Die Fehlerbehandlung entspricht den im Abschnitt Fehlerbehandlung beschriebenen Prinzipien.

5.3.2 Der Agent „AktAntragBearbeitung“

5.3.2.1 Überblick

Jede Bearbeitung des Aktivierungsantrags über die Erstellung hinaus wird nicht mehr vom Anfrageagent durchgeführt, sondern von dem Agent „AktAntragBearbeitung“. Dieser wird über Schaltflächen im Aktivierungsantrag gestartet und bekommt dabei die durchzuführende Aktion mitgeteilt. Die Aktion ist dabei eine der folgenden: den Aktivierungsantrag annehmen oder ablehnen, den Kommentarzyklus starten oder abrechnen oder aber eine Kommentareingabe abschließen. Im folgenden soll kurz die Einbettung beschrieben werden, sowie die Funktionsweise des Agenten.

5.3.2.2 Einbettung in die Datenbank und in die Objektstruktur

Der Agent „AktAntragBearbeitung“ ist wiederum in die Lotus Notes® Datenbank eingebettet. Dies bedeutet, daß dem Agent durch den Server ein Umgebungsdocument zur Verfügung gestellt wird, in dem alle CGI-Variablen, sowie das aktuell in Bearbeitung befindliche Dokument, übermittelt werden.

Ein weiterer Vorteil der Einbettung in die Datenbank ist die automatische Berücksichtigung der Berechtigungen des Benutzers. Diese werden gemäß der Zugriffskontrollliste der Datenbank vom Server abgefragt und das Login steht dann in einem Feld des Umgebungsdocuments zur Verfügung. Der Inhalt dieses Feldes wird von der Klasse „ContextInfo“ und der darin benötigten Klasse „UserInfo“ dazu benutzt den offiziellen Benutzernamen des Benutzers zu ermitteln. Desweiteren nimmt die Klasse „ContextInfo“

eine Initialisierung der wichtigsten benötigten Variablen vor und macht insbesondere das Umgebungsdokument verfügbar, um die weiteren Aufgaben durchführen zu können. Um bei Bedarf die Aktivierung einer Modellierungsanfrage durchführen zu können, benötigt der Agent die Klassen mit den Modellierungsfunktionen.

5.3.2.3 Durchführung einer Aktion

Dem Agent wird durch den Aufrufer, d.h. durch das Skript, welches sich hinter der Schaltfläche im Aktivierungsantrag befindet, in Form eines reservierten Feldes mitgeteilt, welche Aktion durchzuführen ist. Der Feldinhalt wird in einer CASE-Anweisung ausgewertet, und die gewünschte Aktion wird jeweils durchgeführt.

Aktivierungsantrag annehmen

Wie bei allen Aktionen wird zunächst überprüft, ob die Rechte des aktuellen Benutzers, wie er von der Klasse „UserInfo“ ermittelt wurde, ausreichen, um die gewünschte Aktion durchzuführen. Dies geschieht mit Hilfe von Rollen. Außerdem wird geprüft, ob sich das Dokument im korrekten Status befindet, denn ein Aktivierungsantrag darf z.B. nicht angenommen werden, wenn er sich noch im Kommentarzyklus befindet. Die Schaltfläche sollte dann im Aktivierungsdokument gar nicht auftauchen, es muß allerdings auch davon ausgegangen werden, daß jemand versucht die Aktion unberechtigterweise anzustoßen. Das weitere Vorgehen hängt von der im Aktivierungsantrag angegebenen Modellierungsfunktion ab und ist deshalb genau wie die Initiierung in das Modellierungsobjekt ausgelagert. Dieses Objekt wird basierend auf dem Aktivierungsantrag instantiiert, und es werden die Methoden zur Initialisierung und Durchführung der Funktion aufgerufen. Abschließend wird das Aktivierungsdokument archiviert, indem dem Aktivator der Funktion die Rechte entzogen werden und der Status dementsprechend geändert wird. Zu allerletzt wird dem Initiator der Funktion noch mitgeteilt, daß sein Antrag angenommen wurde.

Aktivierungsantrag ablehnen

Da bis zum aktuellen Zeitpunkt noch keine Änderungen effektiv durchgeführt wurden, beschränken sich die Aufgaben bei der Ablehnung des Antrags auf die Überprüfung der Rechte und des Status, die Archivierung und die Benachrichtigung des Initiators. Das Vorgehen entspricht hierbei weitgehend dem Annehmen eines Antrags.

Kommentarzyklus starten

Beim Start des Kommentarzyklus werden Rechte und Status überprüft, sowie alle im Dokument angegebenen Kommentatoren. Diese müssen im Adreßbuch des Servers gefunden werden, damit ihnen eine Nachricht zugesandt werden kann. In der Versendung der Benachrichtigungen besteht auch die wesentliche Aufgabe dieser Unterprozedur des Agenten. Ansonsten wird nur der Status entsprechend geändert und die Zugriffsrechte auf das Dokument werden gelockert, sodaß die angegebenen Kommentatoren den Abschnitt mit der Kommentareingabe verwenden dürfen.

Kommentareingabe abschließen

Auch hier werden sicherheitshalber die Rechte und der Status überprüft. Danach wird der eingegebene Kommentar je nach angegebener Tendenz (neutral, positiv oder negativ) in der richtigen Farbe und mit Angabe von Datum und Namen des Kommentators in die Liste der Kommentare eingetragen. Anschließend wird der Kommentator aus der Liste der Kommentatoren gelöscht, und seine Zugriffsrechte auf das Dokument werden wieder aufgehoben.

Sollte es sich um den letzten Kommentator gehandelt haben, muß der Status des Dokuments wieder zurückgesetzt werden, und eine Benachrichtigung an den Verantwortlichen geschickt werden, der über den weiteren Ablauf entscheidet. Eine eventuelle Zeitschranke kann gelöscht werden.

Kommentarzyklus abbrechen

Neben dem normalen Ablauf des Kommentarzyklus hat der Aktivator noch die Möglichkeit, den Kommentarzyklus vorzeitig abzubrechen. Hierzu werden zunächst Rechte und Status geprüft. Danach werden alle noch eingetragenen Kommentatoren durch eine Benachrichtigung in Kenntnis gesetzt, daß ihr Kommentar nicht mehr benötigt wird, und ihnen werden die entsprechenden Rechte entzogen. Danach wird noch der Status des Dokuments korrigiert, sodaß der Aktivator erneut über das weitere Vorgehen entscheiden kann.

5.3.3 Der Agent „AktAntragZeitkontrolle“

5.3.3.1 Überblick

Der Agent „AktAntragZeitkontrolle“ sorgt dafür, daß in Aktivierungsanträgen gesetzte Zeitschranken für die Kommentierung überprüft werden. Stößt er auf eine überschrittene Zeitmarke, so beendet er automatisch den Kommentierungszyklus.

5.3.3.2 Einbettung in die Datenbank und in die Objektstruktur

Der Agent „AktAntragZeitkontrolle“ ist wiederum in die Lotus Notes® Datenbank eingebettet, allerdings handelt es sich bei diesem Agenten im Gegensatz zu den anderen um einen zeitlich gesteuerten Agenten. Der Agent wird vom Datenbanksystem automatisch in periodischen Abständen ausgeführt und kann dann auf die Dokumente der Datenbank einwirken. Ein Umgebungsdokument ist in diesem Fall nicht vorhanden.

5.3.3.3 Funktionsweise

Zunächst einmal werden alle Dokumente ermittelt, die sich im Kommentarzyklus befinden. Diese werden dann insofern abgearbeitet, daß überprüft wird, ob sie eine Zeitschranke besitzen, und, falls dies der Fall ist, ob diese überschritten ist. Ist genau dies eingetreten, entspricht das weitere Vorgehen nahezu der Aktion „Kommentarzyklus abbrechen“ des Agenten „AktAntragBearbeitung“. Auch hier werden die verbleibenden Kommentatoren benachrichtigt, daß ein Kommentar nicht mehr nötig ist, ihre Zugriffsrechte auf das Dokument werden eingeschränkt, und der Status wird korrigiert. Außerdem erhält der Aktivator eine Benachrichtigung, daß eine Entscheidung über das weitere Vorgehen erforderlich ist.

5.3.4 Weitere Agenten

Die allgemeine Funktionsweise von Lotus Notes® macht an manchen Stellen weitere kleinere Agenten erforderlich. Ihr Umfang ist deutlich geringer, aber sie vereinfachen die Arbeit mit dem System u.U. erheblich. Neben den hier beschriebenen Agenten besitzt auch die ursprüngliche Realisierung noch zahlreiche andere Agenten, die z.T. an die neuen Gegebenheiten angepaßt werden mußten; eine genaue Beschreibung der einzelnen Änderungen würde hier aber zu weit führen und ist auch nur von rein implementierungstechnischem Interesse. Kurz erwähnt werden sollen hier deshalb nur die neu hinzugekommenen Agenten „InitialisiereProjektstruktur“ und „BearbeiteAttachments“.

5.3.4.1 Der Agent „InitialisiereProjektstruktur“

Da dem Konzept des Projektraumes entsprechend bei Erzeugung eines Projekts zumindest eine Person sowie der Kontextbereich Projektmoderation existieren müssen, liegt es nahe, diese Initialisierungsarbeit in einem eigenen Agenten verfügbar zu machen, der bei Erzeugung eines neuen Projektraumes automatisch einmal auszuführen ist. Dieser Agent greift direkt auf die Modellierungsobjekte „Erzeugung eines neuen Akteurs“ und „Erzeugung eines neuen Kontextbereichs“ zurück, ohne dabei einen Aktivierungszyklus zu starten. Die einzige Information, die dafür benötigt wird, nämlich der Name der als neuer Projektmoderator einzufügenden Person, holt sich der Agent aus dem zuvor ausgefüllten Hauptprofilokument des Projekts.

5.3.4.2 Der Agent „BearbeiteAttachments“

Die Maske bzw. das Formular mit den persönlichen Daten eines Benutzers, wie z.B. Name, Funktion im Projekt usw. wurde um die Möglichkeit erweitert, ein Bild an das Web-Formular anzuhängen, das dann automatisch vom System als das Bild des Benutzers behandelt wird. Dazu wird bei Abschluß der Bearbeitung dieser Agent aufgerufen, der unter Berücksichtigung der Einbettung in die Datenbank die Adresse des ersten angehängten Bildes ermittelt und in ein entsprechendes Verwaltungsfeld des Personeninfoblatts (IBP) einträgt. Von dort holt sich dann der Anfrageagent bei Bedarf die Adresse, um sie an das Applet bzw. die externe Applikation weitergeben zu können, ohne selbst das IBP nach dem richtigen Anhang (engl. *attachment*) durchsuchen zu müssen.

Fehlerbehandlung

Da die Fehlerbehandlung in allen größeren Agenten, sowie in den Modellierungsobjekten gleich realisiert wurde und außerdem einem bestimmten Konzept folgt, soll sie hier kurz beschrieben werden.

Zunächst einmal wurde die Ausnahme ‚ForseenException‘ erstellt, die es ermöglicht, den Fehler in die drei Kategorien Benutzerfehler, struktureller Fehler und kritischer Fehler einzuordnen. Außerdem wurde die Klasse ‚Assert‘ erstellt, die als wesentliche Aufgabe die Überprüfung von Bedingungen und das Auslösen dieser typisierten Ausnahme hat. Diese beiden Klassen werden im Abschnitt Hilfsklassen näher beschrieben.

Bei der Programmierung mit Hilfe des Notes API fällt auf, daß Ausnahmen nur zur Meldung wirklich ernster Fehler genutzt werden. Diese Fehler sollen unberücksichtigt bleiben und werden wie kritische Fehler behandelt. Allerdings müssen im Lauf des Programms viele Werte auf Gültigkeit oder Vorhandensein geprüft werden, was für die Programmausführung einer Unterbrechung gleich kommt. Mit den oben erwähnten Klassen kann die Überprüfung solcher Konsistenzbedingungen wie im folgenden beschrieben erfolgen.

Nach jeder Aktion, deren erwartungsgemäßer Abschluß geprüft werden soll, wird die Methode ‚assert‘ der gleichnamigen Klasse aufgerufen, um die Bedingung zu prüfen. Dabei wird der Typ des etwaigen Fehlers und seine Nummer mitgeliefert. Diese Methode löst dann bei erfolgloser Überprüfung der Bedingung eine Ausnahme aus. Um diese neuen Ausnahmen, sowie natürlich alle anderen zu behandeln, wird das gesamte Programm in einen großen TRY-CATCH-Block eingefaßt, der die Auswertung dieser und aller anderen Ausnahmen gewährleistet. Hier wird dann nach der Schwere des Fehler entschieden, welche Maßnahmen zu treffen sind. In den meisten Fällen ist dies das Protokollieren des Fehlers und die Ausgabe auf der Serverkonsole. Außerdem wird als Rückmeldung an den Benutzer entweder eine Meldung im aktuell angezeigten Dokument veranlaßt (z.B. beim Agent ‚AktAntragBearbeitung‘), oder der Fehler wird in die Antwort auf die POST-Anfrage eingetragen (z.B. beim ‚AnfrageAgent‘). In jedem Fall beendet eine Ausnahme die Programmausführung, allerdings wird abschließend noch versucht das Logbuch zu schließen, um einen geordneten Abbruch zu gewährleisten.

5.4 Die Modellierungsfunktionen

Aufbau und Rechtesystem

Aus mehreren Gründen sind die Modellierungsfunktionen jeweils in einem Objekt gekapselt. Zum einen werden die Modellierungsfunktionen nicht global an einer Stelle benötigt, sondern sowohl in der Initialisierung eines Projekts als auch bei der Modellierung, was es unmöglich macht, sie in ein einzelnes Programm als Unterprozeduren einzubauen. Zum zweiten finden sich Teilaufgaben der Modellierungsfunktionen im Anfrageagent (Modellierungsanfrage) und andere im Agent ‚AktAntragBearbeiten‘ (Aktivierung der Funktion). Außerdem sind die Funktionen mit Initialisierung, Erstellung des Aktivierungsantrags, Verschicken von Benachrichtigungen und vielen weiteren Teilaufgaben sehr umfangreich und sprengen ohnehin den sinnvollen Rahmen einer Prozedur. Nicht zuletzt ist der Pool von Funktionen deutlich einfacher erweiterbar, wenn dafür jeweils ein Objekt programmiert werden kann, ohne sich um andere Programmteile zu kümmern.

Der Aufbau der einzelnen Funktionen ähnelt sich sehr und hätte bei besserer Planung in einer gemeinsamen Oberklasse festgelegt werden können. Jedes der Objekte zu den Modellierungsfunktionen umfaßt zumindest Methoden mit folgenden Funktionalitäten:

- **Konstruktor ohne Aktivierungsdokument** – Die Methode bekommt die Datenbank geliefert, in der die Funktion durchgeführt werden soll und initialisiert das Objekt in sofern, als Zeiger auf die Strukturinformationen der Datenbank ermittelt werden, sowie bestimmte darin enthaltene Informationen, die auf jeden Fall benötigt werden. Dies können z.B. Hauptprofilokument, Zugriffskontrolliste, Projektmoderator und andere sein.
- **Konstruktor mit Aktivierungsdokument** – Diese Methode erfüllt zunächst einmal dieselben Aufgaben, wie der andere Konstruktor, bekommt aber vom Aufrufer noch einen Aktivierungsantrag mitgeliefert und führt die vollständige Initialisierung des Objekts anhand des Aktivierungsantrags durch. Welche Variablen hier initialisiert werden hängt von der Funktion ab, die durchgeführt werden soll.
- **Setzen des Initiators** – Diese Methode bekommt den Namen derjenigen Person geliefert, die diese Funktion initiieren möchte. An dieser Stelle wird geprüft, ob für die Initiierung genug Rechte bestehen. Die Überprüfung wird anhand von Strukturdaten oder Rollen durchgeführt und hängt stark von der durchzuführenden Funktion ab.
- **Setzen des Aktivators** – Diese Methode bekommt den Namen derjenigen Person geliefert, die diese Funktion aktivieren möchte. Es wird anhand der Rollen der Person geprüft, ob genügend Rechte für die Aktivierung vorliegen. Abhängig von der Funktion kann es nötig sein, daß die Person Projektmoderator oder Kontextbereichsmoderator ist.
- **Setzen diverser funktionspezifischer Daten** – Mit Hilfe dieser Methoden werden für jede Funktion spezielle Daten gesetzt. Dies kann im Falle des Hinzufügens eines Akteurs z.B. der Name eben derjenigen Person sein, die hinzugefügt werden soll. Jede Information wird automatisch dahingehend geprüft, ob die auf diese Art schrittweise initialisierte Modellierungsfunktion gültig bleibt. So wird im obigen Fall z.B. geprüft, ob der Akteur nicht schon im Projekt vorhanden ist.
- **Erzeugen des Aktivierungsantrags** – Diese Methode erzeugt den Aktivierungsantrag und füllt dabei alle notwendigen Felder aus.
- **Verschicken der Benachrichtigung** – Der Verantwortliche für die jeweilige Funktion wird benachrichtigt und erhält ein Lesezeichen mit einem Link auf den in der zuvor beschriebenen Methode erzeugten Aktivierungsantrag.
- **Aktivierung der Funktion** – Mit Hilfe dieser Methode wird die zuvor initialisierte Modellierung tatsächlich durchgeführt. Dies ist die einzige Methode, die die Strukturdaten manipuliert und nicht nur ausliest.

Wie man sofort erkennt, ist bei manchen Methoden die Reihenfolge des Aufrufs entscheidend. Dies gilt ganz besonders für die Initialisierungsmethoden, da bei fehlenden Daten nicht überprüft werden kann, ob eine bestimmte Aktion durchgeführt werden darf. Es ist auch offensichtlich, daß der Aufrufer dafür verantwortlich ist zu entscheiden, ob bestimmte Aktionen, wie z.B. die Erstellung eines Aktivierungsantrags für den Aktivierungszyklus, überhaupt ausgeführt werden sollen. Dies ist insbesondere bei der Überprüfung der Rechte des Initiators bzw. Aktivators der Fall und hat neben praktischen Gründen – bei der Projekterstellung gibt es keinen Aktivierungszyklus – auch einen handfesten Grund im Aufbau des gesamten Rechtesystems. In Lotus Domino® gibt es zwei Alternativen für die Rechtevergabe: Ein Agent hat entweder die Rechte desjenigen, der den Agenten gespeichert hat, oder die Rechte des aktuellen Web-Benutzers. Die Strukturdokumente sind schreibgeschützt und Änderungen dürfen nur von einer bestimmten Rolle durchgeführt werden, die bei normaler Nutzung nie einer Person, sondern immer nur speziellen Agenten zugesprochen wird. Da nun aber die Agenten und somit auch die darin verwendeten Modellierungsobjekte auf die Strukturdaten zugreifen müssen, können sie nicht mit den Rechten des Benutzers laufen, da dieser sonst allgemein das Recht zur Manipulation der Strukturdaten haben müßte. Die Agenten, welche die Strukturdaten manipulieren, also „AnfrageAgent“, „AktAntragBearbeitung“ und „InitialisiereProjektstruktur“, werden daher mit den Rechten des Speichernden ausgeführt, der in der Datenbank die Rolle zur Veränderung der Strukturdaten besitzt. Die Agenten ermitteln dann bei Bedarf das Login des Web-Benutzers und geben es mit den entsprechenden Methoden als Parameter an die

Modellierungsobjekte weiter. Aus diesem Grund ist bei der Programmierung der Agenten Vorsicht geboten, denn zumindest hinsichtlich der Strukturdokumente haben sie freie Hand. Im folgenden sollen nun kurz die Besonderheiten der einzelnen Modellierungsobjekte beschrieben werden, sowie die überprüften Bedingungen.

5.4.1 Akteur hinzufügen

Diese Funktion fügt eine im Namens- und Adreßbuch des Servers bekannte Person als Mitglied in das Projekt ein. Sie wird dort als Gast behandelt, solange sie keinem Kontextbereich zugeordnet wird. Dies bedeutet, daß ein neues Personeninformationsblatt mit den bekannten Daten des neuen Mitglieds erzeugt wird. Außerdem wird die Person in die Zugriffskontrollliste der Datenbank aufgenommen und bekommt die Rolle eines Mitglieds.

Bei der Initialisierung wird der Name des neuen Akteurs benötigt. Dieser darf noch nicht Mitglied des Projekts sein. Vorgeschlagen werden darf diese Funktion von jedem Mitglied des Projekts, und aktiviert wird sie vom Projektmoderator.

5.4.2 Akteur entfernen

Dies ist die entgegengesetzte Funktion zur vorherigen. Eine Person, die sich als Gast im Projekt befindet wird aus diesem entfernt. D.h. das Personeninformationsblatt und die Einträge in die Zugriffskontrollliste der Datenbank werden gelöscht.

Bei der Initialisierung wird der Name der zu entfernenden Person benötigt. Einzige Bedingung ist, daß sie nur als Gast im Projekt ist, d.h. sie darf keinem Kontextbereich zugeordnet sein. Der Initiator muß mindestens Mitglied sein, und der Aktivator ist der Projektmoderator.

5.4.3 Akteur zu Kontextbereich hinzufügen

Diese Funktion fügt ein Mitglied des Projekts zu einem der bestehenden Kontextbereiche hinzu. Dazu gehört die Erweiterung der Rechte in der Zugriffskontrollliste und der Eintrag des neuen Kontextbereichs in das Personeninformationsblatt. Umgekehrt muß die Person im Profildokument des Kontextbereichs eingetragen werden.

Bei der Initialisierung werden die Nummer des Kontextbereichs und ein Benutzername gesetzt. Der Kontextbereich darf höchstens sieben Mitglieder besitzen, und das neue Mitglied muß bereits dem Projekt aber nicht diesem Kontextbereich angehören. Der Initiator muß ein Mitglied des Kontextbereichs sein, und der Aktivator ist der Moderator des Kontextbereichs.

5.4.4 Akteur aus Kontextbereich entfernen

Dies ist die entgegengesetzte Funktion zur vorherigen. Ein Mitglied eines Kontextbereichs wird aus diesem entfernt. Sollte die Person nunmehr keinem Kontextbereich mehr angehören, so ist sie als Gast zu behandeln. Dazu werden die erweiterten Rechte zurückgenommen, und der Kontextbereich wird aus dem Personeninformationsblatt entfernt. Umgekehrt wird natürlich der Eintrag für diese Person aus dem Profildokument des Kontextbereichs entfernt.

Bei der Initialisierung werden die Nummer des Kontextbereichs und ein Benutzername gesetzt. Der Kontextbereich muß mindestens zwei Mitglieder besitzen, da der Moderator nicht entfernt werden darf. Die Person muß Mitglied des Kontextbereichs sein, darf aber nicht der Moderator des Kontextbereichs oder einer seiner Wechselwirkungen sein. Hierzu muß zunächst die jeweilige Moderatorrolle an eine andere Person übergeben werden. Initiatoren sind alle Mitglieder des Kontextbereichs und die vorgeschlagene Person, d.h. jeder darf selbst vorschlagen, in einen Kontextbereich aufgenommen zu werden. Aktivator ist der Moderator des Kontextbereichs.

5.4.5 Kontextbereich erzeugen

Die Erzeugung eines Kontextbereichs geht immer mit dem Hinzufügen einer Person einher, da jeder Kontextbereich zumindest seinen Moderator als Mitglied haben muß. Zunächst wird für den neuen Kontextbereich ein Profildokument mit allen Informationen (Position, Name,

Mitglieder usw.) erstellt, das dann im Hauptprofildokument eingetragen wird. Das Hinzufügen des Moderators geschieht analog zum allgemeinen Hinzufügen einer Person zu einem Kontextbereich allerdings müssen der Person erweiterte Rechte eingeräumt werden, da es sich ja um den Moderator handelt.

Bei der Initialisierung müssen für den Kontextbereich Name und Position und für die Person der Name übergeben werden. Die Position muß dabei eine der gültigen Positionen von 1 bis 15 umfassen, und die angegebene Person muß mindestens Mitglied des Projekts sein. Initiatoren sind alle Kontextbereichsmoderatoren (inkl. dem Projektmoderator), und aktiviert wird die Funktion vom Projektmoderator.

5.4.6 Wechselwirkung erzeugen

Die Funktion verknüpft zwei bestehende Kontextbereiche durch eine Wechselwirkung und braucht dazu einen Wechselwirkungsmoderator. Für die Wechselwirkung wird ein eigenes Dokument erzeugt, das neben Zeigern auf die beiden verknüpften Kontextbereiche den Namen des Moderators aufnimmt. Da die Wechselwirkung z.Z. noch keine Funktionalität besitzt außer der Anzeige einer starken Beziehung zwischen Kontextbereichen, müssen auch keine Rechte angepaßt oder andere Aktionen ausgelöst werden.

Bei der Initialisierung werden die beiden Kontextbereiche und der Name des neuen Moderators übergeben. Die Nummern der Kontextbereiche werden daraufhin abgeprüft, ob bereits eine Wechselwirkung besteht, und der Wechselwirkungsmoderator muß in den beiden Kontextbereichen Mitglied sein. Als Initiatoren kommt der Projektmoderator oder jedes Mitglied der beiden zu verknüpfenden Kontextbereiche in Frage. Aktiviert wird die Funktion vom Projektmoderator.

5.4.7 Kontextbereichsmoderator ändern

Diese Funktion dient dazu, die Rolle des Kontextbereichsmoderators an ein anderes Mitglied des Kontextbereichs weiterzugeben. Dazu wird einfach im Profildokument des Kontextbereichs das neue Mitglied als Moderator eingetragen und der alte Moderator wieder als normales Mitglied aufgeführt. Neben dieser Änderung der Strukturdaten müssen natürlich die Rechte in der Zugriffskontrolliste geändert werden.

Bei der Initialisierung muß der Kontextbereich angegeben werden, sowie der Name des neuen Moderators. Einzige Bedingung ist, daß diese Person Mitglied des Kontextbereichs ist. Initiiert wird die Funktion von einem Mitglied des Kontextbereichs oder dem Projektmoderator und aktiviert wird sie ebenfalls vom Projektmoderator.

5.4.8 Wechselwirkungsmoderator ändern

Analog zur Änderung des Kontextbereichsmoderators wird in dieser Funktion die Rolle des Wechselwirkungsmoderators weitergegeben. Dazu wird der neue Moderator in das Strukturdokument der Wechselwirkung eingetragen. Wiederum sind keine weiteren Änderungen erforderlich, weil die Wechselwirkung noch keine tatsächliche Funktionalität besitzt.

Bei der Initialisierung wird die Wechselwirkung über die Nummern der verknüpften Kontextbereiche charakterisiert und der Name des neuen Moderators wird übergeben. Der neue Moderator muß wie der alte in beiden Kontextbereichen Mitglied sein. Initiatoren sind die Mitglieder der beiden Kontextbereiche, sowie der Projektmoderator. Dieser ist auch Aktivator der Funktion.

5.5 Der Aktivierungsantrag

Der Aktivierungsantrag wird notwendig, weil eine Modellierungsanfrage keine direkte Änderung der Strukturdaten bewirkt, sondern nur die Erstellung eines Antrags auf eine solche Änderung. Der folgende Abschnitt beschreibt zunächst kurz, welchen Weg der Antrag in seinem Lebenszyklus durchlaufen kann.

5.5.1 Der Aktivierungszyklus

Wie im vorherigen Abschnitt gesehen können Antragsteller, d.h. Initiator, und genehmigende Person, d.h. Aktivator, sich durchaus unterscheiden. Der Antrag stellt das Bindeglied zwischen den beiden dar. Nachdem der Initiator den Antrag durch den Aufruf der Modellierungsfunktion im Applet erzeugt hat, bleibt ihm nur noch abzuwarten, wie der Aktivator über seinen Antrag entscheidet. Der benachrichtigte Aktivator hat die folgenden drei Alternativen:

1. Er aktiviert den Antrag und sorgt somit dafür, daß die beantragte Modellierungsfunktion tatsächlich durchgeführt wird. Erst durch diesen Vorgang werden irgendwelche Strukturdaten manipuliert. Der Initiator erhält eine Benachrichtigung über den Erfolg seines Antrags und der Antrag wird archiviert.
2. Er lehnt den Antrag ab. Dies resultiert in der Archivierung des Antrags und in einer Benachrichtigung des Initiators über den Mißerfolg seines Antrags.
3. Er schickt den Antrag in den Kommentierungszyklus, was bedeutet, daß er den Antrag einer Reihe von Leuten, die er frei wählen kann, zur Kommentierung vorlegt. Diese Kommentierung kann wiederum auf drei Arten enden:
 - 3.1. Alle angegebenen Kommentatoren geben wunschgemäß ihren Kommentar ab. Der Aktivator erhält nach Eingabe des letzten Kommentars eine Benachrichtigung und hat wiederum die oben angegebenen drei Alternativen.
 - 3.2. Zumindest einer der Kommentatoren überschreitet das vom Aktivator gesetzte Zeitlimit für die Kommentierung. Alle noch ausstehenden Kommentatoren werden benachrichtigt, daß ihr Kommentar nicht mehr nötig wird und der Aktivator bekommt die Möglichkeit erneut eine der drei Alternativen zu wählen.
 - 3.3. Der Aktivator bricht den Kommentarzyklus von Hand ab. Alle ausstehenden Kommentatoren werden benachrichtigt, daß ihr Kommentar inzwischen überflüssig ist und der Aktivator hat wieder die Möglichkeit eine der drei Alternativen für die weitere Bearbeitung zu wählen.

Dieser Lebenszyklus des Aktivierungsantrags ist in dem folgenden Petrinetz schematisch dargestellt (Abbildung 5-3). Der Aktivierungsantrag wird hierbei durch das Token repräsentiert.

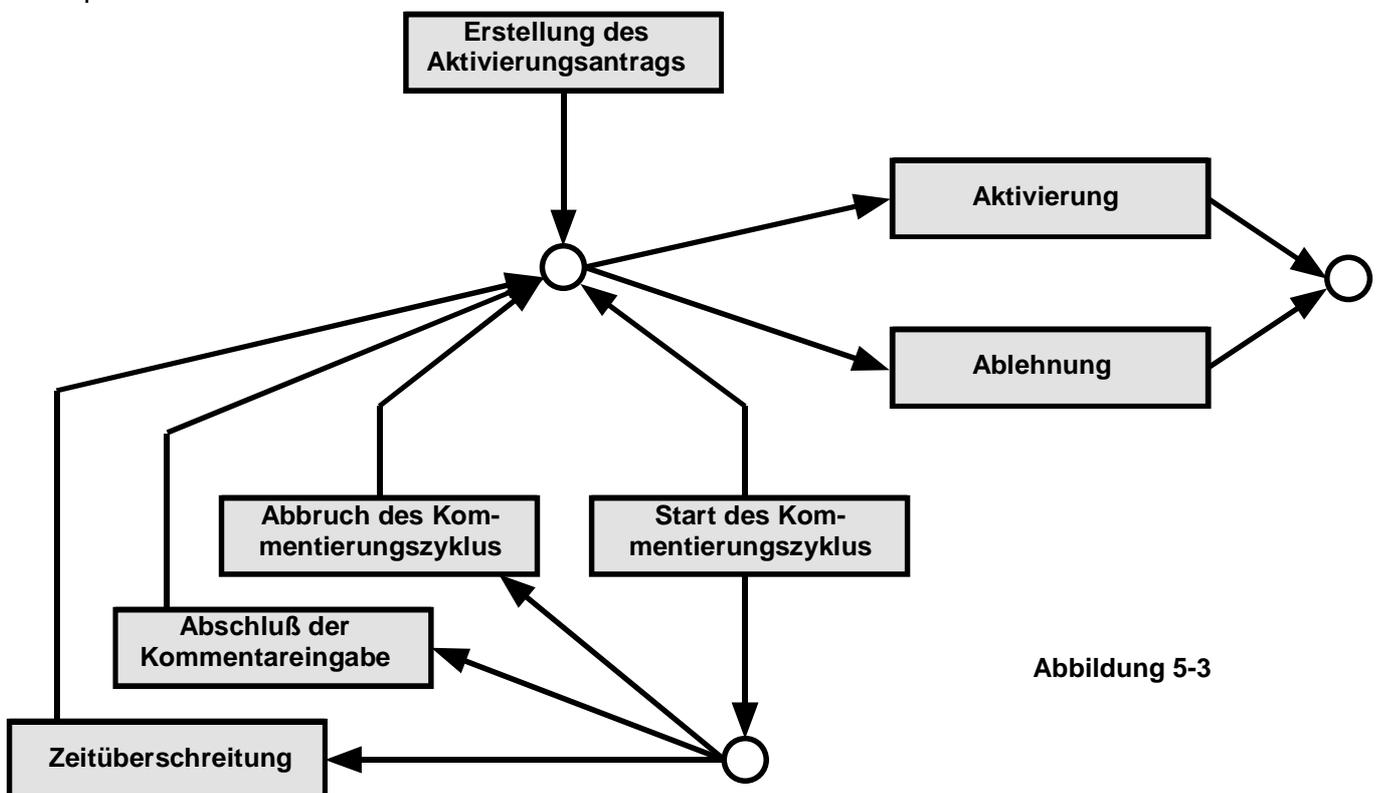


Abbildung 5-3

5.5.2 Die Maske „Aktivierungsantrag“

Der Aktivierungsantrag wird durch ein Lotus Notes® Dokument realisiert. Zu einem solchen Dokument kann unter Lotus Notes® eine Maske definiert werden, die dazu dient, die dort gespeicherten Informationen in ansprechender Form darzustellen. Diese Masken, die gewissermaßen wie eine Lochmaske auf die Daten des Dokuments fungieren, können dabei diverse Aufgaben erfüllen. Eine genaue Beschreibung der vielfältigen Gestaltungsmöglichkeiten würde hier deutlich zu weit führen, aber eine grundlegende Kenntnis der Fähigkeiten ist zum Verständnis der Maske erforderlich und soll kurz vermittelt werden.

Jede Information ist in dem Dokument in einem mit einem Namen versehenen Feld gespeichert. Wird ein Feld mit dem entsprechenden Namen in die Maske eingefügt, so erscheint bei der Betrachtung des Dokuments mit Hilfe dieser Maske der Wert, der im Dokument gespeichert ist. Es gibt zudem die Möglichkeit, Felder in die Maske aufzunehmen, die einen Wert anzeigen, den sie aus einem oder mehreren Feldwerten des Dokuments berechnet haben (z.B. die Summe zweier Felder), oder aber vom System selbst erhalten haben (z.B. den Benutzernamen). Bestimmte Felder des Dokuments können den Zugriff auf Abschnitte des Dokuments sperren oder freigeben (Autorenfelder), außerdem ist es möglich jede einzelne Zeile des Dokuments durch Bedingungen gesteuert zu verstecken oder anzuzeigen. Eine genauere Beschreibung der Funktionalität wird bei der Beschreibung der jeweiligen Felder gegeben.

Die Maske „Aktivierungsantrag“ unterteilt sich grob in die Maske selbst und vier weitere Abschnitte, deren Rechte mit Hilfe von Autorenfeldern einzeln manipuliert werden. Die meisten Felder haben das Kürzel „AA_“ voranstellen, um anzudeuten, daß es sich um Felder des Aktivierungsantrags handelt. Rot und kursiv gedruckte Zeilen sind im allgemeinen unsichtbar, denn sie werden nur dann sichtbar, wenn der aktuelle Benutzer die spezielle Rolle „Debug“ besitzt. Dies erlaubt ein besonders einfaches Testen.

5.5.2.1 Der Dokumentkopf

The screenshot shows a document header area with a dotted border. At the top, there is a line of red, italicized text: "Editoren des Dokuments: AA_autor, eingeloggt als AA_username_anz, mit den Rollen AA_userroles_anz". Below this, there is a grey rectangular box containing the text "Sie sind eingeloggt als:" followed by a field containing "aktuser_anz". To the right of this is the text "Aktivierungsantrag zur Gestaltungsfunktion:" followed by a field containing "AA_art_anz".

Abbildung 5-4

Im Dokumentkopf gibt es eigentlich nur ein Feld, das seinen Wert aus dem Dokument bezieht, nämlich das Feld „AA_autor“. Bei diesem Feld handelt es sich um ein sogenanntes Autorenfeld, das den Zugriff auf das Dokument steuert. Nur in diesem Feld eingetragene Personen oder Rollen können das Dokument verändern. Alle anderen Felder werden zur Anzeige berechnet, was durch die Endung „_anz“ angedeutet wurde. Dabei zeigen „AA_username_anz“ und „aktuser_anz“ den Benutzernamen der eingeloggt Person, „AA_userroles_anz“ die Rollen dieser Person und „AA_art_anz“ berechnet auf Basis des Feldes „AA_fkt“ die textuelle Bezeichnung der Modellierungsfunktion, die durch diesen Aktivierungsantrag beantragt wurde.

5.5.2.2 Der Informationsabschnitt

Editoren des Abschnitts Informationen:

Informationen zum Aktivierungsantrag	ABGELEHNT ANGENOMMEN KOMMENTIERUNG BEARBEITUNG DEFEKT
Initiator:	<input type="text" value="AA_initiator"/>
Beschreibung:	<input type="text" value="AA_aktionsbeschreibung"/>
Begründung:	<input type="text" value="AA_begründung"/>

*** allgemeine Informationen:

Funktion: , Person:

Status des Antrags: , Aktivatorrolle:

UNID des betreffenden Dokuments: (theoretisch nicht mehr nötig)

*** Nur für "Änderung KBModerator", "Person zu KB hinzu" und "Person aus KB weg":

Name des Profildokuments des KBs

*** Nur für "neuen KB":

Neuer KB an Position

*** Nur für "neue WW" und "Änderung WWModerator":

WW zwischen und

Abbildung 5-5

Dies ist der Abschnitt der Maske, der alle Informationen über den Aktivierungsantrag beinhaltet. Alle diese Informationen, außer der berechneten Beschreibung des Modellierungswunsches im Feld „AA_aktionsbeschreibung“, werden vom Erzeuger des Antrags (hier also der „AnfrageAgent“) ausgefüllt. Dies sind im einzelnen: der Initiator des Antrags in „AA_initiator“, die eingegebene Begründung in „AA_begründung“, die Nummer der Modellierungsfunktion in „AA_fkt“, die betroffene Person (z.B. der neue Moderator) in „AA_person“, der aktuelle Status des Antrags (z.B. „in Kommentar“) in „AA_status“ und die Rolle, die der Aktivator innehaben muß, in „AA_aktivatorrolle“. Das Feld „AA_unid“ wurde nur aus Kompatibilitätsgründen übernommen und sollte die eindeutige Identifikationsnummer des betroffenen Dokuments (z.B. des Wechselwirkungsdokuments) enthalten. Die weiteren Felder werden nur in bestimmten Fällen benötigt, was durch den Kommentar deutlich gemacht wird. Dies sind: die eindeutige Identifikationsnummer des Kontextbereichsdokuments in „AA_kbpdunid“, der Name des neuen Kontextbereichs in „AA_newkbname“ und seine gewünschte Position im Projektnavigator in „AA_newkbpos“, sowie die eindeutige Identifikationsnummern der Kontextbereichsdokumente, zwischen denen eine Wechselwirkung eingerichtet werden soll in „AA_wwquelle“ und „AA_wwziel“. Der Status des Antrags wird für den Benutzer durch ein Schlagwort im oberen Teil der Maske angezeigt, hierbei werden durch eine Formel alle ungültigen Schlagworte ausgeblendet, sodaß von diesen fünf immer nur eines angezeigt wird.

5.5.2.3 Der Kommentarabschnitt

Editoren des Abschnitts Kommentare: AA_komautor_anz

Kommentare

eingegeben von: AA_komeingegeben

noch erwartet von: AA_komerwartet

bisherige Kommentare: AA_kommentare

Abbildung 5-6

Dieser Abschnitt ist wie der vorherige für Benutzer nur lesbar. Für Tests wird das Autorenfeld „AA_komautor“ in dem Feld „AA_komautor_anz“ angezeigt. Nach Start des Kommentarzyklus können sich die Benutzer im Feld „AA_komeingegeben“ darüber informieren, wer Kommentare eingegeben hat, und im Feld „AA_komerwartet“ darüber, von wem noch einen Kommentar gewünscht wird. Die jeweiligen Kommentare befinden sich dann in der richtigen Farbe im Feld „AA_kommentare“, wo sie vom Agent „AktAntragBearbeitung“ eingetragen werden.

5.5.2.4 Der Aktivierungsabschnitt

Editoren des Abschnitts Aktivierung: AA_aktautor_anz

Aktivierung

Aktion: AA_aktion

Kommentierung: Kommentare von: AA_kommentatoren

AA_komzeitbeschr

bis: AA_zeitschranke

Abbildung 5-7

Dieser Abschnitt kann nur vom Aktivator und nur im Status „in Bearbeitung“ manipuliert werden, was über das Autorenfeld „AA_aktautor“ gesteuert wird. Für Tests wird dieses Feld in dem zur Anzeige berechneten Feld „AA_aktautor_anz“ sichtbar gemacht. Das Optionsfeld „AA_aktion“ ermöglicht es dem Aktivator eine der drei Aktionen „ablehnen“, „aktivieren“ und „Kommentierungszyklus starten“ auszuwählen. Das Feld wird dann genutzt, um dem Agenten „AktAntragBearbeitung“ die benötigte Aktion mitzuteilen. Das Feld „AA_kommentatoren“ kann der Aktivator mit den Namen der Personen füllen, von denen er einen Kommentar wünscht. Über das Feld „AA_komzeitbeschr“, das nur angekreuzt werden kann, kann er eine die Zeitkontrolle aktivieren. Diese nutzt wiederum das Feld „AA_zeitschranke“, welches den spätesten Zeitpunkt festlegt, zu dem die Kommentare eingetroffen sein müssen.

5.5.2.5 Der Kommentierungsabschnitt und der Dokumentfuß

Editoren des Abschnitts Kommentareingabe: AA_komeingautor_anz

Kommentareingabe

Kommentar: AA_komeingabe

Tenor: AA_komtenor

Aktion aktivieren

Kommentierung abbrechen

Kommentar einreichen

Abbildung 5-8

Wie bei den beiden vorherigen Abschnitten dient das Feld „AA_komeingautor_anz“ dazu, den Inhalt des Autorenfeldes „AA_komeingautor“ für Tests sichtbar zu machen. Dieser Abschnitt wird bei Bedarf für die vom Aktivator angegebenen Kommentatoren freigegeben, die in dem Feld „AA_komeingabe“ ihren Kommentar eingeben können und in dem Feld „AA_komtenor“ eine der drei Optionen „neutral“, „positiv“ und „negativ“ auswählen können, um ihren Kommentar noch zu bewerten. Dies wird dazu genutzt, den Kommentar in der richtigen Farbe (schwarz, grün oder rot) in die Kommentarliste aufzunehmen.

Die drei Schaltflächen gehören nicht zum Abschnitt, sondern zum Hauptteil der Maske. Je nach Status und Benutzer können dabei eine oder mehrere dieser Schaltflächen angezeigt werden. Die Schaltflächen starten allesamt den Agenten „AktAntragBearbeitung“, übergeben ihm aber unterschiedliche Aktionen zur Ausführung. Die Schaltfläche „Aktion aktivieren“, die nur dem Aktivator im Status „in Bearbeitung“ angezeigt wird, erlaubt ihm, die im Feld „AA_aktion“ spezifizierte Aktion anzustoßen. Die Schaltfläche „Kommentierung abbrechen“, die ebenfalls nur dem Aktivator aber diesmal im Status „in Kommentar“ angezeigt wird, erlaubt ihm, den Kommentierungszyklus geordnet abzubrechen. Die Schaltfläche „Kommentar einreichen“ hingegen ist für die Kommentatoren im Status „in Kommentar“ sichtbar und sorgt für ein Umtragen des eingegebenen Kommentars in die Liste der Kommentare.

5.6 Die allgemeinen Hilfsklassen

Einige der Aufgaben, die im Laufe der Entwicklung aufgetreten sind, sind so allgemein und in sich abgeschlossen, daß Sie in eigenen Klassen realisiert wurden. Diese Klassen mit hohem Wiederverwendungswert sollen im folgenden kurz beschrieben werden.

5.6.1 Die Klasse „StringDecoder“

Wie bereits in der Beschreibung des Anfrageagenten erläutert, wird einem Lotus Notes® Agenten der Inhalt der POST-Anfrage in einem Textfeld des Umgebungsdokuments übergeben. Der Inhalt der POST-Anfrage ist per Definition so kodiert, wie es der MIME-Typ „application/x-www-form-urlencoded“ fordert. Dies wurde von uns so definiert, weil dadurch eine vom Applet unabhängige Abfrage der Datenbankschnittstelle über die Standard-Eingabemasken des WWW-Browsers erfolgen kann, was vor allem beim Testen Früchte trägt. Die Codierung sieht vor, daß der Schlüssel von seinem Wert mit einem Gleichheitszeichen und jedes solche Paar durch das Zeichen „&“ vom nächsten getrennt wird (s. Abschnitt 3.3). Jeder Schlüssel und jeder Wert ist so kodiert, wie es die Methode „encode“ der Java-Standardklasse „URLEncoder“ tut. Der Inhalt muß zur weiteren Verwendung also zunächst zurückkonvertiert werden. Hierfür steht leider keine Java-Standardklasse zur Verfügung.

Die „StringDecoder“ genannte Klasse erweitert die im allgemeinen Teil beschriebene abstrakte Klasse „Decoder“ um den Konstruktor und um die Methode „getHashtable“. Der Konstruktor nimmt den String entgegen und checkt ihn zur weiteren Verwendung in einen Java-StringTokenizer ein. Dieser wird dabei so konfiguriert, daß er das Zeichen „&“ sowie Zeilenwechsel als Trennzeichen erkennt. Die Methode „getHashtable“ nutzt den so konfigurierten StringTokenizer dazu, die Tokens aus dem String zu extrahieren und spaltet sie beim Gleichheitszeichen auf. Die erste Hälfte wird mit Hilfe der Methode „decode“ aus der Oberklasse dekodiert und als Schlüssel in die Hashtabelle aufgenommen. Analog wird die zweite Hälfte dekodiert und als Wert unter diesem Schlüssel eingetragen. Nach diesem Vorgang stehen also dem Benutzer die Werte der POST-Anfrage extrem komfortabel zur Verfügung, da sie direkt über den Schlüssel aus der Hashtabelle ermittelt werden können.

5.6.2 Die Klasse „RequestReply“

Wie kurz angedeutet muß ein Lotus Notes® Agent seine Ausgabe an den Erzeuger der POST-Anfrage auf einen Java-PrintStream schreiben. Da aber nun das Applet die Rückgabedaten ebenfalls im MIME-Typ „application/x-www-form-urlencoded“ erwartet, liegt es nahe, auch die Aufbereitung der Rückgabedaten in einer Klasse zu kapseln.

Diese Aufgabe übernimmt die Klasse „RequestReply“, die neben dem Konstruktor die Methoden „clear“, „checkIn“ und „toOutput“ besitzt. Die gesamte Rückgabe wird in einem Container vom Typ String verwaltet, der durch den Konstruktor erzeugt wird. Paare von Schlüssel und Wert werden durch Aufruf der Methode „checkIn“ in diesen Container aufgenommen, wobei automatisch eine Codierung mit Hilfe der Java-Standardklasse „URLEncoder“ vorgenommen wird. Sollte dies nötig sein – z.B. im Fehlerfall – kann der Container mit der Methode „clear“ wieder geleert werden. Die Ausgabe des gesamten Containerinhalts auf einen Strom wird durch die Methode „toOutput“ realisiert, die als Parameter den entsprechenden Ausgabestrom erwartet.

5.6.3 Die Klasse „UserInfo“

Das Login eines Benutzers wird dem Lotus Notes® Agenten über ein Textfeld im Umgebungsdokument verfügbar gemacht. Da nun aber der Web-Server nicht nur ein einziges Login pro Benutzer akzeptiert, sondern eine ganze Vielzahl, wie z.B. vollständiger Name einmal mit und einmal ohne Organisation oder selbstdefiniertes Kürzel, muß zu diesem Login noch der eindeutige Name ermittelt werden. Dieser steht im Namens- und Adreßbuch des Servers und kann von dort geholt werden.

Die Klasse „UserInfo“ kapselt diese elementare Funktionalität. Sie besteht nur aus dem Konstruktor und einer selbstdefinierten Ausnahme. Beim Aufruf wird das Login und ein Zeiger auf die aktuelle Sitzung an den Konstruktor übergeben. Dieser ermittelt daraus die Liste der öffentlichen Namens- und Adreßbücher des Servers und versucht das Login in jedem dieser Adreßbücher in einer speziellen, dafür vorgesehenen Ansicht zu finden. Hat er Erfolg, wird neben einem Objekt vom Typ „Name“ auch das Login und ein Zeiger auf das Dokument im Adreßbuch in der Klasse „UserInfo“ verwaltet. „Name“ ist eine Notes-Klasse, die einen vereinfachten Zugriff auf verschiedene Schreibweisen eines Namens erlaubt, wie z.B. vollständiger Name oder Name inklusive Organisation im kanonischen Format. Konnte das Login nicht gefunden werden, wird die oben erwähnte Ausnahme mit dem Namen „NameNotFoundException“ ausgelöst. Dies kann natürlich bei der Überprüfung des Logins des eingeloggten Benutzers nicht passieren, denn ansonsten hätte er sich ja nicht einloggen können, aber auf diese Weise erlaubt die Klasse allgemein nach Logins bzw. verschiedenen Schreibweisen eines Namens zu suchen.

5.6.4 Die Klasse „ContextInfo“

Weitere ebenso grundlegende Funktionalitäten wie die Überprüfung des Benutzers, die in praktisch jedem Agent benötigt werden, sind die Initialisierung eines Logbuchs, sowie die Ermittlung von Zeigern auf die aktuelle Sitzung, das Umgebungsdokument, die aktuelle Datenbank und den Ausgabestrom. Alle diese elementaren und trivialen Aufgaben werden in

einer Klasse gekapselt, um sich nicht bei der Programmierung eines Agenten jedesmal von Neuem um die Details kümmern zu müssen.

Die Klasse „ContextInfo“ besteht ähnlich wie die Klasse „UserInfo“ nur aus dem Konstruktor. Dieser Konstruktor ermittelt aus dem übergebenen Zeiger auf das Agentenobjekt alle oben beschriebenen Zeiger und speichert sie in entsprechenden öffentlichen Variablen. Außerdem öffnet er das Logbuch für den Agenten und erzeugt mit Hilfe des Logins aus dem Umgebungsdocument ein Objekt vom Typ „UserInfo“. Somit stehen nach Ausführung des Konstruktors alle typischerweise bei der Programmierung eines Agenten erforderlichen Informationen zur Verfügung.

5.6.5 Die Klasse „ForseenException“

Um eine differenzierte Fehlerbehandlung durchzuführen, kommt wohl kein Java-Programm darum herum, eigene Ausnahmen zu definieren. Um nun aber keine endlose Menge von neuen Ausnahmen zu definieren, wurde hier eine einzige neue Ausnahme deklariert, die mehrere Möglichkeiten zur genaueren Beschreibung des Fehlers bietet.

Zunächst werden von dieser Ausnahme drei Fehlertypen unterstützt:

- USER - Trat der Fehler bei der Überprüfung einer Eingabe des Benutzers auf und lag der Fehler aller Wahrscheinlichkeit nach bei ihm, so sollte die Ausnahme diesen Typ besitzen. Beispiel: ein mißachteter Wertebereich.
- STRUCT - Trat der Fehler auf, als geprüft wurde, ob irgendeine Strukturinformation vorhanden oder korrekt ist, kann im Fehlerfall von fehlerhaften Strukturdaten ausgegangen werden, was dieser Fehlertyp andeutet. Beispiel: ein fehlendes Feld in einem Strukturdokument.
- CRITICAL - Trat der Fehler auf, als eine Überprüfung erfolgte, deren Scheitern auf kritische Probleme hinweist, so wird dieser Fehlertyp verwendet. Beispiel: das Scheitern der Speicherung eines Dokuments.

Neben den Fehlertypen kann dann, wie bei Ausnahmen üblich, durch eine Fehlermeldung der Fehler näher erläutert werden. Dies erlaubt es zwar nicht, im Programm auf ganz spezielle Fehler zu reagieren, da eine Kategorisierung nach der Fehlermeldung sehr unsauber wäre, aber der Detaillierungsgrad reicht für die allermeisten Fälle aus. Der Vorteil einer einzigen Fehlerklasse liegt darin, daß alle vorhergesehenen Fehler durch Abfangen dieser Ausnahme behandelt werden können, sofern sie wie hier praktisch nur ausgegeben und nicht tatsächlich einzeln behandelt oder gar behoben werden sollen. Alle anderen Fehler kann man zumindest als Lücken in der Programmierung auffassen, und sie sollten eingehender geprüft werden.

Die Klasse „ForseenException“ ist eine Erweiterung der Java-Standardklasse „Exception“ und stellt mehrere Konstruktoren zur Verfügung, die eine Entgegennahme des Fehlertyps vorsehen. Im wesentlichen bestehen deren Aufgaben aber nur in der Speicherung des Typs und dem Aufruf des Konstruktors der Oberklasse.

5.6.6 Die Klasse „Assert“

Um der Tatsache gerecht zu werden, daß extrem häufig eine einfache Bedingung geprüft wird, deren Scheitern nur das Auslösen einer Ausnahme zur Folge hat, wurde diese Klasse eingeführt. Sie hat die Aufgabe, die Überprüfung von Bedingungen und das Auslösen der entsprechenden Ausnahme im Quellcode zu verkürzen und zu vereinheitlichen. Es wäre damit auch möglich, durch einen Präprozessor diese nun einheitlich formulierten Abfragen herauszufiltern, und zwar sogar für jeden Fehlertyp einzeln. Sie unterstützt nämlich genau die Fehlertypen, die in der Klasse „ForseenException“ beschrieben wurden.

Die angestrebten Aufgaben wurden durch eine zentrale statische Methode „assert“ realisiert, die eine zu prüfende Bedingung, den Fehlertyp, die Fehlernummer und bei Bedarf weitere Informationen zum Fehler (z.B. Name des fehlenden Feldes) erhält. Die Bedingung wird geprüft und bei Scheitern wird eine Ausnahme vom Typ „ForseenException“ generiert, die den übergebenen Fehlertyp besitzt. Die Fehlernummer wird zusammen mit den u.U. übergebenen weiteren Informationen in eine intuitive Fehlermeldung übersetzt, die ebenfalls bei der Erzeugung der Ausnahme mitgeliefert wird.

5.6.7 Die Klassen „DatabaseRes“, „ErrorMessagesRes“ und „UserMessagesRes“

Da beim Zugriff auf die Dokumente der Datenbank nahezu alle Dokumente und Felder über ihren Namen beschrieben werden, der ihnen beim Entwurf der Datenbank zugewiesen wurde, ist es sinnvoll diese Informationen vom eigentlichen Programm zu entkoppeln. Auch die Fehlermeldungen und Statusmeldungen für den Benutzer sollten sinnvollerweise ausgelagert werden, um z.B. eine Übersetzung in andere Sprachen zu ermöglichen. Zu dieser Entkopplung dient die Java-Standardklasse „ResourceBundle“, die es ermöglicht, Strings unter einem bestimmten Schlüssel abzulegen.

Die Klasse „DatabaseRes“, die wie die anderen beiden auch die Klasse „ListResourceBundle“ erweitert, enthält dabei alle Informationen, die sich mit der Struktur der Datenbank beschäftigen, d.h. Maskennamen, Namen von Profildokumenten, Feldnamen usw.. Die Informationen wurden des Überblicks wegen nach den Masken kategorisiert.

Die Klasse „ErrorMessagesRes“ nimmt, wie der Name schon sagt, die Fehlermeldungen auf, die dann von der Klasse „Assert“ benutzt werden, um sie bei der Erzeugung einer Ausnahme anzugeben. Die Fehlermeldungen sind auch hier nach den drei Fehlertypen kategorisiert.

Die Klasse „UserMessagesRes“ nimmt alle Meldungen auf, die zu irgendeinem Zeitpunkt an den Benutzer weitergegeben werden, wie zum Beispiel bei der Erstellung der Benachrichtigungen.

5.7 Bemerkungen

5.7.1 Replizierung

Alle Adressen von Masken, Ansichten und Dokumenten, die den Datenbanknamen oder den Servernamen enthalten, werden dynamisch berechnet. Dies ist die Grundlage für die Replizierung, bei welcher der Datenbestand durch Kopie auf einen anderen Rechner übertragen wird. Ohne die dynamische Berechnung gewisser Adressen würden z.B. die URLs der in den Personeninformationsblättern gespeicherten Bilder bei der Replizierung falsche Werte bekommen.

5.7.2 Integration des Applets

Da die Datenbank als Ganzes auf einen anderen Rechner replizierbar sein soll, ist auch das HTML-Dokument mit dem eingebetteten Applet Teil der Datenbank, genau wie alle Bilddateien, die dieses benötigt. Dies ist dadurch möglich, daß dem Lotus Domino® Server über die Eigenschaften der Datenbank mitgeteilt werden kann, welches Dokument bei Öffnen der Datenbank gestartet werden soll, was in diesem Fall das Dokument mit dem Applet ist. Die Parameter aus der URL werden dabei in das APPLET-Tag eingefügt und stehen dem Applet somit als normale Parameter zur Verfügung. Ein Rücksprung zum Applet wird folglich immer durch einen Sprung zum ersten Dokument der Datenbank realisiert.

5.8 Zusammenfassung

Nachdem nun alle Einzelteile des Systems vorgestellt wurden, sollen diese noch einmal durch das vollständige Objektmodell in Zusammenhang gebracht werden. Dieses ist in Abbildung 5-9 dargestellt.

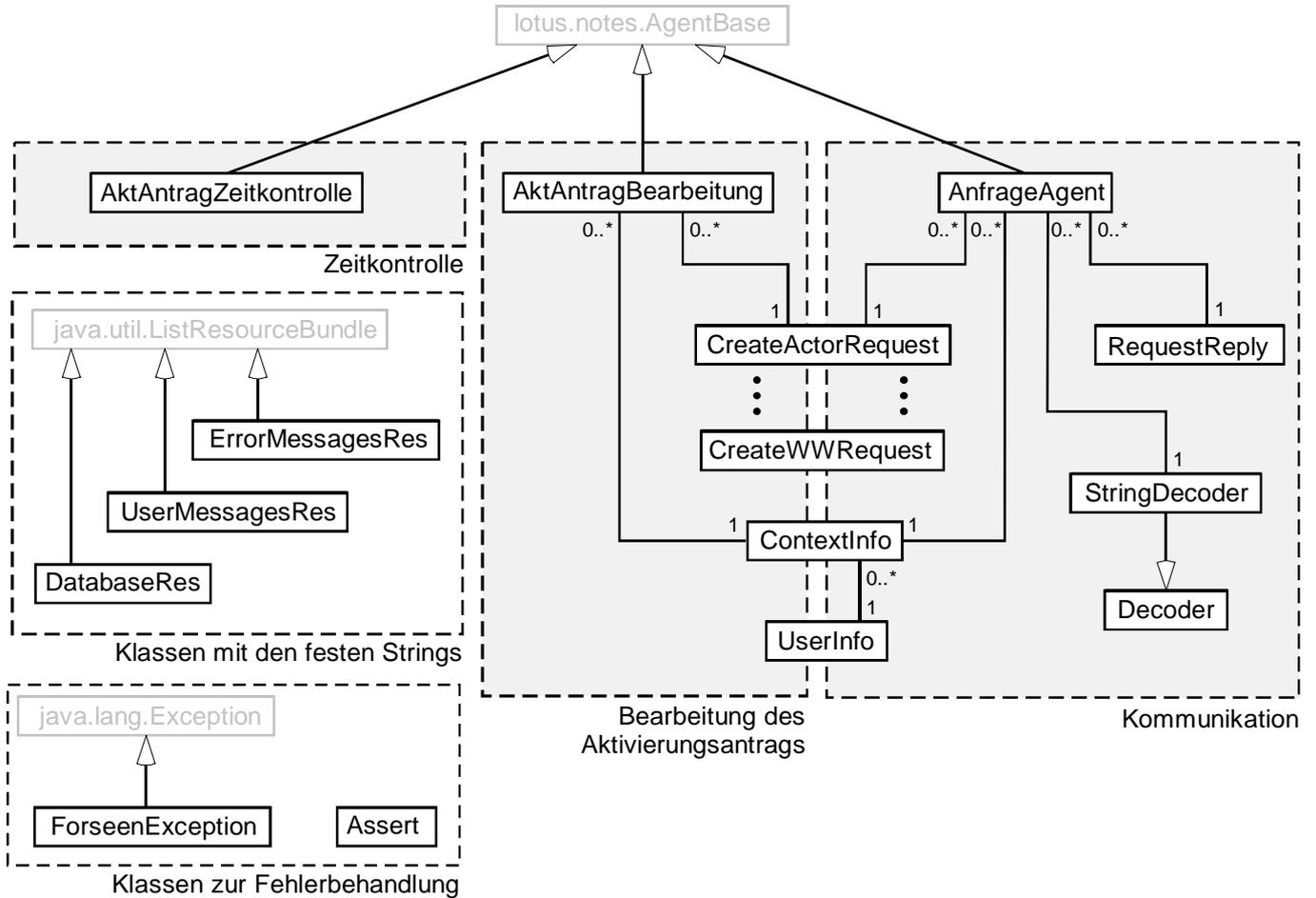


Abbildung 5-9

6 Literaturverzeichnis

- David Flanagan – Java in a Nutshell
O'Reilly Verlag 1998, ISBN 3-89721-100-9
- Klaus Fochler, Primoz Perc, Jörg Ungermann – Lotus Domino 4.6
Addison-Wesley Longman Verlag 1998, ISBN 3-82731-337-6
- Handbücher zu Lotus Domino 4.6
Lotus Development Corporation 1997
- K. C. Hopson, Stephen E. Ingram - Developing professional Java applets
Indianapolis, Ind. 1996, ISBN 1-57521-083-5
- Arthur van Hoff, Sami Shaio, Orca Starbuck - Hooked on Java: creating hot Web sites
with Java applets
Addison-Wesley Longman Verlag 1996, ISBN 0-201-48837-X
- <http://www.javasoft.com>, WWW
Offizielle Seite von Sun Microsystems Inc.
- <http://www.javaworld.com>, WWW
IDG's magazine for the Java community
- RFC1521, RFC1522
MIME Part One & Part Two