

Prüfer: Prof. Dr.-Ing. Heinz Ebert, Institut für Informatik  
Betreuer: Dipl.-Inf. Mathias Ocker  
Beginn am: 01. August 1997  
Beendet am: 23. Dezember 1997  
CR-Klassifikation: D.2.2

Diplomarbeit-Nr.: 1567

Entwurf und Entwicklung eines  
modellbasierten World Wide Web  
Interface einer Service Request und  
Autorisations-Applikation

Volker Kren

Universität Stuttgart  
Institut für Informatik  
Breitwiesenstr. 20/22  
70565 Stuttgart

---

# Inhaltsverzeichnis

## **Kapitel 1: Einführung.....5**

1.1	Motivation .....	5
1.2	Gegenstand der Arbeit.....	6
1.3	Aufbau der Arbeit .....	7
1.4	Danksagung.....	7
1.5	Kurzfassung.....	7

## **Kapitel 2: Anforderungen .....9**

2.1	Überblick über Service Request Applikation.....	9
2.1.1	Grundfunktionalität und Architektur.....	9
2.1.2	Scheduler .....	10
2.1.3	Ausführung von Service Requests.....	11
2.1.4	GUI-Server .....	13
2.1.5	Datenstrukturen .....	13
2.2	Benutzeranforderungen .....	13
2.2.1	Objektrepräsentation und Objekthierarchie.....	13
2.2.2	Aktualität der Daten .....	14
2.2.3	Bildschirmaufteilung und Navigation .....	15
2.2.4	Konzept der Views.....	16
2.2.5	Benutzerkonzept .....	17
2.2.6	Filtermechanismen .....	18
2.2.7	Aliasing der Objektdarstellung.....	18
2.2.8	Aktionen .....	18
2.2.9	Benutzerführung .....	20
2.2.10	Hilfesystem.....	20
2.2.11	Verfügbarkeit .....	20
2.2.12	Zusammenfassung .....	21

---

2.3	Konkretisierung an User/Security Applikation .....	21
2.3.1	Konzept der User/Security Applikation .....	21
2.3.2	Prototyp der User/Security Applikation .....	23
2.3.3	Views für User/Security Applikation .....	23
2.3.4	Benutzung des Darstellungs-Aliasing .....	24
2.3.5	Funktionalität.....	24
 <b>Kapitel 3: Design .....</b>		<b>26</b>
3.1	Realisierung der Minimalfunktionalität .....	26
3.1.1	Interne Objektrepräsentation .....	26
3.1.2	Erstellung von Service Requests .....	27
3.1.3	Navigationsbereich .....	34
3.1.4	Informationsbereich.....	35
3.1.5	Kommunikation mit Applikationsserver .....	35
3.2	Verbesserung der Minimalfunktionalität.....	37
3.2.1	Applikationsunabhängige Objektgliederung .....	37
3.2.2	Verwaltungsbaum für Navigationsbaum .....	42
3.2.3	Lokaler Cache.....	46
3.2.4	Notifikations-Mechanismen .....	48
3.2.5	Erkennen des Darstellungs-Aliasing .....	52
3.2.6	Drag&Drop Aktionen.....	53
3.2.7	Rahmenfunktion zur Objektmarkierung.....	54
3.2.8	Verschlüsselung .....	54
3.2.9	Navigations- und Informationsmodus .....	55
3.2.10	Filtermechanismen .....	56
3.2.11	Hilfesystem.....	56
3.2.12	Scheduling .....	57
3.2.13	Zusätzliche Protokollelemente .....	57
3.2.14	Plattformunabhängige Verfügbarkeit.....	58
3.2.15	Auswahl der Implementierungssprache .....	59
 <b>Kapitel 4: Implementierung .....</b>		<b>60</b>
4.1	Objekte der Benutzungsoberfläche .....	60
4.1.1	Zusammenfassung der Aufgaben .....	60
4.1.2	Aufgabenverteilung .....	60
4.1.3	Schnittstellen .....	62

---

4.2	Entwicklungsumgebung .....	63
4.2.1	Anforderungen.....	63
4.2.2	Kriterienkatalog.....	63
4.2.3	Mögliche Kombinationen.....	64
4.2.4	Evaluation.....	64
4.2.5	Auswahl.....	65
4.3	Implementierte Funktionalität des Prototyps .....	66
4.3.1	Cache Manager.....	67
4.3.2	Display Manager .....	68
4.3.3	Kommunikations-Manager.....	70
4.3.4	Action Manager .....	71
4.3.5	Hilfe Manager.....	71
<b>Kapitel 5: Rück- und Ausblick. ....</b>		<b>73</b>
5.1	Ergebnisse .....	73
5.2	Projektverlauf.....	74
5.3	Offene Punkte.....	75
5.4	Ausblick .....	77
<b>Anhang A: Literaturverzeichnis .....</b>		<b>78</b>
<b>Anhang B: Glossar .....</b>		<b>80</b>
<b>Anhang C: Selbständigkeitserklärung .....</b>		<b>84</b>

# Abbildungsverzeichnis

Abbildung 1	Screenshot von Prototyp der Benutzungsoberfläche .....	8
Abbildung 2	Benutzungsoberfläche als Client des Service Request Applikationsservers .....	10
Abbildung 3	Vorgangnetz zur Autorisierung eines Service Requests .....	12
Abbildung 4	Skizzierte Bildschirmdarstellung.....	15
Abbildung 5	Zusammenhang zwischen Benutzungsoberfläche, Bäumen, Views und Objekten .....	16
Abbildung 6	Kontextbezogene Hilfe .....	20
Abbildung 7	Beispiel einer Rollentemplatehierarchie.....	22
Abbildung 8	Übersicht über Objekte und ihre Aktionen.....	25
Abbildung 9	Erstellung von Service Requests .....	28
Abbildung 10	Vor- und Nachteile der Ansätze zur Dialogsteuerung.....	30
Abbildung 11	Erstellung von Service Requests mit zusätzlichem Dialog .....	33
Abbildung 12	Kommunikationsprotokoll für Queries.....	36
Abbildung 13	Protokollelemente “Benutzungsoberfläche an Applikationsserver” für Minimalfunktionalität .....	37
Abbildung 14	Protokollelemente “Applikationsserver an Benutzungsoberfläche” für Minimalfunktionalität .....	37
Abbildung 15	Beispiel eines Verwaltungsbaumes .....	42
Abbildung 16	Beispiel eines Navigationsbaumes .....	43
Abbildung 17	Implementierungsvariante für Query Cache.....	47
Abbildung 18	Implementierungsvariante für Objekt Cache.....	48
Abbildung 19	Aktualisieren des Query Cache bei Hinzufügen eines Objekts.....	51
Abbildung 20	Zusätzliche Protokollelemente “Benutzungsoberfläche an Applikationsserver” .....	57
Abbildung 21	Zusätzliche Protokollelemente “Applikationsserver an Benutzungsoberfläche” .....	57
Abbildung 22	Grobstruktur der Benutzungsoberfläche.....	61
Abbildung 23	Attribute des Cache Managers.....	67
Abbildung 24	Konstruktor des Cache Managers.....	67
Abbildung 25	Klassenmethoden des Cache Managers.....	67
Abbildung 26	Instanzen-Methoden des Cache Managers .....	67
Abbildung 27	Attribute des Display Managers .....	68
Abbildung 28	Konstruktor des Cache Managers.....	68
Abbildung 29	Instanzen-Methoden des Cache Managers .....	69
Abbildung 30	Attribute des Kommunikations-Managers .....	70
Abbildung 31	Konstruktor des Kommunikations-Managers.....	70
Abbildung 32	Klassenmethoden des Kommunikations-Managers .....	70
Abbildung 33	Instanzen-Methoden des Kommunikations-Managers .....	70
Abbildung 34	Attribute des Action Managers.....	71
Abbildung 35	Instanzen-Methoden des Action Managers .....	71
Abbildung 36	Attribute des Hilfe Managers .....	71
Abbildung 37	Instanzen-Methoden des Hilfe Managers .....	72
Abbildung 38	Projektplan.....	74

# EINFÜHRUNG

---

Our ability to build complex networking infrastructures and applications usually outstrips our ability to manage them.

John McConnell, 1996

## 1.1 Motivation

---

Das Internet hat in den letzten Jahren immer neue Wachstumsrekorde verzeichnet. Immer mehr Firmen präsentieren neben firmennahen Informationen Produktübersichten und Dokumentationsmaterial. Vor allem Software-Firmen sind in diesem Sektor sehr stark vertreten, da sie über das nötige Wissen und die Ausrüstung verfügen.

Ebenso wie das Internet selbst wuchsen Firmen wie Pilze aus dem Boden, die Software auf den Markt warfen, um mit dem neuen Medium Internet umgehen zu können. Das wichtigste Softwareprodukt zur Bedienung des Internet ist der sogenannte Browser. Er stellt die im Internet auf Servermaschinen abgelegten Seiten dar, die in der Sprache HTML (Hyper Text Markup Language) geschrieben sind. Dynamik entsteht dadurch, daß es auf den Seiten sensitive Stellen gibt, mit deren Hilfe der Benutzer zu anderen Seite verzweigen kann. Diese Verknüpfungen (engl. Links) heißen Hyperlinks. Durch sie wird Navigation von Seite zu Seite, von Server zu Server möglich. Die Browser bauen auf einen bestimmten Dienst des Internets (Hyper Text Transfer Protocol, HTTP) auf und spannen dadurch ein weltweites Informationsnetz auf, das World Wide Web (WWW).

Zunächst war das World Wide Web ein Informationsnetz, von dem nur Informationen gelesen werden konnten. Dies liegt vor allem daran, daß der Sprachumfang von HTML nur einen Lesezugriff vorsieht. In HTML können Texte und Bilder definiert werden, nicht aber komplizierte Mechanismen, die zum Einlesen und Verarbeiten von Daten benötigt würden. Die ab 1992 von der Firma Sun entwickelte Sprache Java bietet nun die Möglichkeit, diese Barriere zu überwinden. Bei Java handelt es sich um eine objektorientierte Programmiersprache, die in die in HTML geschriebene Seiten eingebunden werden kann. Natürlich mußten die Internetbrowser darauf eingerichtet werden, jetzt nicht mehr nur HTML, sondern auch Java zu verarbeiten.

Java besitzt eine mit C und C++ verwandte Sprachsyntax, wird aber, im Gegensatz zu C und C++, interpretiert und nicht übersetzt. Die Interpretation und damit die Ausführung des Programms übernimmt die sogenannte Java Virtual Machine (JVM). Auf fast allen Betriebssystemen gibt es bereits eine Implementierung der JVM, wodurch Java zu einer plattformunabhängigen Entwicklungssprache avancierte. Insbesondere ist in den gängigen Internetbrowsern eine JVM implementiert.

Durch die Kombination der Faktoren Java und JVM in Internetbrowsern, verbunden mit dem Versprechen der Betriebssystemhersteller, ihren Internetbrowser bereits mit dem Betriebssystem auszuliefern, wuchs die Idee, eine plattformunabhängige Benutzungsoberfläche zu entwickeln. Vor allem große Firmen, bei denen eine Vielzahl von Betriebssystemen eingesetzt wird, waren begeistert von der Tatsache, von nun an eine Benutzungsoberfläche nur noch einmal zu schreiben, nur noch einmal zu testen und anschließend auf jeder Plattform zur Verfügung zu haben.

## 1.2 Gegenstand der Arbeit

---

Genau an dieser Stelle setzt meine Diplomarbeit an. Es gilt, eine Benutzungsoberfläche zu entwickeln, mit der eine Service Request Applikation bedient werden kann. Sie soll über das World Wide Web bedient werden können, also über einen Internetbrowser. Eine Service Request Applikation ist eine Applikation, die nach Autorisierung Service Requests ausführt. Service Requests sind hierbei Aufträge zur Modifikation von Applikationsobjekten.

Visualisiert werden dynamisch veränderliche Objekte und Objektmodelle der Service Request Applikation. Daneben soll auch die Erstellung von Service Requests, die diese Objekte und Objektmodelle verändern, durch die Benutzungsoberfläche abgewickelt werden. Am Ende dieser Arbeit soll ein als Prototyp lauffähiges System zur Verfügung stehen. Der Prototyp dient als Grundlage für Entwicklungsingenieure zur späteren Weiterentwicklung.

Die hier beispielhaft betrachtete Service Request Applikation ist auf den Bereich User Management und den damit verbundenen Sicherheitsaspekten spezialisiert. Daher sollen die zunächst für eine allgemeine Service Request Applikation gefundenen Anforderungen und der Prototyp selbst für diese spezielle Applikation konkretisiert werden. Sie wird 'User/Security Applikation' genannt.

Ich werde in diese Arbeit gezielt dieser Aufgabenstellung nachgehen und setze beim Leser ein Grundverständnis für Informatik, insbesondere für Computernetzwerke und verteilten Anwendungen, voraus. Deshalb gibt es kein in diese Bereiche einführendes Kapitel. Ich werde nicht die Vor- und Nachteile von Netzwerklösungen diskutieren, Netzwerktopologien beschreiben oder das Client/Server-Prinzip erklären.

Außerdem werde ich den Begriff 'Benutzungsoberfläche' durchgehend verwenden, um zu unterstreichen, daß diese Oberfläche nicht nur für Benutzer, sondern auch für Benutzerinnen gedacht ist. Wenn ansonsten von 'Benutzer', 'Kunde' oder ähnlichem die Rede ist, so ist auch immer die weibliche Form miteinbezogen, auch wenn nur die männliche Form geschrieben wird.

## 1.3 Aufbau der Arbeit

---

Da es sich bei dem Prototypen um ein später weiterzuverwendendes Stück Software handelt, orientiert sich seine Erstellung am Software-Lebenszyklus. Der Aufbau der Arbeit spiegelt dies wieder.

Begonnen wird mit der Untersuchung der Anforderungen aus Kundensicht. Unter dem Begriff 'Kunde' werden hierbei sowohl die Entwicklungsingenieure verstanden, denen der Prototyp als Grundlage dient, als auch die Endkunden, bei denen das Produkt schließlich zum Einsatz kommen soll.

Nachdem die Anforderungen aufgestellt worden sind, wird mit dem Design eines objektorientierten Systems zur Erfüllung dieser Anforderungen fortgefahren.

Das entworfenen System wird anschließend auf eine reales System abgebildet und in einer ausgewählten Sprache implementiert.

Zum Abschluß werden die Ergebnisse zusammengefaßt, offene Probleme diskutiert und ein Ausblick gegeben.

Ich empfehle dem Leser, die Kapitel in der vorgegebenen Reihenfolge zu lesen, da in späteren Kapiteln auf Ergebnisse und Begriffe der vorherigen Kapitel Bezug genommen wird.

## 1.4 Danksagung

---

An dieser Stelle möchte ich mich bei meinem Prüfer, Herrn Prof. Dr.-Ing. Heinz Ebert, herzlich dafür bedanken, mich bei dieser praxisorientierten Diplomarbeit unterstützt und geprüft zu haben.

Mein besondere Dank geht auch an meinen Betreuer Mathias Ocker, der mir stets rat- und tatkräftig zur Seite gestanden ist.

Für umfangreiche und teilweise recht heiße Diskussionen bedanke ich mich bei Roland Heumesser, Aura Schoger, Claus Hümpel, Klaus Wurster und Matthias Schmid.

## 1.5 Kurzfassung

---

Benutzungsoberfläche und Service Request Applikation bilden ein Client/Server-System. Der Applikationsserver stellt Objekte zur Verfügung, die die Benutzungsoberfläche visualisieren soll. Zwischen den Objekte bestehen Beziehungen, die hierarchische Strukturen bilden. Die Benutzungsoberfläche ordnet zur Präsentation diese Objekte in einer Baumstruktur auf dem Bildschirm an, in der der Anwender navigieren kann. Je nach Navigation werden die Objekte über die zum Server aufgebaute Netzwerkverbindung nachgeladen. Der Navigationsbaum befindet sich am linken Rand des Darstellungsbereichs.

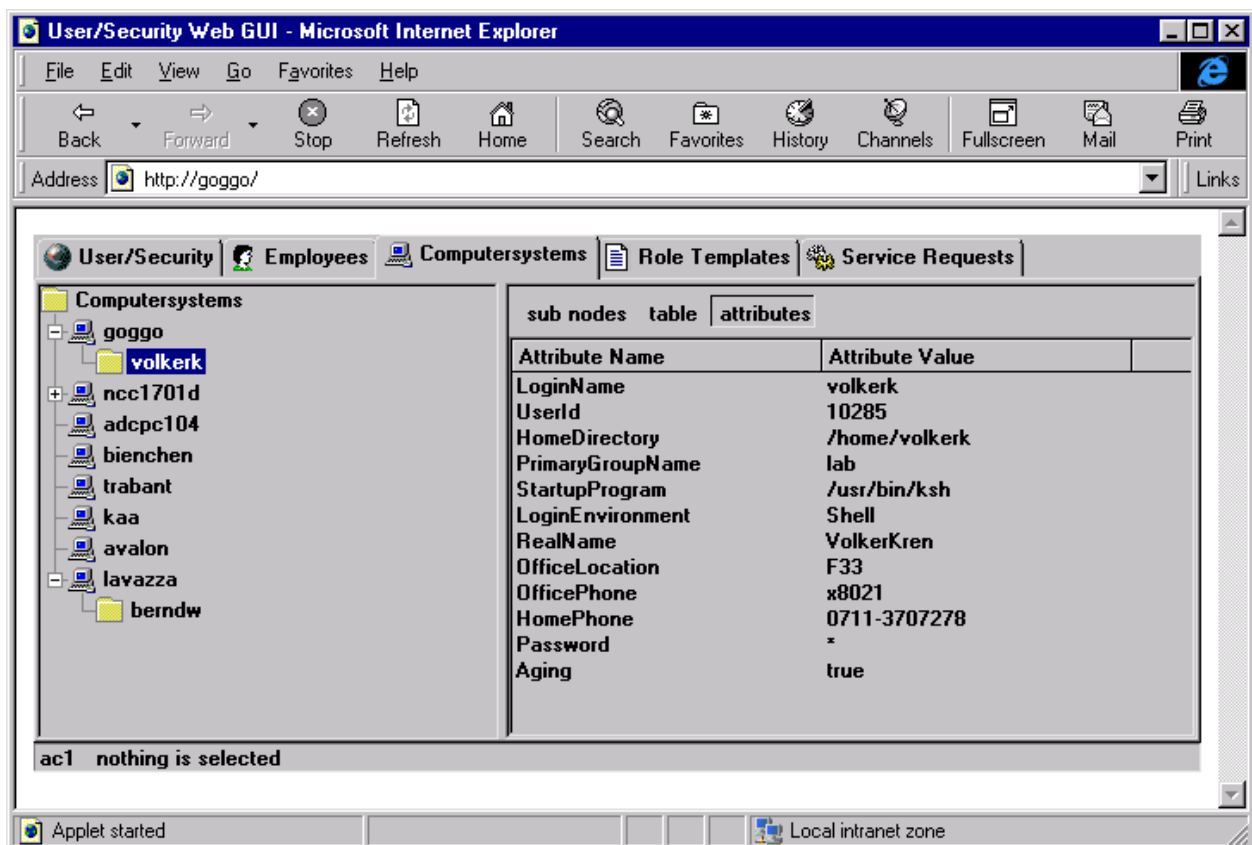
Um nicht nur Navigation in den Objektstrukturen zu ermöglichen, sondern auch Detailinformationen der Objekte darstellen zu können, existiert ein dafür definierter Bildschirmbereich. Er wird Informationsbereich genannt und befindet sich am rechten Rand des Darstellungsbereichs.

Die Benutzungsoberfläche ist als Applet in Java geschrieben und kann in einem Internetbrowser ausgeführt werden. Ihr "Look & Feel" entspricht dem Microsoft Explorer.

Objekte im Kontext der Service Request Applikation können nur durch sogenannte Service Request modifiziert werden. Ein Service Request muß vor seiner Ausführung durch eine entsprechende Stelle autorisiert werden. Die Benutzungsoberfläche bietet einen Mechanismus, um solche Service Request zu erstellen und an den Applikationsserver zu schicken. Die Erstellung eines Service Request geschieht durch die Auswahl einer Aktion und das Einlesen der zu dieser Aktion benötigten Werte. Dabei sind die Dialoge, die das Einlesen der Werte abwickeln, nicht fest in der Benutzungsoberfläche verankert, sondern werden dynamisch vom Applikationsserver geladen.

Um die Funktionalität der Benutzungsoberfläche zu testen, wurde parallel zur Benutzungsoberfläche der Prototyp einer Service Request Applikation geschrieben.

ABBILDUNG 1 Screenshot von Prototyp der Benutzungsoberfläche



# ANFORDERUNGEN

---

Dieses Kapitel befaßt sich mit den Anforderungen, die an die zu entwickelnde Benutzungsoberfläche gestellt werden.

Im ersten Abschnitt wird die Funktionalität der Service Request Applikation vorgestellt, die durch die Benutzungsoberfläche gesteuert werden soll.

Im nächsten Abschnitt werden die Anforderungen an die Benutzungsoberfläche aufgezeigt. Hier werden Fragen bezüglich der Kommunikation mit dem Applikationsserver, der Navigation, der Darstellung, der Zugriffsrechte, der Konfiguration und des Hilfesystems beantwortet.

Im letzten Abschnitt werden die vorgestellten Konzepte am Beispiel der User/Security Applikation konkretisiert.

## 2.1 Überblick über Service Request Applikation

---

### 2.1.1 Grundfunktionalität und Architektur

Die Service Request Applikation ist ein Anwendungssystem mit Client/Server-Architektur. Es besteht aus den drei Komponenten Benutzungsoberfläche, Applikationsserver und Datenbank. Die Benutzungsoberfläche dient der Interaktion mit dem Benutzer und zur Erstellung von Aufträgen an den Server, der diese ausführt und zur persistenten Speicherung der Daten auf die Datenbank zugreift. Zwischen Benutzungsoberfläche und Applikationsserver besteht, genauso wie zwischen Applikationsserver selbst und Datenbank, ein Client/Server-Verhältnis.

Die Aufträge, die die Benutzungsoberfläche an den Server schickt, heißen Service Requests. Sie bestehen aus einer Anzahl von Aktionen, die der Server ausführen soll. Er kann diese allerdings nicht unmittelbar ausführen, sondern er muß diesen Service Request zunächst von einer dafür zuständigen Stelle autorisieren lassen. Erst nach einer Autorisierung werden die im Service Request beschriebenen Aktionen ausgeführt.

Definition "Service Request":

Ein Service Request ist ein Auftrag an den Server, der autorisiert werden muß, bevor er ausgeführt werden darf. Ein Service Request wird dem Server immer asynchron übergeben.

Definition "Applikationsserver" (oder nur "Server"):

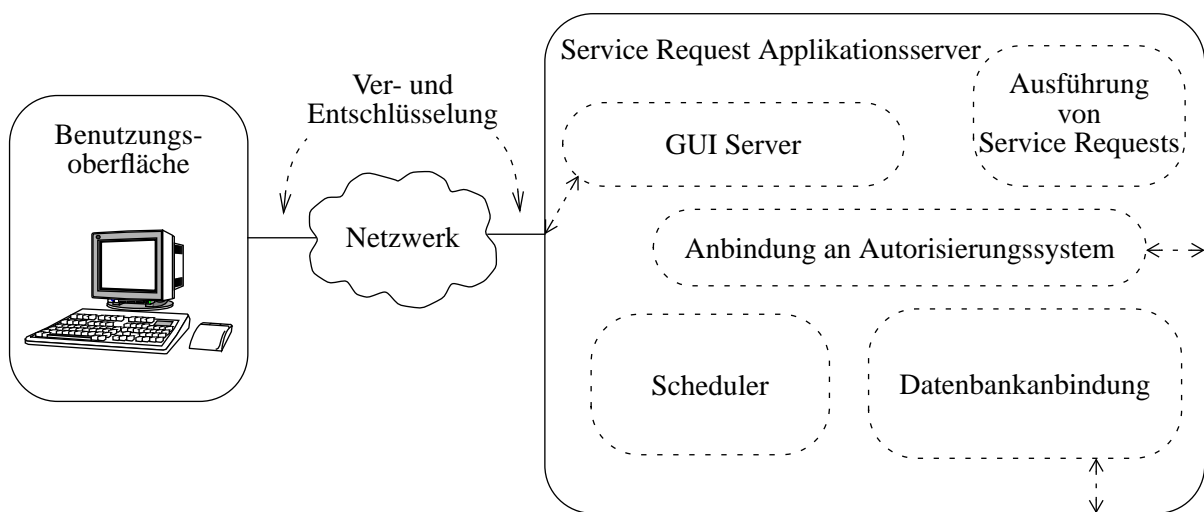
Der Applikationsserver ist diejenige Komponente der Service Request Applikation, die Service Requests ausführen kann.

Ein Service Request wird auf der Grundlage von Objekten erstellt, die ebenfalls durch den Applikationsserver zur Verfügung gestellt werden. Ein Service Request ist im Kontext der Service Request Applikation die einzige Möglichkeit, Objekte zu erschaffen,

zu ändern oder zu löschen. Beispiel hierfür ist ein Auftrag an den Server, einen Benutzeraccount auf einem System einzurichten. Dabei wird ein Objekt (das Computersystem) geändert und ein Objekt (der Benutzeraccount auf dem Computersystem) erzeugt.

Die Ausführung eines Service Requests unterliegt dem Transaktionsprinzip, das bedeutet, daß ein Service Request entweder vollständig oder gar nicht ausgeführt wird. Falls die Durchführung erfolgreich ist, so sind die während der Abarbeitung durchgeführten Änderungen dauerhaft. Die Datenbank befindet sich wieder in einem konsistenten Zustand, falls sie vor Beginn der Ausführung konsistent war.

ABBILDUNG 2 Benutzeroberfläche als Client des Service Request Applikationsservers



Um die Objekte in der Benutzeroberfläche zu visualisieren, muß die Benutzeroberfläche sie zunächst vom Server anfordern. Dies geschieht über sogenannte Queries.

Definition "Query":

Eine Query ist eine Anfrage an den Server. Aus ihr kann der Server eine gewisse Menge an Objekten errechnen, die er als Antwort an die Benutzeroberfläche zurück-schickt. Queries werden stets synchron beantwortet.

Die für die Benutzeroberfläche relevanten Serverkomponenten werden nun im folgenden näher beschrieben.

### 2.1.2 Scheduler

Der Applikationsserver stellt neben der direkten Ausführung von Service Requests auch einen Scheduling-Mechanismus zur Verfügung, der die Ausführung eines Service Requests erst zu einem bestimmten Zeitpunkt in der Zukunft anstößt. Die Durchführung eines Service Request ist ein Prozeß, bei dem es neben einem Start- und einem Endzustand auch Zwischenzustände gibt. Der Applikationsserver stellt Informationen über diese Zwischenzustände zur Verfügung.

### 2.1.3 Ausführung von Service Requests

Service Requests bestehen aus einer Reihen von durchzuführenden Aktionen. Autorisierung eines Service Requests bedeutet folglich, für jede einzelnen Aktion eine Durchführungserlaubnis einzuholen. Sind alle Aktionen autorisiert, so ist auch der ganze Service Request autorisiert.

Der Applikationsserver unterscheidet grundsätzlich zwischen zwei Ausführungsprinzipien.

Ausführungsprinzip (1):

Zum einen gibt es Service Requests, bei denen es auf eine schnelle Durchführung ankommt. Jede Aktion, die autorisiert worden ist, wird sofort ausgeführt. Autorisierung und Ausführung sind bei diesem Prinzip eng miteinander verwoben, und es ergibt sich eine Wechselspiel zwischen "Autorisierung einholen" und "Aktion ausführen". Um das Transaktionsprinzip nicht zu verletzen, muß es möglich sein, alle bereits ausgeführt Aktionen zu annullieren, falls zu einem späteren Zeitpunkt eine Aktion aufgrund fehlender Autorisierung nicht ausgeführt werden darf.

Ausführungsprinzip (2):

Das andere Prinzip der Abarbeitung eines Service Requests ist, zuerst alle notwendigen Autorisierungen einzuholen und anschließend die Aktionen auszuführen. Hierbei ist das Einhalten des Transaktionsprinzip einfacher zu gewährleisten. Bereits vor Ausführung der ersten Aktion sind alle Aktionen autorisiert. Trotzdem muß ein Annullieren von Aktionen möglich sein, falls eine Aktionen aus technischen Gründen nicht durchgeführt werden kann. In diesem Fall ist es möglich, die Ausführung des Service Requests mit Hilfe des Scheduling-Mechanismus auf einen späteren Zeitpunkt zu verschieben.

Autorisierungen sind in zwei Kategorien aufgeteilt: aktive und passive Autorisierungen. Aktiv bedeutet, daß eine Autorisierungsstelle explizit seine Zustimmung geben muß, damit der zugehörige Schritt als autorisiert betrachtet werden darf. Passiv bedeutet, daß der zugehörige Schritt nur dann als nicht autorisiert gilt, wenn die Autorisierungsstelle innerhalb einer gewissen Zeitspanne explizit ein Verbot ausspricht. Verstreicht die Zeitspanne, ohne das ein Verbot ausgesprochen wird, so wird dies als implizite Autorisierung betrachtet.

Aktive und passive Autorisierung spielen mit den zwei zuvor beschriebenen Ausführungsprinzipien zusammen. Passive Autorisierung wird vor allem für Standardvorgänge mit niedrigem Sicherheitslevel verwendet, wie zum Beispiel das Einrichten eines Mail-Accounts für einen neuen Mitarbeiter. Mit Hilfe eines Mail-Accounts ist es nicht möglich, an sensitive Daten zu gelangen. Hier reicht passive Autorisierung aus. Als Ausführungsprinzip wird Prinzip (1) gewählt ("Aktion sofort nach Autorisierung durchführen"), da die Wahrscheinlichkeit hoch ist, daß alle Aktionen autorisiert werden.

Aktive Autorisierung wird bevorzugt, falls Zugang zu einem System geschaffen wird, mit dessen Hilfe ein Benutzer an sensitive Daten gelangen kann, zum Beispiel Zugangsberechtigung zum Finanzsystem des Managements. Ein solcher Autorisierungsvorgang wird mit aktiver Autorisierung modelliert, da hier ein aktives Zustimmung des verantwortlichen Managers notwendig ist. Als Ausführungsprinzip wird am besten

Prinzip (2) benutzt (“Erst alle Autorisierungen einholen, dann durchführen”). Es macht Sinn, erst die Genehmigung aller verantwortlichen Manager einzuholen, bevor einem neuen Mitarbeiter der Zugang zu einem sensitiven System wie einem Finanzsystem gestattet wird. Würden die Aktionen unmittelbar durchgeführt werden, so könnten temporäre Sicherheitslücken entstehen, falls nicht alle Autorisierungen eingeholt werden könnten.

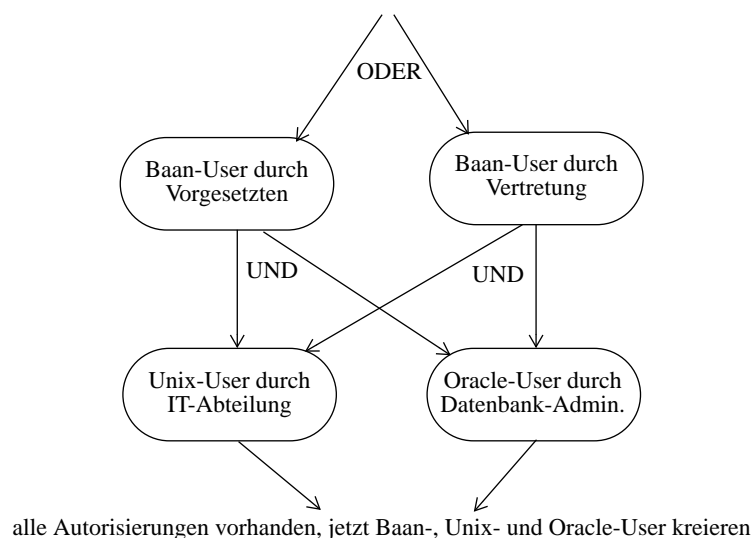
Hinzu kommt, daß der Autorisierungsvorgang nicht einstufig ist, sondern daß vielmehr ein sogenanntes ‘Vorgangnetz’ zugrunde liegt. Innerhalb dieses Vorgangnetzes gibt es Verzweigungen und Parallelität, UND und ODER Verknüpfungen, n aus m Verknüpfungen und ähnliches. Es gibt allerdings keine Zyklen.

Die vorgestellten Konzepte werden im folgenden an einem Beispiel verdeutlicht. Als Beispiel dient das Aufsetzen eines Baan-Users. Er besteht aus einem Unix-User und einem Oracle-User. Um diese drei User (Baan, Unix, Oracle) aufzusetzen zu dürfen, muß jeweils eine Autorisierung eingeholt werden. In diesem Beispiel wird davon ausgegangen, daß die drei Autorisierungsstellen voneinander verschieden sind. Der Baan-User wird vom Vorgesetzten autorisiert, oder in Abwesenheit des Vorgesetzten von dessen Vertretung. Der Unix-User wird von der zentralen IT-Abteilung und der Oracle-User vom Datenbank-Administrator autorisiert. Das Einholen der Autorisierung des Baan-Users muß den beiden anderen Autorisierungen vorausgehen, da der Unix- bzw. der Oracle-User nur im Kontext eines bereits autorisierten Baan-Users sinnvoll sind.

Die nachfolgende Abbildung verdeutlicht das Vorgangnetz zur Autorisierung. Das Ausführungsprinzip ist hierbei Prinzip (2). Jedes abgerundete Kästchen beschreibt einen Autorisierungsschritt, durch Pfeile logisch verknüpft. Ein Autorisierungsschritt besteht aus dem zu autorisierenden Gegenstand und der zugehörigen Autorisierungsstelle.

ABBILDUNG 3

Vorgangnetz zur Autorisierung eines Service Requests



### 2.1.4 GUI-Server

Der GUI-Server wickelt die gesamte Kommunikation mit der Benutzungsoberfläche ab. Er ist zentral gehalten innerhalb des Applikationsservers und verdeckt eine eventuell verteilte Implementierung. Die Kommunikation zwischen Benutzungsoberfläche und GUI-Server basiert auf einem noch zu definierendem Protokoll, das seinerseits auf ein Netzwerkprotokoll aufsetzt. Dieses Netzwerkprotokoll wird ein sowohl für firmeninterne Intranets als auch für das weltweite Internet verwendbares Protokoll sein, zum Beispiel TCP/IP.

### 2.1.5 Datenstrukturen

Grundsätzlich stellt der Server Objekte zur Verfügung, auf die über eine definierte Schnittstelle zugegriffen werden kann. Objekte besitzen einen Namen, eine Objekt ID, die das Objekt eindeutig identifiziert, einen Objekttyp und eine Reihe von Attributen und Aktionen. Die Attribute besitzen Namen und Werte und beschreiben so das Objekt näher. Die Aktionen definieren, was mit diesem Objekt gemacht werden kann.

Objekte können Beziehungen zu anderen Objekten haben. Diese Beziehungen werden nicht in den Attributen der Objekte selbst abgelegt, sondern in einem speziellen Objekt beschrieben, dem Beziehungsobjekt. Objekte, die Beziehungen zwischen Objekten ausdrücken, werden auch Assoziationen genannt. In der restlichen Arbeit wird zur Beschreibung von Objekt-Zusammenhängen stets der Begriff "Beziehung" verwendet werden. Wird nur von einem Objekt gesprochen, so ist immer ein "normales" Objekt und kein Beziehungsobjekt gemeint. Die Objektstrukturen auf Seite des Servers sind meist baumartige Strukturen.

## 2.2 Benutzeranforderungen

---

### 2.2.1 Objektrepräsentation und Objekthierarchie

Vom Server werden Objekte an die Benutzungsoberfläche geliefert. Diese wiederum muß die Objekte in einer internen Datenstruktur ablegen. Diese Datenstruktur soll in der Lage sein, beliebige Objekte aufzunehmen. Im Sinne der Benutzungsoberfläche sind die Applikationsobjekte somit Meta-Objekte. Die Datenstruktur muß eine variable Anzahl von Parametern aufnehmen können. Die Objektrepräsentation auf Seite der Benutzungsoberfläche ist unabhängig von der des Applikationsservers.

Die Objekte haben Beziehungen zueinander, die topologisch betrachtet oftmals eine hierarchische Struktur darstellen. Die Benutzungsoberfläche muß in der Lage sein, mit solchen hierarchischen Datenstrukturen, auch Bäume genannt, umzugehen. Sie reflektieren die hierarchischen Objektbeziehungen.

Jedes Objekt wird dabei auf den Knoten eines Baumes abgebildet, die Beziehungen auf Kanten zwischen den Objekten. Beziehungen sind, genauso wie Objekte, typisiert. Durch den Typ wird eine semantische Eigenschaft der Beziehung ausgedrückt, beispielsweise eine "enthält"-Beziehung. Die Beziehungen haben somit eine "Richtung" und werden damit auf gerichtete Kanten abgebildet.

### 2.2.2 Aktualität der Daten

Die Benutzungsoberfläche ist in ständigem Kontakt mit dem Applikationsserver und stellt damit die Aktualität der Daten von Seite der Benutzungsoberfläche sicher. Daraus folgt aber auch die Notwendigkeit, dem Benutzer mitzuteilen, falls sich ein Datum geändert hat und dieses automatisch zu aktualisieren. Die Richtung der Aktualisierung ist vom Server zur Benutzungsoberfläche.

Damit der Benutzer erkennen kann, an welcher Stelle eine Änderung aufgetreten ist, wird ihm der Gesamtstatus visualisiert, aus dem ersichtlich ist, an welcher Stelle eine Modifikation aufgetreten ist. Er kann durch eine einstufige Aktion zu dieser Stelle springen. Der Applikationsserver bietet einen sogenannten Notifikations-Mechanismus an, der die Benutzungsoberfläche über eine Objektänderung benachrichtigt, falls die Benutzungsoberfläche dieses Objekt schon einmal angefordert hat.

Der Server hat zwei Möglichkeiten, Änderungen festzustellen. Dies sind zum einen Änderungen, die er selbst durchführt und deren Erkennung damit trivial ist. Zum anderen benutzt er einen Synchronisations-Mechanismus, um Änderungen festzustellen, die vorbei am Server vorgenommen wurden. Es werden jeweils unterschiedliche Notifikationen erzeugt.

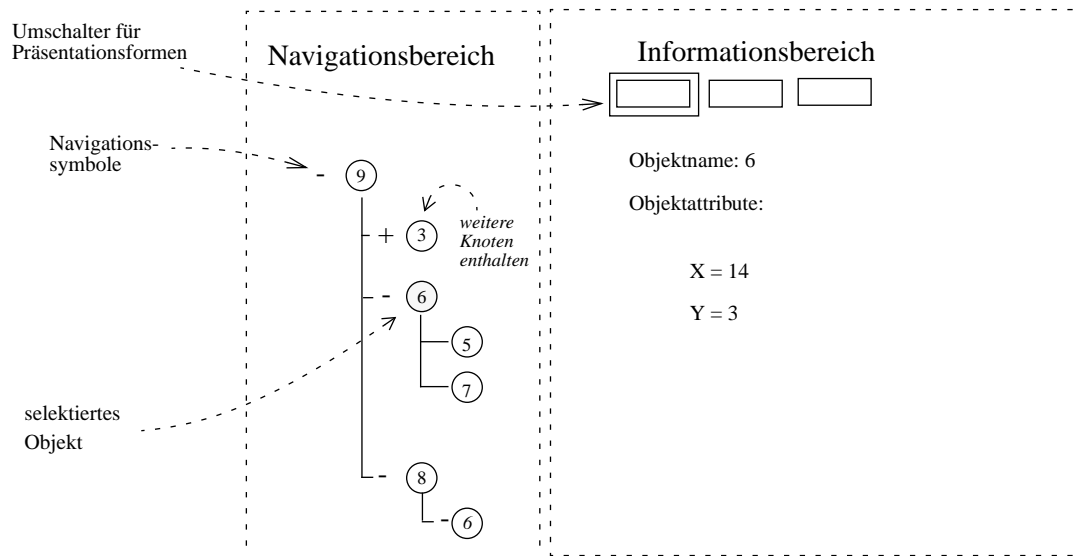
Änderungen sind hier zu verstehen als die Differenz zwischen der Datenbank, in der die Welt aus Sicht des Servers abgespeichert ist, und der realen Welt. Diese Differenz entsteht, da die reale Welt auch ohne Einwirken des Applikationsservers verändert werden kann. Man denke nur an das Von-Hand-Aufsetzen eines neuen Benutzers auf einem Computersystem. Der Synchronisationsmechanismus dient dem Aktualisieren der Datenbank und gleicht sie mit der realen Welt ab. Im Normalfall geschieht dies periodisch, zum Beispiel jede Nacht.

Es gibt Objekte und Attribute, die vom Notifikations-Mechanismus erfaßt werden, bei einer Änderung allerdings nicht explizit markiert werden. Sie werden im Hintergrund geändert, ohne daß dies im Gesamtstatus visualisiert wird. Dies liegt daran, daß die Änderungsrate dieser Objekte so hoch ist, daß von einer kontinuierlichen Änderung gesprochen werden kann. Ein Beispiel hierfür sind der freie Speicher einer Festplatte oder die Warteschlange eines Druckers. Es wäre störend, wenn ständig eine Änderung gemeldet werden würde.

Um den Datenaustausch zwischen Benutzungsoberfläche und Applikationsserver zu verringern und zugleich auch die Performance zu erhöhen, besitzt die Benutzungsoberfläche einen lokalen Cache, in dem möglichst viele Objekte temporär zwischengespeichert werden. Es wird solange mit diesen Objekten gearbeitet, bis der Benutzer auf Objekte zugreifen will, die nicht im lokalen Cache vorhanden sind. Bei einem Cache Miss wird eine Anfrage an den Server geschickt, der die gewünschten Objekte liefert. Der lokale Cache wird anschließend anhand einer Auslagerungsstrategie modifiziert. Sobald die Benutzungsoberfläche über eine Änderung eines Objekts benachrichtigt wird, wird der lokale Cache ebenfalls aktualisiert, falls das Objekt dort zwischengespeichert ist.

## 2.2.3 Bildschirmaufteilung und Navigation

ABBILDUNG 4 Skizzierte Bildschirmdarstellung



Grundsätzlich ist der Bildschirm in einen Navigations- und einen Informationsbereich aufgeteilt. Der Navigationsbereich bietet die Möglichkeit, durch die Objektstrukturen zu navigieren. Dabei ist zunächst kein Objekt selektiert, der Informationsbereich ist leer. Sobald eines selektiert wird, wird der Informationsbereich gefüllt, normalerweise entweder mit den Attributen des Objekts oder mit den untergeordneten Objekten oder auch mit einer Kombination aus Attributen und Objekten.

Für den Informationsbereich gibt es vordefinierte Präsentationsformen für alle vorkommenden Objekte. Der Benutzer hat aber auch die Möglichkeit, eigens erstellte Präsentationsformen einzubinden. Zwischen den einzelnen Präsentationsformen kann einstufig hin- und hergeschaltet werden.

Um die Navigation in sehr großen Objektstrukturen zu erleichtern, gibt es eine Reihe von Navigationshilfen.

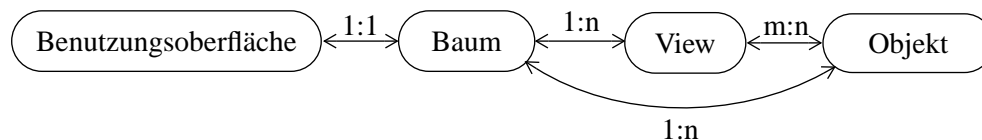
- In einer Statuszeile ist stets ersichtlich, in welchem Hierarchiezweig und in welcher Ebene sich der Fokus befindet.
- Tritt ein Knoten in verschiedenen Hierarchien auf, so kann zwischen den verschiedenen Vorkommen einstufig hin- und hergesprungen werden.
- Die einzelnen Hierarchieebenen im Navigationsbaum können über Navigationssymbole ein- und ausgeblendet werden.
- Knoten können als sogenannte 'Bookmarks' gespeichert werden. Sie können anschließend jederzeit direkt angesprungen werden.
- Innerhalb einer Hierarchieebene kann durch Druck eines Buchstabens direkt zu dem ersten Objekt gesprungen werden, dessen Name mit diesem Buchstaben beginnt. Jeder weitere Druck springt auf das nächste Objekt. Ist das letzte Objekt erreicht, wird wieder auf das erste zurückgesprungen.

## 2.2.4 Konzept der Views

Ein View ist eine Gliederungsmöglichkeit innerhalb des Navigationsbaumes. Er enthält einen definierbaren Teilbaum. Dabei ist dieser Teilbaum nicht nur aus realen Objekten (= vom Server zur Verfügung gestellt) aufgebaut, sondern kann auch Objekte enthalten, die nur der logischen Gliederung dienen und keine Objekte der realen Welt reflektieren. Der Benutzer kann solche Gliederungsobjekte in beliebiger Anzahl erschaffen und an beliebiger Stelle einfügen. Er kann somit seine ganz persönliche Sicht auf die realen Objekte konstruieren, wobei dies optional ist. Es kann auch ganz auf Views verzichtet werden.

Der Navigationsbaum enthält standardmäßig alle Objekte der Service Request Applikation. Dies sind mindestens die Objekte der realen Welt und die Service Requests.

ABBILDUNG 5 Zusammenhang zwischen Benutzungsoberfläche, Bäumen, Views und Objekten



Das Konzept der Views eröffnet dem Benutzer die Möglichkeit, eigene Strukturen aufzubauen und die Objekte innerhalb dieser visualisieren zu lassen. Diese sind unabhängig von den Objektstrukturen auf Seite des Applikationsservers.

Diese Funktionalität ist normalerweise bereits in der Datenbank enthalten (View-Konzept von Datenbanken). Somit erscheint diese Anforderung an die Benutzungsoberfläche zunächst überflüssig zu sein, da man die in der Datenbank integrierte Funktionalität nutzen könnte. Man stelle sich aber vor, daß später einmal Teile aus verschiedenen Datenbanken zusammen präsentiert werden sollen. Dies könnte zum Beispiel die Applikationsdatenbank selbst und eine Personaldatenbank sein, in der auf Namen und Mitarbeiternummern zugegriffen wird. Wenn diese Datenbanken von verschiedenen Herstellern sind, so kann nicht davon ausgegangen werden, daß das View-Konzept auf Datenbankebene benutzt werden kann. Deshalb wird dieser Ansatz auch auf Seite der Benutzungsoberfläche verfolgt. Außerdem erleichtert es die Integration in andere Applikationen und liefert somit einen wesentlichen Beitrag zum generischen Charakter der Benutzungsoberfläche.

Um Views zu definieren, stellt die Benutzungsoberfläche zwei Möglichkeiten zur Verfügung:

- der Benutzer kann Objekte in einer von ihm generierten Baumstruktur beliebige anordnen (interaktiv innerhalb der Benutzungsoberfläche)
- der Benutzer kann, getrennt von der Benutzungsoberfläche, die Navigationsstrukturen textuell definieren

Bei der ersten Möglichkeit kann der Benutzer beliebige Bäume erzeugen und anschließend die Hierarchieebenen innerhalb dieser Bäume mit beliebigen Objekten füllen. Dabei müssen nicht alle Hierarchieebenen vollständig gefüllt werden, es kann durchaus leere Knoten geben, die nur der logischen Gliederung dienen. Insbesondere kann der

Benutzer auch solche logischen Knoten in einen bereits bestehenden Objektbaum einfügen oder wieder löschen. Die Aktionen, die diese Funktionalität abdecken, werden GUI-Aktionen genannt.

Das Problem ist, daß für große Strukturen diese Art der Zusammenstellung sehr langwierig sein kann, außerdem müssen Objekte, die neu hinzukommen, wieder von Hand eingefügt werden. Komfortabler, aber gleichzeitig auch komplizierter ist die zweite Möglichkeit.

Es steht eine Sprache zur Verfügung, in der die Objektstrukturen definiert werden können, quasi extern gehaltene Layouts für die Views innerhalb des Navigationsbaumes. Der Benutzer kann in solchen Layoutbeschreibungen definieren, welcher Typ von Objekten in welchem Kontext an welcher Stelle im Baum visualisiert werden soll. Für jede Hierarchieebene kann eine Auswahl von Objekten definiert werden, wobei die Objekte unterschiedliche Typen haben dürfen. Neben Objekttypen können auch Objekte über IDs spezifiziert werden.

Da sowohl Objekttypen als auch Objekt IDs zugelassen sind, können sowohl Strukturen mit statischen als auch mit dynamischem Charakter aufgebaut werden. Beim Start der Benutzungsoberfläche wird das Layout neu ausgewertet, da sich etwas geändert haben könnte. Falls ein Objekt über seine ID spezifiziert worden ist, so hat dies statischen Charakter, falls dieses Objekt eine sehr niedrige Änderungsrate besitzt. Wird hingegen ein Objekttyp spezifiziert, wobei Objekte dieses Typs eine hohe Änderungsrate haben, so wird das Ergebnis jedesmal anders ausfallen. Ein Beispiel hierfür ist, daß die Service Request Applikation alle anwesenden Mitarbeiter auflistet. Es ist wahrscheinlich, daß diese Liste von Tag zu Tag aufgrund von externen Schulungen, Messebesuchen, Krankheit usw. verschieden sein wird. Die Layoutbeschreibung hat somit dynamischen Charakter.

### 2.2.5 Benutzerkonzept

Bei der Service Request Applikation handelt es sich um ein Mehrbenutzersystem. Beim Start der Benutzungsoberfläche muß sich der Benutzer anmelden und erhält anschließend Zugriff entsprechend der ihm zugewiesenen Rechte. Sowohl auf Objekt- als auch auf Attributebene existieren Rechte. Dabei wird unterschieden zwischen vier verschiedenen Arten:

- Benutzer hat sowohl Lese- als auch Schreibzugriff (z.B. auf einen Namen)
- Benutzer hat nur Schreibzugriff (z.B. auf ein Paßwort)
- Benutzer hat nur Lesezugriff (z.B. auf einen Objekttyp)
- Benutzer hat keinen Zugriff (z.B. auf ein Mitarbeitergehalt)

Das Benutzerkonzept der Service Request Applikation folgt dem Vier-Augen-Prinzip. Es gibt einen dedizierten Benutzer, der das Recht besitzt, neue Benutzer aufzusetzen und ihnen Rechte zu geben. Er selbst besitzt keine weiteren Rechte, er kann also keine Objekte oder Attribute lesen oder schreiben. Seine Service Requests beziehen sich nur auf Benutzer der Service Request Applikation. Er wird als Administrator bezeichnet. Die von ihm aufgesetzten Benutzer, sogenannte Operatoren, können wiederum keine Benutzer aufsetzen, sondern 'nur' die Applikationsobjekte selbst modifizieren. Sie sind im Sinne der Applikation die eigentlichen 'Benutzer'.

## 2.2.6 Filtermechanismen

Der Benutzer kann mittels Filterkriterien steuern, welche Objekte er betrachten möchte. Sowohl für den Navigations- als auch für den Informationsbereich können Filter definiert werden. Im Informationsbereich werden Filter nur auf Objekte, nicht aber auf Attribute der Objekte angewendet.

Im Navigationsbaum kann für jede Hierarchieebene ein Filter definiert werden. Dadurch kann die Ebene in seiner Größe und damit in der vertikalen Ausdehnung auf dem Bildschirm beschränkt werden. Der Filter kann definiert werden, sowohl bevor die Objekte vom Applikationsserver angefordert werden als auch nachdem die Hierarchieebene schon aufgebaut worden ist. Sobald die Anzahl der Objekte einer Hierarchieebene einen gewissen Wert übersteigt, schlägt die Benutzungsoberfläche automatisch vor, einen Filter für diese Hierarchieebene zu definieren. Der Anwender kann aber auch mit der kompletten Objektanzahl arbeiten. In diesem Fall werden die Objekte abschnittsweise dargestellt.

Im Informationsbereich können beispielsweise alle untergeordneten Objekte präsentiert werden. Das bedeutet, daß hier dasselbe Problem wie für den Navigationsbereich vorliegt. Eventuell soll eine unüberschaubar große Menge an Objekten dargestellt werden. Um die Anzahl der darzustellenden Objekte zu reduzieren, kann der Benutzer für diese Präsentationsform einen Filter definieren. Bei zu vielen Objekten geschieht die Darstellung ebenfalls abschnittsweise.

## 2.2.7 Aliasing der Objektdarstellung

Als Aliasing wird der Effekt bezeichnet, daß dasselbe Objekt mehr als einmal referenziert wird. Im Falle einer Benutzungsoberfläche heißt dies, daß dasselbe Objekt mehr als einmal dargestellt wird. Diese Darstellung muß konsistent sein. Falls das Objekt modifiziert wird, so muß jeder Alias aktualisiert werden.

Das Aufspüren von mehrfach dargestellten Objekten ist nicht trivial. Der Algorithmus zur Lösung dieses Problems muß eine praxistaugliche Laufzeit haben.

## 2.2.8 Aktionen

Um nicht nur Lesezugriff auf Objekte und Objekthierarchien zu haben, muß die Benutzungsoberfläche in der Lage sein, Aktionen auszulösen. Eine solche Aktion ist beispielsweise das Anlegen eines Benutzers auf einem System.

Es gibt prinzipiell zwei Arten von Aktionen. Einmal sind dies Aktionen, die die Benutzungsoberfläche nicht selbst ausführt, sondern nur die notwendigen Objekte und Parameter sammelt, die Aktion festlegt und diese Informationen an den Server schickt. Dieser führt anschließend die Aktion durch und schickt entsprechende Kontrollinformationen an die Benutzungsoberfläche zurück. Solche Aktionen heißen Server-Aktionen. Sie bilden die Grundlage von Service Requests.

Die andere Art von Aktionen sind Aktionen, die die Benutzungsoberfläche selbst ausführen kann. Sie haben keine Auswirkung auf die Objekte und dienen nur der Veränderung der Darstellung der Objekte. Diese Aktionen heißen GUI-Aktionen und verändern Views im Navigationsbaum (siehe "Konzept der Views" auf Seite 16).

Aktionen, die sich auf die Benutzungsoberfläche selbst beziehen, sind beispielsweise das Erzeugen eines neuen Views, das Kopieren eines Teilbaums an eine andere Stelle innerhalb desselben Views, das Verschieben einer Teilbaums in einen anderen View, das Erzeugen eines logischen Knotens in einem Baum oder das Zuordnen eines Objekts zu einem logischen Knoten.

Aktionen werden immer kontextbezogen ausgelöst, das heißt aufgrund einer Selektion von einem oder mehreren Objekten sind bestimmte Aktionen möglich. Sie werden in einem eigenen Kontextmenü zusammengefaßt.

Bei einer Einzelselektion ist eindeutig, welche Aktionen für dieses Objekt anwendbar sind, nämlich die für dieses Objekt definierten.

Bei einer Mehrfachselektion werden alle den Objekten gemeinsamen Aktionen zur Auswahl gestellt. Gibt es keine solchen gemeinsamen Aktionen, so kann keine Aktion ausgeführt werden. Aktionen, die für mehr als ein Objekt ausgeführt werden, heißen Massenoperationen.

Zu beachten ist, daß aus Sicht der Benutzungsoberfläche Aktionen nicht an den Typ eines Objekts, sondern an das Objekt selbst gekoppelt sind. Es existiert keine Tabelle, in der Objekttypen und Aktionen zueinander in Beziehung gesetzt werden. Damit stehen nicht zwangsweise für alle Objekte eines Typs dieselben Aktionen zur Verfügung, sondern dem Server steht es frei, für bestimmte Objekte individuelle Aktionen zu definieren.

Nach Selektion der Objekte und Auswahl der Aktion müssen eventuell weitere Parameter eingelesen werden. Will der Benutzer beispielsweise ein Objekt modifizieren, so müssen vor Abschicken des Service Request noch die neuen Werte der zu verändernden Attribute eingelesen werden.

Das Einlesen neuer Werte kann durch Dialog mit dem Benutzer, durch eine Berechnungsvorschrift oder mittels vordefinierter Werte geschehen. Für Massenoperationen muß es noch die Möglichkeit geben, die fehlenden Werte aus einer Datei einzulesen. Dies bedeutet nicht, daß die Benutzungsoberfläche dieses Einlesen abwickeln muß, sie muß aber mindestens in der Lage sein, den Namen einer Datei für diese Aktion einzulesen.

Da sich Aktionen immer auf selektierte Objekte beziehen, ist es für Massenoperationen notwendig, komfortable Selektionsmechanismen zur Verfügung zu stellen. Der Benutzer möchte möglichst einfach viele Objekte selektieren können. Dazu kann er einzelnen Objekte nacheinander, alle untergeordneten Objekte oder Objekte mittels eines Rahmens selektieren. Ein Rahmen ist ein rechteckiger Bereich, in dem alle Objekte selektiert sind.

Beim bisherigen Ansatz bezieht sich jede Aktion auf das Objekt, an die es gekoppelt ist. Dies entspricht dem objektorientierten Ansatz. Es gibt jedoch einen Fall, bei dem dieser Ansatz durchbrochen werden muß. Dies ist für die Aktion "Erzeuge Objekt" notwendig. Diese Aktion kann nicht an das Objekt selbst gebunden werden, da sie es erst erzeugt. Daher muß für die Aktion "Erzeuge" eine Sonderregelung getroffen werden.

In der Baumstruktur gibt es für jedes Objekt ein übergeordnetes Objekt, mit Ausnahme für die Wurzel des Baumes. Die Aktion "Erzeuge Objekt" kann somit an das übergeordnete Objekt gehängt werden. Sie ist eine Aktion, die sich auf einen Objekttyp

bezieht, im Gegensatz zu allen anderen Aktionen. An einem Objekt, dem Objekte vom Typ A untergeordnet sind, hängt zum Beispiel eine Aktion "Erzeuge Objekt vom Typ A". Die Wurzel des Baumes muß folglich ein logischer Knoten, also ein von der Benutzungsoberfläche definierter oder ein vom Server fest vordefinierter Knoten sein, der nicht erzeugt oder gelöscht werden kann.

## 2.2.9 Benutzerführung

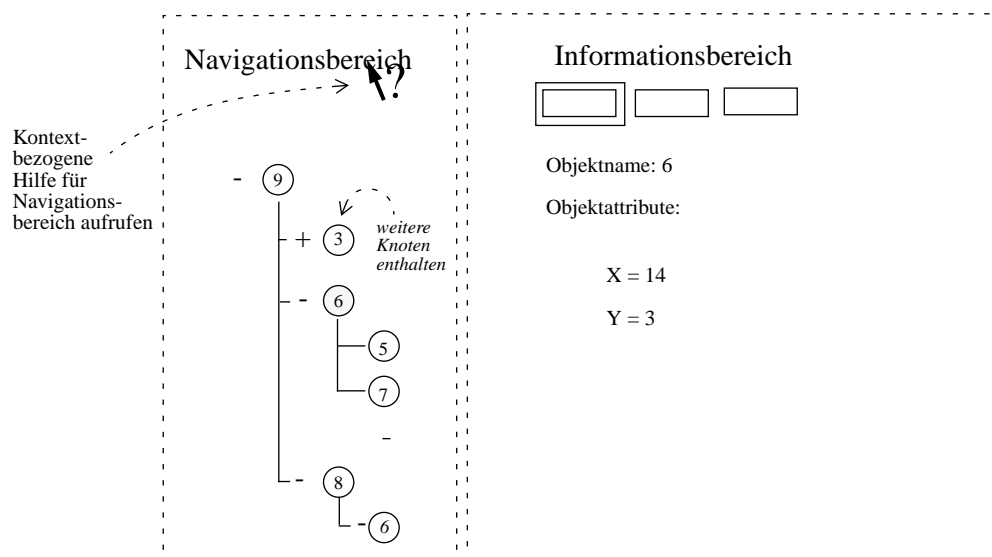
Der Start der Benutzungsoberfläche geschieht über eine World Wide Web Seite, die der Benutzer mit Hilfe seines Internetbrowsers ansteuert. Zunächst muß er sich mit Username und Paßwort beim Applikationsserver anmelden. Falls er in einer Single-Sign-On-Umgebung arbeitet, so geschieht dieser Vorgang automatisch, ansonsten muß er die Login-Informationen explizit eingeben.

Die Benutzungsoberfläche präsentiert sich nach dem Start in einem Anfangszustand. Die Steuerung der Benutzungsoberfläche ist über das Zeigegerät möglich, im Normalfall der Maus.

## 2.2.10 Hilfesystem

Die Benutzungsoberfläche verfügt über ein jederzeit erreichbares Online-Hilfesystem. Hilfe zu einem bestimmten Objekt ist über eine Kontextfunktion aufrufbar.

ABBILDUNG 6 Kontextbezogene Hilfe



## 2.2.11 Verfügbarkeit

Die Benutzungsoberfläche soll auf beliebigen Plattformen verfügbar sein und von beliebiger Stelle in einem Netzwerk aus gestartet und benutzt werden können.

## 2.2.12 Zusammenfassung

Die zu entwickelnde Benutzungsoberfläche für eine Service Request Applikation muß folgende Konzepte bieten:

1. Applikationsunabhängige Objektrepräsentation
2. Benutzungsoberfläche auch mit anderen Applikationen einsetzbar
3. Abhörsichere Kommunikation mit Service Request Applikation durch Datenverschlüsselung
4. Gute Performance durch lokales Caching
5. Visualisierung unterteilt in Navigations- und Informationsbereich
6. Applikationsunabhängige logische Objektgliederung innerhalb des Navigationsbereichs, durch Benutzer definierbar
7. Verschiedene Präsentationsformen für Informationsbereich, auch durch Benutzer definierbar
8. Interaktive Erstellung von Service Requests
9. Auslösen von Service Requests
10. Datenaktualität durch Notifikationsmechanismus
11. Aliasing bezüglich der Darstellung wird behandelt
12. Filtermechanismen
13. Zugriffsrechte auf Objekte und Attribute werden berücksichtigt und optisch visualisiert
14. Kontextbezogene Hilfe
15. Netzweite, plattformunabhängige Verfügbarkeit

Diese Liste ist bisher noch nicht priorisiert und konkretisiert. Dies wird im folgenden Abschnitt anhand der User/Security Applikation vorgenommen.

## 2.3 Konkretisierung an User/Security Applikation

### 2.3.1 Konzept der User/Security Applikation

Um die Anforderungen an die Benutzungsoberfläche konkretisieren und insbesondere priorisieren zu können, muß zunächst das Konzept der User/Security Applikation vorgestellt werden.

Die User/Security Applikation ist eine spezielle Service Request Applikation. Mit ihrer Hilfe ist komfortables und fundiertes User Management möglich. Um das Konzept zu verstehen, müssen zunächst einige grundlegende Begriffe eingeführt werden.

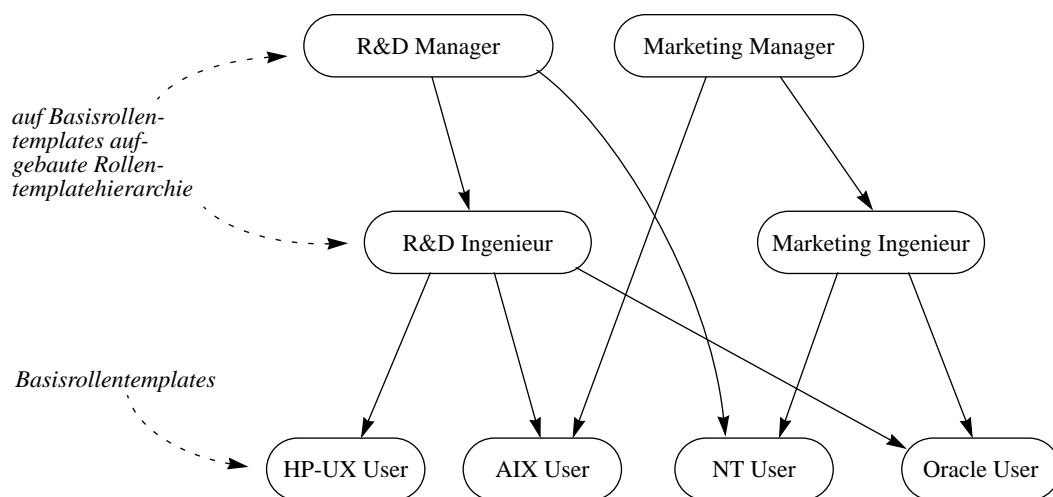
Ein Employee ist ein Angestellter einer Firma. Zur Ausführung seiner Arbeit benötigt er Zugriff auf Ressourcen. Ressourcen sind beliebige Computersysteme und Applikationen, also zum Beispiel Datenbanken, Rechner, CAD-Systeme und ähnliches. Er hat nicht auf beliebige Ressourcen Zugriff, sondern bestenfalls genau auf die, die er zur Ausführung seiner Arbeit benötigt. Der Zugang zu den einzelnen Ressourcen ist durch Zugriffsbeschränkungen geregelt. Jeder Employee braucht einen Account, um eine Ressource benutzen zu können. Accounts sind in der Regel durch Paßwörter vor unberechtigtem Zugriff geschützt.

Auf welche Ressourcen ein Employee Zugriff braucht, hängt von der Art seiner Tätigkeit innerhalb der Firma ab. Durch die Organisationsstruktur der Firma sind ihm eine oder mehrere Rollen zugewiesen. Ein Entwicklungsingenieur einer Software Abteilung hat zum Beispiel die Rolle "R&D Engineer". An die Rolle, die ein Employee innerhalb einer Firma besitzt, knüpfen sich Ressourcen, die zur Ausübung dieser Rolle notwendig sind. Die Verknüpfung von Rollen und Ressourcen geschieht durch sogenannte "Basisrollen", die einen Account in der realen Welt widerspiegeln. Die Beschreibung einer Rolle wird als "Rollentemplate" bezeichnet.

Dieser logischen Zusammenhang zwischen Employee, Rolle, Ressource und Account wird in der User/Security Applikation nachgebildet. Ein Employee besitzt eine Reihe von Rollen, die seine Tätigkeiten in der Firma reflektieren. Die Bindung einer Rolle an einen Employee geschieht durch Instanziierung eines Rollentemplates. Dies erzeugt eine neue Rolle und weist sie einem Employee zu. Wenn an dieses Rollentemplate Basisrollen gebunden sind, so werden sie bei der Instanziierung in konkrete Accounts auf konkreten Ressourcen umgesetzt. Bei der Instanziierung des R&D Ingenieurs 'Mayer' aus dem Rollentemplate 'R&D Engineer' wird beispielsweise das Basisrollentemplate 'HP-UX User' in einen konkreten Account 'mayer' auf der HP-UX Workstation 'hpux01' umgesetzt. Somit erhält der Employee 'Mayer' Zugriff auf die Ressource 'hpux01' vom Typ 'Workstation'.

Um den Aufbau und die funktionelle Organisation einer Firma nachbilden zu können, kann eine Rollentemplatehierarchie aufgebaut werden. Diese Hierarchie wird von unten nach oben aufgebaut, zu Beginn existieren nur eine Reihe von Basisrollentemplates. Darauf aufbauend können höherwertige Rollentemplates konstruiert werden, zum Beispiel das Template eines Entwicklungsingenieurs, der einen Account auf einer Workstation, Zugriff auf ein CAD-System und Zugriff auf eine Datenbank hat.

ABBILDUNG 7 Beispiel einer Rollentemplatehierarchie



Ein Service Request wird bei der User/Security Applikation als Bindeglied zwischen Rollentemplates und realer Welt verwendet. Um beispielsweise einen Account in der realen Welt zu kreieren, muß ein Service Request an die User/Security Applikation geschickt werden, der nach Autorisierung das Basisrollentemplate instanziiert und

dadurch einen neuen Account in der realen Welt schafft. Der Autorisierungsaspekt ist in den Rollentemplates selbst enthalten.

### 2.3.2 Prototyp der User/Security Applikation

Der zu entwickelnde Prototyp soll ein Prototyp werden, der als Basis für die Implementierung des späteren Produkts dienen soll. Er ist damit ein evolutionärer Prototyp und wird nicht nach dem 'Quick & Dirty' Prinzip entwickelt. Die Anwendung des Softwarelebenszyklus-Modells gerechtfertigt.

Um möglichst schnell einen lauffähigen Prototyp zu erhalten, werden die zuvor vorgestellten Anforderungen so priorisiert, daß als erstes die für einen Minimalbetrieb notwendigen Funktionalitäten implementiert werden. Anschließend werden sie verbessert und für den Benutzer optimiert.

Oberste Priorität haben somit die Funktionalitäten, die mindestens zu einem Minimalbetrieb notwendig sind:

- Interne applikationsunabhängige Objektrepräsentation
- Kommunikation mit Applikationsserver (= minimale Kommunikation)
- Erstellen und Auslösen von Service Requests (= minimale Benutzerfunktionalität)
- Navigations- und Informationsbereich zur Visualisierung von Objektstruktur und Attributen (= minimale Visualisierung der Objekte)

Anschließend wird die Minimalfunktionalität verbessert:

- Lokales Caching (= Performance-Verbesserung der Kommunikation)
- Applikationsunabhängige Objektgliederung (= Verbesserung des Navigationsbereichs)
- Verschiedene Präsentationsformen, auch benutzerdefiniert (= Verbesserung des Informationsbereichs)
- Datenaktualität durch Notifikationsmechanismus (= Verbesserung der Benutzerfunktionalität)
- Filtermechanismen
- Zugriffsrechte werden optisch visualisiert
- Hilfesystem

### 2.3.3 Views für User/Security Applikation

Die User/Security Applikation unterteilt sich in vier logische Komponenten. Es handelt sich hierbei um die Verwaltung der Employees, das Management der Rollentemplates, die Verwaltung der daraus abgeleiteten Rollen und die Verwaltung der Service Requests. Die Benutzungsoberfläche spiegelt diese vier Komponenten mit den zugehörigen Objekten in jeweils einem eigenen View innerhalb des Navigationsbaumes wieder: Employee View, Rollentemplate View, Rollen View und Service Request View. Alle vier Views befinden sich auf oberster Stufe des Navigationsbaumes.

Im ersten View werden alle Employees dargestellt. Über eine Aktion, die am übergeordneten logischen Knoten hängt, können Employees erzeugt werden. Dieser Vorgang ist automatisierbar, so daß zum Beispiel Daten aus einer Personaldatenbank verwendet werden können.

Im zweiten werden die verfügbaren Rollentemplates visualisiert. Die Hierarchie der Rollentemplates läßt sich dabei als Navigationsstruktur verwenden. Da aber die Basisrollentemplates auch direkt und nicht nur am Ende der Rollentemplatehierarchie sichtbar sein sollten, werden sie zusätzlich auf erster Stufe eingehängt. Die existierenden Ressourcen werden in einem weiteren View dargestellt. Es existieren somit drei Views innerhalb des Rollentemplate Views. Einer visualisiert die Hierarchie der Rollentemplates, der nächste die Basisrollentemplates und der dritte die Ressourcen.

Der Rollen View stellt die existierenden Instanzen der Rollentemplates dar. Die festgelegte Rollentemplatehierarchie spiegelt sich in der Hierarchie der instanziierten Rollen wieder.

Im letzten dieser vier Views werden alle Service Requests visualisiert, unterteilt in bereits gestellte und ausgeführte Service Requests (eine Art History), Service Requests, die für einen Zeitpunkt in der Zukunft geplant sind und Service Requests, die gerade bearbeitet werden. Ihr Status wird in einem ihrer Attribute reflektiert.

### 2.3.4 Benutzung des Darstellungs-Aliasing

Da ein Objekt in mehreren Views vorkommen kann, verwaltet die Benutzungsoberfläche dieses sogenannte Darstellungs-Aliasing. Notwendig ist dies, da bei der Änderung eines Objekts alle visuellen Kopien geändert werden müssen. Der Benutzer hat zusätzlich noch die Möglichkeit, zwischen den einzelnen visuellen Kopien hin- und herzuspringen. Daß dieses Springen vom Benutzer sinnvoll eingesetzt werden kann, soll im folgenden durch zwei Beispiele belegt werden.

- Der Benutzer ist im Service Requests View und betrachtet einen Service Requests, der eine Rolle erzeugt hat. Diese Rolle ist als untergeordnetes Objekt des Service Requests dargestellt. Er kann nun von dieser Rolle im Service Request View zu ihrer Darstellung im Rollen View springen. Er kann sofort sehen, welchem Employee diese Rolle zugewiesen ist, ohne daß er diese Rollen selbst im Rollen View suchen muß.
- Der Benutzer ist im Rollentemplate View. Er will wissen, welcher Service Request sich auf das gerade selektierte Rollentemplate bezieht. Er springt dazu in den Service Request View, in dem die visuelle Kopie dieses Rollentemplates fokussiert wird, falls eine existiert. Das übergeordnete Objekt ist ein Service Request, der eine Instanzierung dieses Rollentemplates durchgeführt hat.

### 2.3.5 Funktionalität

Die Hauptfunktionalitäten der Benutzungsoberfläche der User/Security Applikation umfassen das Visualisieren der Rollentemplates und der daraus abgeleiteten Rolleninstanzen, das Auslösen eines Service Requests sowie die Statuskontrolle der laufenden Service Requests.

Das Visualisieren wird innerhalb der zuvor beschriebenen Views erledigt. Der Gesamtschritt für laufende Service Requests ist als Attribut dieser Objekte modelliert und wird somit ebenfalls dargestellt. Service Requests können über Aktionen ausgelöst werden, die an Objekte gebunden sind.

Für jedes Objekt gibt es eine Reihe von durchführbaren Aktionen. Im folgenden wird eine Liste der wichtigsten Kombinationen 'Objekt & Aktion' und den benötigten Parametern vorgestellt.

ABBILDUNG 8 Übersicht über Objekte und ihre Aktionen

Objekt	Aktionen	Parameter
Rollentemplate	instantiate: aus dem Rollentemplate wird eine Rolle instanziiert, die einem bestimmten Employee zugewiesen wird modify: ändern, hinzufügen und löschen von Attributen	Employee, alle im Rollentemplate definierten Attribute  alle im Rollentemplate definierten
Rolle	assign: die einem Employee zugewiesene Rolle wird einem anderen Employee zugewiesen	neuer Employee
Employee	create: erzeugt einen Employee delete: löscht einen Employee	Name und Personalnummer keine weiteren notwendig
Ressource	create: erzeugt eine Ressource delete: löscht eine Ressource	Name und Typ keine weiteren Parameter
Service Request	create: eine der obigen Aktionen ausführen, dies erzeugt einen Service Request cancel: Service Request abbrechen	entsprechende Parameter, siehe oben  keine weiteren Parameter

Dieses Kapitel befaßt sich mit dem Design eines objektorientierten Systems zur Realisierung der vorgestellten Benutzungsoberfläche.

Es werden Mechanismen und Algorithmen zur Erfüllung der Benutzeranforderungen erarbeitet. Die Reihenfolge der Erarbeitung orientiert sich an der aufgestellten Prioritätenliste (siehe Seite 23). Für jede beschriebene Realisierung wird der Bezug zu den gestellten Anforderungen hergestellt.

## 3.1 Realisierung der Minimalfunktionalität

---

### 3.1.1 Interne Objektrepräsentation

*Bezug zu Anforderungen: Jedes Applikationsobjekt soll von der Benutzungsoberfläche gleich gespeichert werden. Dafür wird eine universelle Datenstruktur benötigt.*

Die Benutzungsoberfläche darf keine Kenntnis über die interne Objektstruktur der Applikationsobjekte haben, sie muß über eine applikationsunabhängige interne Repräsentation der Objekte verfügen, die für alle Applikationsobjekte gültig ist.

Ein Applikationsobjekt ist für die Benutzungsoberfläche ein Meta-Objekt, dessen ID, Typ, Methoden und Attribute nur gespeichert und nicht ausgewertet werden. Methoden sind hier zu verstehen als die mit diesem Objekt ausführbaren Aktionen. Diese Aktionen kann nur der Applikationsserver ausführen, angestoßen durch Service Requests. Die Anzahl der Attribute und Methoden variiert von Objekt zu Objekt und kann sich sogar zur Laufzeit ändern. Die interne Speicherstruktur muß dieser dynamischen Veränderbarkeit Rechnung tragen.

Somit müssen in der internen Objektrepräsentation die nachfolgenden Informationen gespeichert werden:

- **Objekt ID:**  
Die ID eines Objekts wird vom Server vergeben, die Benutzungsoberfläche kann auf sie keinen Einfluß nehmen. Damit kann eine Objekt ID aus Sicht der Benutzungsoberfläche beliebig sein. Aus Sicht des Servers hat sie eine textuelle Form und wird deshalb auf Seite der Benutzungsoberfläche als String gespeichert.
- **Name des Objekts:**  
Der Name ist ein Text, intern als String gespeichert.
- **Typ des Objekts (applikationsbezogener Typ):**  
Der Typ ist ein Text, er wird nicht von der Benutzungsoberfläche interpretiert, also auch ein String. Eventuell ist dieser Eintrag leer, falls der Server typisierte Objekt IDs verwendet.

- Liste der Attribute:  
Dieser Teil muß dynamisch flexibel sein, Attribute können hinzukommen und wieder gelöscht werden. Die interne Datenstruktur muß zur Laufzeit erweiterbar sein, also beispielsweise eine Listenstruktur. Ein Attribut ist dabei ein Tripel, bestehend aus Name, Wert und Zugriffsrechten.
- Liste der Aktionen:  
Die Aktionen müssen ebenfalls dynamisch erweiterbar sein, intern in einer Listenstruktur gespeichert. Eine Aktion ist ein Paar, bestehend aus dem Namen und dem zugehörigen Aktionsdialog (siehe "Erstellung von Service Requests mit zusätzlichem Dialog" auf Seite 33).

Syntaktisch gesehen sind die Server-Aktionen normale Attribute. Damit ist gewährleistet, daß sie eventuell für Objekte gleichen Typs verschieden sein und sich auch zur Laufzeit ändern können. Diese Tatsache trifft nur für einen geringen Teil der Objekte zu, bei den meisten Objekten gleichen Typs werden die Aktionen übereinstimmen.

Mit dieser internen Objektrepräsentation können beliebige Applikationsobjekte abgespeichert werden, insbesondere auch Service Request Objekte. Sie haben, genauso wie alle anderen Objekte, eine ID, einen Typ und eine beliebige Anzahl von Attributen und Aktionen.

Beispiel für technischen Realisierung (in Java):

```
public class ApplicationObject {
    private String id;
    private String name;
    private String type;
    private Vector attributes; //Vector ist dynamische Datenstruktur
    private Vector actions;
}
```

### 3.1.2 Erstellung von Service Requests

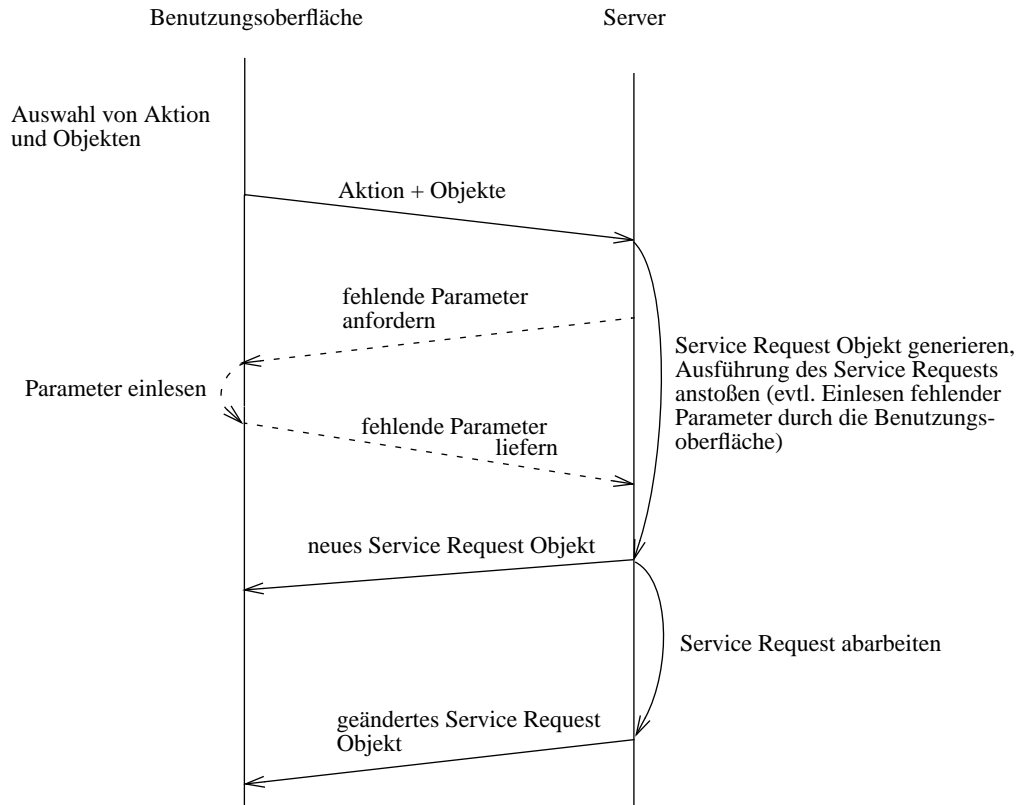
*Bezug zu Anforderungen: Die Benutzungsoberfläche muß diese Kernfunktionalität einer Service Request Applikation realisieren. Ein Service Request ist die einzige Möglichkeit, Applikationsobjekte zu erzeugen, zu verändern oder zu löschen.*

Der Benutzer selektiert eine Auswahl von Objekten und eine Aktion. Die Selektion der Objekte erfolgt durch Markierung auf dem Bildschirm, die Aktion kann über ein Kontextmenü spezifiziert werden. Die ausgewählten Objekte werden sodann, zusammen mit der durchzuführenden Aktion, an den Server geschickt. Nun gibt es zwei Möglichkeiten, wie diese Daten im Server verarbeitet werden:

- Die Aktion kann direkt ausgeführt werden, der Server benötigt keine weiteren Parameter. Er erzeugt ein Service Request Objekt, das den Service Request beschreibt, schickt dieses Objekt an die Benutzungsoberfläche und stellt es in seine interne Warteschlange zur Abarbeitung.
- Zur Ausführung der Aktion sind, neben den bereits geschickten Objekten, noch weitere Parameter notwendig. Der Server schickt diese Information an die Benutzungsoberfläche, die daraufhin die fehlenden Parameter durch Interaktion mit dem Benutzer einließt. Sie schickt diese an der Server, der nun ein Service Request Objekt kreiert und in seine Warteschlange einreicht.

Der Server arbeitet den Service Request irgendwann ab und informiert die Benutzungsoberfläche durch eine Objektänderungs-Notifikation bezüglich des Service Request Objekts. Das Schicken des neues Objekts wird durch eine Query Notifikation abgehandelt (siehe "Notifikations-Mechanismen" auf Seite 48).

ABBILDUNG 9 Erstellung von Service Requests



Für Massenoperationen muß eine Regelung zum Einlesen der fehlenden Parameter getroffen werden. Begonnen wird mit den Parametern des ersten Objekts. Es erfolgt eine Vorbesetzung der Werte mit Default-Werten, falls solche vorhanden sind. Die Werte der Parameter für das nachfolgende Objekt werden mit den Werten des zuvor eingegebenen vorbesetzt. So wird analog mit allen restlichen Objekten weiterverfahren.

Für jeden einzulesenden Parameter können Regeln definiert werden, woher dieser Wert geholt oder wie er errechnet werden soll. So ist beispielsweise möglich, fortlaufende User ID für Accounts zu erzeugen oder einer Reihe von Rechner mit aufeinanderfolgenden Namen zu benennen ("hpux0001", "hpux0002" usw.). Die Regel kann sich aber auch auf das Holen eines Wertes durch Ausführen eines externen Programms auf Seite des Servers beziehen.

Die Spezifikation einer Datei, aus der die fehlenden Parameter eingelesen werden, ist ebenfalls möglich. Die Parameter haben eine bestimmte Reihenfolge, die als Reihenfolge zum Einlesen der Werte benutzt wird. Eine Zeile der Datei entspricht dabei einem Objekt. Die einzelnen Werte sind durch Trennzeichen separiert. Das Trennzeichen

kann optional in der ersten Zeile der Datei spezifiziert werden, ansonsten wird das Zeichen ‘|’ als Trennzeichen verwendet.

Fehlerfälle:

- Datei enthält mehr Zeilen als benötigt -> Rest der Datei wird ignoriert
- Datei enthält weniger Zeilen als benötigt -> alle noch ausstehenden Parameter werden durch Interaktionen mit dem Benutzer eingelesen; zusätzliche Spezifikation einer Quelldatei möglich
- Zeile enthält zu viele Werte -> Rest der Zeile wird ignoriert
- Zeile enthält zu wenig Werte -> für diese Objekte wird eine Vorbesetzung der Werte durchgeführt, die restlichen Werte werden mit Werten des letzten Objekts gefüllt; Bestätigung durch den Benutzer notwendig, der die vorbesetzten Werte noch ändern kann

Die Eingabe der fehlenden Parameter wird über Dialoge abgehandelt. Es stellt sich die Frage nach der Gestaltung dieser Dialoge, und wo und wie diese Dialoge gespeichert werden. Die fehlenden Parameter sind Attribute von Objekten, die beispielsweise Default-Werte haben können oder für die es eine gewisse Auswahl an Werten geben kann. Außerdem wird es Abhängigkeiten zwischen einzelnen Werten geben. Es muß möglich sein, diese Abhängigkeiten in irgendeiner Art und Weise zu beschreiben.

### **Mögliche Ansätze für Dialogmodellierung**

1. Jeder Wert wird auf dieselbe Art eingegeben, es gibt keine Default-Werte oder Auswahl. Es existiert nur eine Art der Dialoggestaltung, die für jede Interaktion mit dem Benutzer angewendet wird (“einfache Dialoge”).
2. Die Benutzungsoberfläche wird auf alle Interaktionen mit dem Benutzer spezifisch zugeschnitten. Für alle Eingaben sind die Dialoge in der Benutzungsoberfläche enthalten (“spezifische Dialoge”).
3. Es existiert eine Modellierungssprache für Dialoge. Der Server schickt zusammen mit den einzugebenden Werten eine Beschreibung des Dialogs mit. Die Benutzungsoberfläche ist in der Lage, diese Beschreibung zu interpretieren und entsprechende Eingabedialoge aufzubauen (“Modellierungssprache für Dialoge”).
4. Der Server schickt zusammen mit den einzugebenden Werten Code, der in irgendeiner Weise ausführbar ist und die Interaktion mit dem Benutzer abwickelt. Es existiert ein definierter Mechanismus / Schnittstelle, um die Eingaben des Benutzers nach Beendigung des Dialogs an den Server zu übertragen (“Ausführbarer Code für Dialoge”).

Diese Ansätze werden nun nachfolgend untersucht und bewertet, um eine Auswahl treffen zu können. Die Ansätze schließen sich nicht grundsätzlich gegenseitig aus, eventuell können sie parallel benutzt werden.

Die Gegenüberstellung wird anhand folgender Kriterien durchgeführt:

- (1) Implementierungsaufwand
- (2) Änderbarkeit (statisch und zur Laufzeit)
- (3) Erweiterbarkeit (statisch und zur Laufzeit)
- (4) individuelle Eingabemöglichkeiten
- (5) zusätzliche Server-GUI-Kommunikation
- (6) Abhängigkeiten zwischen einzelnen Eingabewerten
- (7) Komplexität der Lösung
- (8) Laufzeitverhalten

ABBILDUNG 10

Vor- und Nachteile der Ansätze zur Dialogsteuerung

Ansatz	Vorteile	Nachteile
einfache Dialoge	einfache Implementierung (1) keine zusätzliche Kommunikation (5) geringe Komplexität (7) gutes Laufzeitverhalten (8)	keine dynamische Änderbarkeit (2) keine dynamische Erweiterbarkeit (3) individuelle Eingaben werden nicht unterstützt (4) Abhängigkeiten zwischen einzelnen Eingabewerten können nicht berücksichtigt werden (6)
spezifische Dialoge	individuelle Eingaben möglich (4) Kommunikation auf Applikationsobjekte beschränkt (5) Berücksichtigung von Abhängigkeiten möglich (6) gutes Laufzeitverhalten (8)	nur statisch erweiterbar (2) Änderungen in Server/Datenbank ziehen Änderungen in Benutzungsoberfläche nach sich (3)
Modellierungssprache	dynamisch erweiterbar (2) bei Server-Änderungen keine Änderungen im GUI Code notwendig (3) individuelle Eingabe möglich (4) Abhängigkeiten zwischen einzelnen Eingabewerten modellierbar (6)	großer Implementierungsaufwand (1) zusätzliche Kommunikation für Dialogbeschreibung notwendig (5) Komplexität der Modellierungssprache, individuelle Eingaben und Definition von Abhängigkeiten müssen möglich sein (7) Laufzeit für Parsing der Dialogbeschreibung (8)
Ausführbarer Code	Dialoge können sich zur Laufzeit ändern (2) sehr generische Schnittstelle, spezielle Anpassungen sind möglich (3) individuelle Eingaben möglich (4) Berücksichtigung von Abhängigkeiten möglich (6) direkt ausführbar, kein Parsing auf GUI-Seite notwendig (8)	Kommunikationsoverhead für Code, eventuell sehr groß (5) Definition eines Mechanismus / einer Dialogschnittstelle notwendig (7) zu generische Schnittstelle, hohe Komplexität der Lösung (7) Verlust der Kontrolle des Laufzeitverhalten (8)

## **Bewertung der vier Ansätze**

Der erste Ansatz (“Einfache Dialoge”) wird dem Kriterium individueller Eingaben und Abhängigkeiten zwischen einzelnen Werten nicht gerecht. Dieses Kriterium ist aber ein Muß, deshalb scheidet dieser Ansatz aus.

Der zweite Ansatz (“Spezifische Dialoge”) wird der zwingenden Forderung nach Änderbarkeit und Erweiterbarkeit nicht gerecht und scheidet deswegen ebenfalls aus. Eine speziell zugeschnittene Benutzungsoberfläche ist nicht erwünscht.

Der dritte Ansatz (“Modellierungssprache”) ist ein Ansatz, der die Entwicklung einer komplexen Modellierungssprache bedeuten würde. Da der Endkunde aber später selbst Dialoge erstellen und einbinden können soll, muß eine solch komplizierte Lösung verworfen werden. Eine Modellierungssprache wäre zu formal, sie würde nicht akzeptiert werden. Außerdem würde das Parsen der Dialogbeschreibung sehr viel Laufzeit kosten.

Der letzte Ansatz (“Ausführbarer Code”) ist wohl der interessanteste von allem. Hier ist eine völlig individuelle Gestaltung der Dialoge möglich. Trotzdem können die Dialoge außerhalb der Benutzungsoberfläche und damit unabhängig von ihr erstellt werden. Es muß geprüft werden, inwieweit dieser Ansatz überhaupt realisierbar ist.

## **Machbarkeitsstudie für Ansatz “Ausführbarer Code”**

Noch ist die zur Implementierung ausgewählte Sprache nicht vorgestellt worden. Für die hier durchgeführte Machbarkeitsstudie wird die Sprache Java gewählt, da sie dynamische Sprachkomponenten enthält, die Möglichkeiten zur Implementierung des vierten Ansatzes bieten. Diese Studie verhilft somit gleichzeitig zur Bewertung von Java (siehe “Auswahl der Implementierungssprache” auf Seite 59).

In Java ist es möglich, Klassen zur Laufzeit zu laden und daraus Objekte zu instanzieren. Ein Dialog zum Einlesen von Parametern kann somit in einem eigenen Objekt implementiert werden, dessen Klasse beim Anstoßen einer Aktion dynamisch geladen wird.

Es ist außerdem möglich, dem Konstruktor bei der Instanzierung Parameter zu übergeben. Diese Parametrisierung ist wichtig, um einen Dialog für ein bestimmtes Objekt, für das eine Aktion ausgeführt werden soll, zu personalisieren.

Außerdem kann der Dialog somit auch Funktionalität nutzen, die vom Hauptprogramm angeboten wird, zum Beispiel die bereits aufgebaute Netzwerkverbindung zum Applikationsserver.

Die Eingabe der Attribute kann aus Sicht der Benutzungsoberfläche entweder synchron (keine Eingaben im Hauptprogramm möglich, “modaler” Dialog) oder asynchron verlaufen. Aus Sicht des Servers sollte sie stets asynchron sein, damit der Server nicht warten muß, bis der Dialog ausgefüllt worden ist. Es ist außerdem erstrebenswert, im Server keine Kontextinformationen für einen Dialog zu kreieren, da dies beim Auftreten eines Fehlers eine ganze Reihe von Fehlerfällen nach sich ziehen würde, die abgedeckt sein müßten. Der Server wäre bezüglich der Dialoge und den damit verbundenen Aktionen zustandsbehaftet.

Wenn die Informationen über Aktion und Objekte, mit deren Hilfe der Dialog erzeugt worden ist, dem Dialog als Parameter übergeben werden, so ist er in der Lage, alle für diese Aktion notwendigen Parameter nach Beendigung der Benutzereingaben eigenständig an den Server zu übertragen. Dies bedeutet für den Server, daß er zuvor keine Kontextinformationen erzeugen und speichern muß. Somit muß er auch keine Fehlerbehandlung im Fehlerfall machen. Dies vereinfacht die Logik im Server erheblich. Es wird erreicht, daß der Server bezüglich der Dialoge und Aktionen zustandslos ist.

Die Benutzungsoberfläche muß sich ebenfalls keine Kontextinformationen bezüglich der Dialoge merken. Einem Dialog werden Aktion und Objekte übergeben, durch die er ausgelöst worden ist. Er kann anschließend, nach Eingabe aller notwendigen Parameter, das Auslösen eines Service Request komplett übernehmen. Erst jetzt erzeugt der Server ein Service Request Objekt. Während der Eingabe kann der Dialog mit dem Server kommunizieren, um zum Beispiel Wertüberprüfungen zu machen.

Die gesamte Logik für die Eingabe der Parameter kann im Dialog selbst verankert werden:

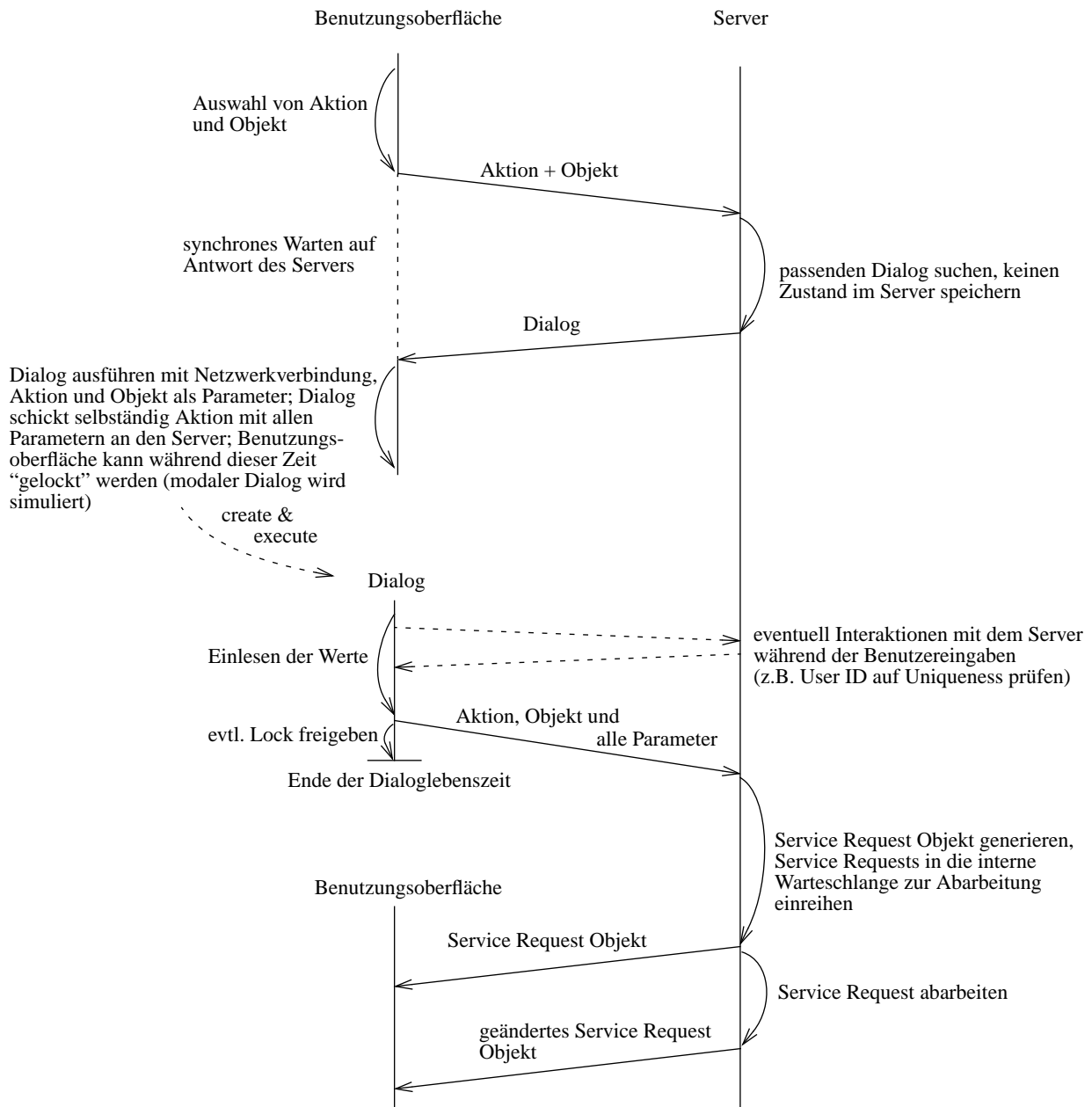
- Konsistenzprüfungen für Werte  
(z.B. Zahl innerhalb eines Wertebereichs)
- statische und dynamische Default-Belegungen  
(z.B. Home-Verzeichnis ist /home/\$USER, wobei \$USER einen zuvor eingegebenen Wert repräsentiert)
- beliebiger Wert aus einer Auswahl  
(z.B. nächste freie Group ID)
- Abhängigkeiten  
(z.B. falls Security Level des System entsprechend, dann ist Password Aging immer eingeschaltet, ansonsten ist es optional)
- beliebige Interaktionen mit dem Server  
(z.B. User ID auf Uniqueness prüfen)

Die Abbildung auf der nächsten Seite beschreibt das Kommunikationsprotokoll zwischen Benutzungsoberfläche und Server zur Erstellung eines Service Request mit zusätzlichem Dialog.

Anmerkung:

Da die Benutzungsoberfläche später so implementiert sein wird, daß dieser Dialog nicht über das Fenstersystem des Betriebssystems modal gestaltet werden kann (Konkret: Das Applet kann keinen modalen Dialog erzeugen, das Sperren des Internetbrowsers ist nicht möglich), muß es die Möglichkeit geben, diese Modalität zu simulieren. Dafür kann ein 'Lock' gesetzt werden, so daß während der Ausführung eines Dialoges keine Eingaben in irgendeinem anderen Fenster der Benutzungsoberfläche möglich sind. Beim Beenden des Dialogs muß dieser Lock wieder freigegeben werden.

ABBILDUNG 11 Erstellung von Service Requests mit zusätzlichem Dialog



### Detailierung

Der Dialog verwendet eine Methode zur Kommunikation mit dem Server. Sie wird als Attribut im Dialog verankert und bei der Dialoginstanziierung gesetzt. Der Wert wird dem Dialog als Parameter übergeben.

```

import java.lang.reflect.*;
public Method send;
    
```

Diese Methode dient zum synchronen Senden an den Server. Sie braucht einen String als Parameter und liefert den Antwortstring des Servers. Ihr Aufruf erfolgt folgendermaßen:

```
// Parametervorbereitung, der zu übergebende String sei 'aString'  
// Parameter müssen als "array of objects" übergeben werden, hier  
// ein "array of strings"  
String[] parameter = new String[1];  
parameter[0] = aString;  
  
// Aufruf der send-Methode  
try {  
    String result = (String)send.invoke(null, parameter);  
} catch (IllegalAccessException e) {  
    // illegal access exception behandeln; tritt auf, wenn das  
    // Aufrufen der Methode nicht erlaubt ist (z.B. private Methode)  
} catch (InvocationTargetException e) {  
    // invocation target exception behandeln; tritt auf, wenn  
    // innerhalb der aufgerufenen Methode eine Exception auftritt  
}
```

Beim Aufruf von 'invoke' wird als erster Parameter eine Referenz auf das Objekt übergeben, das diese Methode implementiert. Er kann 'NULL' sein, wenn es sich um eine statische Klassenmethode handelt.

Der zweite Parameter ist die Parameterliste für den eigentlichen Methodenaufruf. Dies ist der String, der an den Server geschickt wird.

Ausprogrammierte Tests in Java ergaben, daß sich diese Variante der Dialogausführung nicht nur so implementieren läßt, sondern auch bezüglich des Antwortverhaltens performant genug ist, da Dialoge im Normalfall nicht sehr groß sind. Ihre Größe liegt bei ca. 10 KByte pro Dialogfenster, ca. 80% der Dialoge kommen mit zwei oder weniger Fenstern aus. Legt man nun einen schlechten Netzwerkdurchsatz (ca. 10 KByte pro Sekunde) zugrunde, so ist die Antwortzeit der Benutzungsoberfläche für die meisten Dialoge weniger als zwei Sekunden. Dies entspricht der maximal akzeptierten Antwortzeit einer Benutzungsoberfläche, die bei zwei, höchsten drei Sekunden liegen darf.

Damit wird als Ansatz für die Implementierung der Dialoge der Ansatz der dynamisch geladenen ausführbaren Dialoge gewählt. Gleichzeitig muß festgehalten werden, daß sich Java zur Implementierung dynamischer Komponenten eignet.

### 3.1.3 Navigationsbereich

*Bezug zu Anforderung: Benutzungsoberfläche visualisiert die Hierarchie der Applikationsobjekte. Dafür wird ein spezieller Bereich auf dem Bildschirm reserviert.*

In einem ersten Ansatz soll die Struktur des Navigationsbaumes der Struktur der Objekte im Server entsprechen. Ein Element des Baumes entspricht dabei einem Objekt. Ausgehend von einem Wurzelobjekt werden die Beziehungen übernommen, die der Server liefert. Dafür benötigt die Benutzungsoberfläche nur eine Query, die der Server für jedes Objekt beantworten können muß. Sie wird als Default-Query bezeichnet.

“Fordere alle Objekte an, die zu einem Objekt X die Beziehung Y haben.”

Da die Beziehungen typisiert sind, muß zusätzlich zur Parametrisierung durch eine Objekt ID auch ein Beziehungstyp spezifiziert werden. Beispiele für Beziehungen sind "ist-abgeleitet-von", "enthält" oder "ist-Instanz-von".

Damit ist die Benutzungsoberfläche in der Lage, ausgehend von einem Objekt, das die Wurzel bildet, alle Applikationsobjekte entsprechend ihrer Struktur in der Datenbank anzufordern und darzustellen.

### 3.1.4 Informationsbereich

*Bezug zu Anforderungen: Nicht nur die Objekthierarchie, sondern auch detailliertere Informationen zu den Objekten soll dargestellt werden. Dies wird im Informationsbereich erledigt.*

Um die Darstellung der Attribute zu ermöglichen, wird eine Präsentationsform für Attribute als Basis implementiert, die für alle Objekte gültig ist. Dies ist eine Listendarstellung aller Attribute inklusive der Attributwerte.

Für die Präsentation von untergeordneten Objekten werden zwei Darstellungsmethoden als Basis implementiert:

- kompakte Darstellung mit Icons und Text
- Tabellendarstellung mit jeweils allen Attributen und Werten

Analog zum dynamischen Laden von Dialogen werden die Präsentationsformen für den Informationsbereich ebenfalls in Klassen implementiert, die zur Laufzeit geladen werden. Dies bietet auch die Möglichkeit, beliebige Präsentationsformen für spezielle Darstellungen einzubinden. Jede Präsentationsform besitzt einen eigenen Auswahlknopf, der am oberen Rand des Informationsbereichs dargestellt wird.

Die Präsentation erfolgt durch die Instanziierung einer Präsentationsklasse. Im Konstruktor wird eine Referenz auf den Informationsbereich übergeben, in dem nachher die Darstellung erfolgt. Daneben werden noch das im Navigationsbaum selektierte Objekt, eine Liste aller untergeordneten Objekte und die Kommunikationsschnittstelle zum Server übergeben. Dadurch kann auch ein Präsentationsobjekt über die bereits aufgebaute Verbindung mit dem Applikationsserver kommunizieren.

### 3.1.5 Kommunikation mit Applikationsserver

*Bezug zu Anforderungen: Alle Objekte werden vom Server zur Verfügung gestellt. Der Server ist über das Netzwerk erreichbar, die Benutzungsoberfläche kommuniziert mit ihm über ein zu definierendes Protokoll.*

Die Benutzungsoberfläche wird nicht isoliert implementiert sein, sondern zusammen mit anderen Komponenten integriert. Die Basisfunktionalität der Netzwerkkommunikation wird später durch andere Teile abgedeckt werden. Diese sind allerdings noch nicht implementiert und können somit von dem zu erstellenden Prototyp nicht verwendet werden. In dieser Arbeit wird die Netzwerkkommunikation deshalb konzeptionell mitentwickelt und prototypisch implementiert werden.

Die Kommunikation wird sowohl von Seite der Benutzungsoberfläche als auch von Serverseite angestoßen. Der Datenverkehr ist dazu in zwei Kanäle aufgeteilt. Auf

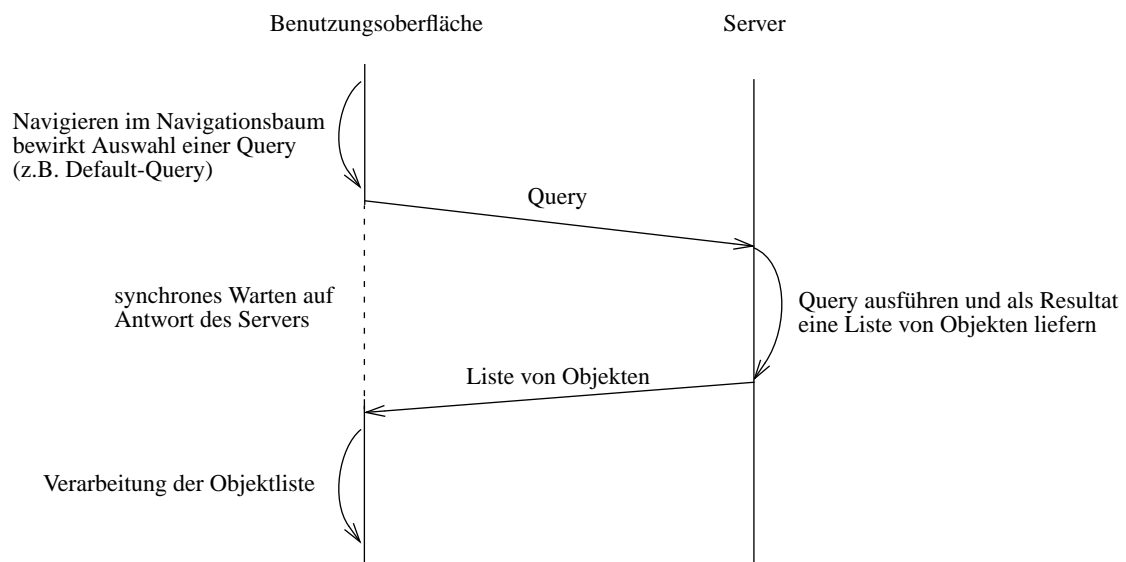
einem Kanal schickt die Benutzungsoberfläche Informationen an den Applikationsserver, auf dem anderen umgekehrt.

Grundsätzlich schickt die Benutzungsoberfläche Queries und Service Requests an den Server. Umgekehrt schickt der Server Applikationsobjekte an die Benutzungsoberfläche. Diese hat die Objekte entweder durch eine Query aktiv angefordert, oder durch die Ausführung eines Service Requests wurde ein Objekt modifiziert, das in der Benutzungsoberfläche visualisiert wird und deshalb aktualisiert werden muß (siehe “Notifikations-Mechanismen” auf Seite 48).

Aktionen bestehen aus einer Liste von Objekten, einer Parameterliste und der Aktion selbst. Die Objektliste besteht aus Objekt IDs, die Parameterliste besteht aus Attributnamen und Attributwerten. Aktionen bilden die Basis für Service Requests. Diese werden vom Server asynchron abgearbeitet und damit auch asynchron beantwortet.

Queries werden vom Server synchron beantwortet, er liefert stets ein Ergebnis zurück, das gegebenenfalls auch leer sein kann. Das Ergebnis einer Query ist eine Liste von Applikationsobjekten.

ABBILDUNG 12 Kommunikationsprotokoll für Queries



Um die Kommunikation zwischen Benutzungsoberfläche und Server möglichst einfach zu gestalten und auch schnell implementieren zu können, wird als Basisdatenpaket für das Protokoll ein String gewählt. Die Benutzungsoberfläche schickt dem Server einen String und kann von ihm einen String empfangen. Dies hat aufgrund der “lesbaren” Notation zusätzlich den Vorteil, Kommunikationselemente über eine Konsole eingeben und dadurch gezielt Tests durchführen zu können.

Neben Queries und Service Requests müssen noch Verwaltungsinformationen wie das Anmelden des Benutzers, das Eröffnen einer Sitzung mit dem Server oder das Holen eines Aktionsdialogs berücksichtigt werden. Als Query ist bisher nur die Default-Query vorgestellt worden (siehe “Navigationsbereich” auf Seite 34).

Die einzelnen Protokollelemente werden durch Tokens repräsentiert, gefolgt von Parametern. Die Trennung erfolgt durch mindestens ein Leerzeichen.

ABBILDUNG 13

Protokollelemente "Benutzungsoberfläche an Applikationsserver" für Minimalfunktionalität

Protokollelement	Bedeutung	Kommunikationsart
L <username>	Login des Users beim Server	asynchron
P <encrypted password>	Verschlüsseltes Paßwort des Benutzers	asynchron
Q <query id> "any" <object id>	Default-Query für Objekt <object id>	synchron
A <object id> <action> <parameter #1> ... <parameter #n>	Aktion <action> mit Objekt <object id> und #n Parametern	asynchron
D <object id> <action>	Dialog zu Objekt und Aktion	synchron

ABBILDUNG 14

Protokollelemente "Applikationsserver an Benutzungsoberfläche" für Minimalfunktionalität

Protokollelement	Bedeutung
O <id> <name> <type> "Attributes" <attribute #1> ... "Actions" <action #1> ...	beschreibt ein Applikationsobjekt
S <user name> <session id>	Sitzung für diesen Benutzer wurde eröffnet
s <session id>	Sitzung wurde geschlossen
Q <query id> <session id>	die folgenden Objekte sind Ergebnis dieser Query
q <query id>	Ende der Folge von Objekten dieser Query
D <dialog name>	Name eines Aktionsdialogs
E <error> <command>	der Fehler <error> ist bei der Ausführung des Befehls <command> im Server aufgetreten

## 3.2 Verbesserung der Minimalfunktionalität

### 3.2.1 Applikationsunabhängige Objektgliederung

*Bezug zu Anforderungen: Beliebige Strukturierung der Objekte ist erwünscht, es muß möglich sein, die durch den Server vorgegebene Struktur zu "überschreiben".*

Um die applikationsunabhängige Objektgliederung in der Benutzungsoberfläche zu realisieren, wird eine interne Beschreibung benötigt, die festlegt, welches Objekt an welcher Stelle im Navigationsbaum präsentiert werden soll. Zur Definition dieser Zusammenhänge wird eine Layout-Sprache benötigt.

Falls eine für eine Person leserliche Syntax gewählt wird, so kann die Beschreibung auch außerhalb der Benutzungsoberfläche definiert und somit dem Benutzer zugänglich gemacht werden.

Die Semantik der Layout-Sprache läßt sich wie folgt beschreiben:

1. Gegeben ist ein leerer Navigationsbaum.
2. Definiere als erstes einen Startknoten, der Wurzel des Navigationsbaumes ist.
3. Definiere für jeden Knoten eine Menge Knoten, die ihm untergeordnet sind.

Die Knoten des Navigationsbaumes werden mit Objekten vom Server gefüllt werden. Um diese Objekte vom Server anzufordern, werden Queries benutzt (Definition siehe Seite 10). Das Layout des Navigationsbaumes besteht somit aus geschachtelten Queries. Die Schachtelung spiegelt die Baumstruktur wieder.

Um den Prototyp schneller implementieren zu können, wird die Syntax der Layout-Sprache der Query-Syntax angepaßt. Das bedeutet, daß die Layout-Beschreibung aus Queries besteht, die direkt an den Server geschickt werden können. Im späteren Produkt wird eine andere Syntax verwendet werden, für Queries beispielsweise SQL-Syntax.

In der Layout-Sprache muß es möglich sein, beliebige Baumstrukturen zu definieren. Um die Möglichkeiten der Objektgruppierung zu erweitern, können logische Knoten definiert werden, die kein Applikationsobjekt repräsentieren, sondern nur der logischen Gliederung dienen.

In einem Baum treten somit zwei Arten von Knoten auf:

- Knoten, die ein Applikationsobjekt repräsentieren
- Knoten, die der logischen Gliederung der Applikationsobjekte dienen

Für jeden Knoten kann eine Query angegeben werden, der die untergeordneten Knoten beschreibt. Wird keine Query für einen Knoten definiert, so wird die Default-Query (siehe "Navigationsbereich" auf Seite 34) angewendet, um die untergeordneten Knoten zu bestimmen, es sei denn, der Knoten ist als Blatt des Baumes definiert. Er besitzt in diesem Fall keine untergeordneten Knoten.

Ein Beispiel soll nun verdeutlichen, was bisher mit der Definition eines Layouts erreicht werden kann. Man stelle sich eine Applikation vor, die Benutzer, Benutzergruppen und benutzbare Software auf UNIX-Computersystemen verwaltet. Dabei wird der Zugriff auf Software durch Benutzergruppen modelliert. Alle Benutzer einer Gruppe haben Zugriff auf bestimmte Software.

Die Applikation speichert alles in Objekten ab, die Objekte haben Beziehungen zueinander. Es sind nur die Beziehungen Benutzer zu Benutzergruppe und Benutzergruppe zu benutzbarer Software gespeichert. Ferner sei die Beziehung von einem Benutzer zu der von ihm benutzbaren Software durch eine Query berechenbar.

Mit dem vorgestellten Layout läßt sich der Navigationsbaum für diese Applikation beliebig aufbauen. Wenn kein Layout spezifiziert wird, so wird folgenden Schachtelung auf dem Bildschirm präsentiert: "Benutzer -> Benutzergruppe -> benutzbare Software". Man kann diese vorgegebene Strukturierung "überschreiben", indem ein Layout definiert wird, daß durch eine Query die einem Benutzer zugängliche Software unmittelbar unter dem Benutzer präsentiert, ohne zuerst die Gruppe des Benutzers anzeigen zu müssen.

Das Beispiel zeigt, daß es nützlich sein kann, in einem speziellen Kontext die Struktur der Objekte zu überschreiben. Falls die Benutzer an mehreren Stellen im Navigationsbaum auftreten, so wird nicht für jedes Vorkommen die Beziehung "Benutzer -> Benut-

zergruppe -> installierte Software" überschrieben werden sollen, sondern nur an einer bestimmten Stelle.

Andererseits kann es aber auch notwendig sein, für alle Vorkommen von Objekten gleichen Typs ein Layout zu definieren. Dieses Layout ist für den gesamten Navigationsbaum und jedes Auftreten eines Objekts diesen Typs gültig. Es hat einen generellen Charakter.

Soll in dem oben beschriebenen Beispiel die Beziehung von installierter Software zu einem Computersystem hergestellt werden und ist diese Beziehung durch eine Query berechenbar, so kann gewünscht sein, daß zu jedem Vorkommen eines Objekts "installierte Software" das zugehörige Computersystem visualisiert wird.

Aus diesen beiden Anforderungen ("Struktur für speziellen Kontext" und "generelle Struktur") folgt, daß es zwei verschiedene Arten von Layouts geben muß, ein kontextabhängiges und ein allgemeingültiges. Das erstere wird "spezielles Layout", das andere "generelles Layout" genannt. Das spezielle Layout gilt nur für einen bestimmten Kontext, also einen bestimmten Zweig im Navigationsbaum, das generelle gilt für alle Objekte eines bestimmten Typs und somit für den gesamten Navigationsbaum.

Diese beiden Layouts müssen in Zusammenhang gebracht werden. Es wird eine Regel benötigt, die eine Priorisierung der beiden Layouts ausdrückt. Das spezielle Layout wird so definiert, daß es eine höhere Priorität als das generelle hat. Zuerst wird geprüft, ob für einen bestimmten Knoten ein spezielles Layout existiert. Falls ja, so wird dieses genommen. Falls nein, so wird im generellen Layout nachgeschaut, ob eine Definition für diesen Objekttyp existiert. Nur falls auch keine generelle Definition existiert, wird die Default-Query benutzt, um untergeordnete Objekte anzufordern. Die Regel zur Erzeugung des Verwaltungsbaumes lautet somit:

"Erst spezielles Layout, dann generelles, dann Default-Query."

Betrachtet man das obige Beispiel der auf einem Computersystem installierten Software, so wird klar, warum diese Regel Sinn macht. Man möchte stets den Zusammenhang "installierte Software" zu "Computersystem" sehen. Nur für einen bestimmten Kontext will man beispielsweise eine Beziehung "installierte Software" zu "Softwarehersteller" sehen. Das spezielle Layout muß in diesem Fall das generelle "überschreiben" und muß folglich eine höhere Priorität als das generelle haben.

Die Beantwortung von Queries ist ein sehr komplexer Sachverhalt. Um in dieser Diplomarbeit eine Chance zu haben, den beschriebenen Sachverhalt überhaupt prototypisch implementieren zu können, wird ein Server mit der Fähigkeit benötigt, Queries zu beantworten. Diese Queries müssen möglichst einfach sein, da sonst der Implementierungsaufwand zu groß wäre.

Zum anderen wird sich zeigen, daß durch die Einfachheit der Queries der Weg für ein weiteres Designkriterium geebnet wird. Es eröffnet sich die Möglichkeit, Queries und ihre Ergebnisse lokal auf Seite der Benutzungsoberfläche zu speichern und somit einen Caching-Mechanismus auf Ebene der Queries zu etablieren (siehe "Lokaler Cache" auf Seite 46). Dies wäre nicht möglich, wenn im Layout SQL-Queries verwendet werden würden, wobei dies wiederum nicht automatisch bedeutet, daß der Server später keine SQL-Queries beantworten kann.

Um der Anforderung nach Einfachheit der Queries zu entsprechen, bezieht sich jede Query immer nur auf einen Objekttyp oder auf bestimmte Objekte. Da innerhalb einer Hierarchieebene Objekte verschiedenen Typs erlaubt sein sollen, muß folglich mehr als eine Query pro Hierarchieebene erlaubt sein. Ein Objekt wird allerdings nur einmal als untergeordnetes Objekt präsentiert, auch wenn es von mehr als einer Query geliefert wird (entspricht Aliasing bezüglich der Darstellung in der gleichen Hierarchieebene). Liefert ein Befehl in einem anderen Teilbaum dasselbe Objekt, so wird es zweimal präsentiert.

Eine Query in einer bestimmten Schachtelungstiefe des Layouts bezieht sich nicht automatisch auf Objekte der übergeordneten Query, sie kann sich auf beliebige Objekte und Objekttypen beziehen und ist somit von der übergeordneten unabhängig. Dadurch kann eine willkürliche Struktur der Applikationsobjekte erzeugt werden. Soll das übergeordnete Objekte einbezogen werden, so kann es innerhalb der Query referenziert werden.

Die Möglichkeit der Einbeziehung des übergeordneten Objekts ist für eine sinnvolle Struktur des Navigationsbaumes elementar wichtig. Man stelle sich vor, daß unter Benutzern ihre Benutzergruppen präsentiert werden sollen. Wenn man eine generelles Layout spezifiziert, daß einfach für jeden Benutzer Benutzergruppen vom Server anfordert, so würden unter jedem Benutzer alle Benutzergruppen hängen, und nicht nur die, denen er angehört. Das übergeordnete Objekt, in diesem Fall der Benutzer, muß einbezogen werden, damit seine Benutzergruppen ermittelt werden können.

Definition der Variable '\$Parent':

Die Variable '\$Parent' meint das einem anderen Objekt übergeordnete Applikationsobjekt im Navigationsbaum.

Wichtig ist hier der Begriff 'Applikationsobjekt'. Gemeint ist also das zuletzt vom Server geholte, reale Objekt und nicht ein eventuell übergeordnetes logisches Objekt (entspricht Inhalt eines logischen Knotens). Der Algorithmus zu Bestimmung des Wertes '\$Parent' lautet:

Gegeben ein beliebiger Baum, bestehend aus logischen Objekten und Applikationsobjekten, und ein bestimmtes Objekt dieses Baumes. Von der Wurzel des Baumes bis zum gegebenen Objekt existiert ein Pfad. Verfolge diesen Pfad in umgekehrter Richtung, ausgehend vom gegebenen Objekt. Das erste erreichte Objekt, das kein logisches Objekt ist, ist der Wert von '\$Parent'.

Um ein bestimmtes Applikationsobjekt als Wurzel des Navigationsbaumes zu definieren, muß es außerdem die Möglichkeit geben, ein Objekt über seine Objekt ID anzufordern.

Insgesamt gibt es drei verschiedene Arten von Queries:

- Queries, die eine beliebige Anzahl von Objekten anfordern, indem sie einen bestimmten Objekttyp spezifizieren ("liefere alle Objekte vom Typ 'Computersystem'")
- Queries, die eine beliebige Anzahl von Objekten anfordern, indem sie einen bestimmten Objekttyp und zusätzlich noch die ID eines Objekts spezifizieren, mit dem die Objekte in Beziehung stehen ("liefere alle Objekte vom Typ

- ‘installierte Software’, die mit dem Objekt mit Objekt ID ‘Group10’ in Beziehung stehen”)
- Queries, die eine bestimmtes Objekt mit Hilfe seiner Objekt ID anfordern (“liefere Objekt mit ID ‘root01’”)

Um die Schachtelung der Layout-Definitionen möglichst lesbar zu gestalten, wird auf unübersichtliche Klammerung verzichtet. Sie wird stattdessen durch Einrückungen ausgedrückt.

### Beispiel für Definition “Spezielles Layout”

```
ROOT Virtual="User/Security"
  NODE Virtual="Employees"
    NODE Query Type="Employee"
  NODE Virtual="Roles"
    NODE Query Type="Role"
  NODE Virtual="Rollentemplates"
    NODE Query Type="Roletemplate"
  NODE Virtual="Service Requests"
    NODE Query Type="ServiceRequest"
```

Dieses Beispiel beschreibt eine mögliche Navigationsstruktur für die User/Security Applikation. Die Wurzel bildet der logische Knoten “User/Security”, dem wiederum die vier logischen Knoten “Employees”, “Roles”, “Rollentemplates“ und “Service Requests” untergeordnet sind.

Da für die untergeordneten Knoten jeweils nichts definiert ist, wird zur Navigation die Default-Query benutzt, es sei denn, es existiert noch ein generelles Layout.

### Beispiel für Definition “Generelles Layout”

```
TYPE Employee NODE Query Type="Role" Association=$Parent
```

Diese generelle Layout-Definition ist für alle Objekte vom Typ “Employee” gültig. Unter jedem Employee werden alle Objekte vom Typ “Role” präsentiert, die eine Beziehung zu diesem Employee haben. Damit kann die Tatsache “Rolle ist Employee zugewiesen” ausgedrückt werden. Die Spezifikation der Beziehung zum Vaterobjekt ist wichtig, damit nicht unter jedem Employee alle Rollen präsentiert werden.

Es ist möglich, die Sichtweise auf die Applikationsobjekte vollständig anzupassen, indem jeder Baum bis hin zu allen Blättern definiert wird. Neben dem Schlüsselwort ‘NODE’ existiert noch das Schlüsselwort ‘LEAF’, das ausdrückt, daß die hier definierten Objekte Blätter des Baumes sind und daß beim Versuch des Öffnens der untergeordneten Hierarchieebene keine Objekte vom Server angefordert werden, auch nicht durch die Default-Query.

Die Strukturierung der Objekte, so wie sie durch die Strukturierung auf Seite des Servers vorgegeben ist, kann auch vollständig übernommen werden. Dazu muß das spezielle Layout minimal definiert werden, es wird nur ein Objekt als Wurzel angegeben. Das generelle Layout bleibt in diesem Fall leer:

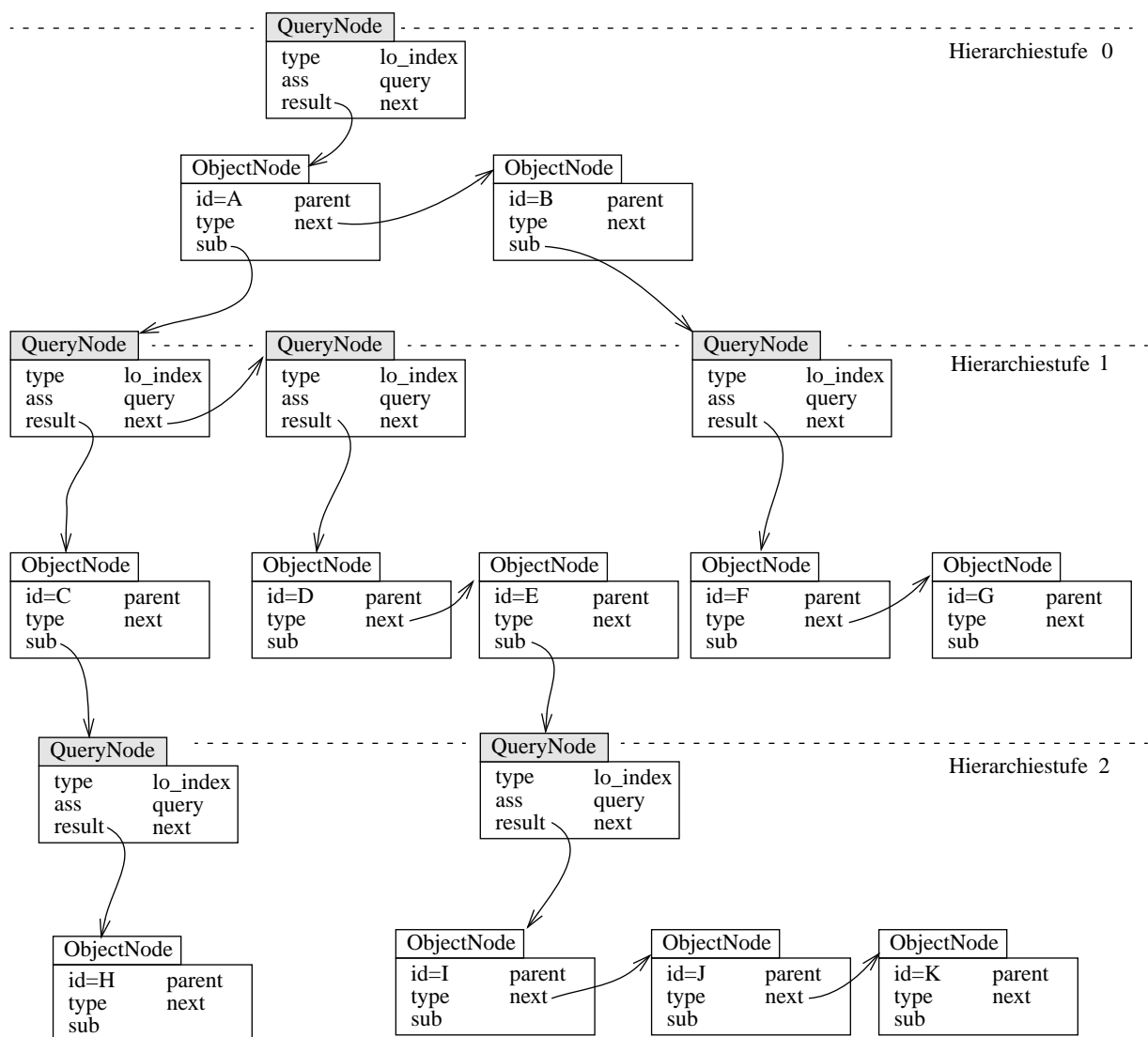
```
ROOT Query ID="theRootObjectID"
```

### 3.2.2 Verwaltungsbaum für Navigationsbaum

Bezug zu Anforderungen: Die Datenstruktur "Verwaltungsbaum" ist für die Realisierung der applikationsunabhängigen Objektgliederung notwendig (siehe auch vorherigen Abschnitt). Zur Realisierung werden Verwaltungsinformationen benötigt, die in einer eigenen Datenstruktur abgelegt werden.

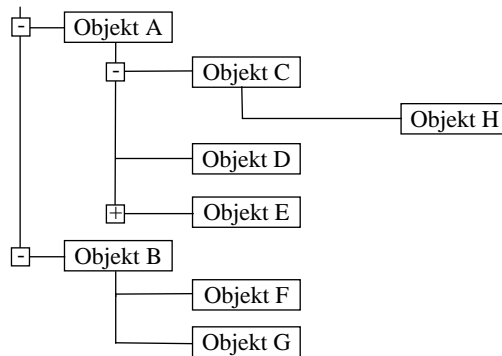
Der Verwaltungsbaum ist eine Datenstruktur, die hinter dem sichtbaren Navigationsbaum steht. Seine Struktur entspricht der Struktur des Navigationsbaumes. Er enthält Verwaltungsinformationen, die den Navigationsbaum mit den Layouts verbinden. Diese Verwaltungsinformationen sind notwendig, um bei Expansion des Navigationsbaumes die passenden Komponenten der Layout-Definitionen zu finden.

ABBILDUNG 15 Beispiel eines Verwaltungsbaumes



Aus den sogenannten ‘ObjectNodes’ der angedeuteten Hierarchiestufen werden die einzelnen Hierarchieebenen des sichtbaren Navigationsbaumes gebildet. Es ergäbe sich der folgende Navigationsbaum.

ABBILDUNG 16 Beispiel eines Navigationsbaumes



Zunächst scheint es überflüssig, eine solche Datenstruktur zusätzlich einzuführen. Man könnte die Verwaltungsinformationen ebenso als Attribute der sichtbaren Elemente des Navigationsbaumes hinzufügen. Es wird sich jedoch zeigen, daß es möglich sein muß, den Verwaltungsbaum zu modifizieren, ohne auch gleichzeitig den Navigationsbaum zu verändern (siehe ‘Lokaler Cache’ auf Seite 46).

Der Verwaltungsbaum wird nach der Regel ‘zuerst spezielles Layout, dann generelles, dann Default-Query’ aufgebaut, der sichtbare Navigationsbaum wird aus dem Verwaltungsbaum abgeleitet, in dem die Struktur übernommen wird und für jeden ObjectNode ein sichtbarer Knoten im Navigationsbaum erzeugt wird. Die sogenannten ‘QueryNodes’ werden nicht in den Navigationsbaum übertragen.

Der Verwaltungsbaum besteht aus zwei verschiedenen Arten von Knoten. Eine Knotenart repräsentiert die Queries (‘QueryNode’), die andere beinhaltet die Objekte, die als Ergebnis einer Query vom Server geliefert werden (‘ObjectNode’).

Bei jeder Expansion wird zunächst aus den Layout-Definitionen die entsprechende Query geholt und an den Server geschickt. Anschließend wird aus der Query ein entsprechender QueryNode erzeugt und die Ergebnisse der Query als ObjectNodes eingehängt. Falls das Ergebnis der Query leer ist, so wird ein leerer ObjectNode eingehangen. Jeder Expansionsschritt erzeugt somit eine Reihe von QueryNodes mit dazugehörigen ObjectNodes. Falls keine weitere Query in einem Layout definiert ist, so wird die Default-Query angewendet, falls der Knoten nicht als ‘LEAF’ definiert ist. Falls doch, so bleibt die Expansion wirkungslos.

Beispieldefinition ‘QueryNode’:

```

public class queryNode {
    public String type;           // Objekttyp der Query
    public String ass;           // Objekt ID des Assoziationsobjekts
    public QueryNode next;       // nächster QueryNode auf gleicher Ebene
    public int lo_index;         // Referenz auf spezielles Layout
    public String query;         // Referenz auf Query
    public ObjectNode result;    // Resultat dieser Query
}
  
```

Beispieldefinition 'ObjectNode':

```
public class objectNode {
    public String id;           // Objekt ID
    public int type;           // Typ Objekts
    public ObjectNode next;    // nächster ObjectNode auf gleicher Ebene
    public ObjectNode parent;  // übergeordneter ObjectNode
    public QueryNode sub;      // untergeordneter QueryNode
}
```

Mit diesen Datenstrukturen läßt sich der Verwaltungsbaum aufbauen. Jede Hierarchieebene besteht dabei aus eine Anzahl von QueryNodes, über die 'next'-Referenz verkettet. Jeder QueryNode verweist über 'result' auf das Ergebnis dieser Query. Dieser Verweis ist niemals leer, falls eine Query kein Objekt als Ergebnis liefert, so wird ein leerer ObjectNode eingehängt.

### Algorithmus zur Erzeugung einer Hierarchieebene

Gegeben sind der Verwaltungsbaum und der dazugehörige Navigationsbaum. Als Eingabe benötigt der Algorithmus einen ObjectNode "theObjectNode", der expandiert werden soll.

```
//erzeuge nächste Hierarchieebene im Verwaltungsbaum
hole Layoutbeschreibung der untergeordneten Objekte für theObjectNode
Start = theObjectNode
while (Layoutbeschreibung nicht leer) {
    generiere Query aus Layoutbeschreibung
    if (Query.Type == virtual) {
        //Query nicht an Server schicken, da für logischen Knoten
        //Liste der "Ergebnisobjekte" direkt verfügbar
        generiere verkettete ObjectNode Liste
        generiere QueryNode mit QueryNode.Result = ObjectNode List
        if (1. Schleifendurchlauf) {
            hänge QueryNode bei Start.sub ein
            Next = QueryNode
        }
        else {
            hänge QueryNode bei Next ein
            Next = Next.next
        }
    }
    else { //Query.Type == real
        //Query an Server schicken, da reale Knoten
        //Liste der "Ergebnisobjekte" wird vom Server geschickt
        generiere verkettete ObjectNode Liste
        generiere QueryNode mit QueryNode.Result = ObjectNode List
        if (1. Schleifendurchlauf) {
            hänge QueryNode bei Start.sub ein
            Next = QueryNode
        }
        else {
            hänge QueryNode bei Next ein
            Next = Next.next
        }
    }
}
```

```
    }
    hole nächste Layoutbeschreibung dieser Hierarchieebene
  }

//erzeuge nächste Hierarchieebene im Navigationsbaum
//QueryHilf ist "Schleifenzähler" über alle QueryNodes
//ObjectHilf ist "Schleifenzähler" für ObjectNodes eines QueryNode
QueryHilf = ObjectNode.sub
while (QueryHilf nicht leer) {
  ObjectHilf = QueryHilf.Result
  while (ObjectHilf nicht leer) {
    if (ObjectHilf.Type == real) {
      hole Objekt aus Objekt Cache
      erzeuge entsprechendes sichtbares Element
    }
    elseif (ObjectHilf.Type == virtual) {
      erzeuge entsprechendes sichtbares Element
    }
    else {
      //"leerer" ObjectNode, nichts machen
    }
    ObjectHilf = ObjectHilf.next //nächster ObjectNode
  }
  QueryHilf = QueryHilf.next //nächster QueryNode
}
```

### Startzustand

Beim Start der Benutzungsoberfläche soll der Navigationsbaum in einem definierten Zustand präsentiert werden. Der Zustand ist zu verstehen als die Gesamtinformation, bis zu welcher Tiefe die Teilbäume geöffnet sind. Diese Gesamtinformation kann aber nicht einfach abgespeichert werden und beim nächsten Start benutzt werden, da sie durch eventuelle Änderungen ungültig geworden sein könnte. Dies könnte im schlimmsten Fall sogar bedeuten, daß der gesamte Navigationsbaum leer ist. Man stelle sich vor, der Navigationsbaum ist minimal definiert über die ID eines Wurzelobjekt. Wird dieses Objekt gelöscht, so greift die Query, die die Wurzel des Baumes definiert, ins Leere, und es kann kein Baum dargestellt werden. Deshalb empfiehlt es sich, die Wurzel des Baumes als logischen Knoten zu definieren und in der gesamten Layoutbeschreibung möglichst auf IDs zu verzichten.

Um beim Start der Benutzungsoberfläche nicht nur einen Wurzelknoten zu haben, wird ein Öffnen des Wurzelknotens durchgeführt und die nächste Hierarchieebene erzeugt. Das Öffnen hat außerdem noch die Selektion des Wurzelknotens zur Folge. Dadurch wird auch der Informationsbereich mit der entsprechenden Präsentationsform gefüllt. Der Informationsbereich ist folglich auch nicht leer.

Im folgenden Abschnitt werden die beiden Caches erläutert. Anschließend wird gezeigt, wie der Notifikations-Mechanismus auf diesen Caches aufsetzt.

### 3.2.3 Lokaler Cache

*Bezug zu Anforderungen: Um die Performance der Kommunikation von Benutzungsoberfläche und Applikationsserver zu verbessern, wird ein lokaler Cache benutzt.*

Der lokale Cache besteht aus zwei Teilen, einem Cache für die Objekte und einem für die Queries.

Der Objekt Cache speichert Applikationsobjekte zusammen mit ihren Attributen. Der Schlüssel ist dabei die Objekt ID. Zum Schreiben in den Cache wird ein Applikationsobjekt selbst, zum Lesen nur eine Objekt ID benötigt.

Der Objekt Cache wird zu Beginn mit einer gewissen Größe angelegt. Ist er voll, so wird zunächst versucht, ihn zu vergrößern. Falls dies fehlschlägt, so wird eine Auslagerungsstrategie angewendet, um Platz zu schaffen.

Analog gilt für den Query Cache, daß er zu Beginn mit einer Anfangsgröße initialisiert wird, später aber dynamisch wachsen kann. In ihm werden die Queries gespeichert. Eine Query besteht aus einer Query ID, die von der Benutzungsoberfläche vergeben wird, einem Typ und/oder einer Objekt ID und dem vom Server gelieferten Resultat. Das Resultat ist eine Liste von Objekt IDs. Das Lesen aus dem Query Cache erfolgt über die Query ID.

Um eine geeignete Wahl der Auslagerungsstrategie treffen zu können, muß zunächst untersucht werden, was 'Auslagern' für jeden der beiden Caches überhaupt bedeutet.

Der Objekt Cache ist, für sich betrachtet, eine Sammlung von Applikationsobjekten, die in keiner Weise zusammenhängen. Daher ist jedes Objekt ein Kandidat zur Auslagerung, falls der Cache überläuft. Auslagern heißt beim Objekt Cache folglich: "Benutze eine Auslagerungsstrategie auf alle Objekte im Cache; dadurch wird ein Objekt ausgewählt, daß dann gelöscht wird; es müssen keine Abhängigkeiten zu anderen Objekten im Cache berücksichtigt werden." Als Auslagerungsstrategie wird am besten die Strategie "Last Recently Used" (LRU-Strategie) angewendet, bei der das Objekt aus dem Cache gelöscht wird, auf das am längsten nicht mehr zugegriffen worden ist.

Anders verhält es sich hingegen beim Query Cache. Die Struktur des Verwaltungsbaumes kann zu einem verbesserten Auslagerungsmechanismus beitragen, da sich aus ihrer Abhängigkeiten zwischen Queries ableiten lassen. Verbesserung bedeutet hier, daß eventuell nicht nur eine, sondern mehrere Queries gelöscht werden können und somit wieder mehr Platz im Query Cache frei wird. Dadurch muß weniger oft ausgelagert werden.

Wird mit Hilfe einer Auslagerungsstrategie eine Query zum Löschen ausgewählt, so können mit Hilfe des Verwaltungsbaumes Queries gefunden werden, die von dieser Query abhängen. Sie können potentiell auch aus dem Query Cache gelöscht werden. Dies ist aber nicht generell der Fall, sondern davon abhängig, ob diese Query aus dem speziellen oder generellen Layout stammt, oder ob es sich um eine Default-Query handelt.

- Query stammt aus speziellem Layout:  
Diese Query kann gelöscht werden, da sie nur in diesem Kontext von Bedeutung ist.

- Query stammt aus generellem Layout:  
Eine solche Query darf nicht gelöscht werden, da auf sie noch an anderer Stelle zugegriffen wird.
- Query ist eine Default-Query:  
Damit ist sie ebenfalls eine nur für diesen Kontext gültige Query und kann gelöscht werden.

Unter Anwendung der obigen Regel und der LRU-Strategie kann nun folgende Auslagerungsstrategie für den Query Cache angewendet werden: "Bestimme nach der LRU-Strategie eine Query und lösche sie. Bestimme zu dieser Query alle Verwendungspunkte im Verwaltungsbaum. Traversiere für alle diese Verwendungspunkte den Navigationsbaum unter Anwendung der obigen Regel. Lösche alle so gefundenen Queries. Markiere alle entsprechenden QueryNodes und ObjectNodes als ungültig und lösche sie, falls sie Blätter des Verwaltungsbaumes sind."

Jetzt wird deutlich, warum eine Auftrennung in Navigations- und Verwaltungsbaum sinnvoll ist. Wenn im Verwaltungsbaum gelöscht wird, so bedeutet dies nicht, daß auch der Navigationsbaum verändert wird. Das Löschen geschieht somit verdeckt für den Anwender und kann im Hintergrund ablaufen. Die Objekte bleiben sichtbar, obwohl sie eventuell nicht mehr im Cache sind. Dies ist genau das gewünschte Verhalten eines Cache.

### Implementierungsvarianten

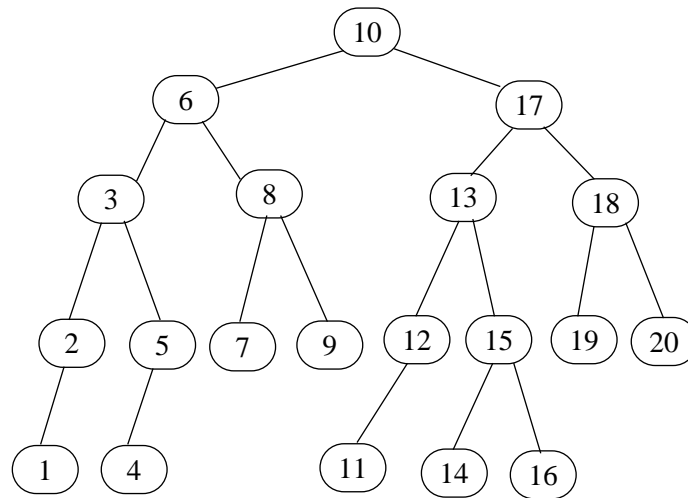
Eine mögliche Implementierungsvariante für den Query Cache ist eine Hashtabelle, da die Query IDs von der Benutzungsoberfläche selbst vergeben werden. Hier können Zahlen als IDs benutzt werden, die zur Indizierung der Hashtabelle dienen.

Für die Speicherung der Objekte ist ein Suchbaum zweckmäßig, um die Suchzeiten zu minimieren. Dazu muß ein Ordnungskriterium für die Objekt IDs existieren.

ABBILDUNG 17 Implementierungsvariante für Query Cache

ID	Type	Ass.Obj.	Object ID	Verwendungspunkte im Verwaltungsbaum
0	A	5	1,4,7,19	.....
1	B		1,5,6,8,10,20	.....
2	C	14	4,5,6	.....
3	D		13,14,17,20	.....
4	E		14,15,16,17,18	.....
5	F		2,3,7,11,20	.....
6	G	12	4	.....
7	H	3	1,2	.....
8	I	8	13,20	.....
9	J	19	18,20	.....
...	...	...	...	.....

Query Cache als Hashtabelle



Objekt Cache als binärer balancierter Suchbaum

### 3.2.4 Notifikations-Mechanismen

*Bezug zu Anforderungen: Die dargestellten Objekte sollen den aktuellen Objekten auf Seite des Servers entsprechen. Der Anwender soll von einer eingetretenen Änderung unterrichtet werden.*

Der Server muß nicht nur auf Objekt-, sondern auch auf Queryebene einen Notifikations-Mechanismus zur Verfügung stellen. Dazu verwaltet der Server eine Liste der von der Benutzungsoberfläche gestellten Queries. Daneben muß er auch eine Liste der geschickten Objekte führen. Es existieren auf Seite des Servers somit zwei Listen:

- Liste der verschickten Objekte, notwendig für Notifikation auf Objektebene
- Liste der eingegangenen Queries, notwendig für Notifikation auf Queryebene

Wird ein Objekt modifiziert, so kann der Server anhand dieser beiden Listen ermitteln, ob der Benutzungsoberfläche eine Notifikation geschickt werden muß.

Die Benutzungsoberfläche verfügt ihrerseits über den Objekt und den Query Cache. Der Objekt Cache stellt die Summe aller vom Server empfangenen Objekte dar, der Query Cache die Summe aller geschickten Queries, respektive einer Obergrenze aufgrund begrenzter Cachegröße.

Zunächst werden die sich entsprechenden Caches (auf Seite der Benutzungsoberfläche) und Listen (auf Seite des Servers) dieselben Objekte referenzieren. Sobald die Benutzungsoberfläche beginnt, Objekte oder Queries auszulagern, wird der Inhalt verschieden. Die Listen auf Serverseite sind im Normalfall größer als die Caches auf Seite der Benutzungsoberfläche. Deshalb kann und wird es vorkommen, daß der Server unnötige Notifikationen schickt. Es ergeben sich hier zwei mögliche Strategien:

**Strategie (1):**

Es ist gewünscht, daß entsprechender Cache bei Benutzungsoberfläche und Liste auf Serverseite unterschiedliche Inhalte haben können. Es wird damit in Kauf genommen, daß der Server potentiell Notifikationen von Objekten/Queries schickt, die nicht mehr im Cache der Benutzungsoberfläche gespeichert sind. Die Benutzungsoberfläche reagiert in solch einem Fall nicht auf die Notifikation. Durch das Schicken nicht berücksichtigter Notifikationen wird “unnötig” zwischen Benutzungsoberfläche und Server kommuniziert.

**Strategie (2):**

Es werden Mechanismen geschaffen, die dafür sorgen, daß die entsprechenden Caches und Listen stets denselben Inhalt haben. Wird auf einer Seite der Cache oder die Liste verändert, so wird die andere Seite über diese Änderung notifiziert (“Cache Notifikation”), der Cache oder die Liste kann entsprechend korrigiert werden. Es entsteht keine “unnötige” Kommunikation bei Objekt/Query Notifikationen, sehr wohl aber aufwendige Kommunikation für Cache Notifikationen.

Diese beiden Strategien sind gegensätzlich, die Vorteile der einen sind die Nachteile der anderen. Grundsätzlich handelt es sich um ein Synchronisationsproblem, das zudem auch noch verteilt ist. Strategie (2) ist dabei die auf jeden Fall die kompliziertere von beiden, es müßte zusätzlicher Code für Cache Notifikationen geschrieben werden und dazu noch eine ganze Reihe von Fehlerfällen abgefangen werden, um die Lösung trotz des unsicheren Faktors ‘Netzwerk’ robust zu gestalten. Das Kommunikationsprotokoll müßte um den Fall ‘Cache Notifikation’ erweitert werden.

Aufgrund dieser Komplexität ist Strategie (1) für die Benutzungsoberfläche vorzuziehen, zumal der Kommunikationsaufwand für Cache Notifikation dem Aufwand einer “normalen” Notifikation gleichgesetzt werden kann. Man kann durch Cache Notifikation keine Kommunikationseinheiten sparen, es bleibt letztlich gleich viel Aufwand. Bei Strategie (1) reagiert die Benutzungsoberfläche einfach nicht, wenn sie feststellt, daß das notifizierte Objekt nicht mehr im lokalen Cache ist.

Auf Seite des Server kann Strategie (1) nicht gewählt werden, da der Fluß der Notifikationen nur in Richtung der Benutzungsoberfläche ist. Wird aus der Liste auf Serverseite ein Objekt gelöscht, so muß die Benutzungsoberfläche benachrichtigt werden, da sie sonst für dieses Objekt keine Notifikationen mehr bekommt und somit auch nicht benachrichtigt werden würde, falls dieses Objekt modifiziert wird. Sie würde mit einem falschen Objekt weiterarbeiten und es nicht merken, solange dieses Objekt nicht aus dem lokalen Cache ausgelagert werden würde. Auf Seite des Servers muß also Strategie (2) implementiert werden. Für die Benutzungsoberfläche bedeutet dies, daß eine spezielle Notifikation für ein Objekt implementiert sein muß, die dieses Objekt nur aus dem Cache löscht und alle Queries, die dieses Objekt als Ergebnis haben, unverändert läßt.

Die Wahl der Strategie (1) bedeutet für die Benutzungsoberfläche also, daß der Server eventuell Notifikationen schickt, die sich auf nicht mehr im Cache gehaltene Objekte beziehen. Erhält die Benutzungsoberfläche eine Notifikation, so muß sie zunächst prüfen, ob und wenn ja welche Auswirkungen diese Notifikation auf den Objekt und den Query Cache hat.

Der Algorithmus, der den Einfluß der Notifikation behandelt, ist von der Art der Notifikation abhängig. Dabei werden drei Arten unterschieden:

- Ändern eines Objekts:  
Bei einem Objekt haben sich Werte der Attribute geändert, Attribute sind hinzugekommen oder gelöscht worden. (Spezialfall: Typ des Objekts hat sich geändert, siehe unten)
- Löschen eines Objekts:  
Das Objekt wurde von einem anderen Benutzer gelöscht.
- Hinzufügen eines Objekts:  
Ein anderer Benutzer hat ein Objekt hinzugefügt.

### **Ändern eines existierenden Objekts**

Wenn sich ein Objekt ändert, kann der Server in der Liste der verschickten Objekte nachschauen, ob das betreffende Objekt schon an die Benutzungsoberfläche geschickt worden ist. Ist dies der Fall, so notifiziert er die Benutzungsoberfläche über die Objektänderung.

Auf Seite der Benutzungsoberfläche ist eine Änderungsnotifikation nur für den Objekt Cache relevant. Die Benutzungsoberfläche fordert das entsprechende Objekt neu vom Server an, falls es noch in seinem lokalen Cache gespeichert ist.

Die lokal gespeicherten Queries, die sich auf dieses Objekt über seinen Objekttyp beziehen, bleiben weiterhin gültig, solange sich die Änderung nicht auf den Typ des Objekts bezieht. Ist dies der Fall, so zieht dies noch Modifikationen im Query Cache nach sich. Dieser Spezialfall kann behandelt werden, wie wenn das alte Objekt gelöscht und das neue hinzugefügt wird.

### **Löschen eines Objekts**

Falls ein Objekt gelöscht wird, so kann der Server ebenfalls über die Liste der verschickten Objekte feststellen, ob die Benutzungsoberfläche bereits auf dieses Objekt zugegriffen hat. Ist dies der Fall, so schickt der Server eine entsprechende Notifikation an die Benutzungsoberfläche. Diese kann anschließend das Objekt im lokalen Cache löschen. Bei Queries, die dieses Objekt als Ergebnis haben, wird es aus dem Ergebnis gelöscht. Die Queries selbst bleiben aber weiterhin gültig.

### **Hinzufügen eines Objekts**

Hier greift der Server auf die Liste der beantworteten Queries zu und vergleicht den Typ des neuen Objekts mit denjenigen Objekttypen, auf die sich die Queries beziehen. Für alle Queries, bei denen der Objekttyp gleich dem Typ des neuen Objekts ist, wird eine Querynotifikation an die Benutzungsoberfläche geschickt. Nun gibt es zwei Möglichkeiten, den Query Cache zu aktualisieren:

Methode (1):

Alle entsprechenden Queries werden als ungültig markiert, sie müssen gegebenenfalls nochmals ausgeführt werden.

Methode (2):

Das neu hinzugefügte Objekt wird dem Resultat aller entsprechenden Queries hinzugefügt.

Betrachtet man die drei Arten von Queries, so stellt sich heraus, daß zwischen den beiden Möglichkeiten keine 'Entweder-Oder'-Auswahl getroffen werden kann, sondern daß anhand der Query-Arten unterschieden werden muß.

ABBILDUNG 19

Aktualisieren des Query Cache bei Hinzufügen eines Objekts

Art der Query	angewendete Methode
Query bezieht sich auf ein Objekt	eine solche Query kann nicht existieren, da Objekt IDs eindeutig sind; das neu hinzugekommene Objekt hat immer eine noch nicht existente ID
Query bezieht sich nur auf Objekttyp, keine Beziehung zu anderem Objekt	neues Objekt kann dem Resultat hinzugefügt werden --> Methode (2)
Query bezieht sich auf einen Objekttyp, Beziehung zu einem anderen Objekt wird berücksichtigt	es ist zunächst nicht entscheidbar, ob dieses Objekt eine Beziehung zu dem in der Query spezifizierten Objekt hat, dazu müßte die Query selbst ausgeführt werden; also kann man sie gleich als ungültig markieren --> Methode (1)

Nachdem die Notifikation verarbeitet worden ist, muß noch die Darstellung auf dem Bildschirm aktualisiert werden. Dazu muß die entsprechende Stelle im Navigationsbaum gefunden werden. Sie kann sie über die Verbindung 'Objekt/Query Cache -> Verwaltungsbaum -> Navigationsbaum' bestimmt werden.

Verbindungen zwischen Verwaltungsbaum und Navigationsbaum werden bei jeder Expansion des Navigationsbaumes angelegt (siehe "Algorithmus zur Erzeugung einer Hierarchieebene" auf Seite 44).

Verbindungen zwischen Query Cache und Verwaltungsbaum werden bei Erzeugung des Verwaltungsbaumes im Query Cache gespeichert (siehe "Implementierungsvariante für Query Cache" auf Seite 47).

Es ist aber bisher noch nicht beschrieben worden, wie Verbindungen zwischen Objekt Cache und Navigationsbaum zustande kommen. Dies wird im folgenden Kapitelabschnitt beschrieben (siehe "Erkennen des Darstellungs-Aliasing" auf Seite 52).

Mit der vorgestellten Funktionalität ist die Benutzungsoberfläche in der Lage, Notifikationen entgegenzunehmen und weiterzuverarbeiten. Die Darstellung auf dem Bildschirm wird aktualisiert. Der Anwender wird über eine eingetretene Modifikation unterrichtet, indem in der Statuszeile ein entsprechender Text erscheint, gefolgt von zwei Buttons. Er kann dies nur zur Kenntnis nehmen, indem er den Button 'OK' drückt, oder er kann zur der Stelle verzweigen, an der die Änderung eingetreten ist. Dazu klickt er den Button 'Go To' an. Die Benutzungsoberfläche springt anschließend an die entsprechende Stelle im Navigationsbaum. Falls mehrere Änderungen auf einmal eingetreten sind, so werden alle aufgelistet, jeweils mit den Buttons 'OK' und 'Go To'. Zusätzlich kann er alle Änderungen auf einmal bestätigen.

Für die Notifikationen werden neue Protokollelemente für die Kommunikation zwischen Benutzungsoberfläche und Server benötigt (siehe "Zusätzliche Protokollelemente" auf Seite 57).

### 3.2.5 Erkennen des Darstellungs-Aliasing

*Bezug zu Anforderungen: Zur Implementierung des Notifikations-Mechanismus wird das Finden von visuellen Kopien gebraucht, um die Datenaktualität sicherzustellen. Dazu wird eine Verbindung zwischen Objekt Cache und Verwaltungsbaum geschaffen, die das Auffinden der entsprechenden Stelle in Navigationsbaum möglich macht. Außerdem kann eine komfortablere Navigation ermöglicht werden.*

#### **Algorithmus (1): Traversierung des Navigationsbaumes**

Zur Erkennung des Darstellungs-Aliasing wird als erster Algorithmus die Traversierung des Navigationsbaumes untersucht.

Traversieralgorithmus:

Gegeben ist der Navigationsbaum, eine leere Menge H zu Vergleichszwecken und eine leere Menge A, die nach Beendigung des Algorithmus Objekte enthält, für die ein Alias existiert.

```
void searchAliasedObjects(NavTreeType NavTree) {
    if (Navtree.Content in H) { //Content=Applikationsobjekt
        //ja, das Objekt war schon mal da, damit ist es aliased
        if (NavTree.Content in A) {
            //ja, aliased object Liste existiert schon in H, also diesen
            //NavTree der Liste hinzufügen
            A.NavTree.Content.add(NavTree)
        }
        else {
            //nein, aliased object Liste existiert noch nicht, deshalb
            //mit diesem Knoten erzeugen
            A.createAliasedList(NavTree.Content, NavTree)
        }
    }
    else {
        //nein, das Objekt war noch nie da, also zur Hilfsmenge hinzu-
        //fügen
        H.add(NavTree.Content)
    }
    //Funktion rekursiv für Unterbäume aufrufen
    Menge U = getAllSubTrees()
    NavTreeType HilfTree
    for all HilfTree in U do {
        searchAliasedObjects(HilfTree) //Rekursion
    }
}
```

Nach Beendigung des Algorithmus enthält Menge A alle Objekte, für die es einen Alias gibt, zusammen mit den Stellen, an denen sie im Navigationsbaum auftreten.

Der Algorithmus ist laufzeitintensiv, falls der Navigationsbaum groß ist. Er kann verbessert werden, indem nur die wirklich geöffneten Zweige des Navigationsbaum traversiert werden.

```

...
NavTreeType HilfTree
for all HilfTree in U do {
    if (HilfTree.isOpen()) searchAliasedObjects(HilfTree)
}
...

```

Da alle Objekte aus dem Objekt Cache geholt werden, kann ein wesentlich schnellerer Algorithmus benutzt werden.

### Algorithmus (2): Referenzlisten erzeugen

Um ein Suchen nach den Aliases und damit eine Traversierung des Navigationsbaumes zu vermeiden, werden Verwaltungsinformationen beim Holen der Objekte aus dem Objekt Cache generiert, mit deren Hilfe Aliasing schneller erkannt werden kann.

Der Objekt Cache verwaltet für jedes gespeicherte Objekt eine Liste der Stellen, an denen es im Navigationsbaum dargestellt wird (Referenzliste). Dazu stellt er eine Methode zur Verfügung, die einen Knoten des Navigationsbaum in eine solche Liste eines Objekts einfügt. Der Algorithmus, der die nächste Hierarchieebene im Navigationsbaum erzeugt (siehe Seite 45), ruft diese Funktion auf, nachdem er das Objekt aus dem Cache geholt hat und das sichtbare Element erzeugt hat:

```

...
if (Hilf1.Type == real) {
    hole Objekt aus Objekt Cache
    erzeuge entsprechendes sichtbares Element
    trage dieses sichtbare Element in die aliased Liste im
    Objekt Cache ein
}
...

```

Jetzt kann das Darstellungs-Aliasing durch lineares Durchsuchen des Objekt Cache aufgelöst werden. Für jedes Objekt im Objekt Cache muß geprüft werden:

“Anzahl der Objekte in Referenzliste > 1? Wenn ja, dann Darstellungs-Aliasing”

Dieser Algorithmus kann für jedes Objekt angewendet werden. Ist es mehrfach dargestellt, so kann zwischen den einzelnen Vorkommen hin- und hergesprungen werden.

## 3.2.6 Drag&Drop Aktionen

*Bezug zu Anforderungen: Drag&Drop Aktionen sollen die intuitive Bedienung der Benutzungsoberfläche verbessern (Standard-Funktionalität grafischer Benutzungsoberflächen).*

Neben den durch Kontextmenüs auswählbaren Aktionen muß es auch die Möglichkeit geben, per Drag&Drop eine Aktion anzustoßen. Die Benutzungsoberfläche verfügt dafür über eine Tabelle, die Drag-Quellen Drop-Zielen zuordnet. Diese Tabelle wird vom Server zur Verfügung gestellt und beim Start der Benutzungsoberfläche geladen.

Der Ablauf einer Drag&Drop Aktion unterscheidet sich im Grunde nicht von einer Aktion, die über das Kontextmenü angestoßen wird. Bei Anforderung des Dialogs wird

ein Objekt mehr (das Zielobjekt) geschickt, und damit ist das Zielobjekt der Aktion festgelegt. Es muß nicht mehr durch Interaktion mit dem Benutzer eingelesen werden.

Zur Modellierung als Drag&Drop Aktionen eignen sich besonders diejenigen Aktionen, die zwei (oder mehr) Objekt miteinander in Verbindung setzen. Bei der User/Security Applikation ist dies beispielsweise das Zuweisen einer Rolle zu einem Employee. Hierfür wird ein Rollentemplate auf einen Employee gezogen.

Bei der Ausführung von Drag&Drop Aktionen ist es ebenfalls möglich, daß noch Parameter zur Ausführung der Aktion fehlen. Der Dialog, der die fehlenden Parameter einließt, wird mit dem Zielobjekt vorbesetzt. Diese Vorbesetzung kann nicht mehr durch den Benutzer modifiziert werden. Analog zu den anderen Aktionen schickt der Dialog nach Eingabe aller Werte die Aktion an den Server.

Vom Ablauf her entspricht eine Drag&Drop Aktion somit einer normalen Aktion (siehe Seite 33).

### 3.2.7 Rahmenfunktion zur Objektmarkierung

*Bezug zu Anforderungen: Der Anwender soll eine Auswahl von Objekten komfortabel erstellen können.*

Der Benutzer hat die Möglichkeit, eine Auswahl von Objekten mittels eines Rahmens zu definieren. Ein Rahmen ist ein rechteckiger Bildschirmbereich, der mit dem Zeigergerät (im Normalfall der Maus) gezeichnet wird. Alle in diesem Rahmen enthaltenen Objekte werden selektiert. Rahmen können nur Ziel von Drag&Drop Aktionen sein.

Es ist mehr als ein Rahmen erlaubt. Hat man bereits einen Rahmen gezeichnet, so zerstört der als nächstes gezeichnete Rahmen alle vorher definierten. Dies ist der eingestellte Default, der durch zusätzliches Drücken und Halten einer Taste ausgeschaltet werden kann. Dadurch können mehrere Rahmen gezeichnet werden. Dies erhöht die Benutzerfreundlichkeit.

### 3.2.8 Verschlüsselung

*Bezug zu Anforderungen: Dies ist ein notwendiger Sicherheitsaspekt der Kommunikation zwischen Benutzungsoberfläche und Server.*

Um eine verschlüsselte Kommunikation mit dem Applikationsserver zu realisieren, gibt es grundsätzlich zwei Strategien:

- Verschlüsselung durch die Benutzungsoberfläche
- Benutzung eines Netzwerkprotokolls mit Verschlüsselung

Bei der ersten Möglichkeit können die verschlüsselten Daten mit einem Standardprotokoll übertragen werden. Dies hat den Vorteil, daß jedes beliebige netzweite Protokoll benutzt werden kann und trotzdem eine sichere Kommunikation von jedem Punkt im Netzwerk möglich ist.

Aus einem anderen Blickwinkel ist der Vorteil auch gleichzeitig Nachteil. Die Verschlüsselung muß explizit programmiert werden, was einen zusätzlichen Aufwand bedeutet. Schneller und billiger wäre es, ein Netzwerkprotokoll als Grundlage der Kommunikation zu benutzen, daß bereits eine verschlüsselte Kommunikation anbietet.

Dies hat zur Folge, daß eine sichere netzweite Kommunikation der Benutzungsoberfläche mit dem Server nicht gewährleistet werden kann, da dieses Protokoll eventuell nicht überall verfügbar ist. Außerdem hat die Benutzungsoberfläche keine Kontrolle mehr über die Verschlüsselung.

Da moderne Betriebssysteme in der Regel Bibliotheksaufrufe zur Verschlüsselung anbieten, liegt der zusätzliche Aufwand in einem vertretbaren Rahmen. Die Benutzungsoberfläche wird die Verschlüsselung selbst vornehmen. UNIX-Betriebssysteme bieten beispielsweise Verschlüsselung nach dem Data Encryption Standard (DES) an.

Für einen speziellen Fall muß die Benutzungsoberfläche in jedem Fall eine Verschlüsselung vornehmen, und zwar wenn ein Paßwort gespeichert werden soll. Auch wenn das Kommunikationsprotokoll zwischen Benutzungsoberfläche und Server auf Ebene eines Netzwerkprotokolls verschlüsselt werden würde, sollte ein Paßwort niemals in lesbarer Form in der Datenbank gespeichert werden. Daher muß es in jedem Fall verschlüsselt werden. Somit muß diese Funktionalität sowieso zur Verfügung stehen und kann daher auch für das Kommunikationsprotokoll verwendet werden.

### 3.2.9 Navigations- und Informationsmodus

*Bezug zu Anforderungen: Verbesserung des Antwortverhaltens der Benutzungsoberfläche.*

Da der Zugriff auf Objekte und ihre Attribute über ein Netzwerk läuft und die Kommunikation zwischen Server und Benutzungsoberfläche von der Netzlast abhängt, ist es ratsam, nur die Daten zu übertragen, die man visualisieren möchte. Der Benutzer kann dazu zwischen zwei verschiedenen Modi wählen.

Im Navigationsmodus greift er nur auf administrative Komponenten der Objekte zu. Dies sind der Name und die Objekt ID. Er bewegt sich nur im Navigationsbereich der Benutzungsoberfläche, der Informationsbereich ist verdeckt. Dem Benutzer präsentiert sich eine navigationsorientierte Sicht der Objekte. Dies ist sinnvoll, wenn der Benutzer nur an bestimmten Objekten Interesse hat. Er möchte keine Präsentationsform im Informationsbereich sehen und somit auch nicht darauf warten, bis die Benutzungsoberfläche sie aufgebaut hat. Sobald er auf ein Objekt stößt, das er präsentiert haben möchte, kann er in den Informationsmodus wechseln, bei dem eine bestimmte Präsentationsform im Informationsbereich angezeigt wird.

Der Wechsel zwischen diesen beiden Modi geschieht nicht durch Wahl eines bestimmten Modus, sondern ist in der Benutzerführung verankert. Zunächst ist im Navigationsbereich kein Objekt selektiert, der Informationsbereich ist verdeckt oder verkleinert, da er noch nicht benötigt wird. Der Benutzer kann mit Hilfe der vor den Objekten dargestellten Navigationssymbole die Unterstruktur eines Objekts öffnen und wieder schließen. Dazu muß kein Objekt selektiert werden. Sobald der Benutzer ein Objekt selektiert, wechselt die Benutzungsoberfläche in den Informationsmodus und stellt die ausgewählte Präsentationsform im Informationsbereich dar. Dieser Vorgang wird allerdings sofort unterbrochen, falls der Benutzer ein anderes Objekt selektiert. Es werden dann die Attribute des neu selektierten Objekts geladen. Greift der Benutzer allerdings wieder auf ein Navigationssymbol zu, so wechselt die Benutzungsoberfläche wieder in den Navigationsmodus. Das zuletzt selektierte Objekt bleibt selektiert. Dementsprechend bleibt die Präsentationsform unverändert.

### 3.2.10 Filtermechanismen

*Bezug zu Anforderungen: Um dem Anwender ein selektives Betrachten einer gewissen Auswahl an Objekte zu ermöglichen und uninteressante Objekte zu verbergen, können Filter sowohl für den Navigations- als auch für den Informationsbereich definiert werden.*

Die Benutzungsoberfläche ist nur für die Eingabe, nicht aber die Berechnung eines Filters verantwortlich. Die Berechnung erfolgt auf Seite des Servers im Kontext der Berechnung einer Query. Das bedeutet für das Kommunikationsprotokoll zwischen Benutzungsoberfläche und Server, daß Queries um die Spezifikation von Filtern erweitert werden müssen. Da Filter das Ergebnis von Queries beeinflussen, müssen zwei Queries, die sich nur aufgrund des Filters unterscheiden, insgesamt als unterschiedlich betrachtet werden. Für den Query Cache bedeutet dies, daß jede dieser beiden Queries zwischengespeichert wird.

Filter können über Kontextmenüs spezifiziert werden. Für den Navigationsbereich bedeutet dies, daß neben den für ein bestimmtes Objekt ausführbaren Aktionen ein Menüpunkt namens 'Definiere Filter' existiert. Dieser Filter bezieht sich auf die diesem Objekt untergeordneten Objekte und ist nicht für Objekte verfügbar, die durch eine Layout-Definition als Blätter des Navigationsbaumes definiert sind.

Im Informationsbereich kann durch Klicken eines Auswahlalters einer Präsentationsform mit der rechten Maustaste ein Kontextmenü aufgerufen werden, in dem ebenfalls der Befehl 'Definiere Filter' zur Verfügung steht. Dieser Filter bezieht sich auf die jeweilige Präsentationsform. Für jede von ihnen kann ein Filter definiert werden.

Wird 'Definiere Filter' aufgerufen, so erscheint ein Fenster, in dem der Filter definiert werden kann. Er kann sich auf Namen, Typen, Attribute oder Aktionen der Objekte beziehen. Ebenfalls möglich ist die Spezifikation bestimmter Bereiche, zum Beispiel die ersten zehn Objekte. Syntaktisch betrachtet ist ein Filter ein regulärer Ausdruck.

### 3.2.11 Hilfesystem

*Bezug zu Anforderungen: Keine Benutzungsoberfläche ist völlig intuitiv, der Anwender wird zu ihrer Bedienung Beschreibungen und erklärende Texte benötigen.*

Das Hilfesystem besteht aus einer ausführlichen Beschreibung der Funktionalität auf Web-Seiten, also im HTML-Format, und Online-Hilfen, die in zusätzlichen Fenstern angezeigt werden.

Die Startseite der Web-Seiten-Dokumentation ist eine Inhaltsübersicht, von der aus alle Kapitel angesprungen werden können. Ebenso steht ein Index, ein Glossar und eine Suchfunktion zur Verfügung. Der Inhalt ist unterteilt in einen Teil für die Benutzungsoberfläche selbst und in einen Server-Teil.

Die Online-Hilfen liefern zu einem selektierten Bildschirmobjekt einen Hilfetext, der die Bedeutung dieses Objekts kurz erklärt. Jeder Hilfetext verfügt über einen Hyperlink auf das entsprechende Kapitel in der Webseiten-Beschreibung. Die Online-Hilfe kann über das Kontextmenü aufgerufen werden.

### 3.2.12 Scheduling

*Bezug zu Anforderungen: Das Auslösen eines Service Requests kann für einen Zeitpunkt in der Zukunft geplant werden. Die Benutzungsoberfläche muß hierfür einen Mechanismus zur Definition dieses Zeitpunkts bereitstellen.*

Nachdem die für einen Service Request benötigten Parameter eingegeben worden sind (siehe "Erstellung von Service Requests" auf Seite 27), hat der Anwender die Möglichkeit, diesen Service Request erst zu einem bestimmten Zeitpunkt in der Zukunft ausführen zu lassen. Anstelle des unmittelbaren Auslösens durch Druck auf den 'OK'-Button wählt der Benutzer den Button 'Schedule'. Ein neues Fenster mit einem Kalender wird dargestellt. Hier kann ein Zeitpunkt in der Zukunft definiert werden, zu dem dieser Service Request ausgeführt werden soll. Erst wenn der Benutzer den 'OK'-Button des Kalenderfensters drückt, wird der Service Request inklusive der Scheduling informationen an den Server übertragen.

Das Kommunikationsprotokoll muß um diesen Fall erweitert werden. Der Zeitpunkt wird in einem bestimmten Format geschickt, zum Beispiel im 'Universal Time Code'-Format (UTC).

### 3.2.13 Zusätzliche Protokollelemente

Durch die Verbesserung der Minimalfunktionalität werden noch zusätzliche Protokollelemente benötigt. Im einzelnen wird das Kommunikationsprotokoll um die nachfolgenden Fälle erweitert.

- zwei zusätzliche Queries
- Filter für Queries
- drei Objekt-Notifikationen
- zwei Cache-Notifikationen
- Scheduling-Informationen für Service Requests

ABBILDUNG 20 Zusätzliche Protokollelemente "Benutzungsoberfläche an Applikationsserver"

Protokollelement	Bedeutung	Kommunikationsart
Q <query id> <object type>	alle Objekte vom Typ <object type> anfordern	synchron
Q <query id> <object type> "ass" <association type> <object id>	alle Objekte vom Typ <object type> anfordern, die Beziehung <association type> zum Objekt <object id> haben	synchron
Q aQuery "reg" <regular expression>	eine der drei Queries mit zusätzlichem Filter <regular expression>	synchron
A anAction "schedule" <date&time>	Ausführung dieses Service Requests erst zum Zeitpunkt <date&time>	asynchron

ABBILDUNG 21 Zusätzliche Protokollelemente "Applikationsserver an Benutzungsoberfläche"

Protokollelement	Bedeutung
N <notification type> <object/query id list>	diese Liste von Objekten/Queries werden notifiziert, wobei für <notification type> fünf Werte möglich sind (für fünf Notifikations-Typen)

### 3.2.14 Plattformunabhängige Verfügbarkeit

*Bezug zu Anforderungen: Dies ist eine Kernanforderung an die Benutzungsoberfläche, damit sie nicht an ein bestimmtes Betriebssystem gebunden ist.*

Dieses Designkriterium ist der entscheidende Punkt für die Auswahl der Implementierungssprache.

Die Benutzungsoberfläche wird als World Wide Web Interface gestaltet und ist damit von jeder Plattform aus über einen Browser zugänglich. Die Auswahl an Implementierungssprachen, mit der diese Anforderung erfüllt werden kann, ist nicht sehr groß, es kommen vier Möglichkeiten in Frage. Sie schließen sich nicht gegenseitig aus und sind eventuell kombinierbar.

Java hat sich als die Sprache zur Programmierung des World Wide Web am Markt durchgesetzt. Zahlreiche Klassenbibliotheken und Entwicklungswerkzeuge stehen zur Verfügung. Ein in Java geschriebenes Programm wird ausgeführt, indem es durch die Java Virtual Machine interpretiert wird. Die Java Virtual Machine ist auf allen wichtigen Betriebssystemen verfügbar, insbesondere ist sie in die wichtigsten World Wide Web Browser integriert. Dies gewährleistet eine Ausführbarkeit auf fast allen Systemen. Es gibt allerdings einige Unterschiede im Umfang der benutzbaren Funktionalität, falls das Java Programm von einer sogenannten 'Standalone Virtual Machine' oder von der eines Browsers ausgeführt wird. Den ersten Fall der Ausführung nennt man 'Applikation', den zweiten 'Applet'. Wird ein Java-Programm als Applikation ausgeführt, so steht ihm uneingeschränkte Funktionalität bezüglich des Systems zur Verfügung, auf dem es ausgeführt wird. Wenn es hingegen als Applet von einem Browser ausgeführt wird, so unterliegt es Sicherheitsbeschränkungen. Es ist in diesem Fall beispielsweise nicht möglich, ein anderes Programm zu starten oder in irgendeiner Weise auf das lokale Dateisystem zuzugreifen.

Die zweite Möglichkeit sind die von Microsoft entwickelten ActiveX Controls. Sie sind nicht auf allen Plattformen verfügbar, da sie nur in Verbindung mit dem Internet Explorer der Firma Microsoft einsetzbar sind. Dieser wiederum ist nur auf Microsoft-Betriebssystemen erhältlich. ActiveX Controls sind grafische Elemente, mit ihnen läßt sich eine Benutzungsoberfläche aufbauen.

In HTML-Seite können CGI-Skripte eingebunden werden. CGI-Skripte sind auf Seite des Servers ausführbare Programme, meist in einer mächtigen Skript-Sprache wie beispielsweise Perl geschrieben, deren Ergebnis auf der Internet-Seite präsentiert wird. Die Darstellungsmöglichkeiten beschränken sich dadurch auf die in HTML verfügbaren Elemente. Theoretisch könnten so auch grafische Elemente über Bilder simuliert werden.

Im Gegensatz zu CGI-Skripten wird JavaScript auf Seite des Clients ausgeführt und ist direkt als Source in der HTML-Seite enthalten. In JavaScript sind bereits viele grafische Elemente enthalten, die zum Aufbau einer grafischen Benutzungsoberfläche durchaus ausreichen würden. Andere Bereiche wie zum Beispiel Netzwerkkommunikation oder Threads sind nicht im Sprachumfang von JavaScript enthalten.

### 3.2.15 Auswahl der Implementierungssprache

*Bezug zu Anforderungen: Notwendige Auswahl, um die Benutzungsoberfläche letztlich implementieren zu können.*

Da das Kriterium "Plattformunabhängige Verfügbarkeit" ein Muß ist, scheiden ActiveX Controls aus. Microsoft sieht eine Portierung ihres Internet Explorer auf andere Plattformen nicht vor, und eine Beschränkung nur auf Microsoft-Betriebssysteme wäre keine wirkliche Plattformunabhängigkeit.

CGI-Skripte bieten nur die grafischen Möglichkeiten, die HTML bietet. Damit ist der Darstellungsbereich auf jeden Fall auf den Browser fixiert. Es können keine neuen Fenster geschaffen werden. Außerdem bietet HTML nicht ausreichend viele grafische Elemente, um eine Benutzungsoberfläche komfortabel aufbauen zu können. Es stehen nur Text und Grafik zur Verfügung. Alle grafische Elemente müßten damit dargestellt werden. Dies wäre unnötig aufwendig und kompliziert zugleich. Diese Implementierungsalternative wird daher ebenfalls verworfen.

JavaScript bietet zwar ein Vielzahl an grafischen Elementen, es fehlen aber Programmiermöglichkeiten in wesentlichen Bereichen wie Netzwerkkommunikation oder Thread-Behandlung. Diese sind aber für die Erstellung der Benutzungsoberfläche grundsätzliche Forderungen, die erfüllt sein müssen. JavaScript scheidet somit aus.

Für Java stehen umfangreiche Bibliotheken und Werkzeuge zur Verfügung. Java ist eine Programmiersprache mit vielen hilfreichen Konzepten wie zum Beispiel Threads, Event Handling und Netzwerkfunktionalitäten. Nicht umsonst hat sie sich als De-Facto-Standard zur Programmierung des Internets etabliert. Java wird deshalb als Implementierungssprache gewählt.

Obwohl in dieser Arbeit bereits zu Anfang über Java geschrieben worden ist und für einige Implementierungsbeispiele Java-Code vorgestellt worden ist, war es noch nicht klar, daß Java die Sprache der Wahl sein wird. Die oben vorgestellten Vorteile, der reiche Sprachumfang, die Eigenschaft der Objektorientierung und die dynamischen Spracheigenschaften von Java (siehe "Machbarkeitsstudie" auf Seite 31) machen die Wahl jedoch einfach.

# IMPLEMENTIERUNG

Dieses Kapitel beschreibt die konkrete Implementierung des objektorientierten Systems in der ausgewählten Implementierungssprache.

Die im vorigen Kapitel erarbeiteten konzeptionellen Lösungen und Algorithmen werden auf verschiedene Programmobjekte verteilt. Diese Objekte werden detailliert bezüglich ihrer Attribute, Methoden und Schnittstellen erläutert.

Zur tatsächlichen Implementierung des Prototyps muß eine Entwicklungsumgebung ausgewählt werden. Hierzu wird ein Kriterienkatalog aufgestellt und aus den am Markt erhältlichen Produkten eine Entwicklungsumgebung anhand dieser Kriterien zusammengestellt.

Anschließend werden für jede implementierte Klasse alle Attribute, Konstruktoren, Klassen- und Instanzenmethoden beschrieben.

## 4.1 Objekte der Benutzungsoberfläche

### 4.1.1 Zusammenfassung der Aufgaben

Die Hauptaufgaben der Benutzungsoberfläche sind:

- Verbindung zum Applikationsserver auf- und abbauen, Anforderung von Objekten zur Darstellung und Verschicken von Service Requests
- Verwaltung des lokalen Cache
- Navigation durch Objekthierarchien ermöglichen (Navigationsbereich), Attribute und untergeordnete Objekte visualisieren (Informationsbereich)
- Erstellung von Service Requests
- Hilfesystem

### 4.1.2 Aufgabenverteilung

Der **Cache Manager** übernimmt die Verwaltung der lokalen Caches. Er speichert darin die vom Server gelieferten Objekte und die dazugehörigen Queries. Wird ein Objekt, das nicht im Cache gespeichert ist, von einer anderen Komponente der Benutzungsoberfläche angefordert, so holt es der Cache Manager über den Kommunikations-Manager vom Applikationsserver (siehe unten). Sobald ein Cache voll ist, entscheidet er anhand der entsprechenden Auslagerungsstrategie, welches Objekt oder welche Query als nächstes aus dem Cache gelöscht werden soll, um Platz zu schaffen.

Der **Display Manager** ist für die Darstellung auf dem Bildschirm zuständig. Er visualisiert die einzelnen Bäume mit ihren Views, Applikationsobjekten und Kontextmenüs und behandelt alle Benutzereingaben. Er ermöglicht dem Benutzer die Navigation durch die Objekthierarchie und stellt die Präsentationsformen im Informationsbereich

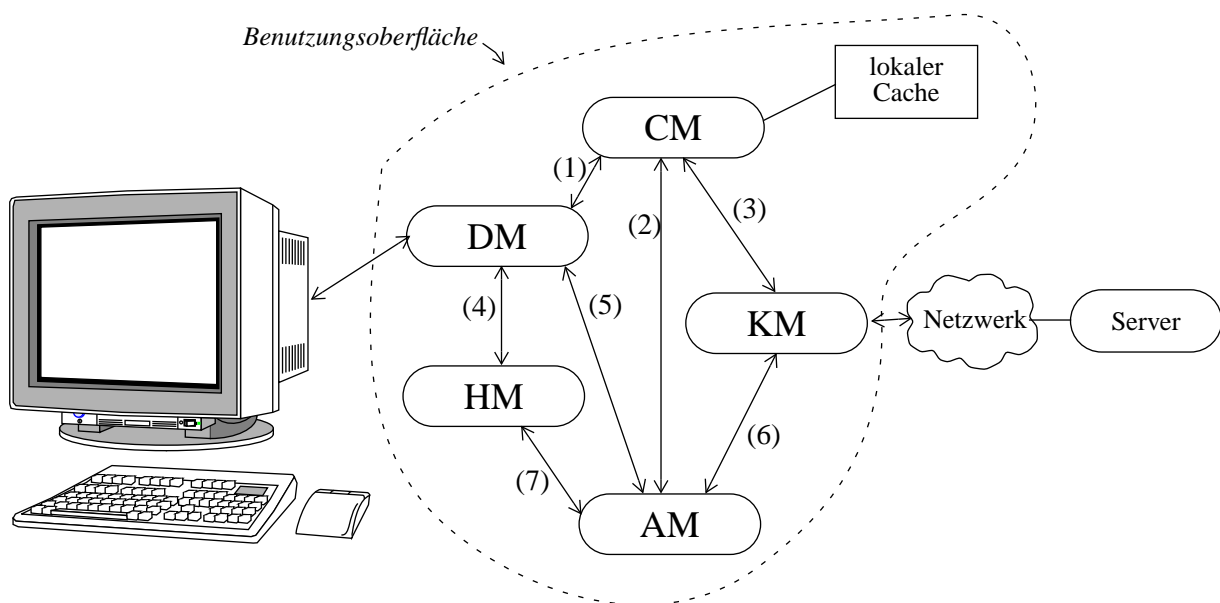
dar. Er verfügt über einen speziellen Zeigemodus, den Help-Modus, in dem ein Bildelement selektiert werden kann, zu dem ein Hilfetext angezeigt werden soll.

Der **Kommunikations-Manager** stellt die Konnektivität zum Applikationsserver her. Er ist für den gesamten Datenaustausch mit ihr verantwortlich. Beim Start wird eine Sitzung eröffnet, der Server weist dafür eine ID zu. Diese ID muß die Benutzungsoberfläche für jedes Datenpaket zur Authentifikation verwendet. Der Kommunikations-Manager kann sowohl synchron als auch asynchron mit dem Applikationsserver kommunizieren.

Der **Action Manager** ist für die Ausführung von Aktionen zuständig. Stößt der Benutzer eine Aktion an, so bekommt der Action Manager eine Liste der fokussierten Objekte und die darauf anzuwendende Aktion. Diese Informationen schickt er an den Server, der ihm darauf den entsprechenden Aktionsdialog, falls notwendig, schickt. Der Action Manager führt diesen Dialog aus. Bei Beendigung schickt der Dialog die Aktion inklusive aller notwendigen Parameter selbständig an den Server.

Der **Hilfe Manager** ist für die Online-Hilfe zuständig. Er verwaltet die Dokumente des Hilfesystems und liefert zu einem Bildelement den passenden Hilfetext.

ABBILDUNG 22 Grobstruktur der Benutzungsoberfläche



### 4.1.3 Schnittstellen

#### **Schnittstelle Display Manager - Cache Manager (1)**

Der Display Manager fordert alle darzustellenden Objekte mit ihren Attributen vom Cache Manager über die Funktion `getObject` an.

#### **Schnittstelle Action Manager - Cache Manager (2)**

Damit der Action Manager Aktionsdialoge vor ihrer Ausführung mit Objekten parametrisieren kann, benötigt er die entsprechenden Objektdaten, die er vom Cache Manager über `getObject` anfordert.

#### **Schnittstelle Cache Manager - Kommunikations-Manager (3)**

Der Cache Manager fordert über die Methode `sendSync` des Kommunikations-Managers Objekte synchron vom Applikationsserver ab.

#### **Schnittstelle Display Manager - Hilfe Manager (4)**

Der Display Manager ruft die Methode `executeHelp` des Hilfe Managers auf, sobald die Online-Hilfe über den speziellen Zeigemodus aktiviert wird.

#### **Schnittstelle Display Manager - Action Manager (5)**

Wird eine Aktion angestoßen, so ruft der Display Manager eine entsprechende Methode des Action Managers auf. Für Drag&Drop Aktionen ist dies die Methode `executeDragDrop`, für Aktionen aus dem Kontextmenü `executeAction`. Der Action Manager holt sich daraufhin die Liste der selektierten Objekte (Methode `getSelectionList` des Display Managers). Der Action Manager kann anschließend die Aktion ausführen.

#### **Schnittstelle Action Manager - Kommunikations-Manager (6)**

Zur Ausführung von Aktionen holt der Action Manager zunächst den entsprechenden Aktionsdialog vom Server, indem er ihn über die Methode `sendSync` des Kommunikations-Managers anfordert. Er führt diesen Dialog aus, der fehlende Werte einließt und über die Methode `sendAsync` des Kommunikations-Managers den Service Request asynchron an den Server schickt.

#### **Schnittstelle Action Manager - Hilfe Manager (7)**

Falls der Benutzer die Hilfefunktion über das Kontextmenü aktiviert, so ruft der Action Manager die Methode `executeHelp` des Hilfe Managers auf.

---

## 4.2 Entwicklungsumgebung

---

### 4.2.1 Anforderungen

Als plattformunabhängige Sprache wurde, wie bereits in Abschnitt “Auswahl der Implementierungssprache” auf Seite 59 erläutert, Java gewählt. Die für Java am Markt verfügbaren Entwicklungsumgebungen und Werkzeuge sind vielfältig.

Parallel zur Entwicklung der Benutzeroberfläche für die Service Request Applikation läuft noch die Entwicklung einer verwandten Benutzeroberflächen, in die die zu entwickelnde Benutzeroberfläche integrierbar sein soll. Daher muß bei der Auswahl der Entwicklungsumgebung, der Klassenbibliotheken und der Werkzeuge möglichst viel mit dem parallel laufenden Projekt übereinstimmen, dies wird eine Integration erleichtern.

### 4.2.2 Kriterienkatalog

Projektabhängigkeit:

- Entwicklungsumgebung des Parallelprojekts muß beachtet werden

Source Code Entwicklungsumgebung:

- Komfortabler Editor für verschiedene Arten von Code
- schneller, optimierender Compiler
- Debugger
- Appletviewer
- Projektunterstützung
- Klassenbrowser
- Dialogeditor
- nicht nur Compiler für Java-Code, sondern auch HTML-Generierung für Applets
- Integration beliebiger Klassenbibliotheken
- gutes Zusammenspiel der Entwicklungskomponenten
- Entwicklungsumgebung muß konfigurierbar sein, z.B. Einbindung eines anderen Java-Compilers oder Appletviewers
- namhafter Hersteller, der Support zur Verfügung stellt

Klassenbibliotheken:

- Verfügbarkeit auf möglichst vielen Plattformen
- einfache Integration in die Entwicklungsumgebung
- gute Dokumentation vorhanden
- namhafter Hersteller, der Support zur Verfügung stellt

Sonstige Werkzeuge:

- Internetbrowser

### 4.2.3 Mögliche Kombinationen

Im folgenden werden einige Kombinationen von Entwicklungsumgebung und Compiler näher betrachtet. Es können nicht alle am Markt verfügbaren Produkte untersucht werden, die Auswahl ist vor allem davon abhängig, welche Produkte in der Abteilung verfügbar sind.

1. Symantec Visual Café 1.0 mit Symantec Visual Compiler
2. WinEdit mit JDK 1.1 von Sun
3. Microsoft J++ 1.1 mit J++ Compiler
4. Microsoft J++ 1.1 mit Microsoft Java SDK Compiler

Klassenbibliotheken:

- Application Foundation Classes von Microsoft (AFC)
- Java Development Classes von Sun (JDC)

Internetbrowser:

- Internet Explorer 4.0 von Microsoft
- Netscape Communicator 4.03 von Netscape Corp.

### 4.2.4 Evaluation

#### **Möglichkeit (1): Symantec Visual Café 1.0**

Bei der Entwicklungsumgebung des Symantec Visual Café sind Source Code und ausführbares Programm eng miteinander verknüpft, man kann jederzeit zwischen Programmteil und Source Code hin- und herspringen. Mein erster Eindruck war sehr positiv, doch bei genauerer Betrachtung stellte ich fest, daß dieser erste Version des Visual Café das neue Event-Modell von Java nicht unterstützt. Damit schied diese Möglichkeit aus.

Mit Java 1.1 führte Sun ein neues Event-Handling ein. Jedes Objekt, daß auf Events reagieren will, kann eine vordefinierte Schnittstelle implementieren und erhält anschließend automatisch die zugehörigen Events. Zuvor gab es eine zentrale Stelle (die Methode "handleEvent"), an die alle Events geschickt wurden. Diese Methode mußte an den gewünschten Stelle redefiniert werden. Zur korrekten Event-Behandlung waren eine Filterung (meist über lange 'CASE'-Strukturen) notwendig. Mit Java 1.1 ist eine wesentlich komfortablere Event-Behandlung möglich.

#### **Möglichkeit (2): WinEdit mit JDK 1.1 von Sun**

Da das Java Development Kit von Sun über keinen komfortablen Editor verfügt, wurde ein sehr guter Editor aus dem Shareware-Bereich für den Gebrauch mit JDK konfiguriert. Nun war beispielsweise das automatische Ausführung des Codes nach erfolgreicher Compilierung möglich, oder auch das Springen an die vom Compiler gelieferten Fehlerstellen.

Diese Lösung hätte den Vorteil gehabt, den originalen Compiler benutzen zu können. Leider konnte WinEdit nicht Anforderungen wie Klassenbrowser, Dialogeditor oder Projektunterstützung erfüllen, weshalb diese Möglichkeit ebenfalls ausschied.

### **Möglichkeit (3): Microsoft J++ 1.1 mit J++ Compiler**

Microsoft bietet eine integrierte Entwicklungsumgebung mit Klassenbrowser, Dialogeditor, komfortabler Verwaltung verschiedener Source Code Dateien, Projektunterstützung und dem Einbinden verschiedener Ressourcen wie Icons oder HTML-Seiten an. Diese Möglichkeit stellte die bisher beste Wahl dar. Sie wurde letztlich nicht in der Form gewählt, da der zugrunde liegende Compiler nicht der neueste war.

### **Möglichkeit (4): Microsoft J++ 1.1 mit Microsoft Java SDK Compiler**

Diese Möglichkeit stellt eine Verbesserung der Möglichkeit (3) dar. Es werden die Vorteile der komfortablen Entwicklungsumgebung übernommen. Zusätzlich werden in diesem SDK die Application Foundation Classes (AFC) mitgeliefert, die zahlreiche Klassen zur Erstellung grafischer Benutzungsoberflächen wie beispielsweise Tree, Tab-Viewer, Kontextmenü, ScrollViewer und desgleichen mehr zur Verfügung stellen. Es ist außerdem der neueste Java-Compiler von Microsoft enthalten, der eine deutliche Verbesserung gegenüber dem mitgelieferten J++-Compiler darstellt.

## 4.2.5

### Auswahl

Die Hauptkriterien sind eine eventuelle Integration in die parallel entwickelte Benutzungsoberfläche und ein komfortable Entwicklungsumgebung mit Projektunterstützung. Das Parallelprojekt baut auf den AFC auf. Daher wurde folgende Auswahl getroffen:

- Betriebssystem:  
Windows NT 4.0 mit Service Pack 1, Fa. Microsoft
- Source Code Entwicklungsumgebung:  
Visual Studio 97 mit J++ 1.1, Fa. Microsoft
- Compiler:  
Microsoft SDK 2.0 Java Compiler, Fa. Microsoft
- Klassenbibliotheken:  
Application Foundation Classes, Fa. Microsoft
- Java Virtual Machine:  
JVT des Internet Explorer 4.0, Fa. Microsoft
- Versionsmanagement:  
ClearCase 2.1.1.NT für Windows NT mit Patch "ccp211nt3", Fa. Atria

Das Produkt ClearCase wird bereits seit mehreren Jahren erfolgreich in der gesamten Abteilung zum Versions- und Dokumentenmanagement eingesetzt.

---

## 4.3 Implementierte Funktionalität des Prototyps

---

Um den Prototyp testen zu können, wurde ein Applikationsserver entwickelt, ebenfalls in Java. Die Benutzungsoberfläche konnektiert sich beim Starten zu diesem Server, der Objekte zur Verfügung stellt.

### **Benutzungsoberfläche**

- Applikationsunabhängige Objektrepräsentation mit dynamischer Anzahl von Attributen
- Synchrone und asynchrone Kommunikation mit dem Server
- Navigationsbaum mit applikationsunabhängiger Objektstruktur, durch Textdatei konfigurierbar
- Informationsbereich mit Präsentationsformen für Objekte und Attribute; die Präsentationsformen werden dynamisch geladen
- Aktionen können sowohl über Kontextmenüs als auch über Drag&Drop angestoßen werden
- Lokales Caching sowohl von Objekten als auch von Queries
- Eingabe von Werten, die zur Ausführung von Aktionen notwendig sind, werden von Dialogen eingelesen, die dynamisch geladen werden; diese Dialoge stoßen die Ausführung der Aktion selbst an, wobei sie die von der Benutzungsoberfläche zum Server bereits aufgebaute Netzwerkverbindung benutzen (keine neues Login notwendig)
- Hilfe ist über Kontextmenü verfügbar
- Initialfokus und Refreshfunktion

### **Kennzahlen der Benutzungsoberfläche**

- ca. 6000 Zeilen Code (ohne Kommentare)
- 33 Klassen
- 22 Klassenmethoden
- 278 Instanzenmethoden
- 24 verwendete AFC-Klassen

### **Server**

- Flat File System Datenbank
- Socket-Kommunikation zur Benutzungsoberfläche
- mehrere Benutzungsoberflächen, die gleichzeitig auf die Datenbank zugreifen können; Zugriffe sind dabei durch Monitor synchronisiert
- einige beispielhafte Aktionen mit entsprechenden Aktionsdialogen: Erstellen von Employees, Kreieren von Rollentemplates, Instanzieren eines Rollentemplates, Erstellen eines Accounts auf einer Ressource

### 4.3.1 Cache Manager

#### Attribute des Cache Manager

ABBILDUNG 23

Attribute des Cache Managers

public Method getObjectMethod	enthält eine Referenz auf die Klassenmethode 'getObjectFromCache'
private Vector objectCache	Objekt Cache; er enthält lokale Kopien von Applikationsobjekten, organisiert als binärer Suchbaum. Bei einem Cache Miss wird das Objekt synchron vom Server angefordert
private Vector queryCache	Query Cache; hier sind lokale Kopien von Queries enthalten, die an den Server geschickt worden sind

#### Konstruktor des Cache Manager

ABBILDUNG 24

Konstruktor des Cache Managers

public CacheMan()	legt einen leeren Objekt- und Query Cache an und generiert Verwaltungsinformationen, die für einen späteren Aufruf der Klassenmethode 'getObjectFromCache' aus einer dynamisch geladenen Klasse notwendig sind
-------------------	--

#### Klassenmethoden des Cache Managers

ABBILDUNG 25

Klassenmethoden des Cache Managers

public static ApplObj getObjectFromCache(String s)	bildet die Schnittstelle zum dynamischen Aufruf der Instanzen-Methode 'getObject'
--	---

#### Instanzen-Methoden des Cache Managers

ABBILDUNG 26

Instanzen-Methoden des Cache Managers

private int queryIsCached (String query)	prüft, ob eine Query im Cache enthalten ist; falls ja, so gibt sie den Index zurück, falls nein ist der Rückgabewert -1
private void putQuery(String query, String resultOfQuery)	legt eine Query und ihr Resultat im Query Cache ab
private void putQueryResultAt(String resultOfQuery, int position)	erneuert das Resultat einer Query an eine bestimmten Positions im Query Cache
public String executeQuery (String query)	führt eine Query aus; falls Query im Query Cache enthalten, so wird das Resultat aus dem Cache zurückgeliefert, ansonsten wird die Query synchron an den Server geschickt und dann im Cache abgelegt
public void invalidateQuery (String query)	markiert die Query als ungültig im Query Cache
public void printQueryCache()	gibt den Query Cache auf der Console aus
private int objectIsCached (String id)	prüft, ob ein Objekt im lokalen Cache enthalten ist; falls ja, so wird der Index des Objekts zurückgegeben, anderenfalls -1

ABBILDUNG 26 Instanzen-Methoden des Cache Managers

public void putObject(String id, String name, String type, String attributes, String actions)	prüft über isObjectCached ab, ob das Objekt bereits im Cache enthalten ist; falls ja, passiert nichts, anderenfalls wird es erzeugt und im Objekt Cache abgelegt
public ApplObj getObject (String id)	liefert das Objekt mit der spezifizierten id; ist es im lokalen Cache enthalten, so wird es von dort geholt, ansonsten vom Server angefordert und im Cache abgelegt
public void printObjectCache()	gibt den Objekt Cache auf der Console aus

### 4.3.2 Display Manager

Klassendefinition: class DisplayMan extends Thread implements Constants

Der Display Manager ist von der Klasse Thread abgeleitet.

#### Attribute des Display Manager

ABBILDUNG 27 Attribute des Display Managers

public boolean dragndrop public boolean dragndrophelp	Flags zur Realisierung von Drag&Drop
public UITabViewer tab public int numberTabs	TabViewer mit #numberTabs Tabs
public UISplitViewer[] split	Array von SplitViewern, für jeden Tab einer
public USecTree[] nav public USecInfo[] info	jeder SplitViewer enthält einen Navigations- und einen Informationsbereich
public QueryNode[] tree public Vector[] specialLayout public Vector generalLayout	jeder Navigationsbereich enthält einen Baum; zu jedem Baum gehört ein spezielles Layout; daneben gibt es noch ein generelles Layout
private Vector[] selectionListNav private Vector[] selectionListInfo	für jeden Navigations- und Informationsbereich wird eine eigene Selektionsliste verwaltet
private Vector dragList private Vector dropList	Listen für Drag&Drop Funktionalität
public UIStatus status	Statuszeile

#### Konstruktor des Display Managers

ABBILDUNG 28 Konstruktor des Cache Managers

public DisplayMan()	legt leere Drag&Drop Listen an
---------------------	--------------------------------

#### Klassenmethoden des Display Managers

Der Display Manager enthält keine Klassenmethoden.

## Instanzen-Methoden des Display Managers

ABBILDUNG 29

Instanzen-Methoden des Cache Managers

<pre>public void initDragDrop() public void addDragDropTarget(Object o) public void addDragDropSource(Object o) public void deleteDragDropSource() public void deleteDragDropTarget() public Object getDragDropSource() public Object getDragDropTarget()</pre>	<pre>initialisiert die Drag&amp;Drop Listen fügt neues Drag&amp;Drop Ziel hinzu fügt neue Drag&amp;Drop Quelle hinzu löscht Drag&amp;Drop Quelle löscht Drag&amp;Drop Ziel hole Drag&amp;Drop Quellen hole Drag&amp;Drop Ziele</pre>
<pre>public String getDefaultQuery(String id)</pre>	<pre>hole Default-Query für Navigationsbereich</pre>
<pre>private void init_generalLayout() public String getGeneralLayout(String type)</pre>	<pre>initialisiere generelles Layout für Navigationsbereich hole in Abhängigkeit eines Typs (eines Applikations- objekts) das generelle Layout (eine Query); falls für den Typ kein Layout gefunden wird, liefere einen leeren String zurück</pre>
<pre>private void init_specialLayout(int i)</pre>	<pre>initialisiere spezielles Layout für Navigationsbereich i</pre>
<pre>\$X == (Nav    Info) public String getSelectionList\$X()  private int isInselectionList\$X(String s) public synchronized void addToSelection- List\$X(String s) public synchronized void removeFromSe- lectionList\$X(String s)</pre>	<pre>Nav = Navigationsbereich, Info = Informationsbereich liefert einen String, der alle zur Zeit im selektierten Objekte enthält, getrennt durch Leerzeichen prüft, ob ein Objekt schon in der Selektionsliste ist fügt ein Objekt zur Selektionsliste hinzu, falls es nicht schon enthalten ist löscht ein Objekt aus der Selektionsliste</pre>
<pre>public void init()</pre>	<pre>initialisiert den TabViewer, die SplitViewer, die Naviga- tions- und Informationsbereiche, die Navigations- bäume, die Layouts, die Selektionslisten und den Initialzustand; diese Methode wird automatisch bei Starten des Display Managers als Threads aufgerufen</pre>

### 4.3.3 Kommunikations-Manager

Klassendefinition: `public class KommMan extends Thread`

Der Kommunikations-Manager ist als Thread implementiert.

#### Attribute des Kommunikations-Managers

ABBILDUNG 30

Attribute des Kommunikations-Managers

<code>public Socket socket</code> <code>private String server</code> <code>private int port</code> <code>public ReadFromServer reader</code> <code>public WriteToServer writer</code>	Socket für die Kommunikation mit dem Server Name (oder IP-Adresse) des Servers Nummer des TCP Ports Thread, der am Port horcht und Strings vom Server empfängt Thread, der auf den Port Strings schreibt
<code>public Method sendSyncMethod</code> <code>public Method sendAsyncMethod</code>	Referenz auf Klassenmethode "sendSyncToServer" Referenz auf Klassenmethode "sendAsyncToServer" ermöglichen das Benutzen der Kommunikationsverbindung durch dynamisch geladenen Klassen

#### Konstruktor des Kommunikations-Managers

ABBILDUNG 31

Konstruktor des Kommunikations-Managers

<code>public KommMan(String s, int p)</code>	das Recht, eine socket connection aufzubauen, wird in der Policy-Engine eingeräumt; Aufbau der socket connection; Initialisieren der Threads zum asynchronen Lesen und Schreiben; Initialisierung der Referenzen auf "sendSyncToServer" und "sendAsyncToServer"
--	---

#### Klassenmethoden des Kommunikations-Managers

ABBILDUNG 32

Klassenmethoden des Kommunikations-Managers

<code>private static String getid()</code>	generiert eine neue ID für synchrone Kommunikation
<code>public static String sendSyncToServer(String s)</code> <code>public static void sendAsyncToServer(String s)</code>	ruft die Instanzen-Methode zum synchronen Schreiben an den Server auf und wartet auf das Ergebnis ruft die Instanzen-Methode zum asynchronen Schreiben an den Server auf

#### Instanzen-Methoden des Kommunikations-Managers

ABBILDUNG 33

Instanzen-Methoden des Kommunikations-Managers

<code>public synchronized void sendSync(String s)</code> <code>public synchronized void sendAsync(String s)</code> <code>public synchronized String read()</code>	verschickt den Strings s an den Server und wartet, bis der Server antwortet verschickt den Strings s an den Server und wartet nicht auf eine Antwort diese Methode realisiert das Warten auf eine Antwort des Servers
---	---

### 4.3.4 Action Manager

Klassendefinition: class ActionMan extends Thread implements Constants

#### Attribute des Action Manager

ABBILDUNG 34

Attribute des Action Managers

public final String defaultActionsForVirtual	definiert die Default Actions für logische Knoten
--	---

#### Konstruktor des Action Manager

Der Konstruktor des Action Managers ist leer.

#### Klassenmethoden des Action Managers

Die Klasse ActionMan enthält keine Klassenmethoden.

#### Instanzen-Methoden des Action Managers

ABBILDUNG 35

Instanzen-Methoden des Action Managers

void executeDragDrop()	holt die Objekte von den Drag&Drop Quell- und Ziellisten und führt die dadurch spezifizierte Drag&Drop Aktion aus
public void executeAction(int fromObjectType, String action)	wird aufgerufen, falls eine Aktion im Kontextmenü ausgewählt wird; ObjectType enthält den Typ des selektierten Knotens (logisch (0) oder real(1)), action enthält die ausgewählte Aktion; 'refresh', 'Hilfe' und Aktionen für logische Knoten können direkt ausgeführt werden; bei Aktionen für reale Knoten wird der zugehörige Dialog vom Server geholt und parametrisiert instanziiert

### 4.3.5 Hilfe Manager

Klassendefinition: class HelpMan extends Thread implements Constants

#### Attribute des Hilfe Manager

ABBILDUNG 36

Attribute des Hilfe Managers

public final String defaultHelpEntry	definiert die Aktion im Kontextmenü zum Aufruf der Hilfe
--------------------------------------	--

#### Konstruktor des Hilfe Manager

Der Konstruktor des Hilfe Managers ist leer.

#### Klassenmethoden des Hilfe Managers

Die Klasse HelpMan enthält keine Klassenmethoden.

**Instanzen-Methoden des Hilfe Managers**

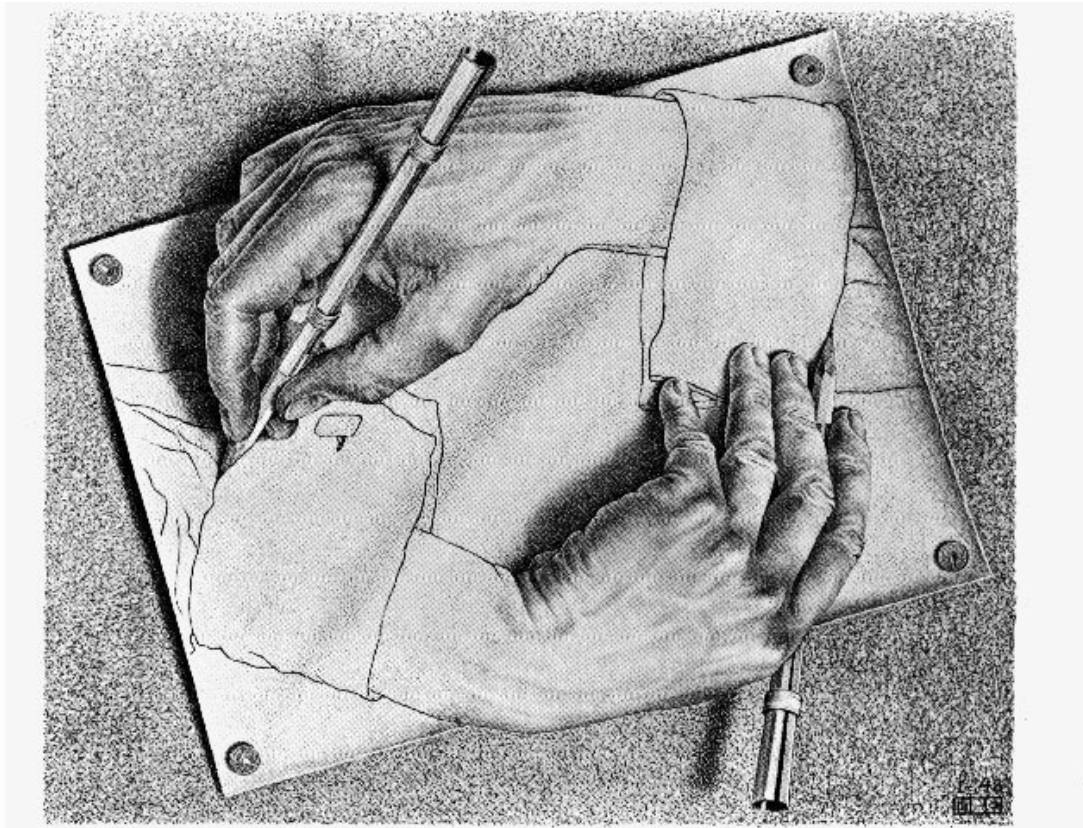
ABBILDUNG 37

Instanzen-Methoden des Hilfe Managers

---

public void executeHelpForReal(String id)	Hilfe für realen Knoten anzeigen
public void executeHelpForVirtual(String id)	Hilfe für logischen Knoten anzeigen

# RÜCK- UND AUSBLICK



## 5.1 Ergebnisse

In dieser Diplomarbeit wurde ein in Java geschriebener Prototyp einer Benutzungsoberfläche entwickelt. Der Prototyp ist als Front-End einer Service Request Applikation gedacht, hier speziell die User/Security Applikation. Er kann durch Konfiguration und das Einbinden von dynamisch geladenen Dialogen und Präsentationsformen jedoch in eine beliebige solche Applikation eingebunden werden.

Seine Entwicklung wurde nach dem Modell des Softwarelebenszyklus durchgeführt. Er ist kein Weg-Werf-Prototyp, sondern hat vielmehr evolutionären Charakter und kann als Basis für die Implementierung einer speziellen Benutzungsoberfläche einer Service Request Applikation dienen.

Er ist komplett in Java als Applet geschrieben, wobei die von Microsoft entwickelten Application Foundation Classes (AFC) verwendet wurden. Microsoft stellt diese in

ihrem Software Development Kit (SDK) für Java zur Verfügung, welches nicht nur auf Microsoft eigenen Betriebssystemen, sondern auch auf anderen lauffähig ist. Der Prototyp ist in diesem Sinne eine plattformunabhängige Software.

## 5.2 Projektverlauf

Entsprechend dem Softwarelebenszyklus-Modell habe ich neben den inhaltlichen Zielen auch einen Projektplan erstellt, in dem die insgesamt zur Verfügung stehende Zeit in Abschnitte (“Phasen”) zerlegt und Ziele für jeden Abschnitt definiert wurden. Um von einem zum nächsten Abschnitt zu gelangen, mußten die Ziele (“Meilensteine”) erfüllt und vom zugehörigen Management (in diesem Fall mein Betreuer) angenommen werden.

Um den Projektplan mit konkreten Zahlen füllen zu können, wurde zunächst eine Gewichtung der einzelnen Phasen bezüglich der Zeit vorgenommen. Anders ausgedrückt bedeutet dies, es wurde abgeschätzt, welche Phase wieviel Prozent der Gesamtzeit beanspruchen wird. Danach wurde die Anzahl der zur Verfügung stehenden Arbeitstage bestimmt, abzüglich Einarbeitungszeit und Vorarbeiten. Mit Hilfe der zuvor geschätzten Zeitanteile entstand nun die folgenden Tabelle.

ABBILDUNG 38

Projektplan

Phase	Meilenstein	Dauer in Tagen
Untersuchung der Anforderungen	Liste der Anforderungen an Prototyp	11
Systemdesign	Designdokumente	22
Implementierung	lauffähiger Prototyp	44
Projektabschluß	abgabefertige Diplomarbeit	34

Dieser Projektplan entspricht nicht vollständig dem im Softwarelebenszyklus vorgeschlagenen Modell. Es fehlen die Phasen “Integration”, “Systemtest” und “Abnahme durch den Endbenutzer”.

Die Phase “Integration” dient zum Verbinden von einzelnen Softwareteilen zu einem Gesamtsystem. Dafür verfügen die Einzelkomponenten über zuvor definierte und verbindliche Schnittstellen. Da mein Prototyp ‘aus einem Stück’ besteht, entfiel diese Phase.

In der Phase “Systemtest” wird die Software von unabhängigen Softwaretester geprüft. Wichtig ist hier, daß der Entwickler selbst nicht beteiligt ist. Er hat seine eigenen Tests, um die generelle Funktionsfähigkeit seiner Software sicherzustellen, bereits vollzogen (der sogenannte “Unit-Test”). Der Unit-Test wurde bei meinem Prototyp als Teil der Phase “Implementierung” durchgeführt.

Bei der Abnahme durch den Endbenutzer wird das bereits getestete System beim Endbenutzer installiert und durch weitere Tests geprüft. Aus Unternehmenssicht ist dies die wichtigste Phase. Kundenzufriedenheit hat oberste Priorität. Bei meinem Prototyp gab es dafür keine explizite Phase, “Kunde” zur Abnahme war vor allem mein Betreuer, der

die Projektinteressen desjenigen Projekts vertrat, dem mein Prototyp als Basis dienen sollte. Den Abschluß meines Projekts "Diplomarbeit" bildeten Reviews meiner Dokumente, zusammen mit meinem Betreuer, Überarbeitungszyklen und eine abschließende Demo für das Projektteam.

## 5.3 Offene Punkte

---

Dieser Abschnitt zeigt zwei Arten von offenen Punkten auf. Zunächst sind dies diejenigen Punkte, die in den Anforderungen zwar definiert worden sind, für die aber keine praxistaugliche Lösung gefunden werden konnte bzw. die aus Zeitmangel nicht implementiert werden konnten. Zum anderen sind dies Punkte, die während der Erstellung des Prototyp und dieser Arbeit erst aufgetaucht sind und deshalb zuvor nicht berücksichtigt werden konnten.

### Offene Punkte dieser Diplomarbeit

Hier eine Liste von Anforderungen, die zwar formuliert worden sind, aber aus Zeitnot nicht implementiert werden konnten. Zu jedem Punkt wird jeweils eine Abschätzung über die Machbarkeit bzw. den Aufwand gegeben.

- **Rahmen:**  
Sie erhöhen ebenfalls die Benutzerfreundlichkeit und sind leicht zu implementieren.
- **Vier-Augen-Prinzip und Zugriffsrechte:**  
Zugriffsrechte sind ein Muß, das Vier-Augen-Prinzip kann implementiert werden. Ohne Zugriffsrechte kann die Benutzungsoberfläche in sicherheitskritischen Unternehmensbereichen nicht zum Einsatz kommen. Der Aufwand ist von Seite der Benutzungsoberfläche eher gering, die Hauptarbeit muß im Server geleistet werden. Da das Vier-Augen-Prinzip nicht in allen Firmen benutzt wird, muß es nicht unbedingt implementiert werden.
- **Filtermechanismen:**  
Sie ermöglichen ein selektiveres Navigieren in großen Objektstrukturen und erhöhen dadurch die Benutzerfreundlichkeit. Der Aufwand zur Implementierung liegt vor allem auf Seite des Servers, der Queries mit zusätzlichem Filter beantworten können muß.
- **GUI-Aktionen:**  
Diese Aktionen dienen der interaktiven Verwaltung von logischen Knoten. Ihre Implementierung ist nach meiner Abschätzung mit einigem Aufwand verbunden. Sie verändern sowohl das spezielle als auch das generelle Layout des Navigationsbaumes und sind per Drag&Drop oder über das Kontextmenü ausführbar.
- **Bookmarks:**  
Meiner Meinung nach ist dies sehr hilfreich für die Navigation in großen Objekthierarchien. Man vergleiche hierzu den Wert von Bookmarks in einer Objekthierarchie namens "Internet". Der Aufwand zur Realisierung ist gering, der Nutzen hoch. Es lohnt sich, diese Funktionalität noch zu implementieren.

### Allgemeine offene Punkte

Java wurde von der Firma Sun als plattformunabhängige Sprache entwickelt. Dies zeigt sich vor allem daran, daß das Java Development Kit auf praktisch allen Plattformen verfügbar ist. Während ich diese Diplomarbeit schrieb (und vor allem nachdem mein Prototyp fast war) wurde immer offenkundiger, daß Microsoft diesem Bestreben entgegenwirken wollte und dies auch mit Äußerungen wie "Java is just another programming language" auch tat. Microsoft sah und sieht durch eine Sprache, mit der Software unabhängig vom Betriebssystem entwickelt werden kann, eine Gefährdung ihrer Monopolstellung im PC-Software-Markt. Wenn Software vorhanden ist, die unabhängig vom Betriebssystem eingesetzt werden kann, so ist es letztlich egal, welches der Endbenutzer hat. Und dies kann durchaus eines sein, das nicht von Microsoft kommt, wie beispielsweise Linux oder OS/2.

Microsoft kommt eine allgemeine Unzufriedenheit der Java-Entwicklungsgemeinde gerade recht. Immer mehr Entwickler scheinen den Glauben in die "Write Once, Run Any-where" Eigenschaft von Java zu verlieren, da natürlich nicht alle Implementierungen der Java Virtual Machine auf allen Plattformen gleich sind. Die Java Virtual Machine ist auch ein Stück Software, daß irgendwo irgendwelche Fehler enthält. Ein Artikel von PointCast, einem Online-Nachrichten-Anbieter (<http://www.pointcast.com>), weist im November auf Umfrageergebnisse hin, bei denen zwei Drittel aller befragter Software-Entwickler Zweifel an der faktischen Plattformunabhängigkeit von Java zum Ausdruck brachten. Hier einige Punkte, die als Ursache für den Unmut genannt wurden:

- Inkompatibilität der einzelnen Java-Versionen (vor allem zwischen JDK 1.02 und 1.1)
- Schlechte Performance des Just-In-Time Compiler
- Keine Standards
- Trotz proklamierter Plattformunabhängigkeit muß auf allen Plattformen separat getestet werden

Demgegenüber stehen die Bemühungen von Sun, Java in immer neuen Versionen zu verbessern, ihre Virtual Machine zu "entwanzen" und immer mehr Plattformen zu unterstützen.

Keiner weiß, wie dieser Kampf der Giganten ausgehen wird. Wird es einmal ein standardisierte Programmiersprache namens "Java" geben, zum Beispiel ANSI-Java? Oder werden sich zwei Java-Versionen entwickeln, eine gestützt von Sun und von Netscape, die andere von Microsoft? Oder wird Java, das aufstieg wie Phönix aus der Asche, genauso schnell wieder in einer Versenkung verschwinden? Letzteres wohl eher kaum, da Java die Programmiersprache ist, mit der das immer weiter wachsende Internet komfortabel programmiert werden kann.

## 5.4 Ausblick

---

Ein nicht abzusehendes Ereignis beeinflusste den Fortgang und die Zukunft des Parallel-Projekts negativ. Auf einmal war nicht mehr klar, wie und an welcher Stelle eine Integration der Benutzungsoberfläche, für die mein Prototyp als Basis dienen sollte, stattfinden sollte. Markteinflüsse, die man nicht beherrschen kann, wiesen einen anderen Weg in die Zukunft, und auf einmal war vieles unklar, wovon ich am Anfang meiner Diplomarbeit ausgegangen war. Diejenigen Bedingungen, die man so schön einfach als Rahmenbedingungen bezeichnen kann, änderten sich mit einem Schlag, und das bereits gebaute Haus begann zu wanken. Deshalb ist zu diesem späten Zeitpunkt der Arbeit nicht mehr gesichert, wie und in welchem Umfang mein Prototyp als Implementierungsbasis benutzt werden wird.

Ungeachtet dieser Tatsache können die von mir entwickelten Konzepte, zum Beispiel das Konzept des dynamischen Ladens von Aktionsdialogen, weiterverwendet werden. Dies ist der wesentliche Nutzen meiner Diplomarbeit.

Für mich selbst sehe ich meine Diplomarbeit als großen Gewinn an. Ich habe viel bei ihr gelernt und schätze dieses erworbene Wissen für meine persönliche Zukunft.

# LITERATURVERZEICHNIS

---

Ken Arnold und James Gosling: *Java - Die Programmiersprache*; Addison-Wesley, 1996

David Flanagan: *Java in a Nutshell*; O'Reilly & Associates, 1996, Internet: <http://www.ora.com/catalog/javanut>

Gary Cornell und Cay S. Horstmann: *Core Java*; SunSoft Press, 1997

Brad J. Cox und Andrea J. Novobilski: *Object-Oriented Programming - An evolutionary approach*; Addison-Wesley, 1991

Dave Collins: *Designing Object-Oriented User Interfaces*; Addison-Wesley, 1995

Ben Shneiderman: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*; Addison-Wesley, 1992

James Martin und James J. Odell: *Object-Oriented Methods - A Foundation*; Prentice Hall, 1995

Peter Coad und Jill Nicola: *Object-Oriented Programming*; Prentice Hall, 1993

David A. Taylor: *Object-Oriented Technology: A Manager's Guide*; Addison-Wesley, 1990

Rul Gunzenhäuser: *Skriptum "Interaktive Systeme"*; Institut für Informatik, Universität Stuttgart, Wintersemester 1994/95

Heidi Heilmann: *Skriptum "Informations- und Kommunikationssysteme"*; Institut für Wirtschaftsinformatik, Universität Stuttgart, Wintersemester 1994/95, Sommersemester 1995

Desktop Management Task Force: *Common Information Model*; April 1997; Internet: <http://www.dmtf.org/cim>, <ftp://ftp.dmtf.org/cim/cimdoc11.pdf>

Internet-Seiten zum Thema Java:

<http://java.sun.com/sfaq>

<http://java.sun.com/products/jdk/1.1/docs/api/packages.html>

<http://www.javasoft.com>

<http://www.javasoft.com/docs/books/tutorial>

<http://www.microsoft.com/visualj>

<http://cafe.symantec.com/cafe>

<http://www.gamelan.com>

---

# GLOSSAR

---

**Account**

Ein Account beschreibt die Zugriffsberechtigung eines Employees auf eine Ressource.

**Aliasing**

Unter Aliasing versteht man den Effekt, daß ein und dasselbe Objekt mehr als einmal zur gleichen Zeit referenziert wird.

**Assoziation**

Eine Assoziation ist eine typisierte Beziehung zwischen zwei oder mehr Objekten.

**Autorisierung**

Dies beschreibt den Vorgang des Einholens der Erlaubnis, eine Aktion auszuführen.

**Basisrollentemplate**

Die Rollentemplatehierarchie baut auf Basisrollentemplates auf. Aus ihnen werden Basisrollen abgeleitet, die Accounts auf Ressourcen widerspiegeln.

**Bookmark**

Ein Bookmark ist eine gespeicherte Referenz auf eine bestimmte Stelle in einem Netzwerk oder einer Hierarchie.

**Employee**

Der Begriff "Employee" beschreibt einen Angestellten bzw. einen Mitarbeiter einer Firma. Einem Employee kann über ein Rollentemplate eine Rolle zugewiesen werden.

**GUI-Aktionen**

Dies sind Aktionen, die die Benutzungsoberfläche unabhängig vom Applikationsserver ausführen kann. Applikationsobjekte bleiben von ihnen unberührt.

**GUI-Server**

Der GUI-Server ist diejenige Komponente im Server, die sämtliche Interaktionen mit der Benutzungsoberfläche abwickelt.

**Informationsbereich**

Dies ist der Bereich, in dem Präsentationsformen auf dem Bildschirm dargestellt werden. Er befindet sich auf der rechten Seite des Darstellungsbereichs.

**Layout**

Das Layout beschreibt den Aufbau des Verwaltungsbaumes und damit des Navigationsbaumes.

**Lokaler Cache**

Hier werden lokale Kopien von Objekten und Queries gehalten. Der lokale Cache besteht aus einem Objekt und einem Query Cache.

**Massenoperation**

Eine Aktion, die auf mehr als eine Objekt angewendet werden kann bzw. wird, heißt Massenoperation.

**Navigationsbereich**

Er visualisiert den Navigationsbaum und ist auf der linken Seite des Darstellungsbereichs zu finden.

**Notifikation**

Eine Notifikation ist eine vom Server ausgelöste Benachrichtigung über die Änderung von Objekten, die die Benutzungsoberfläche betreffen.

**Notifikations-Mechanismus**

Dies ist der Vorgang der Notifikation mit der entsprechenden Reaktion der Benutzungsoberfläche auf die Notifikation.

**Rahmen**

Rechteckiger Bildschirmbereich, der durch das Zeigegerät definiert werden kann und zur erweiterten Objektmarkierung verwendet werden kann.

**Ressource**

Ein Ressource ist Gerät, das zur Ausführung der Arbeit eines Employees notwendig ist, beispielsweise ein Computersystem oder eine Applikation. Seine Benutzung ist durch Accounts beschränkt.

**Rolle**

Um die Mitarbeiter bezüglich ihrer in einem Unternehmen zugewiesenen Tätigkeiten zu erfassen, existieren in der User/Security Applikation sogenannte Rollen, die diese Zusammenhänge auf Datenmodelle abbilden. Diese Datenmodelle sind sogenannte Rollentemplates.

**Rollentemplate**

Ein Rollentemplate ist die Datenstruktur zur Beschreibung einer Rolle. Zum Erschaffen einer Ausprägung dieses Rollentemplates kann es instanziiert werden.

**Rollentemplatehierarchie**

Um Modularisierung innerhalb des Datenmodells "Rollentemplate" zu ermöglichen und gleichzeitig Replikation von Daten zu vermeiden, ist der Aufbau einer Hierarchie aus Rollentemplates möglich. Dabei wird von Basisrollentemplates ausgegangen, die die Blätter dieser Hierarchie bilden. Auf sie aufsetzend können neue Rollentemplates definiert werden.

**Scheduling-Mechanismus**

Der Applikationsserver bietet die Möglichkeit an, Service Request nicht sofort, sondern erst zu einem definierten Zeitpunkt in der Zukunft auszuführen.

**Server-Aktionen**

Dies sind Aktionen, die nur der Applikationsserver und nicht die Benutzungsoberfläche ausführen kann. Ein Service Request besteht aus Server-Aktionen.

**Service Request**

Unter einem Service Request versteht man einen Auftrag zur Änderung der bestehenden IT-Umgebung. Ein Service Request muß vor seiner Durchführung autorisiert werden.

**Service Request Applikation**

Ein solche Aktion kann Service Requests ausführen. Dabei muß sie nicht zwingend alle einzelnen Aktionen selbst ausführen können, sie kann diese auch an andere Stellen delegieren.

**Synchronisation**

Um Änderungen in der bestehenden IT-Umgebung feststellen zu können, muß ein Abgleich mit einem zuvor erstellten Datenbestand durchgeführt werden. Dies wird als Synchronisation bezeichnet und dient der Erkennung von Änderungen, die von Hand und nicht durch die Service Request Applikation durchgeführt worden sind.

**User/Security Applikation**

Dies ist eine spezielle Service Request Applikation, deren Service Requests sich nur auf die Benutzer einer IT-Umgebung beziehen. Sie leistet damit einen wesentlichen Beitrag zur Sicherheit dieser IT-Umgebung.

**Vier-Augen-Prinzip**

Vier-Augen-Prinzip bedeutet, daß es nicht eine dedizierte Person gibt, die alles machen darf, sondern das zu Kernfragen der Sicherheit immer mindestens zwei Personen als Entscheidungsträger notwendig sind.

**View**

Ein View ist die technische Realisierung einer ganz bestimmten Sichtweise auf Datenobjekte, meist gekoppelt an einen bestimmten Benutzer.

**Vorgangnetz**

Ein Vorgangnetz beschreibt den Ablauf zur Ausführung einer Tätigkeit, in diesem Fall eines Service Requests. Für jeden Schritt ist derjenige definiert, der ihn ausführt, und diejenige Stelle, bei der vor der Ausführung eine Genehmigung eingeholt werden muß.

# SELBSTÄNDIGKEITSERKLÄRUNG

---

Hiermit versichere ich, daß ich diese Arbeit selbst erstellt und nur die angegebenen Hilfsmittel verwendet habe.

Stuttgart, 23. Dezember 1997