

# Universität Stuttgart

## Institut für Parallele und Verteilte Höchstleistungsrechner

Prüfer: Prof. Dr.-Ing. A. Reuter

Betreuer: Dipl.-Inform. F. Schwenkreis

Begonnen am: 27.05.1996

Beendet am: 10.12.1996

CR-Klassifikation: H.5.2, H.4.1, C.2.4, J.1.1

Diplomarbeit-Nr. 1419

### **Eine Javaoberfläche zur Administration**

Tobias Hummel

# Inhaltsverzeichnis

<b>1 EINLEITUNG</b>	<b>6</b>
<b>1.1 Workflows</b>	<b>6</b>
1.1.1 Definition bzw. informelle Beschreibung	6
1.1.2 Die Bedeutung von Workflows für die moderne Unternehmung	7
1.1.3 Workflow Management Systeme	9
<b>1.2 Das ConTract-Modell</b>	<b>10</b>
1.2.1 Was ist ein ConTract?	10
1.2.2 Programmiermodell	11
1.2.3 Fazit	11
<b>1.3 Aufgabenstellung dieser Arbeit</b>	<b>12</b>
<b>2 APRICOTS</b>	<b>13</b>
<b>2.1 Was ist APRICOTS?</b>	<b>13</b>
<b>2.2 Architektur</b>	<b>13</b>
2.2.1 ConTract Manager	14
2.2.2 Transaction Manager	14
2.2.3 Context Manager	15
2.2.4 Automated Step Server	15
2.2.5 Resource Manager	15
2.2.6 User Agent	15
2.2.6.1 Monitor Agent	15
2.2.6.2 Tasklist Agent	16
2.2.7 Storage Agent	16
2.2.8 Der Naming Service	16
2.2.9 User Interface	16
<b>2.3 Der Namensraum von APRICOTS</b>	<b>17</b>
2.3.1 Hierarchie und Rollen	18
2.3.2 Modellierungsmöglichkeiten von Organisationsstrukturen	19
<b>2.4 Realisierung</b>	<b>20</b>
2.4.1 Implementierungsgrundsätze	20
2.4.2 Kommunikation	21
<b>2.5 Beurteilung</b>	<b>21</b>
<b>3 CORBA</b>	<b>23</b>
<b>3.1 Verteiltes Objekt Management</b>	<b>23</b>
<b>3.2 Spezifikation und Arbeitsweise von CORBA</b>	<b>24</b>
<b>3.3 CORBA-Services</b>	<b>26</b>
3.3.1 Naming Service	26
3.3.2 OTS als Service für die verteilte Transaktionsverwaltung	26
3.3.3 Persistent Object Service	27

<b>3.4 Die Eignung von CORBA für die Realisierung eines verteilten, objektorientierten Systems</b>	<b>27</b>
<b>3.5 IONA Orbix als CORBA-Implementierung in APRICOTS</b>	<b>28</b>
3.5.1 Die Verwendung des Naming Services	28
3.5.2 Die Verwendung von OTS	28
<b>4 DAS ADMINISTRATIVE WERKZEUG IM DETAIL</b>	<b>30</b>
<b>4.1 Aufgaben</b>	<b>30</b>
4.1.1 Der System Monitor	30
4.1.2 Die System Administration	31
<b>4.2 Anforderungen an die graphische Benutzungsoberfläche</b>	<b>32</b>
4.2.1 Allgemeine Anforderungen an graphische Benutzungsoberflächen	32
4.2.2 Spezielle Anforderungen im Bezug auf das administrative Werkzeug	33
<b>5 BENUTZUNGSKONZEPT DES ADMINISTRATIVEN WERKZEUGS</b>	<b>35</b>
<b>5.1 Objekte und Funktionalität</b>	<b>35</b>
5.1.1 Knotenorientiert	35
5.1.2 Komponentenorientiert	36
5.1.2.1 Benutzer/-innen	36
5.1.2.2 Automated Step Server	37
5.1.2.3 Resource Manager	39
5.1.2.4 Der Naming Service	39
5.1.2.5 Speicher Agenten, ConTract Manager, ConText Management, Object Request Broker	39
5.1.3 Zusammenfassung	39
<b>5.2 Sicherheitsaspekte</b>	<b>40</b>
<b>6 REALISIERUNG</b>	<b>41</b>
<b>6.1 Programmiersprache und Werkzeuge</b>	<b>41</b>
6.1.1 Java	41
6.1.2 Java-WorkShop und Visual	42
<b>6.2 Architektur</b>	<b>43</b>
<b>6.3 Entwurf</b>	<b>46</b>
6.3.1 Objekte	46
6.3.2 Administration	47
6.3.2.1 Das Einbringen von Komponenten	47
6.3.2.2 Ändern von Komponenten-Attributen	50
6.3.2.3 Entfernen von Komponenten	51
6.3.3 Überwachung / Monitoring	52
6.3.3.1 Knotenorientiert	53
6.3.3.2 Komponentenorientiert	54
<b>6.4 Anforderungen an Benutzer Agenten bzw. Step Server</b>	<b>55</b>
6.4.1 Aufruf-Syntax	55
6.4.2 Aktionen beim erstmaligen Anstarten	56

<b>6.5 Die graphische Benutzeroberfläche</b>	<b>56</b>
6.5.1 Zugang zum administrativen Werkzeug	56
6.5.2 Der System Monitor	58
6.5.3 Die System Administration	60
<b>6.6 Der Einsatz von Multi-Threading im administrativen Werkzeug</b>	<b>62</b>
6.6.1 Administration	62
6.6.2 Monitoring	62
<b>6.7 Recovery-Fähigkeiten</b>	<b>63</b>
6.7.1 Speicher Agent (Storage Agent)	64
6.7.2 Direkte Datenbankanbindung mit JDBC	64
<b>7 ABSCHLIEßENDE DISKUSSION</b>	<b>65</b>
<b>7.1 Praktikabilität des Werkzeugs</b>	<b>65</b>
<b>7.2 Ausblick</b>	<b>65</b>
7.2.1 Lastbalancierung und Migration	65
7.2.2 System Monitor Agent	66
<b>8 LITERATUR</b>	<b>67</b>
<b>9 ANHANG</b>	<b>70</b>
<b>9.1 Die IDL-Schnittstelle zum Speicher Agenten</b>	<b>70</b>
<b>9.2 Der Entwurf der administrativen Datenbank</b>	<b>73</b>

## Abbildungs- und Tabellenverzeichnis

Abb. 2.1:	Die Architektur von APRICOTS	14
Abb. 2.2:	Der Namensraum von APRICOTS	17
Abb. 2.3:	Hierarchische Funktionsbereichsstruktur einer fiktiven Unternehmung	19
Abb. 2.4:	Abbildung auf den Namensraum von APRICOTS	20
Abb. 3.1:	Die Struktur des ORB-Interfaces nach [OMG95]	25
Abb. 6.1:	Die drei Basis-Verteilungsmodelle nach [Clie95]	43
Abb. 6.2:	Die Architektur des administrativen Werkzeugs	45
Abb. 6.3:	Das Einbringen einer Komponente	48
Abb. 6.4:	Übergänge der Soll-Zustände eines Automated Step Servers	49
Abb. 6.5:	Der Zugang zum administrativen Werkzeug	56
Abb. 6.6:	Der System Monitor	57
Abb. 6.7:	Der Host Monitor	58
Abb. 6.8:	Die Benutzer Administration	59
Abb. 6.9:	Das Editieren der Rollen	60
Tabelle 5.1:	Attribute eines Rechnerknotens	35
Tabelle 5.2:	Attribute einer Benutzerin bzw. eines Benutzers	36
Tabelle 5.3:	Attribute eines Automated Step Servers	37
Tabelle 6.1:	Mögliche Zustände eines Rechnerknotens	52
Tabelle 6.2:	Mögliche Ist-Zustände einer Komponente	54
Tabelle 6.3:	Administrationsberechtigungen	56
Tabelle 9.1:	Die Relation <i>SysAdmin_Components</i>	72
Tabelle 9.2:	Die Relation <i>SysAdmin_ComponentRoles</i>	72
Tabelle 9.3:	Die Relation <i>SysAdmin_Hosts</i>	73
Tabelle 9.4:	Die Relation <i>SysAdmin_Settings</i>	73

# 1 Einleitung

„*Workflow-Management*“ hat sich in der letzten Zeit zu einem „Modethema“ entwickelt wie Anfang der 80er Jahre relationale Datenbanksysteme und Anfang der 90er Jahre Objektorientierung. Der Begriff „Workflow“ tritt erstmals in den 70er Jahren in der betriebswirtschaftlichen Literatur in Erscheinung. Man versuchte dort insbesondere Büroabläufe zu strukturieren und zu automatisieren. In diesem Zusammenhang ist der Begriff „*business-process-engineering*“ ebenfalls bekannt geworden. Der heutige Begriff „Workflow“ umfaßt jegliche Art von Prozessen, wie sie in betriebswirtschaftlichen Unternehmungen vorkommen können.

In der Informatik wird der Begriff Workflow-Management häufig im Zusammenhang mit sogenannten „nicht-standard“-Anwendungen genannt. Unter „nicht-standard“-Anwendungen versteht man beispielsweise Büroautomatisierung, Steuerung von Produktionsanlagen oder CAD-Anwendungen. Im Gegensatz dazu sind z.B. Banksysteme, Lagerverwaltungen oder Buchhaltungssysteme typische Vertreter von „Standard-Anwendungen“. Gegenüber klassischen (Standard-)Anwendungen haben die „nicht-standard“-Anwendungen unter anderem die Eigenschaft, daß eine große Anzahl von Objekten oft über einen großen Zeitraum bearbeitet werden müssen. Die Informatik versucht Ausführungsplattformen für die Abarbeitung von Workflows zur Verfügung zu stellen.

Diese Arbeit entstand im Rahmen des Workflow-Management-Systems APRICOTS, das auf dem ConTract-Modell nach [WäRe91] basiert.

Im folgenden werden zunächst Workflows genauer betrachtet. Der Begriff wird abgegrenzt und motiviert. Danach wird das ConTract-Modell kurz vorgestellt. Die Einleitung wird mit der Aufgabenstellung dieser Arbeit abgeschlossen.

## 1.1 Workflows

### 1.1.1 Definition bzw. informelle Beschreibung

[Jab195] stellt das Fehlen von anerkannten theoretischen Grundlagen im Bereich von Workflows fest. Dies führt zu einer wenig exakten, eher informellen Definition des Begriffs „Workflow“. Ein Workflow ist durch folgende Eigenschaften charakterisiert:

- An der Ausführung eines Workflows sind mehrere Personen beteiligt (Arbeitsteiligkeit).
- Ein Workflow besteht aus verschiedenen Teilaufgaben.
- Die Teilaufgaben eines Workflows unterliegen einer vordefinierten Ablaufstruktur.
- Die Behandlung von Fehlerfällen ist vordefiniert.

- Es gibt eine Instanz, welche die Koordination der Teilaufgaben übernimmt.
- Bei der Ausführung von Workflows werden Daten verwaltet und transferiert.

Eine Eigenschaft, die in dieser informellen Beschreibung nicht explizit enthalten ist, ist der Zeitbedarf eines Workflows. Typischerweise dauert die Bearbeitung eines Workflows eine größere Zeitspanne, die je nach Anwendung bis zu Tagen, Wochen oder Monaten betragen kann.

Die Ablaufstruktur eines Workflows kann durch alle denkbaren Kontrollkonstrukte aufgebaut werden. Als Kontrollkonstrukte sind Sequenzen, Schleifen, Parallelverarbeitung und Synchronisation mehrerer paralleler Abläufe möglich. Aus diesen Konstrukten lassen sich alle Arten von Ablaufstrukturen modellieren.

### **1.1.2 Die Bedeutung von Workflows für die moderne Unternehmung**

Workflows gewinnen in der heutigen Zeit zunehmend an Bedeutung für betriebswirtschaftliche Unternehmungen. Dieser Wandel ist durch die rasche und stetig fortschreitende Veränderung der Unternehmungsumwelt bedingt. Die heutige Unternehmungsumwelt ist geprägt durch enormen Konkurrenzdruck und sehr kurze Produktlebenszyklen. Unter dem Produktlebenszyklus versteht man die Zeit zwischen der Markteinführung eines Produkts und dem drastisch sinkenden Absatz, so daß das Produkt vom Markt genommen werden muß. Die Folge von verkürzten Produktlebenszyklen ist die Notwendigkeit, Folgeprodukte in kürzerer Zeit verfügbar zu haben.

Als Ursachen für diese Veränderungen sieht [Zahn96] die zunehmende Internationalisierung und die weltweite Markttransparenz. Zunehmende Internationalisierung bedeutet die Verteilung der Unternehmung in verschiedene Länder. Dadurch werden neue Märkte erschlossen und es können die Vorteile eines speziellen Standorts genutzt werden (z.B. niedrige Lohnkosten, wenige gesetzliche Vorschriften etc.). Zusätzlich zu der zunehmenden Internationalisierung der Unternehmungen, d.h. das „Vorortsein“ beim Kunden, ermöglichen moderne Kommunikationstechniken und Logistik dem Kunden die Auswahl eines Produkts aus dem weltweiten Angebot. Diese Entwicklung wird beispielsweise durch die globale Vernetzung gefördert. Bestellungen bei Versandhäusern z.B. können bereits heute „online“ durchgeführt werden. Dabei ist die Lokation des Versandhauses von geringer Bedeutung.

In der betriebswirtschaftlichen Literatur findet man verschiedene Ansätze, mit den veränderten Bedingungen umzugehen. Zwei - in meinen Augen vielversprechende - Ansätze sollen hier näher betrachtet werden.

[Zahn91] sieht einen Ausweg in der Konzentration auf Kernkompetenzen. Kernkompetenzen sind diejenigen Fähigkeiten von Unternehmungen, welche in der Lage sind, einen langfristigen Wettbewerbsvorteil zu sichern. Kernkompetenzen sind durch langwierige Lern- und Erfahrungsprozesse entstanden und sind das eigentliche

„Vermögen“ der Unternehmung. Als Konsequenz der Konzentration auf Kernkompetenzen läßt sich beobachten, daß Unternehmen Bereiche, die nicht zu ihren Kernkompetenzen zählen, ausgliedern („*Outsourcing*“). Dadurch wird versucht, die Kompetenzen anderer Unternehmungen für die Qualitäts- und Preisstruktur des eigenen Produkts zu nutzen. Vorausgesetzt werden muß allerdings der reibungslose Ablauf des Warenaustauschs zwischen Zulieferer und Einkäufer. Insbesondere muß auf Zeit- und Qualitätstreue geachtet werden. Aus diesem Grund werden heute oft bestimmte Arten von Partnerschaften zwischen Zulieferer und Einkäufer geschlossen, um den Ablauf des Warenaustauschs zu optimieren. Aus diesem Ansatz ist erkennbar, daß durch die Konzentration auf Kernkompetenzen neue Ablaufstrukturen entstehen, die sowohl unternehmungsinterner wie -externer Natur sind. Diese Abläufe sind besonders sensibel, da sie zum einen kostenintensive Bereiche (z.B. Fertigung mit hoher Kapitalbindung) und zum anderen den Absatz der Produkte unmittelbar betreffen. Abläufe dieser Art lassen sich gut durch Workflows abbilden und abarbeiten. Denkbar sind automatisierte Bestellungen beim Zulieferer auf elektronischem Wege, bei entsprechenden Verträgen der Verzicht auf Eingangskontrollen, „*just-in-time*“-Anlieferung von Teilen, so daß auf Eingangslager verzichtet werden kann etc..

[Wald94] zeigt den Wandel weg von der Strukturierung nach Funktionen, hin zu der Strukturierung nach Produkten oder Märkten. Mit der Strukturierung nach Funktionen ist die klassische Abteilungsorganisation gemeint mit Verantwortlichkeiten für die Funktionsbereiche einer Abteilung. Das Augenmerk liegt hier auf der Optimierung des eigenen Funktionsbereichs. Durch den Blick durch das „Sichtfenster“ des eigenen Funktionsbereichs wird das Gesamtprodukt und damit die Kundenbedürfnisse bzw. der Markt aus den Augen verloren. Außerdem lassen sich Zusammenhänge bei der Produktzusammensetzung nur schwierig erkennen. Das Ziel der Verantwortlichen eines Funktionsbereichs ist häufig der positive Quartalsabschluß der eigenen Abteilung. Es ist jedoch bekannt, daß die Summe aus einzelnen Optima nicht zwingend das Optimum des Gesamten ergibt. Es fehlt die integrierte Sichtweise, die Strukturen und Zusammenhänge erkennen läßt. Diese Erkenntnis führt zu der Strukturierung nach Produkten oder Märkten („*horizontal*“) und lenkt das Augenmerk auf Prozesse bzw. Abläufe in der Unternehmung.

Die horizontale Organisation von Verantwortlichkeiten hat einige Vorteile:

- Erhebliche Reduzierung der Durchlaufzeiten.
- Verbesserte Flexibilität.
- Weniger Schnittstellen, weniger Doppelarbeit, weniger Aufwand, weniger Bestände.
- Erhebliche Kostenreduzierung.
- Kleine Regelkreise durch die personifizierte Verantwortung für das gesamte Produkt (es müssen keine Hierarchien durchlaufen werden).
- Kunden- und Marktnähe.

Die Konzentration dieses Ansatzes liegt also in dem Erkennen und der Gestaltung von den wesentlichen Prozessen. Die Prozesse werden vom Beginn bis zum Ende durchdacht und optimiert. Durch das Erkennen der Zusammenhänge in der Ablaufstruktur lassen sich relativ leicht Optimierungsmöglichkeiten finden.

Die gerade vorgestellten Ansätze zielen auf die Umorganisation der Unternehmung von vertikalen Strukturen (Abteilungsorganisation) zu horizontalen Strukturen (Prozeßorientierung) ab. Dabei werden die Ablaufstrukturen innerhalb der Unternehmung zum wesentlichen Faktor für den Unternehmungserfolg. Die Arbeitsabläufe oder Prozesse lassen sich durch Workflows darstellen und abarbeiten. Daher wird die effiziente Beherrschung von Workflows zum Schlüsselfaktor für den Unternehmungserfolg.

### 1.1.3 Workflow Management Systeme

Eine exakte Definition des Begriffs „Workflow Management System“ ist nach [Jabl95] kaum möglich. [McB191] gibt die folgende informelle Definition an:

*Workflow management software is a proactive computer system which manages the flow of work among participants, according to a defined procedure consisting of a number of tasks. It coordinates user and system participants [...]. The coordination involves passing tasks from participant to participant in correct sequence [...].*

Damit handelt es sich bei einem Workflow Management System um aktive Software, welche die Abarbeitung von Workflows ermöglicht. Als Ziel von Workflow Management Systemen wird die „generelle Qualitätsverbesserung“ genannt. Diese wird durch fünf Faktoren erreicht:

- *Effektivität*: Ein Prozeß wird durch einen Workflow repräsentiert. Ein Workflow kann leicht an veränderte Situationen angepaßt werden.
- *Effizienz*: Durchlaufzeiten werden durch Parallelisierung und zielgerichtete Übermittlung von Informationen zwischen Produzent und Konsument verkürzt.
- *Informationstransparenz*: Information ist durch Workflow Management Systeme unabhängig vom Ort ihrer Erfassung und Speicherung unternehmensweit verfügbar.
- *Konsistenz*: Das Workflow Management System muß Konsistenz garantieren.
- *Innovation*: Workflow Management erfordert die Überarbeitung („Reengineering“) der für eine Unternehmung wesentlichen Prozesse. Dadurch wird Innovationskraft freigesetzt.

[Jabl95] beschreibt als „technische“ Anforderungen an ein Workflow Management System insbesondere drei Aspekte:

- *Integrierbarkeit bestehender Softwarekomponenten*. Gemeint sind damit nicht nur Anwendungsprogramme, sondern auch bereits existierende Datenbestände.
- *Skalierbarkeit*, da nicht abgesehen werden kann, wieviele Geschäftsprozesse welcher Komplexität auf dem Workflow Management System realisiert werden.
- *Anpaßbarkeit bzw. Erweiterbarkeit* des Workflow Management Systems an sich ständig verändernde Problemstellungen im Bezug auf die Gestaltung von Geschäftsprozessen (Anpaßbarkeit von Beschreibungselementen).

Eine wesentliche Anforderung fehlt in der Aufzählung von [Jab195]. Workflows sind in der Regel langandauernde Prozesse, so daß die Zuverlässigkeit des Workflow Management Systems als elementare Voraussetzung für das erfolgreiche Arbeiten mit dem System angesehen werden muß. Nur ein zuverlässiges Workflow Management System kann Konsistenz, Informationstransparenz und insbesondere Effektivität und Effizienz gewährleisten.

## 1.2 Das ConTract-Modell

Das klassische Transaktionsmodell hat sich für kurzlebige Transaktionen sehr gut bewährt. Der wohl größte Vorteil dieses Paradigmas ist in der vollständigen Trennung der eigentlichen Applikation von der Ausführungskontrolle zu sehen. D.h. die Applikation beinhaltet keinerlei Funktionalität im Hinblick auf Ausnahme- oder Fehlerbehandlung. Das Transaktionssystem garantiert in jedem Falle die ACID-Eigenschaften<sup>1</sup>.

Im Hinblick auf Workflows ist das klassische Transaktionsmodell nicht geeignet. Das Fehlen der Einflußnahme auf den Kontrollfluß - aus klassischer Sicht hat das die Vorteile beschert - wird bei langandauernden und viele Objekte berührenden Abläufen zum untragbaren Nachteil. Beispielsweise könnten Objekte über einen großen Zeitraum, evtl. Wochen oder Monate, gesperrt sein, obwohl die Bearbeitung dieses Objekts schon längst beendet ist. Die Parallelität des Systems wird dadurch möglicherweise stärker eingeschränkt, als notwendig. Eine weitere Einschränkung ergibt sich aus der „Atomicity-Eigenschaft“ des Transaktionsmodells. Im Fehlerfall werden alle Veränderungen dieser Transaktion zurückgesetzt (*Rollback*), so daß wieder der (konsistente) Zustand vorliegt, bevor die Transaktion begonnen hat. Bei langandauernden Prozessen ist diese Art der Fehlerbehandlung nicht sinnvoll. Beispielsweise kann bei einer CAD-Anwendung nicht eine sich über Monate hingezogene Konstruktion ungeschehen gemacht werden, nur weil vielleicht fast am Ende der Transaktion ein Rechner ausfällt. Der Verlust an bereits sinnvoll geleisteter Arbeit wäre nicht tragbar. Der logische Abschluß einer solchen Transaktion läge im erfolgreichen zu Ende bringen dieser Transaktion (*Rollforward*, vgl. hierzu [ReScWä92]). Diese Möglichkeit schließt das klassische Transaktionsmodell jedoch aus.

Das in [WäRe91] vorgestellte ConTract-Modell stellt einen Lösungsansatz für diese Problematik dar.

### 1.2.1 Was ist ein ConTract?

[WäRe91] definieren einen ConTract (Controlled Activities Controlled Transaction Compound) so:

---

<sup>1</sup> ACID-Eigenschaften: Atomicity, Consistency, Isolation, Durability.

*Ein ConTract ist eine konsistente und fehlertolerante Ausführung einer willkürlichen Sequenz von vordefinierten Aktionen (Steps) gemäß einer explizit definierten Kontrollfluß-Beschreibung (Skript).*

Das Ziel ist wiederum die Trennung der Applikation von der Ausführungskontrolle. Der Kontrollfluß wird in dem Skript beschrieben, während die Applikation durch die einzelnen Steps determiniert ist. Dadurch kann in den Steps auf die Fehlerbehandlung verzichtet werden. Sie wird durch das ConTract-System übernommen.

### **1.2.2 Programmiermodell**

Die Basisidee des ConTract-Modells liegt im Aufbau von großen Applikationen aus einzelnen kurzen ACID-Transaktionen. Zusätzlich wird ein applikationsunabhängiger Systemservice bereitgestellt, der die Kontrolle über die Transaktionen ausübt.

Steps sind die elementaren Aktionen, die im ConTract abgearbeitet werden. Jeder Step repräsentiert eine Basiseinheit einer Applikation. Der Kontrollfluß wird durch das Skript definiert. Die einzelnen Steps können in aus Programmiersprachen bekannte Kontrollelemente wie bedingte Verzweigung, Sequenzen oder Schleifen eingebettet werden. Zusätzlich stehen Kontrollelemente zur Parallelausführung von Steps zur Verfügung.

Steps können unter gewöhnlichem ACID-Transaktionsschutz ablaufen. Es ist jedoch auch möglich, mehrere Steps in eine Transaktion zu fassen. Zwischen die einzelnen Steps können Invarianten gelagert werden, die bei der Abarbeitung des ConTracts evaluiert werden. Abhängig vom jeweiligen Ergebnis wird der ConTract entsprechend des Kontrollflusses fortgesetzt. Durch die Invarianten kann zum einen die Fehlerbehandlung gesteuert werden, zum anderen können parallel ablaufende Steps synchronisiert werden. Die graphische Darstellung eines Skripts gleicht einem Petri-Netz.

Bei der Erstellung des Skripts kann jedem Step ein Kompensations-Step zugeordnet werden. Nach [GrRe93] bedeutet Kompensation im Zusammenhang mit ConTracts das „logische Ungeschehenmachen“ von Effekten einer abgeschlossenen (*committed*) Transaktion. Dies kann nur durch eine „neue“ Transaktion erfolgen, da die Effekte einer abgeschlossenen Transaktion sichtbar sind. Im ConTract-Modell hat der Skript-Programmierer die Möglichkeit zu definieren, durch welche Aktionen die Effekte eines Steps „ungeschehen“ gemacht werden können. Diese Aktionen werden in einen Kompensations-Step gefaßt. Grundsätzlich kann Kompensation auf Wunsch des Benutzers oder durch eine Fehlersituation initiiert werden.

### **1.2.3 Fazit**

Ein ConTract ist ein Programm, das einen Kontrollfluß besitzt, das persistente lokale Variablen besitzt, das auf gemeinsame Objekte über Synchronisationsmechanismen zugreift und das eine präzise Fehler-Semantik besitzt. Das Ziel des ConTract-Modells ist die Bereitstellung einer zuverlässigen, fehlertoleranten Ausführungsplattform für große, verteilte Applikationen. Dabei werden, ähnlich dem klassischen Transaktionsmodell, alle Aufgaben, welche die Parallelität, die Synchronisation und die zuverlässige Ausführung von Steps betreffen, von der Applikation getrennt und einem Systemdienst übertragen.

Das ConTract-Modell kann damit folgende Eigenschaften garantieren: Dauerhaftigkeit, Konsistenz, Synchronisation, Kooperation und Recovery.

Nähere Informationen bezüglich des ConTract-Modells finden sich in [WäRe91] und [ReScWä92].

### **1.3 Aufgabenstellung dieser Arbeit**

APRICOTS ist eine prototypische Implementierung des ConTract-Modells. Für diese Implementierung soll ein Werkzeug entworfen werden, mit welchem das System administriert werden kann. Insbesondere soll die Überwachung von Systemkomponenten ermöglicht werden. Die Systemkomponenten können auf verschiedene Rechnerknoten verteilt sein. Die Kommunikation zwischen den Komponenten erfolgt in APRICOTS auf der Basis von CORBA. Es müssen daher entsprechende Schnittstellen auf der Basis von CORBA entwickelt werden. Des weiteren soll untersucht werden, wie neue Komponenten in das System eingebracht werden können. Die Realisierung des administrativen Werkzeugs soll objektorientiert in der Programmiersprache Java durchgeführt werden.

## 2 APRICOTS

### 2.1 Was ist APRICOTS?

APRICOTS (A Prototype Implementation of a ConTract System) ist die prototypische Implementierung eines ConTract verarbeitenden Systems. APRICOTS wird derzeit am IPVR (Institut für Parallele und Verteilte Höchstleistungsrechner) Abteilung Anwendersoftware der Fakultät Informatik der Universität Stuttgart durch eine Projektgruppe neu implementiert.

Für den Benutzer soll APRICOTS die „Middleware“ für ein verteiltes System darstellen, auf der eine individuelle Applikation aufgebaut werden kann. Die Eigenschaften eines verteilten Systems werden in Kapitel 3 näher diskutiert. Als wesentliche Ziele von APRICOTS sind *Zuverlässigkeit* und *Fehlertoleranz* zu nennen.

### 2.2 Architektur

APRICOTS wird streng modular aufgebaut. Funktional zusammenhängende Teile werden in Komponenten gefaßt. Dies hat den Vorteil, daß das System auf einfache Art und Weise erweitert und gepflegt werden kann. Abb. 2.1 gibt einen Überblick der Architektur wieder. Die Aufgaben der einzelnen Komponenten werden in den folgenden Abschnitten beschrieben. Komponenten, die in der Abbildung mit einem Schatten versehen sind, können im Namensraum mehrfach vorhanden sein. Die Pfeile können interpretiert werden als: „Komponente A nimmt Dienste von Komponente B in Anspruch“. Die Pfeile, welche die Benutzeroberflächen betreffen, sind der Übersichtlichkeit halber gestrichelt dargestellt.

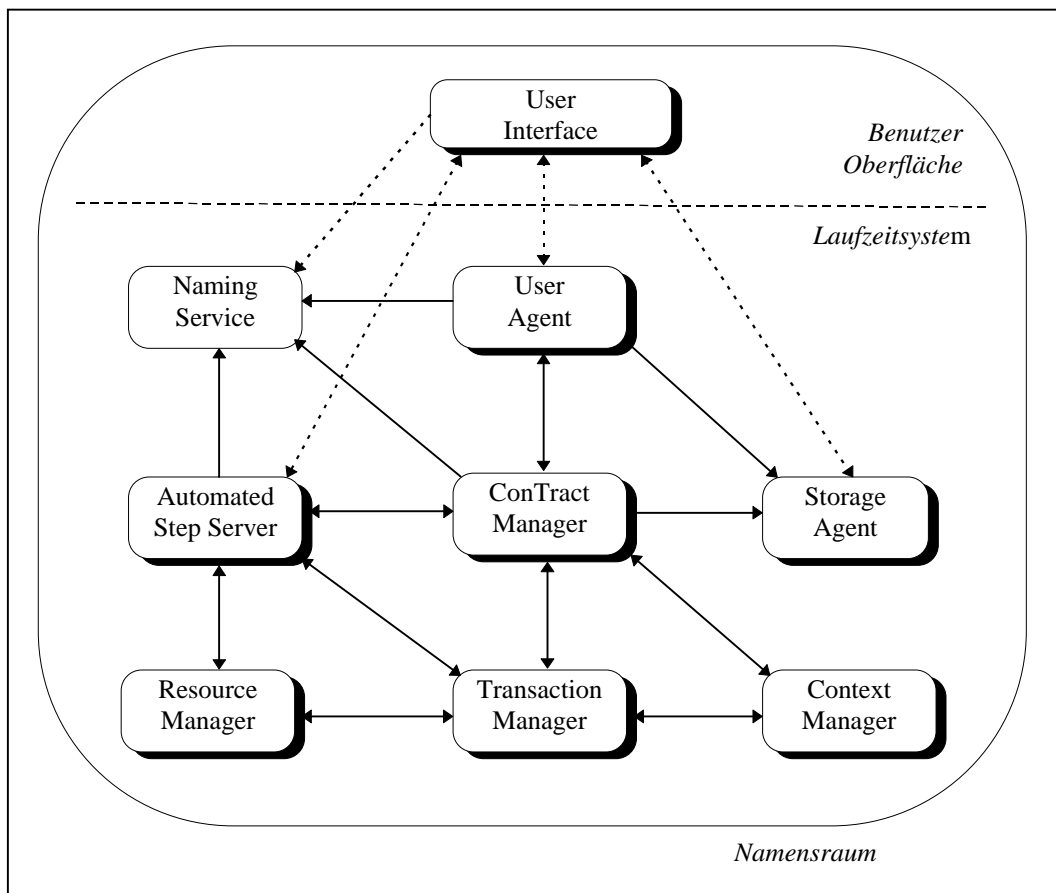


Abb. 2.1: Die Architektur von APRICOTS

### 2.2.1 ConTract Manager

Der ConTract Manager ist die zentrale Komponente des Systems. Er erzeugt bzw. löscht die sogenannten ConTract-Engines, welche die Abarbeitung einer Instanz eines Skripts übernehmen. Nach [Seif96] können in einem Namensraum mehrere ConTract Manager vorhanden sein, jedoch nicht auf einem Rechnerknoten.

### 2.2.2 Transaction Manager

Die einzelnen Steps eines Skripts laufen je nach Skriptdefinition unter Transaktionsschutz ab. Der Transaction Manager stellt die dafür benötigten Dienste zur Verfügung. Es wird ein 2-Phasen-Commit-Protokoll angewandt.

### 2.2.3 Context Manager

Steps können untereinander Daten austauschen. Diese Daten werden Kontextelemente genannt. Die Kontextelemente werden durch das Context Management während der gesamten Dauer einer ConTract-Instanz verfügbar gehalten. Kontextelemente ähneln daher globalen Variablen in herkömmlichen Programmiersprachen. Es werden alle Versionen eines Kontextelements gespeichert, um evtl. später einzelne Kompensationen korrekt durchführen zu können.

### 2.2.4 Automated Step Server

Automated Step Server sind diejenigen Komponenten, welche die Semantik der Applikation bestimmen. Sie müssen einzeln programmiert und in das System eingebracht werden. Automated Step Server implementieren Steps, welche keine Interaktion mit menschlichen Benutzern benötigen. Alle Entscheidungen können durch den Automated Step Server selbst erfolgen. Automated Step Server können Dienste der Resource Manager in Anspruch nehmen.

### 2.2.5 Resource Manager

Resource Manager verwalten Objekte wie Datenbanken, Drucker, Maschinen oder ähnliches. Sie werden nie direkt durch eine ConTract-Engine initiiert, sondern ausschließlich durch Step Server. Je nach Skriptdefinition nehmen sie am 2-Phasen-Commit-Protokoll der Transaktion teil.

### 2.2.6 User Agent

Jeder Benutzer des Systems wird durch genau einen Benutzer Agenten (User Agent) repräsentiert. Der Benutzer Agent dient als Schnittstelle zwischen dem System und der Benutzeroberfläche (vgl. unten). Per Konvention muß der Benutzer Agent ständig im System vorhanden sein, auch wenn der menschliche Benutzer gerade kein User-Interface gestartet hat. Die Verwendung eines Benutzer Agenten ist durch die Langlebigkeit einer ConTract-Instanz motiviert. Der Benutzer Agent ist zweigeteilt aufgebaut. Er beinhaltet den Monitor Agent und den Tasklist Agent.

#### 2.2.6.1 Monitor Agent

Der Monitor Agent erhält von jeder ConTract-Engine Informationen über den Fortschritt der Bearbeitung aller ConTract-Instanzen, die der Benutzer gestartet hat. Falls die Benutzeroberfläche „*Monitoring Interface*“ läuft, d.h. der Benutzer ist momentan am Rechner, dann werden diese Informationen an die Oberfläche weitergeleitet und dort

visualisiert. Ist die Oberfläche jedoch nicht gestartet, dann speichert der Monitor Agent alle Informationen, so daß der Benutzer sie zu einem späteren Zeitpunkt abrufen kann.

### **2.2.6.2 Tasklist Agent**

Unter einer *Tasklist* versteht man eine Liste von Aufgaben, die der menschliche Benutzer im Rahmen von Steps bearbeiten muß. Der Tasklist Agent ist für die Weiterleitung der Aufträge an die Benutzer-Oberfläche verantwortlich. Die Aufträge können dabei von jeder beliebigen ConTract-Engine kommen. In umgekehrter Richtung werden Resultate von bearbeiteten Aufträgen von der Oberfläche über den Tasklist Agent der entsprechenden ConTract-Engine mitgeteilt. Im Falle der Abwesenheit des menschlichen Benutzers übt der Tasklist Agent die Funktion einer Mailbox aus. Eingehende Aufträge werden gespeichert, bis der Benutzer zur Bearbeitung bereit ist.

### **2.2.7 Storage Agent**

Der Speicher Agent (Storage Agent) bietet eine abstrakte Schnittstelle zu stabilem Speicher an. Jede APRICOTS-Komponente, die stabilen Speicher benötigt, kann über den Speicher Agenten Daten ablegen. Beispielsweise speichert der Benutzer Agent im Falle der Abwesenheit der Benutzer-Oberfläche alle eingehenden Aufträge und Systeminformationen über den Speicher Agenten.

### **2.2.8 Der Naming Service**

APRICOTS definiert einen logischen Namensraum um Komponenten auffinden zu können. Außerdem werden physischen Adressen logische Namen zugewiesen, so daß ein hohes Maß an Lokationstransparenz (vgl. Kapitel 3) erreicht werden kann. Der Namensraum wird durch den Naming Service verwaltet.

### **2.2.9 User Interface**

Das User Interface setzt sich aus dem *Tasklist Interface*, *Monitoring Interface*, *Programming Interface* und der *System Administration* zusammen. In Abbildung 2.1 sind die einzelnen Benutzeroberflächen der Übersichtlichkeit halber nicht gezeigt. Das Tasklist Interface übernimmt im Rahmen einer Stepbearbeitung die Interaktion mit dem Benutzer. Beispielsweise können hier Formulare ausgefüllt, Anträge genehmigt etc. werden. Das Tasklist Interface kommuniziert mit einer ConTract-Engine über den Tasklist-Teil des Benutzer Agenten. Das Monitoring Interface visualisiert den gegenwärtigen Zustand einer ablaufenden ConTract-Instanz. Außerdem lassen sich neue ConTract-Instanzen starten. Die Kommunikation mit der ConTract-Engine wird hier über den Monitoring-Teil des Benutzer Agenten durchgeführt. Das Programming Interface bietet die Möglichkeit, visuell Skripte

zu erstellen und zu verwalten. Die System Administration ist Gegenstand dieser Arbeit und wird ab Kapitel 4 näher erläutert.

### 2.3 Der Namensraum von APRICOTS

In APRICOTS wird jedem Objekt ein logischer Name zugeordnet. Logische Namen abstrahieren von physischen Eigenschaften wie z.B. der Lokation oder dem Servernamen, der dieses Objekt implementiert. Dadurch wird ein hoher Grad an Lokationstransparenz (vgl. Kapitel 3) erreicht. Namen können hierarchisch aufgebaut werden, ähnlich einer Verzeichnisstruktur eines Dateisystems. Durch die Zuordnung von hierarchischen Namen an Objekte kann mit diesen eine bestimmte „Rolle“ assoziiert werden. Auf diese Art und Weise kann eine ConTract-Engine einen Server finden, der einen bestimmten Step implementiert. Die Abb. 2.2 zeigt den Namensraum von APRICOTS. In einer „Hierarchieebene“ neu hinzugekommene Namen werden jeweils kursiv dargestellt. Die in der Abbildung dargestellten Punkte deuten auf mögliche Unterbäume hin, die applikationsspezifisch sind.

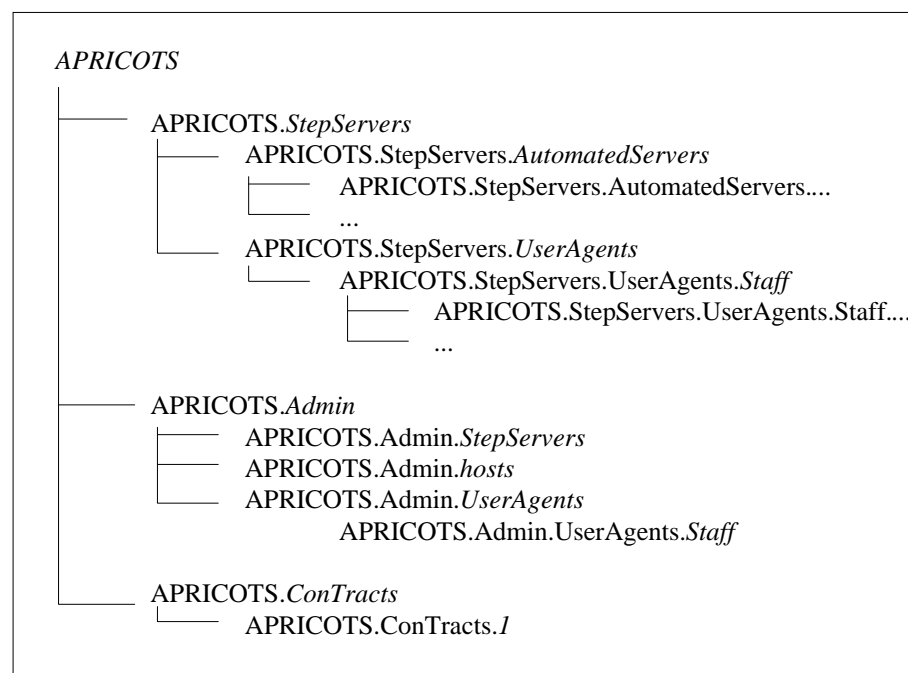


Abb. 2.2: Der Namensraum von APRICOTS

### 2.3.1 Hierarchie und Rollen

Jeder Step eines Skripts beschreibt eine bestimmte (Teil-)Aufgabe. Bei der Abarbeitung einer ConTract-Instanz muß die ConTract-Engine für jeden Step einen passenden Server finden, der diesen Step implementiert. Dies geschieht über die den Servern zugeordneten Rollen. Anhand der zu bearbeitenden Aufgabe kann die Engine einen Server aus dem Namensraum auswählen, dem eine solche Rolle zugeordnet ist.

*APRICOTS* stellt die Wurzel des Namensraums von APRICOTS dar. Unterhalb von *APRICOTS* existieren drei verschiedene Namen: *StepServers*, *Admin*, *ConTracts*.

#### *APRICOTS.ConTracts*

Unterhalb des Asts *APRICOTS.ConTracts* finden sich Informationen über alle ConTract-Instanzen, die gerade abgearbeitet werden. Dort finden sich u.a. Referenzen von beteiligten Komponenten, wie zum Beispiel des Monitor Agenten, der ConTract-Engine oder des Context Managements.

#### *APRICOTS.Admin*

Die eigentliche Applikation wird durch die in das System eingebrachten Benutzer, Step Server und die am System teilnehmenden Rechnerknoten bestimmt. Diese Information ist unterhalb des Asts *APRICOTS.Admin* im Namensraum von APRICOTS gespeichert. Der *Admin*-Zweig beginnt mit der Unterteilung in *StepServers*, *hosts* und *UserAgents*. Unterhalb dieser Namen existieren keine weiteren Hierarchieebenen. *APRICOTS.Admin.StepServers* enthält für jeden am System teilnehmenden Step Server einen Eintrag. Jeder Step Server muß dazu einen in diesem Unterbaum eindeutigen Namen besitzen. Der Eintrag eines Step Servers enthält neben dem logischen Namen insbesondere eine Referenz des Serverobjekts.

Analoges gilt für den Unterbaum *APRICOTS.Admin.UserAgents*. Hier werden alle dem System bekannten Benutzer bzw. Benutzerinnen gespeichert. Jeder Benutzer benötigt einen unter *APRICOTS.Admin.UserAgents* eindeutigen Benutzernamen. Unterhalb des Benutzernamens finden sich dann die Referenzen für den Tasklist und den Monitor Agenten des Benutzers.

#### *APRICOTS.StepServers*

Dieser Ast wird verwendet, um die Rollen von Step Servern und Benutzern abzubilden. Die Hierarchiestruktur unterhalb von *APRICOTS.StepServers* ist applikationsabhängig und kann beliebig entworfen werden. Eine Benutzerin oder ein Step Server kann hier unter verschiedenen Ästen mehrfach eingetragen sein, je nach Aufgabenumfang der Mitarbeiterin bzw. des Step Servers. Der folgende Abschnitt erläutert die Verwendung der Rollen anhand eines Beispiels.

### 2.3.2 Modellierungsmöglichkeiten von Organisationsstrukturen

APRICOTS ermöglicht dem Benutzer durch die freie Bindung von hierarchisch aufgebauten Namen an Objekte die individuelle Organisation von Funktionsbereichen einer Unternehmung. Das folgende Beispiel soll die Gestaltung von Aufgabenbereichen deutlich machen.

Die Abbildung 2.3 zeigt einen Ausschnitt aus einer hierarchischen Aufgabenstruktur, wie sie in einer fiktiven Unternehmung vorkommen könnte. Es werden der Übersichtlichkeit halber nur Mitarbeiter dargestellt. Die Rollen von Step Servern können analog modelliert werden. Ausgefüllte Punkte bezeichnen Kontextknoten, d.h. die Funktionalität eines Unternehmungsbereichs. Nicht ausgefüllte Punkte stellen einzelne Objekte dar.

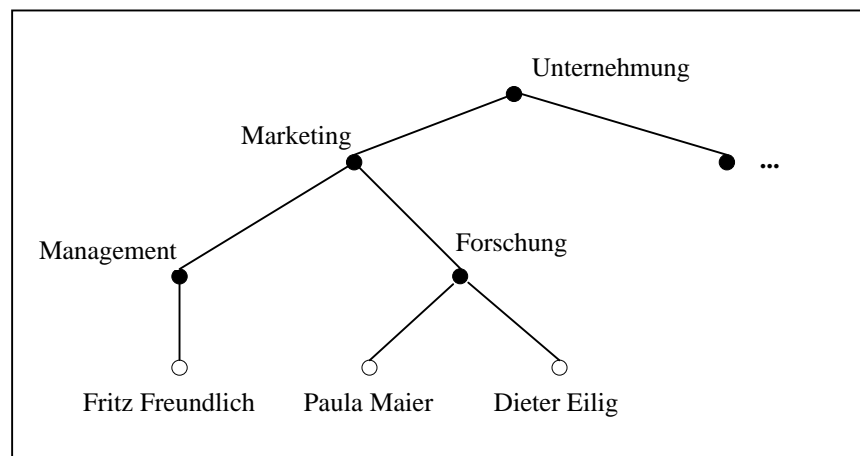


Abb. 2.3: Hierarchische Funktionsbereichsstruktur einer fiktiven Unternehmung

Aus der Abbildung läßt sich ableiten, daß Fritz Freundlich Manager der Abteilung Marketing ist. Paula Maier und Dieter Eilig sind im Funktionsbereich Forschung der Abteilung Marketing tätig. Beide können die gleichen Aufgaben ausführen. Abbildung 2.4 zeigt die Abbildung dieser Struktur auf den Namensraum von APRICOTS. Kursiv dargestellte Namen bezeichnen Objekte, während „normal“ dargestellte Namen Kontextknoten sind bzw. Rollen repräsentieren.

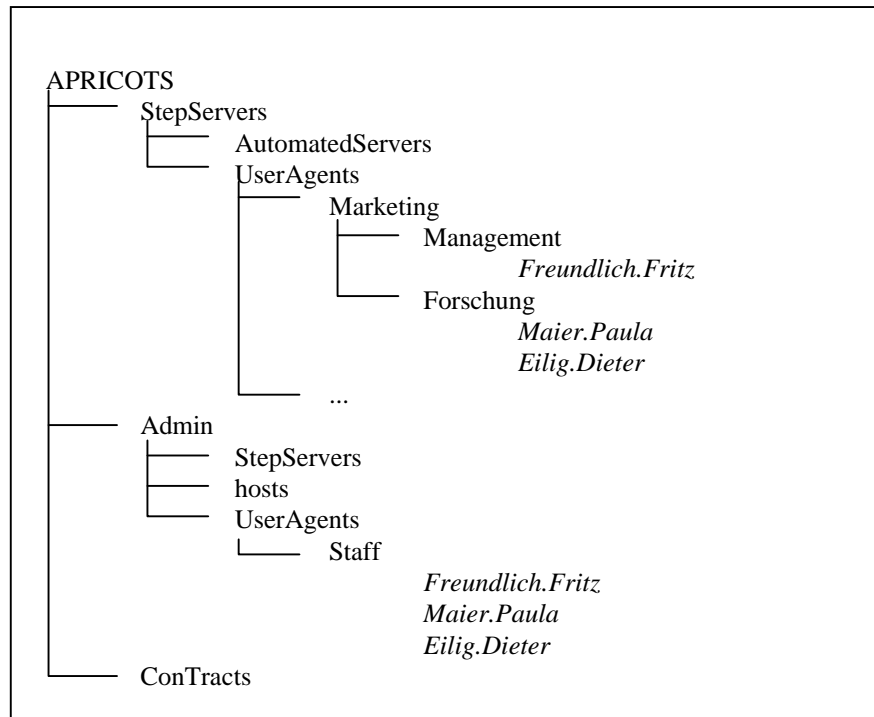


Abb. 2.4: Abbildung auf den Namensraum von APRICOTS

Soll bei der Abarbeitung einer ConTract-Instanz beispielsweise ein Mitarbeiter der Abteilung Marketing-Forschung eine Aufgabe erledigen, so kann die ConTract-Engine im Namensraum unterhalb des Asts „*APRICOTS.StepServers.UserAgents.Marketing.Forschung*“ Referenzen von Benutzer Agenten finden, die diese Aufgabe ausführen können.

Die Modellierung von Rollen im Namensraum von APRICOTS stellt also die hierarchische Strukturierung der in einer Unternehmung anfallenden Aufgaben dar. Die Hierarchie entsteht durch Abstraktion von konkreten Aufgaben. Dieser Ansatz läßt die Organisationsform der Unternehmung außer Betracht und ist damit für jede Art von Unternehmungsorganisation einsetzbar.

## 2.4 Realisierung

### 2.4.1 Implementierungsgrundsätze

Bei der Implementierung von einzelnen Komponenten sollen insbesondere die zwei Anforderungen Zuverlässigkeit und Fehlertoleranz berücksichtigt werden.

Graphische Benutzeroberflächen werden in der derzeitigen Implementierung vollständig in

der Programmiersprache „Java“ implementiert, während die anderen Komponenten in C++ realisiert werden. Als Java-Entwicklungsumgebung steht Java Workshop (SunSoft) zur Verfügung.

### 2.4.2 Kommunikation

Die APRICOTS-Komponenten können in verschiedenen Adreßräumen lokalisiert sein. Die Kommunikation zwischen APRICOTS-Komponenten basiert auf dem CORBA 2.0 Standard der Object Management Group (OMG). Als Implementierung dieses Standards wird Orbix der Firma IONA Technologies Ltd. verwandt. Näheres zu CORBA und IONA/Orbix findet sich in Kapitel 3.

## 2.5 Beurteilung

In diesem Abschnitt soll überprüft werden, ob APRICOTS ein Workflow Management System ist. Dazu werden die Kriterien aus Abschnitt 1.1 herangezogen.

- *Integrierbarkeit:* Bestehende Softwarekomponenten können in APRICOTS in Resource Managern bzw. Step Servern integriert werden, so daß diese an der Abarbeitung von Workflows teilnehmen können. Bereits existierende Datenbestände lassen sich auf die gleiche Weise integrieren. Für die Interaktion mit Benutzern stellt der elektronische Eingangskorb (Tasklist Interface) gängige Applikationen zur Verfügung.<sup>1</sup> Damit ist die Integrierbarkeit gewährleistet.
- *Skalierbarkeit:* APRICOTS-Komponenten können auf beliebig vielen Rechnerknoten, die durch ein Verbindungsnetzwerk gekoppelt sind, verteilt werden. Durch die Rollenzuordnung können mehrere Step Server einen bestimmten Step implementieren. Es können beliebig viele Benutzer am System teilnehmen. Eine Einschränkung der Skalierbarkeit ist in der derzeitigen Implementierung jedoch im Naming Service zu sehen. Der Naming Service existiert nur einfach und könnte dadurch zum „Hotspot“ werden.
- *Anpaßbarkeit bzw. Erweiterbarkeit:* Skripte können in APRICOTS durch Kontrollelemente, wie sie aus gewöhnlichen Programmiersprachen bekannt sind und zusätzlich durch Parallelkonstrukte mit Synchronisation modelliert werden. Steps können beliebig in Transaktionen gefaßt werden. Damit lassen sich sehr viele denkbare Kontrollflußstrukturen modellieren. Um alle denkbaren Strukturen modellieren zu können, müßten beliebige Sprünge möglich sein. Das ConTract-Modell sieht jedoch beliebige Sprünge nicht vor. In meinen Augen ist dies keine echte Einschränkung.

---

<sup>1</sup> näheres in [Bay96]

- *Zuverlässigkeit*: APRICOTS hat als Hauptziele der Implementierung Zuverlässigkeit und Fehlertoleranz. Dadurch wird ein hohes Maß an Verfügbarkeit des Systems erreicht. Durch die Verwendung des Naming Services der Firma IONA entsteht allerdings ein „*single-point-of-failure*“. Die derzeitig verwendete Version 1.0 (beta) des Naming Services ist nicht als zuverlässig und hochverfügbar einzustufen.<sup>1</sup>

APRICOTS kann demnach als ein Workflow Management System betrachtet werden, unter der Voraussetzung, daß eine zuverlässige und hochverfügbare Version des Naming Services eingesetzt wird.

---

<sup>1</sup> vgl. Abschnitt 3.5

### 3 CORBA

APRICOTS bildet eine Plattform für den Aufbau eines verteilten Systems. Die Kommunikation zwischen den APRICOTS-Komponenten basiert auf dem CORBA-Standard. Einzelne Komponenten werden gemäß des CORBA-Standards als „Objekte“ angesehen. Die genauere Betrachtung von CORBA ist daher an dieser Stelle notwendig. Zunächst wird der Begriff „verteiltes Objekt Management“ näher erläutert.

#### 3.1 Verteiltes Objekt Management

[ÖzDaVa94] definieren den Begriff „Verteiltes Objekt Management“ wie folgt:

*Distributed object management has the [...] objective [...]: transparent management of „objects“ that are distributed across a number of sites. Thus users have an integrated, „single image“ view of the objectbase while it is physically distributed among a number of sites.*

Diese Definition beinhaltet zwei zentrale Punkte. Erstens die Verteilung von Objekten auf verschiedene Knoten. Der Begriff „Objekt“ wird dabei nicht näher spezifiziert. Unter einem Knoten versteht man CPU und Speicher. Die Knoten sind durch ein Netzwerk miteinander verbunden. Es wird angenommen, daß sich die Knoten keinen gemeinsamen Speicher (*Shared Memory*) teilen.

Als zweiter zentraler Punkt dieser Definition ist die Transparenz anzusehen. Der Benutzer darf von der Verteilung der Objekte auf mehrere Knoten nichts merken. Für ihn erscheint das verteilte System wie ein einzelner, gewöhnlicher Rechner. Die angesprochene Transparenz setzt sich nach [ÖzDaVa94] aus folgenden Teilen zusammen:

- *Netzwerk- bzw. Verteilungstransparenz*: Die Existenz eines Netzwerks und die Verteilung von Daten bleiben dem Benutzer verborgen („*network / distribution transparency*“). Diese Art von Transparenz kann untergliedert werden in *Zugriffstransparenz* und *Lokationstransparenz*. *Zugriffstransparenz* bedeutet, daß aus Benutzersicht kein Unterschied besteht zwischen dem Zugriff auf ein lokales und ein entferntes Objekt. *Lokationstransparenz* bedeutet, daß dem Benutzer verborgen bleibt, auf welchem Knoten ein Objekt physikalisch lokalisiert ist, bzw. daß es gleichgültig ist, auf welchem Knoten eine Aufgabe bearbeitet wird.
- *Replikationstransparenz*: Die eventuelle Existenz von Kopien logischer Daten bleibt dem Benutzer verborgen („*replication transparency*“). Die Datenkonsistenz muß vom System gewährleistet werden.

- *Fragmentierungstransparenz*: Der Benutzer wird vor der Notwendigkeit des Partitionierens von Datenobjekten bewahrt. Diese Aufgabe übernimmt vollständig das verteilte System („*fragmentation transparency*“).

Transparenz kann in diesem Zusammenhang gesehen werden als die Erweiterung des Konzepts der *Datenunabhängigkeit* im Hinblick auf verteilte Systeme.

### 3.2 Spezifikation und Arbeitsweise von CORBA

CORBA (Common Object Request Broker Architecture) wurde von der „Object Management Group“ (OMG) spezifiziert. OMG ist eine internationale, 1989 gegründete Organisation von verschiedenen System-Händlern, Software-Herstellern und Benutzern. Das Selbstverständnis der OMG liegt in der Unterstützung von objekt-orientierter Technologie im Bereich der Software-Entwicklung. Es werden Richtlinien und Spezifikationen erstellt, um ein umfassendes Rahmenwerk für die Applikationsentwicklung zu schaffen. Die verfolgten Hauptziele der OMG sind nach [OMG95] Wiederverwendbarkeit, Portabilität und Interoperabilität von objekt-orientierter Software in verteilten, heterogenen Umgebungen.

CORBA ist die Definition eines Standards zur Kommunikation zwischen Objekten in einem verteilten System. CORBA versteht unter einem Objekt eine identifizierbare, gekapselte Entität, die einen oder mehrere Dienste anbietet. Diese Dienste können durch Clienten in Anspruch genommen werden. Dazu muß jedes Objekt eine Schnittstelle definieren. CORBA spezifiziert zu diesem Zweck die implementierungsunabhängige Sprache „*OMG Interface Definition Language*“ (OMG IDL). Die Realisierung eines Objekts (das Server-Objekt) muß diese IDL-Schnittstelle implementieren. Es ist möglich, daß ein Server-Objekt im Rahmen einer Dienstausführung zum Clienten eines anderen Server-Objekts wird. Für die Implementierung der Clienten und Server-Objekte können jeweils verschiedene Programmiersprachen gewählt werden. Es wird ein Konvertierungsmechanismus verwendet, der die IDL-Konstrukte auf die Zielsprache abbildet („*Mapping*“). Die CORBA-Spezifikation standardisiert das Mapping von IDL auf C, C++, Ada und Smalltalk. Neben den statischen IDL-Schnittstellen besteht die Möglichkeit zur dynamischen Verwendung von Server-Objekten, deren Schnittstelle erst zur Laufzeit bekannt wird. Dazu wird das sogenannte „*Interface Repository*“ verwendet.

Ein Compiler erzeugt aus den Schnittstellendefinitionen „Stubs“. Die Stubs sind für die Anpassung an das zugrundeliegende Kommunikationsprotokoll und für das Packen und Entpacken von Nachrichten (*Marshalling*, *Unmarshalling*) zuständig. Als Kommunikationsprotokoll kann TCP/IP oder OSF/DCE<sup>1</sup> verwendet werden.

---

<sup>1</sup> DCE: Distributed Computing Environment

Der *Object Request Broker* (ORB) ist der Kern von CORBA. Der ORB ermöglicht Clienten die transparente Inanspruchnahme von Diensten von Server-Objekten. Dabei ist der ORB verantwortlich für das Auffinden des Server-Objekts, für das Vorbereiten des Server-Objekts auf die Dienstinanspruchnahme und für das Übermitteln der Daten des Dienstaufrufs. Will ein Client Dienste eines Servers in Anspruch nehmen, setzt er einen Aufruf an den ORB ab. Man nennt diesen Vorgang das „*Binden*“ an ein Server-Objekt. Als Effekt des Bindens eines Clienten an ein Server-Objekt wird im Client ein sogenanntes „*Proxy-Objekt*“ angelegt. Der Rückgabewert der *\_bind()*-Operation ist eine Referenz auf das Proxy-Objekt. Zwischen Client und Server-Objekt ist nun eine Verbindung hergestellt. Der Client kann jetzt Aufrufe an das Server-Objekt absetzen. Die Aufrufe werden nicht über den ORB, sondern direkt an das Server-Objekt geleitet. Durch die Verwendung des ORBs werden dem Clienten physische Eigenschaften des Server-Objekts verborgen. Die Abb. 3.1 verdeutlicht die Struktur des ORB-Interfaces nach [OMG95].

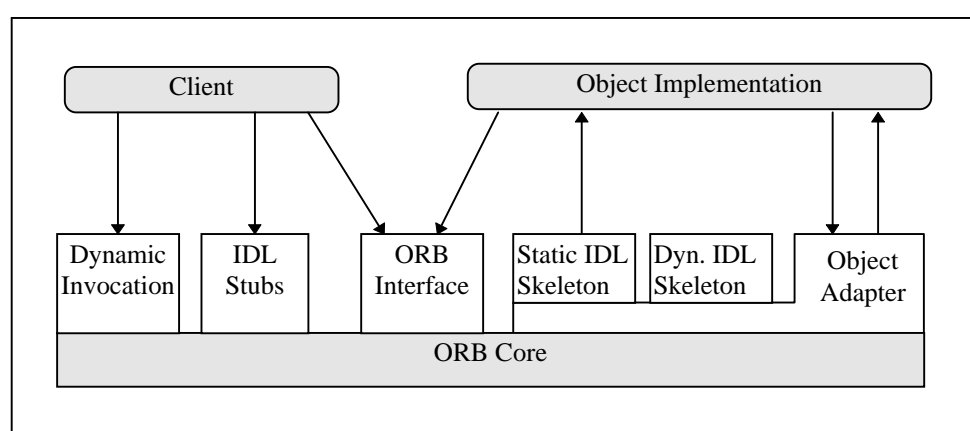


Abb. 3.1: Die Struktur des ORB-Interfaces nach [OMG95]

Server-Objekte werden durch den Namen des Servers und den Namen des Objekts („*marker*“) identifiziert. Für das Auffinden des Server-Objekts ist der ORB zuständig. Dazu wird der „*Locator*“ verwendet. Einem ORB können andere ORBs bekannt gemacht werden, so daß im Falle des Nichtauffindens eines Server-Objekts der ORB die Anfrage an ihm bekannte ORBs weiterleiten kann. Wie oben erwähnt, ist eine weitere Aufgabe des ORBs das Vorbereiten eines Server-Objekts auf die Inanspruchnahme durch einen Clienten. Das „*Vorbereiten*“ bedeutet im wesentlichen das Anstarten des Betriebssystemprozesses, in welchem das Server-Objekt läuft. Die Prozesse von Server-Objekten, welche längere Zeit nicht verwendet werden, werden gestoppt, damit keine System-Ressourcen unnötig verbraucht werden. Um ein Server-Objekt anstarten zu können, muß dem ORB die Zuordnung zwischen dem Servernamen und den zu diesem Server gehörenden ausführbaren Programmdateien bekannt sein. Diese Zuordnung wird im „*Implementation Repository*“ verwaltet.

Grundsätzlich gibt es beim Aufruf von Methoden in einem Server-Objekt verschiedene Kommunikationsmodi. Beispielsweise gibt es beim klassischen RPC<sup>1</sup> sowohl den synchronen, als auch den asynchronen Modus. CORBA stellt drei verschiedene Kommunikationsmodi zur Verfügung.

- *Synchron*: der Client wird blockiert, bis das Server-Objekt den Auftrag bearbeitet hat. Dieser Modus entspricht im wesentlichen dem synchronen RPC.
- *Verzögert synchron* („*deferred synchron*“): Der Client wird nicht blockiert. Intern wird die Kommunikation jedoch synchron abgewickelt. Der Client kann jederzeit den Eingang der Antwort nachfragen.
- *Einweg* („*oneway*“): Methoden können in der IDL-Schnittstellendefinition als „*oneway*“ markiert werden. Diese Methoden können keine Rückgabewerte haben. Der Client wird nicht blockiert.

### 3.3 CORBA-Services

Die OMG hat für unterschiedliche Aufgaben bereits Dienste standardisiert. Hier sollen einige dieser Dienste kurz vorgestellt werden.

#### 3.3.1 Naming Service

Der Naming Service ermöglicht die Bindung eines logischen Namens an ein Server-Objekt. Dadurch wird von physischen Eigenschaften (z.B. Servername und Marker) eines Server-Objekts abstrahiert. Clienten brauchen nur den logischen Namen zu wissen, um sich an ein Server-Objekt binden zu können. Die Namen können hierarchisch aufgebaut werden, ähnlich einer Verzeichnisstruktur eines Dateisystems.

#### 3.3.2 OTS als Service für die verteilte Transaktionsverwaltung

OTS (*Object Transaction Service*) ist ein Dienst zur Verwaltung von verteilten Transaktionen. Genaue Informationen über OTS finden sich in [OMG94].

---

<sup>1</sup> RPC: Remote Procedure Call

### 3.3.3 Persistent Object Service

Mit Hilfe des *Persistent Object Services* (POS) sollen CORBA-Objekte persistent gespeichert werden können. Bislang existieren keine brauchbaren Implementierungen dieses Services.

### 3.4 Die Eignung von CORBA für die Realisierung eines verteilten, objektorientierten Systems

Die Eignung von CORBA für die Realisierung eines verteilten, objektorientierten Systems soll anhand der oben dargestellten Anforderungen überprüft werden.

- *Verteilung von Objekten auf verschiedenen Knoten*: Jedes Server-Objekt kann auf einem beliebigen Rechnerknoten lokalisiert sein. Die einzige Bedingung ist die Existenz eines ORBs auf diesem Rechnerknoten.
- *Zugriffstransparenz*: Der Aufruf von Methoden von lokalen und entfernten Objekten im Clienten unterscheidet sich nicht.
- *Lokationstransparenz*: Der Client „sieht“ nicht, wo das Objekt lokalisiert ist. Der *Locator* ermöglicht das Binden eines Clients an ein Server-Objekt, von dem nur der Servername und der Objektname bekannt ist.
- *Replikationstransparenz*: die Replikationstransparenz ist gewährleistet, da der CORBA-Standard bisher keine Replikate vorsieht.
- *Fragmentierungstransparenz*: Die Fragmentierungstransparenz wird bereits durch TCP gewährleistet.

CORBA bietet noch einige weitere Möglichkeiten, die in der obigen Definition nicht explizit von einem verteilten, objektorientierten System gefordert werden:

- Server-Objekte, die längere Zeit nicht verwendet werden, werden gestoppt, so daß sie keine Systemressourcen verbrauchen.
- Nachdem die Verbindung zwischen Client und Server hergestellt ist, wird keine (zentrale) Instanz mehr benötigt für die Kommunikation.
- *Interoperabilität*: Der CORBA-Standard spezifiziert Sprachanpassungen (*Mapping*) für die Programmiersprachen C, C++, Ada und Smalltalk. Außerdem wird das ORB-Interface spezifiziert, so daß in realen Anwendungen ORBs verschiedener Hersteller zusammenwirken können.

- Schnittstellendefinitionen von Server-Objekten können zur Laufzeit über das *Interface Repository* erfragt werden.
- Schnittstellendefinitionen von Server-Objekten können objektorientiert entworfen werden.

Zusammenfassend kann gesagt werden, daß sich CORBA sehr gut für die Implementierung eines verteilten, objektorientierten Systems eignet.

### **3.5 IONA Orbix als CORBA-Implementierung in APRICOTS**

Orbix ist eine vollständige Implementierung des CORBA 2.0-Standards. Orbix wird von der Firma IONA Technologies Ltd. angeboten und kommt in APRICOTS als Kommunikationsplattform zum Einsatz. Orbix bietet Sprachanpassungen für die Programmiersprachen C, C++, Ada, Smalltalk und Java. Die Sprachanpassung für Java (*OrbixWeb*) konnte zu Beginn dieser Arbeit nur für die Implementierung von Clienten verwendet werden. Orbix-Server konnten in Java nicht implementiert werden. Die inzwischen verfügbare Version 2.0 von OrbixWeb behebt jedoch diese Einschränkung. Durch die umfangreiche Sprachanpassung von Orbix lassen sich Clienten und Server in einer für den jeweiligen Zweck gut geeigneten Programmiersprache implementieren. Dies ist als Vorteil einzustufen. Ein weiterer Vorteil von Orbix liegt in der Verfügbarkeit von Orbix für verschiedene Plattformen (z.B. UNIX, Windows NT). Dadurch wird die Portierbarkeit unterstützt. Die Verfügbarkeit von CORBA-Services wie z.B. der *Naming Service* runden das Bild von Orbix ab. Nähere Informationen bezüglich Orbix finden sich in [IONA95a] und [IONA95b].

#### **3.5.1 Die Verwendung des Naming Services**

Der Namensraum von APRICOTS wird durch den Naming Service der Firma IONA Technologies Ltd. implementiert. Die vorliegende Version 1.0 (beta) des Orbix Naming Services ist als relativ unzuverlässig einzustufen. Als stabiler Speicher wird ein „einfaches“ Dateisystem ohne jegliche Fehlertoleranz verwendet. Da im Falle des Ausfallens des Naming Services das ganze APRICOTS-System früher oder später zum Stillstand kommt, muß der Naming Service als „*single-point-of-failure*“ eingestuft werden. Nähere Informationen zum Orbix Naming Service sind in [IONA96] zu finden.

#### **3.5.2 Die Verwendung von OTS**

Zur Verwaltung von Transaktionen verwendet der ConTract-Manager einen Transaktionsverwalter auf der Basis von OTS. Ein im Hinblick auf APRICOTS großer

Vorteil von OTS liegt in der Möglichkeit von geschachtelten Transaktionen und in der Existenz einer IDL-Schnittstelle. Näheres über die Verwendung von OTS in APRICOTS findet sich in [Seif96].

## 4 Das administrative Werkzeug im Detail

### 4.1 Aufgaben

#### 4.1.1 Der System Monitor

APRICOTS implementiert eine Umgebung, in der ConTract-Skripte zuverlässig abgearbeitet werden können. Der Begriff *Zuverlässigkeit* ist nach [GrRe93] definiert als:

*Zuverlässigkeit*: Zeit zwischen dem Initialzustand eines Moduls und dem darauf folgenden Fehlerfall. In der Statistik spricht man von „*mean-time-to-failure*“ (MTTF). Die Zeitdauer, die im Fehlerfall die Dienstauführung unterbricht, wird „*mean-time-to-repair*“ (MTTR) genannt. Dies entspricht dem Zeitbedarf für die Erkennung und die Beseitigung des Defekts.

Die Zuverlässigkeit eines Gesamtsystems wird durch die Verfügbarkeit seiner einzelnen Komponenten determiniert. Der Begriff *Verfügbarkeit* ist definiert als:

*Verfügbarkeit*: die Zeit, in der ein Dienst korrekt ausgeführt wird, im Verhältnis zur gesamten dafür benötigten Zeit. Statistisch heißt das:

$$\text{Verfügbarkeit} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Aus dieser Beziehung läßt sich ableiten, daß die Verfügbarkeit eines Moduls auf zwei Arten erhöht werden kann. Zum einen kann die MTTF erhöht werden. Man spricht in diesem Fall von „Fehler-Vermeidung“. [Somm92] stellt zu diesem Thema fest, daß ein Modul maximal im Sinne seiner Spezifikation fehlerfrei sein kann. Es ist jedoch denkbar, daß die Spezifikation Fehler enthält oder daß potentiell mögliche Situationen in der Spezifikation vergessen wurden. Dies hat zur Folge, daß auch das Modul nicht fehlerfrei sein kann. Den Aufwandsverlauf, ein Modul „fehlerfreier“ zu machen, beschreibt [Somm92] als exponentiell wachsend. Bei sehr großen und komplexen Systemen ist dieser Ansatz daher nur begrenzt möglich bzw. sinnvoll.

Die zweite Möglichkeit, die Verfügbarkeit eines Moduls zu erhöhen, liegt in der Verringerung der MTTR. Im Grenzfall ( $\text{MTTR} \rightarrow 0$ ) strebt die Verfügbarkeit gegen 100%, unabhängig davon, wie gut oder schlecht die Zuverlässigkeit des Moduls ist. Das Problem dieses Ansatzes liegt in der Tatsache, daß ein Fehler erst beseitigt werden kann, nachdem er erkannt worden ist. D.h. je schneller ein Fehler erkannt wird, desto schneller kann reagiert werden und damit die Verfügbarkeit des Moduls erhöht werden.

Die Verfügbarkeit von APRICOTS-Komponenten läßt sich also u.a. durch die Verringerung der MTTR erhöhen. Mit der steigenden Verfügbarkeit von einzelnen

APRICOTS-Komponenten steigt auch die Verfügbarkeit des Gesamtsystems, welche ihrerseits Voraussetzung für die zuverlässige Abarbeitung von ConTract-Skripten ist. Das Ziel, die MTTR von APRICOTS-Komponenten gering zu halten, impliziert, daß eine Möglichkeit gegeben ist, ausgefallene oder potentiell gefährdete Komponenten möglichst schnell und auf einfache Art und Weise identifizieren zu können. Dies bedeutet insbesondere die automatisierte Zustandsermittlung, die mühsames und langwieriges Arbeiten an der Kommandozeile überflüssig macht. Die erste Hauptaufgabe des administrativen Werkzeugs ist daher die Bereitstellung eines *System Monitors*.

Um sich einen umfassenden Überblick über den Systemzustand verschaffen zu können, muß der System Monitor folgende Fragen beantworten können:

- Welche Rechnerknoten sind ausgefallen?
- Welche APRICOTS-Komponenten des Gesamtsystems sind ausgefallen?
- Welche Komponenten sind auf Rechnerknoten X lokalisiert?
- Welche Komponenten, die auf Rechnerknoten X lokalisiert sind, sind ausgefallen?

Für die Beantwortung dieser Fragen empfiehlt sich die Trennung des System Monitors in eine „knotenorientierte“ und eine „komponentenorientierte“ Sichtweise. Der knotenorientierte Ansatz überwacht einen Rechnerknoten mit allen auf ihm laufenden Komponenten. Er ist damit in der Lage, die beiden letzten Fragen zu beantworten. Der komponentenorientierte Ansatz fokussiert auf einzelne Komponenten unabhängig von dem Rechnerknoten, auf dem sie lokalisiert sind. Er beantwortet somit die ersten beiden Fragen.

#### 4.1.2 Die System Administration

Ein ConTract besteht aus verschiedenen einzelnen Steps. Steps sind Arbeitsschritte, die durch die sogenannten Step Server oder durch Mitarbeiter ausgeführt werden. Die Semantik des Gesamtsystems (die eigentliche Applikation) wird durch die Funktionalität der einzelnen Step Server bzw. durch die Fähigkeiten der Benutzer determiniert. Es lassen sich alle denkbaren Arten von Applikationen unter APRICOTS realisieren. Beispielsweise können relativ einfache Reisebuchungssysteme genauso wie komplexe Produktionsprozesse oder langandauernde Büroabläufe implementiert werden. Um die jeweilige Applikation realisieren zu können, muß es eine Möglichkeit geben, die individuellen Komponenten in das System einbringen zu können. Dies ist die Hauptaufgabe der *System Administration*.

Gleichzeitig bildet die System Administration die Basis für den System Monitor. Der System Monitor soll Teil eines Regelkreises sein, mit dem das APRICOTS-System hochverfügbar gehalten werden kann (vgl. oben). Das Prinzip der Regelung basiert auf Vergleichen zwischen Soll- und Ist-Zuständen und dem ergebnisabhängigen Eingreifen. Der Soll-Zustand des Systems wird durch die eingebrachten individuellen Komponenten bestimmt. Da die System Administration für das Einbringen von Komponenten verantwortlich ist, kennt sie den Soll-Zustand des Systems. Anhand dieser Information ermittelt der System Monitor den tatsächlich vorliegenden Zustand.

Zusammenfassend lassen sich die Aufgaben des administrativen Werkzeugs beschreiben als:

- *System Monitor*: Der Anwender soll in der Lage sein, sich jederzeit über den gegenwärtigen Systemzustand informieren zu können. Die relevanten Daten sollen in einer graphischen Oberfläche dargestellt werden.
- *System Administration*: Neue Komponenten sollen anhand der graphischen Oberfläche in das System eingebracht bzw. bestehende Komponenten entfernt werden können.

## **4.2 Anforderungen an die graphische Benutzungsoberfläche**

### **4.2.1 Allgemeine Anforderungen an graphische Benutzungsoberflächen**

[Shnei92] gibt für das Gestalten von Mensch-Maschine-Dialogen acht „goldene“ Regeln an:

- *Bemühung um Konsistenz*: Die Konsistenz bezieht sich auf Abläufe, Terminologie, Aufbau von Dialogen.
- *Ermöglichen von Shortcuts für erfahrene Benutzer(innen)*: Erfahrene Benutzer sollten die Möglichkeit erhalten, schnell bzw. direkt die gewünschte Funktion ausführen zu können.
- *Informative Rückmeldungen des Systems*: Je nach Häufigkeit des Auftretens sollte die Rückmeldung ausführlich bzw. kurz sein.
- *Geschlossene Aktionen*: Folgen von Aktionen sollten gruppiert angeboten werden. Die Gliederung in Anfang, Mitte und Ende mit entsprechenden Rückmeldungen gibt dem Benutzer das Gefühl des erfolgreichen Abschlusses einer Aktion.
- *Einfache Fehlerbehandlung*: Folgeschwere Benutzungsfehler sollten durch entsprechendes Oberflächendesign vermieden werden. Fehlerhafte Eingaben sollten derart korrigiert werden können, daß minimaler Aufwand entsteht.
- *Bereitstellung von UNDO-Funktionalität*: Möglichst viele Schritte sollten ungeschehen gemacht werden können. Die Realisierung von mehrstufiger UNDO-Funktionalität erfordert allerdings einen sehr hohen Aufwand.

- *Gefühl der Kontrolle über das System vermitteln*: Der Benutzer sollte das Gefühl der Kontrolle über das System haben, nicht von ihm kontrolliert werden. Dies läßt sich durch die Vermeidung von „überraschenden“ Systemaktionen und durch das Sichtbarmachen von Aktionseffekten erreichen. Das Prinzip der „*direkten Manipulation*“<sup>1</sup> unterstützt diesen Aspekt.
- *Für den menschlichen Betrachter übersichtliche Dialoge*: das menschliche, kurzfristige Informationsaufnahmevermögen beträgt ca. sieben Informationen. Dialoge sollten dementsprechend nicht mit Information überladen sein.

Als Vorteile von gut strukturierten graphischen Benutzeroberflächen lassen sich festhalten:

- Leichte Erlernbarkeit. Die anfänglich von Benutzern entgegengebrachte Skepsis läßt sich spielerisch und mit geringem Zeitaufwand abbauen.
- Der Benutzer hat mehrere Fenster zur Verfügung, mit denen er simultan arbeiten kann. Ausgaben von gerade nicht bearbeiteten Tasks können trotzdem wahrgenommen werden.
- Die Interaktion mit dem System ist auf einfache Art und Weise möglich. Das Erlernen von Kommandosprachen oder ähnlichen Interaktionsmitteln entfällt. So können auch Benutzer, die sehr selten mit dem System arbeiten, den vollen Funktionsumfang nutzen.

Die Nachteile einer graphischen Benutzeroberfläche liegen in dem relativ hohen Aufwand der Erstellung und in den benötigten Systemressourcen zur Laufzeit. Der Erstellungsaufwand kann durch die Verwendung von entsprechenden Werkzeugen drastisch gesenkt werden. Der Nachteil der zur Laufzeit benötigten Systemressourcen ist heute durch die Verfügbarkeit von sehr leistungsfähigen und kostengünstigen Rechnern in den Hintergrund getreten.

#### 4.2.2 Spezielle Anforderungen im Bezug auf das administrative Werkzeug

APRICOTS kann als „Middleware“ eines verteilten Transaktionssystems betrachtet werden. Eine Eigenschaft von verteilten Systemen ist die Transparenz der Lokation von Objekten. Auch das administrative Werkzeug sollte diesem Grundsatz folgen. Das heißt, daß die graphische Oberfläche des administrativen Werkzeugs von jedem beliebigen Rechnerknoten im System aus auf die gleiche Art und Weise eingesetzt werden kann. Dies impliziert, daß das Werkzeug von jedem Rechnerknoten aus gleichartig auf die Datenobjekte des Soll-Zustands zugreifen können muß (Zugriffstransparenz). Folglich kann lokaler stabiler Speicher nicht für die Speicherung des Soll-Zustands des Systems

<sup>1</sup> Prinzip der direkten Manipulation: Permanente Sichtbarkeit der interessierenden Objekte; schnelle, einstufige Benutzeraktionen mit unmittelbarer Rückmeldung; Ersetzung komplexer Kommandos durch physische Aktionen wie Mausclicks etc.. Näheres siehe [Shnei92], [Somm92], [Gunz91].

verwendet werden. Das Werkzeug muß logisch unterteilt werden in die Applikation und die Datenhaltung.

## 5 Benutzungskonzept des administrativen Werkzeugs

In diesem Kapitel wird die Sichtweise des Benutzers des administrativen Werkzeugs auf das APRICOTS-System dargestellt. Zunächst werden die Betrachtungsgegenstände und deren Funktionalität diskutiert und festgelegt. Außerdem werden Sicherheitsaspekte und Fehlersituationen diskutiert.

### 5.1 Objekte und Funktionalität

#### 5.1.1 Knotenorientiert

Für die Abarbeitung eines ConTract-Skripts werden verschiedene Step Server bzw. Benutzer Agenten benötigt. Diese Komponenten sind auf einem oder mehreren Rechnerknoten lokalisiert. Die Verfügbarkeit einer Komponente setzt daher zunächst die Verfügbarkeit des entsprechenden Rechnerknotens voraus. Aus diesem Grund ist die Überwachung von Rechnerknoten eine Aufgabe des System Monitors. Der System Monitor soll die Verfügbarkeit eines Rechnerknotens für APRICOTS ermitteln und visualisieren.

Grundsätzlich muß ein Monitor Kenntnis über die zu überwachenden Entitäten besitzen. Das administrative Werkzeug muß daher die Rechnerknoten, die an der APRICOTS-Konfiguration beteiligt sind, kennen. Dies muß in der System Administration verwaltbar sein. Es soll möglich sein, neue Rechnerknoten in die Liste der zu überwachenden aufzunehmen bzw. andere von der Überwachung auszunehmen.

#### *Attribute eines Rechnerknotens*

In der gegenwärtigen Implementierung von APRICOTS basiert das Verbindungsnetzwerk zwischen Rechnerknoten auf dem TCP/IP-Protokoll (Transport Control Protocol / Internet Protocol). Die Adressierung eines Rechnerknotens in einem TCP/IP-Netzwerk kann auf zwei Arten geschehen: eine aus vier Nummern zusammengesetzte IP-Adresse oder durch einen symbolischen Namen. Für die Verwendung von symbolischen Namen wird ein DNS (*Domain Name System*) benötigt, welches die Zuordnung zwischen Namen und IP-Adressen verwaltet. Der symbolische Name setzt sich aus dem Rechnernamen und dem Namen der Organisation, dem der Host angehört, zusammen<sup>1</sup>. Der Rechnerknoten mit dem Namen „salome“ des Instituts für Informatik der Universität Stuttgart beispielsweise hat die symbolische Adresse „salome.informatik.uni-stuttgart.de“ und die IP-Adresse „129.69.216.167“. Symbolische Adressen haben für den menschlichen Betrachter gegenüber IP-Adressen den Vorteil der einfacheren Handhabbarkeit und der höheren Aussagekraft. Beide Adressierungsarten sollten in dem administrativen Werkzeug zur Verfügung stehen.

---

<sup>1</sup> näheres in [Glass93]

Die Attribute eines Rechnerknotens werden in Tabelle 5.1 nochmals dargestellt.

Attribut	Typ	Erläuterung
HostName	String	Symbolischer Name des Rechnerknotens. Falls Name unbekannt, dann IP-Adresse.
IP	String	IP-Adresse des Rechnerknotens.

Tabelle 5.1: Attribute eines Rechnerknotens

## 5.1.2 Komponentenorientiert

### 5.1.2.1 Benutzer/-innen

In APRICOTS wird jeder menschliche Benutzer durch einen Agenten (Benutzer Agent) repräsentiert. Dieser dient als Puffer zwischen den User Interfaces „Elektronischer Eingangskorb“ (Tasklist Interface) und „ConTract-Fortschrittsanzeige“ (Monitoring Interface) und einer ConTract-Engine. Die Pufferfunktion besteht im Zwischenspeichern von Aufträgen bzw. Systeminformationen, die von der ConTract-Engine an den Benutzer während dessen Abwesenheit gesandt werden. In umgekehrter Richtung informiert der elektronische Eingangskorb den Benutzer Agenten über die Beendigung eines Arbeitsauftrags. Der Benutzer Agent teilt dies daraufhin der ConTract-Engine mit. Damit ist der Benutzer Agent an der Abarbeitung von ConTracts, die interaktiv mit Benutzern sind, beteiligt und wird zum Betrachtungsgegenstand des System Monitors.

Im Bezug auf die System Administration läßt sich festhalten, daß menschliche Benutzer(innen) einer relativ hohen Veränderlichkeit unterliegen. Denkbar sind Veränderungen wie Neueinstellungen von Mitarbeitern, das Ausscheiden von Mitarbeitern, Abteilungswechsel, Beförderung etc.. Diese Veränderungen sollen durch die System Administration verwaltbar sein.

Jeder Mitarbeiterin bzw. jedem Mitarbeiter einer Unternehmung sind bestimmte Rollen zugeordnet. Diese Rollen repräsentieren die Befähigung oder die Berechtigung, bestimmte Aufgaben zu bearbeiten. Der Namensraum von APRICOTS ermöglicht die Zuordnung von Rollen zu Benutzern. Dies sollte durch die System Administration verwaltbar sein.

#### *Attribute eines Benutzers*

Die Identifikation eines Benutzers erfolgt in APRICOTS durch einen systemweit eindeutigen Namen (*UserName*). Alle Benutzernamen sind im Namensraum unterhalb des Asts *APRICOTS.Admin.Users* lokalisiert. Die Eindeutigkeitsforderung des Benutzernamens kann so auf einfache und effiziente Art und Weise gesichert werden.

Damit ein klarer Bezug zwischen Benutzername und der menschlichen Person hergestellt werden kann, wird das Attribut „*NameInRealLife*“ eingeführt. Auf weitere Eigenschaften einer menschlichen Person, wie z.B. Geburtsdatum, Adresse etc., wird an dieser Stelle bewußt verzichtet, da das administrative Werkzeug nicht die Funktionalität einer Mitarbeiterkartei bereitstellen soll. Die Verbindung zu einer Mitarbeiterkartei könnte bei Bedarf über den Schlüssel „*UserName*“ leicht hergestellt werden. Wie oben erwähnt, werden jedem Benutzer eine oder mehrere Rollen zugeordnet. Dies wird durch das Attribut „*Roles*“ repräsentiert.

Das APRICOTS-System stellt die Implementierung eines generischen Benutzer Agenten zur Verfügung. Bei der Neuaufnahme eines Benutzers sollte angegeben werden können, ob dieser generische Agent verwendet werden soll oder ein eventuell eigens implementierter Agent, der beispielsweise zusätzliche Funktionalität bietet. Daraus ergeben sich die zwei Attribute „*GenericUserAgent*“ und „*UserAgentExecutable*“. Jeder Benutzer Agent kann auf einem beliebigen Rechnerknoten, der am System teilnimmt, lokalisiert sein. Dies wird durch das Attribut „*onHost*“ berücksichtigt.

Zusätzlich zu den bisher genannten Eigenschaften werden die Attribute „*Rights*“ und „*Passwd*“ eingeführt. Diese Attribute beziehen sich auf die Zugangsberechtigung zum administrativen Werkzeug. Der Zweck wird in Abschnitt 5.2 (Sicherheitsaspekte) näher spezifiziert.

Die Tabelle 5.2 stellt die Attribute einer Benutzerin bzw. eines Benutzers nochmals in der Übersicht dar.

Attribut	Typ	Erläuterung
UserName	String	Systemweit eindeutiger Benutzername
NameInRealLife	String	Vor- und Nachname der Person
GenericUserAgent	Boolean	Flag, ob generischer oder individueller Benutzer Agent verwendet wird
UserAgentExecutable	String	Pfad und Name des individuellen Benutzer Agenten (falls verwendet)
Rights	AdminRights	Berechtigung für das administrative Werkzeug
Passwd	String	Passwort für den Zugang zum administrativen Werkzeug
Roles	String Sequence	Sequenz von Namenstrings, welche die Rollen des Benutzers repräsentieren
onHost	String	Name des Rechnerknotens, auf dem der Benutzer Agent lokalisiert ist

Tabelle 5.2: Attribute einer Benutzerin bzw. eines Benutzers

### 5.1.2.2 Automated Step Server

Ein Automated Step Server implementiert einen einzelnen Step, der nicht mit menschlichen Benutzern interagiert. Jede automatisierte Ressource wird über einen Automated Step Server dem System zugänglich gemacht. Die Inanspruchnahme von automatisierten Ressourcen erfolgt in APRICOTS ausschließlich über Automated Step Server. Automated Step Server bilden also die Schnittstelle zwischen einer ConTract-Engine und den Ressourcen-Verwaltern. Die Verfügbarkeit von Automated Step Servern sollte deshalb durch den System Monitor überwacht werden können.

Wie in Kapitel 4 bereits erwähnt, bilden die Automated Step Server in Verbindung mit den Benutzer Agenten die individuelle Semantik des Systems und damit die eigentliche Applikation. Dem administrativen Benutzer muß die Möglichkeit gegeben werden, auf einfache Art und Weise die Automated Step Server zu verwalten. Neue Server sollten integriert, bestehende aus dem System entfernt und der Status eines Servers verändert werden können. Beispielsweise sollte es möglich sein, einen Automated Step Server zu stoppen, um eine neue Version des ausführbaren Programmcodes zu installieren.

Die Semantik eines Automated Step Servers wird analog zu den Benutzern im Namensraum von APRICOTS abgebildet. Die Rollen eines Automated Step Servers finden sich im Namensraum unterhalb des Asts *APRICOTS.StepServers.AutomatedServers*. Dort wählt die ConTract-Engine bei der Abarbeitung eines Skripts die passenden Automated Step Server aus.

#### *Attribute eines Automated Step Servers*

Jeder Automated Step Server wird in APRICOTS durch einen eindeutigen Namen identifiziert. Das administrative Werkzeug muß die Eindeutigkeitsforderung gewährleisten. Neben dem Pfad und Name des ausführbaren Programmcodes des Servers muß angegeben werden können, auf welchem Rechnerknoten der Server lokalisiert sein soll. Wie oben erwähnt, sind jedem Automated Step Server  $n$  ( $n \geq 0$ ) Rollen zugeordnet. Dies wird durch das Attribut „Roles“ berücksichtigt. Der (Soll-)Zustand eines Automated Step Servers wird durch das Attribut „State“ repräsentiert.

Die Tabelle 5.3 stellt die Attribute eines Automated Step Servers nochmals dar.

Attribut	Typ	Erläuterung
Name	String	Eindeutiger Name des Automated Step Servers
Executable	String	Pfad und Name des ausführbaren Programmcodes
Roles	String Sequence	Sequenz von Namestrings, welche die Rollen des Automated Step Servers repräsentieren
onHost	String	Name des Rechnerknotens, auf dem der Automated Step Server lokalisiert ist
State	SS_STATE	Soll-Zustand des Automated Step Servers.

Tabelle 5.3: Attribute eines Automated Step Servers

### **5.1.2.3 Resource Manager**

Ressourcen-Verwalter (Resource Manager) verwalten Objekte wie Datenbanken etc.. Nach [Seif96] initiiert die ConTract-Engine nie direkt Dienste von Resource Managern, sondern immer über Step Server. Das administrative Werkzeug sollte deshalb die Überwachung und Verwaltung von Resource Managern auf ähnliche Weise wie Automated Step Server bereitstellen. Außer dem Attribut „Roles“, das hier nicht benötigt wird, können die Attribute eines Automated Step Servers auch die Eigenschaften eines Resource Managers beschreiben.

### **5.1.2.4 Der Naming Service**

Der Naming Service kann als „systeminterne“ Komponente betrachtet werden. Per Konvention muß er einmal pro Namensraum vorhanden sein und ist damit nicht Teil einer individuellen Konfiguration bzw. Applikation. Der administrative Benutzer hat nur implizit Einfluß auf den Namensraum, indem er Benutzern bzw. Step Servern Rollen zuordnet. Der Naming Services wird daher ausschließlich implizit durch das Einrichten von Rollen administriert.

### **5.1.2.5 Speicher Agenten, ConTract Manager, ConText Management, Object Request Broker**

Diese Komponenten sind ebenfalls als „systemintern“ einzustufen. D.h. sie sind nicht Teil einer speziellen Applikation, sondern implementieren Basisdienste des Systems. Ist eine solche Komponente ausgefallen, dann muß ein Neustart des Gesamtsystems erfolgen.

## **5.1.3 Zusammenfassung**

Der Benutzer des administrativen Werkzeugs versteht sich als Administrator der individuellen Applikation, die unter APRICOTS realisiert ist. Sein Blickwinkel beschränkt sich daher auf Komponenten, welche die individuelle Applikation ausmachen. Diese Komponenten sollen durch das administrative Werkzeug verwaltet und überwacht werden können. Konkret bedeutet dies das Verwalten von Mitarbeitern bzw. Benutzern, das Verwalten von Step Servern und das Verwalten von Resource Managern. Im Hinblick auf die Überwachung sollen diejenigen Komponenten betrachtet werden, welche Mitarbeiter, Step Server und Resource Manager implementieren. Alle anderen Komponenten sind

Bestandteile von APRICOTS, ohne die das Gesamtsystem nicht lauffähig ist. Die Überwachung dieser Art von Komponenten ist daher nicht Bestandteil des administrativen Werkzeugs.

## **5.2 Sicherheitsaspekte**

Ein wesentliches Ziel von APRICOTS ist die Bereitstellung einer *zuverlässigen* Ausführungsplattform für ConTract-Skripte. Mit Hilfe des administrativen Werkzeugs ist es möglich, die applikationsspezifischen Komponenten in das System einzubringen und zu entfernen. Dies kann die Funktionsfähigkeit des Gesamtsystems beeinflussen und sollte deshalb nur durch sachkundige und dafür autorisierte Benutzer erfolgen.

Das administrative Werkzeug enthält Informationen über die Rollen der Benutzer. Die Rolle eines Benutzers spiegelt dessen Kompetenzen oder Befähigungen wieder. In der Praxis dürfte die Kompetenzverteilung innerhalb einer Unternehmung eine nicht-öffentliche Information sein und muß daher vor unbefugtem Zugriff geschützt werden. Der Zugang zum administrativen Werkzeug muß aus diesen Gründen kontrolliert werden.

## 6 Realisierung

### 6.1 Programmiersprache und Werkzeuge

#### 6.1.1 Java

Die für die Realisierung zu verwendende Programmiersprache stand nicht zur Diskussion. In der Aufgabenstellung dieser Arbeit ist vorgegeben, daß das Werkzeug in der Programmiersprache „Java“ zu implementieren ist.

Java ist eine junge Programmiersprache, die von Sun Microsystems entwickelt wurde. Die Sprache wird in [GosGil96] so charakterisiert:

*Java: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language.*

Java bietet gegenüber „konventionellen“ Programmiersprachen einige Vorteile:

- *Verteilt:* Unter Java können verteilte Applikationen auf einfache Art und Weise erstellt werden. Java unterstützt „Sockets“ zur zuverlässigen Interprozeßkommunikation und erlaubt den Zugriff auf „entfernte“ Objekte über das Internet.
- *Robustheit und Sicherheit:* Typen werden in Java zur Übersetzungszeit strenger überprüft, als in C++. C++ hat insbesondere im Bereich der Methodendeklaration die Mängel der Programmiersprache C „geerbt“. Das Speichermodell von Java wurde hauptsächlich durch die Eliminierung von Zeigervariablen zuverlässiger gemacht. Das Problem der „dangling pointer“ wird in Java durch einen in der virtuellen Maschine eingebauten „Garbage Collector“ umgangen. Java-Code enthält daher keine Anweisungen, um Objekte zu löschen. Im Hinblick auf verteilte Systeme implementiert Java verschiedene Mechanismen, um sich vor unbefugtem Zugriff über das Netzwerk schützen zu können.
- *Architektur-neutral, Portabilität und Interpretierung:* Java ist eine plattformunabhängige Sprache. Dies wird erreicht durch die Implementierung einer „virtuellen Maschine“ auf verschiedenen Plattformen. Der Übersetzer übersetzt Java-Code in einen Byte-Code, der zur Laufzeit von der virtuellen Maschine interpretiert wird. Damit können Applikationen ohne Neuübersetzung auf verschiedene Plattformen portiert werden.
- *Multithreaded:* Threads sind Bestandteil der Sprache. Synchronisationsmechanismen sind verfügbar.

Im Bezug auf die Performance müssen gegenüber C++ allerdings Abstriche gemacht werden. Java-Applikationen laufen durch die interpretative Ausführung nach [Flan96] durchschnittlich 20 mal langsamer ab, als in Maschinencode übersetzte C++-Programme. Dieser Nachteil kann beseitigt werden, wenn Übersetzer verfügbar werden, die Java-Code in Maschinencode übersetzen. Die Plattformunabhängigkeit geht dabei jedoch verloren. Eine andere Möglichkeit besteht in sogenannten „just-in-time-compilers“. Diese übersetzen zur Laufzeit Java-Byte-Code in Maschinencode der Zielmaschine. Dadurch wird die Plattformunabhängigkeit bewahrt und trotzdem ein gutes Laufzeitverhalten erreicht. Sun gibt an, daß in Maschinencode übersetzter Byte-Code vom Laufzeitverhalten kaum schlechter ist als in Maschinensprache übersetzter C++-Code. Just-in-time-compiler sind derzeit noch nicht verfügbar.

Als weiterer Nachteil von Java muß gesagt werden, daß die Implementierung des Interpreters noch nicht völlig ausgereift ist. Das öffentliche Interesse an Java ist jedoch so groß, daß die kontinuierliche Verbesserung des Laufzeitsystems erwartet werden kann.

Alles in allem ist Java sehr gut geeignet, um graphische Benutzeroberflächen zu implementieren. Der Geschwindigkeitsnachteil fällt bei dieser Art von Applikation nicht so sehr ins Gewicht.

### 6.1.2 Java-WorkShop und Visual

Aufgrund der relativ jungen Sprache waren zu Beginn dieser Arbeit nur wenige Werkzeuge verfügbar. Als Entwicklungsumgebung stand zunächst eine frühe Vorversion des „Java-WorkShop“ der Firma SunSoft zur Verfügung (Version Dev. 5, Mai 1996). Der Java-WorkShop ist unterteilt in die Entwicklungsumgebung, in der Projekte verwaltet und bearbeitet werden können, und den graphischen Interface-Builder „Visual“. Visual erlaubt das interaktive Erstellen von graphischen Benutzeroberflächen sowohl für Applets, als auch für „standalone“-Applikationen. Das Arbeiten mit diesem Werkzeug war durch häufige Systemabstürze und unakzeptablen Antwortzeiten nicht sinnvoll möglich. Mit dem Erscheinen der ersten Vollversion des Java-WorkShops (Release: 1.0) waren die meisten Probleme beseitigt, so daß das graphische Userinterface (GUI) mit diesem Werkzeug erstellt wurde.

Nachdem das GUI interaktiv erstellt ist, läßt sich in Visual Sourcecode erzeugen. Der Code läßt sich ohne weitere Modifikation zu einem lauffähigen Programm übersetzen. Der übersetzte Code ist sowohl als Applet, als auch als „standalone“-Applikation lauffähig. Oberflächenkomponenten erzeugen bei der Interaktion durch den Benutzer sogenannte „Events“. Diese Events können in eigenen Routinen behandelt werden. Die Applikation wird also durch die Event-Behandlungsroutinen an die graphische Oberfläche gebunden.

Die Dokumentation des Java-WorkShops ist in der vorliegenden Version nur „online“ verfügbar. Die Klassenhierarchie von Visual beispielsweise läßt sich nur relativ mühsam

und mit einigem Zeitaufwand verstehen. Es sind nur die Klassendefinitionen dokumentiert, meist ohne oder mit sehr wenigen Kommentaren. Ein umfassendes Programmierhandbuch existiert beispielsweise nicht.

## 6.2 Architektur

In Abschnitt 4.2 wird die Anforderung gestellt, daß das administrative Werkzeug von jedem an APRICOTS teilnehmenden Rechnerknoten aus einsetzbar ist. Das administrative Werkzeug muß also Lokationstransparenz gewährleisten. Dies impliziert, daß lokaler stabiler Speicher für die Speicherung des Soll-Zustands des Systems nicht verwendet werden kann. Lokationstransparenz kann durch die Verteilung des administrativen Werkzeugs erreicht werden. Es stellt sich die Frage, wie die Verteilung vorgenommen wird.

Betrachtet man allgemein die Aufgaben einer Anwendung, so lassen sich drei logische Komponenten erkennen, die folgende Aufgaben erfüllen:

- *Präsentation*: Die Schnittstelle zwischen Benutzer(-in) und Anwendung.
- *Anwendungslogik*: Die Vermittlung zwischen Benutzerschnittstelle und Datenhaltungsschnittstelle. Gleichzeitig ist hier die Anwendungslogik lokalisiert.
- *Datenhaltung*: Die Verwaltung von persistenten Speicherobjekten.

Die logischen Komponenten lassen sich auf drei verschiedene Arten verteilen. Abb. 6.1 verdeutlicht die drei Basis-Verteilungsmethoden nach [Clie95]:

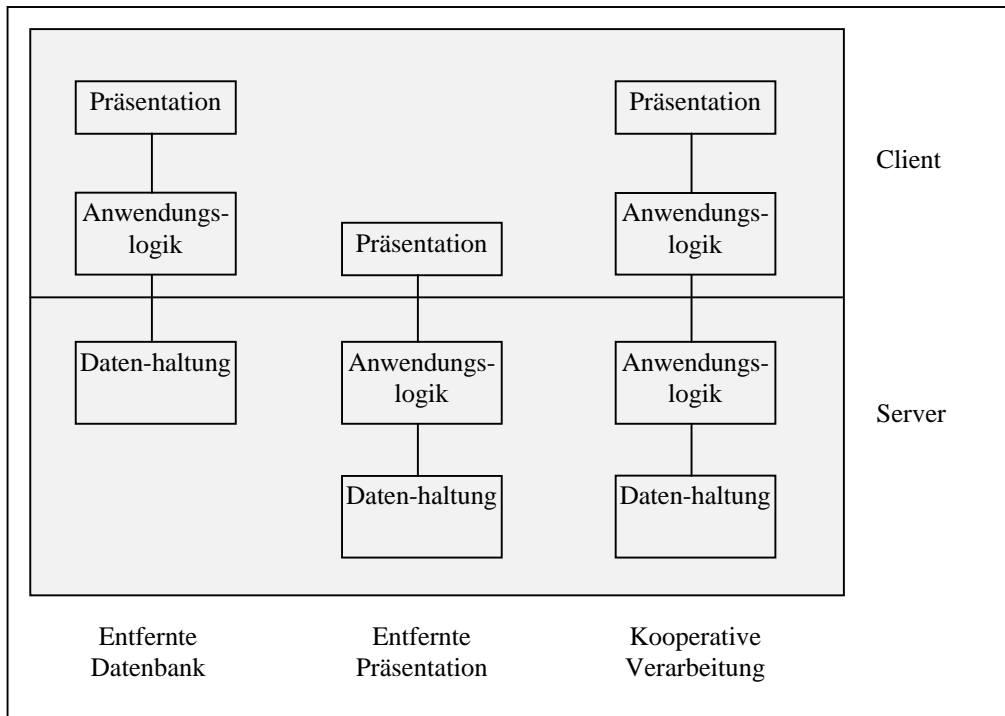


Abb. 6.1: Die drei Basis-Verteilungsmodelle nach [Clie95]

*Entfernte Datenbank (Remote Data Access)*

Die Anwendung wird vollständig im Client ausgeführt. Server werden lediglich für Datenzugriffe genutzt. Im Server befindet sich keinerlei Anwendungslogik.

*Entfernte Präsentation (Remote Presentation)*

Die Anwendungslogik ist vollständig im Server lokalisiert. Der Client fungiert lediglich als Interaktionsschnittstelle zwischen Benutzer und Anwendung. Jede Interaktion des Benutzers muß zum Server übertragen und dort behandelt werden. Ergebnisse werden zurückübertragen und im Client dargestellt. Ein gängiges Beispiel für diese Architekturvariante ist das X-Window-System.

*Kooperative Verarbeitung (Distributed Processing)*

Die Anwendung selbst ist auf Client und Server verteilt. Ein Teil der Anwendung bedient die Schnittstelle zur Präsentation während der andere Teil den Datenzugriff übernimmt. Beide Teile enthalten Funktionalität der Anwendung.

Aus verschiedenen Gründen fiel die Entscheidung auf die Variante „Entfernte Datenbank“. Gegenüber der Variante „Entfernte Präsentation“ ist die Netzbelastung durch „Entfernte Datenbank“ deutlich geringer, da nur Zugriffe auf persistente Daten über das Netzwerk geleitet werden müssen. Die Interaktion mit dem Benutzer und die Anwendungslogik laufen vollständig im Client ab. Der Realisierungsaufwand für „Entfernte Datenbank“ ist relativ gering. Es genügt die einfache Definition einer Datenbankschnittstelle. Dieser Ansatz wird bereits durch den modularen Aufbau unterstützt. Ein- und Ausgaberroutinen können leicht in ein Modul gekapselt werden.

Letztendlich den Ausschlag gegeben für die Methode „Entfernte Datenbank“ hat jedoch, daß zu Beginn dieser Arbeit OrbixWeb (Sprachanpassung von Orbix für Java) nur als Client agieren konnte und nicht als Server. Dadurch konnten in der Programmiersprache Java keine Server implementiert werden, die über ORB's mit Clients kommunizieren. Da jedoch in APRICOTS jegliche Kommunikation über Orbix realisiert werden soll, scheiden die Möglichkeiten „Entfernte Präsentation“ und „Kooperative Verarbeitung“ aus.

Eine weitere Einschränkung der Möglichkeiten des Datenzugriffs lag zu Beginn dieser Arbeit im Fehlen des direkten Datenbankzugriffs von der Programmiersprache Java aus. Die letzte Möglichkeit für das administrative Werkzeug, Daten auf stabilem Speicher zu halten und die oben genannten Anforderungen zu erfüllen, besteht in der Verwendung des Speicher Agenten von APRICOTS.

Die sich damit ergebende Architektur des administrativen Werkzeugs ist in Abb. 6.2 dargestellt.

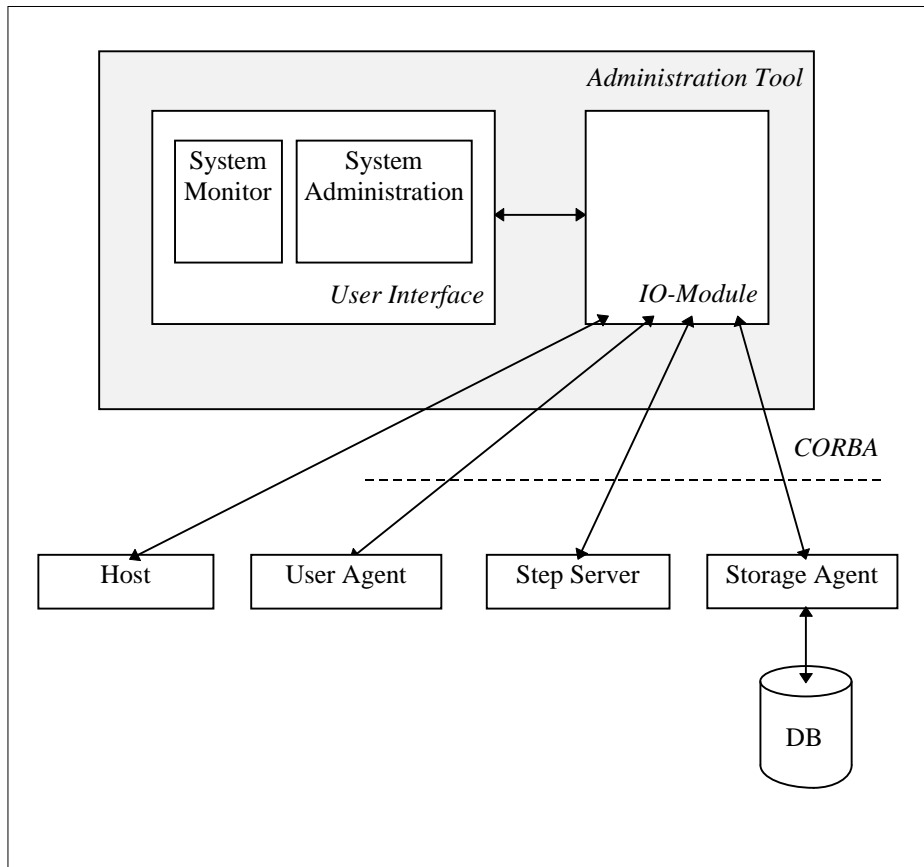


Abb. 6.2: Die Architektur des administrativen Werkzeugs

### 6.3 Entwurf

#### 6.3.1 Objekte

Resource Manager sind nicht explizit Betrachtungsgegenstand des administrativen Werkzeugs. Das Einbringen eines Resource Managers ist aus der Sicht des administrativen Werkzeugs eine Unterfunktion des Einbringens eines Step Servers. Step Server müssen individuell programmiert werden, so daß in diesem Schritt die vom Step benötigten Resource Manager integriert werden können. Resource Manager können als eine Art Unterklasse von Step Servern betrachtet werden, da eine ConTract-Engine mit Resource Managern ausschließlich über Step Server kommuniziert. Das Einbringen eines Resource Managers, dem kein Step Server zugeordnet ist, ist daher sinnlos. Aus diesem Grund unterscheidet das administrative Werkzeug nicht zwischen Resource Managern und Step Servern.

Aus der Sicht einer ConTract-Engine unterscheiden sich auch Step Server und Benutzer Agenten nicht. Step Server und Benutzer Agenten werden auf die gleiche Art und Weise initiiert. Aus Systemsicht könnte daher auf die Unterscheidung dieser beiden Objekte ebenfalls verzichtet werden. Die Sicht des Benutzers des administrativen Werkzeugs ist jedoch eine andere. Der menschliche Administrator möchte nach wie vor zwischen Benutzern und automatisierten Step Servern unterscheiden können. Dies spiegelt sich in den Attributen, die Benutzer beschreiben, wider. Zur klaren Zuordnung eines Benutzer Agenten an eine Person wurde in Kapitel 5.1 das Attribut „*NameInRealLife*“ eingeführt. Außerdem können Benutzern Rechte im Bezug auf die Benutzung des administrativen Werkzeugs eingeräumt werden. Diese Rechte werden anhand eines Passwortes beim Zugang zum Werkzeug überprüft. Die gerade genannten Eigenschaften sind im Bezug auf Step Server nicht sinnvoll. Ein weiterer Grund für die Unterscheidung zwischen Benutzer Agenten und Step Servern durch das administrative Werkzeug liegt in der Möglichkeit, bei der Rolleneditierung den für Benutzer bzw. Step Server korrekten Ast im Namensraum vorzuselektieren. Die Rollen von Benutzern finden sich unter „*APRICOTS.StepServers.UserAgents*“, während die Rollen von Step Servern im Namensraum unter „*APRICOTS.StepServer.AutomatedServers*“ zu finden sind. Die Unterscheidung durch das Werkzeug vermeidet daher die Entstehung von Fehlern.

### 6.3.2 Administration

Die System Administration ermöglicht das Einbringen bzw. das Entfernen von Komponenten in APRICOTS.

#### 6.3.2.1 Das Einbringen von Komponenten

Beim *Einbringen* einer neuen Komponente in das System müssen einige Arbeitsschritte durchgeführt werden:

- Der Name der Komponente muß auf Eindeutigkeit geprüft werden. Der Name kann anhand des Soll-Zustands des Systems überprüft werden, da der Soll-Zustand alle im System befindlichen Komponenten kennt. Dies bedeutet einen Zugriff auf den stabilen Speicher des administrativen Werkzeugs.

Die zweite Möglichkeit besteht in der Überprüfung anhand des Namensraums. Jeder Automated Step Server ist im Namensraum unterhalb von „*APRICOTS.Admin.StepServers*“ registriert bzw. jeder Benutzer Agent unter „*APRICOTS.Admin.UserAgents*“. Da die vorliegende Version des Orbix Naming Services keine Zugriffspfade unterstützt, müssen alle vorhandenen Namen unterhalb des entsprechenden Asts überprüft werden. Dies bedeutet einen relativ großen Aufwand.

Der stabile Speicher des administrativen Werkzeugs wird durch ein Datenbanksystem realisiert. Die Funktionalität des Datenbanksystems kann daher für die Eindeutigkeitsprüfung implizit verwendet werden.

- Die neue Komponente muß in den Soll-Zustand des Gesamtsystems aufgenommen werden. Dies bedeutet den Zugriff auf den stabilen Speicher des administrativen Werkzeugs.
- Die neue Komponente muß in dem Implementation Repository desjenigen ORBs registriert werden, auf welchem das Server-Objekt der Komponente lokalisiert sein soll. Im Implementation Repository stehen Zuordnungen zwischen Namen von Server-Objekten und den zugehörigen ausführbaren Programmdateien. Das Implementation Repository wird von dem ORB verwendet, um einen Server, der zum Zeitpunkt einer Dienstanforderung nicht läuft, anstarten zu können. Die Registrierung im Implementation Repository kann durch das Orbix-Shell-Kommando „*putit -h <hostname> <ServerName> <CommandLine>*“ erfolgen. Die zweite Möglichkeit besteht in der Verwendung des IT\_Daemon Interfaces. Der Orbix-Daemon ist selbst ein Orbix-Server, dessen IDL-Interface „*IT\_Daemon*“ heißt. Das Interface bietet Methoden zum Zugriff auf das Implementation Repository an.
- Falls die neue Komponente ein Automated Step Server ist, muß im Namensraum von APRICOTS unterhalb von „*APRICOTS.Admin.StepServers*“ eine Zuordnung des eindeutigen Namens und der Referenz des Server-Objekts eingetragen werden. Der Orbix Naming Service bietet ein IDL-Interface namens „*CosNaming*“ an, über das der Namensraum verwaltet werden kann. Von Interesse in diesem Zusammenhang sind die Methoden „*new\_context()*“ und „*bind\_new\_context()*“.
- Falls die neue Komponente ein Benutzer Agent ist, muß im Namensraum von APRICOTS unterhalb von „*APRICOTS.Admin.UserAgents*“ eine Zuordnung des eindeutigen Namens und der Referenz des Server-Objekts eingetragen werden. Der Eintrag kann ebenfalls über das Interface *CosNaming* erfolgen.
- Falls die neue Komponente ein Automated Step Server ist, müssen im Namensraum von APRICOTS unterhalb von „*APRICOTS.StepServers.AutomatedServers*“ alle Rollen des Step Servers eingetragen werden. Ein Eintrag besteht dabei wiederum aus der Zuordnung des eindeutigen Namens des Step Servers und der Referenz seines Server-Objekts. Die Eintragungen können ebenfalls über das Interface *CosNaming* erfolgen.
- Falls die neue Komponente ein Benutzer Agent ist, müssen im Namensraum von APRICOTS unterhalb von „*APRICOTS.StepServers.UserAgents*“ alle Rollen des Benutzers eingetragen werden. Ein Eintrag besteht dabei aus der Zuordnung des eindeutigen Namens des Benutzer Agenten und der Referenz seines Server-Objekts. Die Eintragungen können ebenfalls über das Interface *CosNaming* erfolgen.

Nach [Stein96] führt der Benutzer Agent beim erstmaligen Anstarten einen Teil der oben genannten Aufgaben selbst durch. Demnach wird die Registrierung des Server-Objekts in dem Implementation Repository durch den Benutzer Agenten automatisch durchgeführt. Des weiteren trägt sich der Benutzer Agent in den Namensraum von APRICOTS unterhalb „APRICOTS.Admin.UserAgents“ ein. Step Server verhalten sich in dieser Hinsicht analog. Der einzige Unterschied besteht in der Eintragung im Naming Service. Ein Automated Step Server trägt seinen Namen und seine Objektreferenz nicht unter „APRICOTS.Admin.UserAgents“, sondern unter „APRICOTS.Admin.StepServers“ ein.

Aus der Sicht des administrativen Werkzeugs reduzieren sich damit die Arbeitsschritte beim Einbringen einer neuen Komponente auf die Registrierung der Komponente im Soll-Zustand des Systems, dem erstmaligen Anstarten der Komponente mit den entsprechenden Parametern (vgl. Abschnitt 6.4) und dem Eintragen der verschiedenen Rollen der Komponente in den Naming Service.

Zu klären bleibt noch die Reihenfolge der einzelnen Arbeitsschritte. Unter der Annahme, daß die Funktionalität des Datenbanksystems für die Eindeutigkeitsüberprüfung des Namens der neuen Komponente implizit genutzt wird, muß die Aufnahme in den Soll-Zustand des Systems als erster Schritt erfolgen. Bei einer Verletzung der Schlüsselbedingung, d.h. Zurückweisung der append-Operation, liegt noch immer der konsistente Zustand vor, wie zu Beginn der Operation.

Die Eintragungen der Rollen der neuen Komponente können erst erfolgen, nachdem die Komponente erstmals angestartet wurde und ihre Startup-Aktivitäten zu Ende gebracht hat. Erst zu diesem Zeitpunkt ist die Referenz des Server-Objekts bekannt. Diese Referenz wird benötigt, um die Rollen der Komponente im Naming Service eintragen zu können.

Die Abbildung 6.3 stellt den Ablauf beim Einbringen einer neuen Komponente nochmals als Flußdiagramm dar.

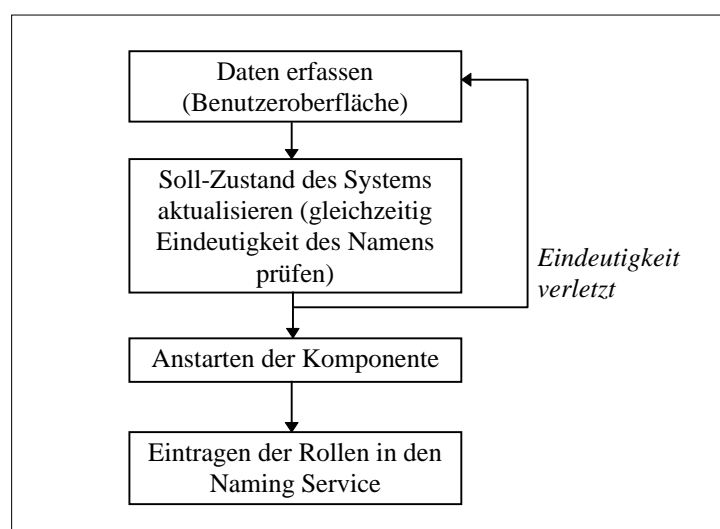


Abb. 6.3: Das Einbringen einer Komponente

Der oben aufgezeigte Ablauf ist logisch richtig. Neue Komponenten sollten erst dann durch eine ConTract-Engine verwendet werden, wenn sie vollständig in das System eingebracht worden sind. Da die ConTract-Engine Komponenten anhand ihrer Rollen auffindet, wird eine Komponente frühestens dann initiiert, nachdem ihre Rollen öffentlich gemacht wurden. Dies geschieht jedoch erst nachdem die Komponente logisch dem System zur Verfügung steht. Damit können keine Konflikte entstehen.

### 6.3.2.2 Ändern von Komponenten-Attributen

Änderbare Attribute einer Komponente sind insbesondere die Rollen der Komponente. In der Praxis ist beispielsweise die Betrauung eines Mitarbeiters mit anderen Aufgaben denkbar oder ein Automated Step Server, der z.B. den Zugriff auf eine Datenbank ermöglicht, wird zusätzlich einer anderen Abteilung zur Verfügung gestellt. Diese Art der Anpassung sollte direkt möglich sein.

Das Hinzufügen von Rollen ist problemlos möglich. Es entspricht dem letzten Arbeitsschritt beim Einbringen einer neuen Komponente (vgl. oben). Das Ändern von Rollen einer Komponente kann durch das Hinzufügen der neuen Rolle und dem anschließenden Entfernen der alten Rolle erfolgen. Die Soll-Zustandsänderung eines Automated Step Servers wird ebenfalls durch das Entfernen bzw. Hinzufügen der Rollen durchgeführt. Die Abb. 6.4 zeigt die möglichen Soll-Zustandsübergänge eines Step Servers.

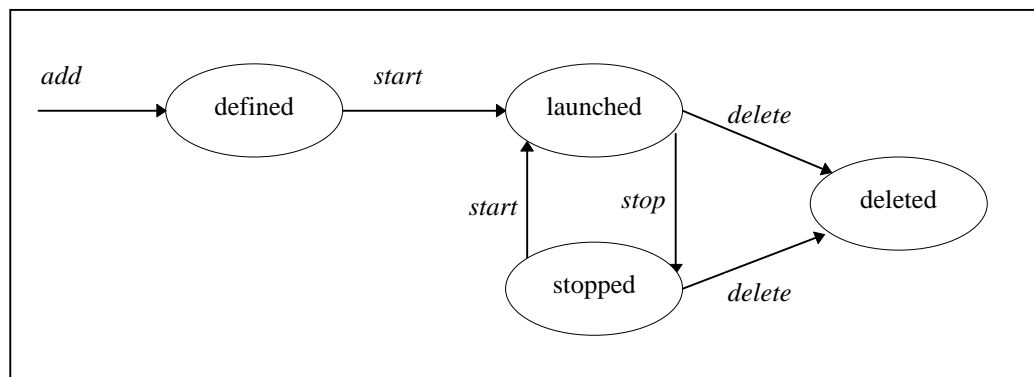


Abb. 6.4: Übergänge der Soll-Zustände eines Automated Step Servers

Attribute, die das System nicht betreffen, können ohne Konflikte verändert werden. Dies entspricht einer einfachen Update-Operation auf der Datenbank des administrativen Werkzeugs. Attribute dieser Art sind die Administrationsberechtigung eines Benutzers und das Passwort eines Benutzers.

Änderungen von anderen Attributen werden vom administrativen Werkzeug nicht zugelassen, da sie im wesentlichen dem vollständigen Entfernen und Neueinbringen der Komponente entsprechen. Der folgende Abschnitt zeigt jedoch, daß das Entfernen von Komponenten nicht problemlos durchgeführt werden kann.

### 6.3.2.3 Entfernen von Komponenten

Das Entfernen von Komponenten erweist sich als problematisch. Eine Komponente kann nicht sorglos aus dem System entfernt werden. Vielmehr müssen Überlegungen angestellt werden, wie entschieden werden kann, ob eine Komponente aus dem System entfernt werden darf oder nicht. Bei diese Überlegungen müssen die Garantien, die das ConTract-Modell dem Benutzer zusichert, berücksichtigt werden. In diesem Zusammenhang ist die folgende Garantie von Bedeutung.

Das ConTract-Modell garantiert dem Benutzer, daß zu jedem Step, der gerade abgearbeitet wird, die Kompensation dieses Steps möglich ist. Diese Garantie darf durch das administrative Werkzeug nicht aufgehoben werden. Es stellt sich daher zunächst die Frage, zu welchem Zeitpunkt entschieden werden kann, ob eine Komponente entfernt werden darf oder nicht. Es sind drei Möglichkeiten denkbar. Zum Zeitpunkt einer abgeschlossenen Skript-Definition ist bekannt, welche Komponenten im Skript enthalten sind. Es kann jedoch nicht ausgesagt werden, ob jemals eine Instanz dieses Skripts gestartet werden wird und welche Komponenten bei der Abarbeitung einer Instanz dieses Skripts verwendet werden. Die zweite Möglichkeit ist der Zeitpunkt des Startens einer ConTract-Instanz. Auch hier kann noch nichts über die tatsächlich benötigten Komponenten gesagt werden, da bedingte Verzweigungen beispielsweise erst zur Laufzeit evaluiert werden. Diese beiden Ansätze sind daher viel zu restriktiv und „überdimensioniert“, um die oben zugesicherte Garantie von Seiten des administrativen Werkzeugs sicherzustellen.

Die dritte Möglichkeit besteht zur Laufzeit. Soll eine Komponente aus dem System entfernt werden, so muß zu diesem Zeitpunkt überprüft werden, ob die Komponente Teil eines Kompensationssteps ist, welcher momentan zur Verfügung stehen muß. D.h. die zu entfernende Komponente selbst muß eine Schnittstelle bieten, mit der überprüft werden kann, ob diese Komponente Teil eines Kompensations-Steps dessen zugeordneter Step momentan abgearbeitet wird. Ist dies der Fall, dann kann zu diesem Zeitpunkt die Komponente nicht entfernt werden. Damit wird die oben zugesicherte Garantie mit geringst möglicher Restriktion gewährleistet.

Die derzeitige Implementierung der Komponenten Step Server und Benutzer Agent sehen eine solche Methode in ihrer Schnittstellendefinition nicht vor. Das Entfernen von Komponenten durch das administrative Werkzeug beschränkt sich daher auf das Entfernen der Rollen der Komponente aus dem Naming Service und auf das Entfernen der Komponente aus dem Soll-Zustand des Systems.

### 6.3.3 Überwachung / Monitoring

Grundsätzlich gibt es zwei Methoden, ein Objekt zu überwachen. Die erste Methode besteht im Abfragen des Zustands durch eine externe Instanz. Die zweite Methode besteht in der Mitteilung einer Zustandsveränderung durch das zu überwachende Objekt selbst. Die letztere Methode setzt jedoch voraus, daß das Objekt noch in der Lage ist, eine Zustandsveränderung mitzuteilen. Die „passive“ Zustandsermittlung, d.h. die überwachende Komponente wartet auf Signale der zu überwachenden Objekte, ist beispielsweise sehr gut geeignet, um Veränderung in einem Datenbanksystem anderen Komponenten (z.B. der Benutzeroberfläche) mitteilen zu können. In modernen Datenbanksystemen kann dies durch die Definition eines sogenannten „*Triggers*“ realisiert werden.

Der Ansatz der passiven Zustandsermittlung ist im Bezug auf APRICOTS nicht geeignet. Es kann bei den zu überwachenden Komponenten nicht davon ausgegangen werden, daß sie im Falle der Fehlfunktion bzw. des Absturzes dies noch nach außen mitteilen können. Die Grundvoraussetzung für die passive Zustandsermittlung ist damit nicht erfüllt. Bei der ausschließlichen Verwendung der passiven Zustandsermittlung könnten so ausgefallene Komponenten möglicherweise nicht erkannt werden.

In APRICOTS muß es daher eine Instanz geben, die aktiv die Zustände der Komponenten ermittelt. Diese Aufgabe erfüllt der System Monitor. Im Bereich der aktiven Zustandsermittlung können wiederum zwei Methoden unterschieden werden. Die Zustandsermittlung kann initiiert werden, wenn der Benutzer dies wünscht. Diese Methode wird „*Snapshot*“ genannt und setzt die Anwesenheit des Benutzers voraus. Die zweite Möglichkeit besteht in der automatischen Zustandsermittlung in regelmäßigen Zeitabständen. Dabei wird in bestimmten Zeitabständen ein Snapshot durchgeführt. Diese Methode wird „*Polling*“ genannt.

Die aktive Zustandsermittlung hat den Vorteil, daß latente Fehler erkannt werden können. Unter latenten Fehlern versteht man Fehler, die bisher nicht erkannt wurden oder die noch keinen Ausfall des Systems verursacht haben („*latent fault*“, vgl. [GrRe93]). Je kleiner das Pollingintervall gewählt wird, desto schneller können latente Fehler erkannt werden und damit die Verfügbarkeit des Gesamtsystems gesteigert werden. Kleine Pollingintervalle haben jedoch den Nachteil, daß System-Ressourcen aktiv verbraucht werden. Insbesondere wird Prozessorzeit benötigt und das Verbindungsnetzwerk belastet.

Der System Monitor des administrativen Werkzeugs verwendet für die Zustandsermittlung die Methoden „*Snapshot*“ und „*Polling*“. Im Polling-Modus kann das Zeitintervall frei gewählt werden, damit ein optimaler Kompromiß zwischen der Ressourcenbelastung und der Latenzzeit von Fehlern gefunden werden kann.

### 6.3.3.1 Knotenorientiert

Jegliche Kommunikation zwischen verschiedenen Adressräumen basiert in APRICOTS auf CORBA.<sup>1</sup> Nach der CORBA-Spezifikation muß auf jedem Rechnerknoten, auf dem ein Server oder ein Client lokalisiert ist, ein ORB existieren. Der ORB hat die Aufgabe, bei der Dienstanforderung eines Clienten die Verbindung zum entsprechenden Server herzustellen. Im Kontext von APRICOTS wird aus diesem Grund die Verfügbarkeit eines Rechnerknotens durch folgende Bedingungen determiniert:

- Die Verfügbarkeit des Rechnerknotens und des Verbindungsnetzwerks zu diesem Knoten. Diese Art der Verfügbarkeit läßt sich beispielsweise mit dem UNIX-Shell-Kommando „*ping <host>*“ feststellen.
- Die Verfügbarkeit des Object Request Brokers (ORB). In Orbix wird der ORB durch den sogenannten „Orbix daemon“ repräsentiert. Die Funktionsfähigkeit kann unter UNIX mit dem Orbix-Shell-Kommando „*pingit <host>*“ ermittelt werden.

Die zweistufige Zustandsermittlung eines Rechnerknotens hat den Vorteil, daß zwischen der Erreichbarkeit bzw. des „Amlebenseins“ eines Hosts und der Verfügbarkeit des Hosts für APRICOTS unterschieden werden kann.

Es ergeben sich folgende mögliche Zustände eines Rechnerknoten:

Zustand	Erläuterung
ALIVE	Rechnerknoten ist physisch erreichbar und reagiert auf Aufforderung.
DOWN	Rechnerknoten bzw. Verbindungsnetzwerk ist ausgefallen.
AVAILABLE	Rechnerknoten ist ALIVE, außerdem existiert dort ein funktionsfähiger Orbix daemon.
IT_DOWN	Rechnerknoten ist ALIVE, jedoch existiert dort kein funktionsfähiger Orbix daemon.

Tabelle 6.1: Mögliche Zustände eines Rechnerknotens

#### Zustandsermittlung von Rechnerknoten

<sup>1</sup> Es gibt eine Ausnahme zu diesem Grundsatz: seit der Verfügbarkeit von JDBC gibt es die Möglichkeit, eine direkte Verbindung zu einer Datenbank herzustellen (vgl. Abschnitt 6.7).

Der Soll-Zustand des Systems verwaltet eine Liste von Rechnerknoten, die am System teilnehmen. Die Liste enthält symbolische Rechnernamen bzw. IP-Adressen. Die Zustände „ALIVE“ bzw. „DOWN“ können durch das Unix-Shell-Kommando „*ping <hostname>*“ ermittelt werden. Das Kommando sendet ein ICMP-Paket (Internet Control Message Protocol) an den zu untersuchenden Rechnerknoten. Antwortet dieser, wird der Exitstatus auf 0 gesetzt, sonst auf 1.

Ist der zu untersuchende Knoten erreichbar und antwortet dieser, dann kann in einem zweiten Schritt geprüft werden, ob dort ein ORB verfügbar ist. Dies geschieht mittels des mit Orbix mitgelieferten Shell-Programms „*pingit*“. Diesem kann als Parameter der Name bzw. die IP-Adresse des zu untersuchenden Rechnerknotens mitgegeben werden. Das Ergebnis kann wiederum durch den Exitstatus des Kommandos ermittelt werden.

### 6.3.3.2 *Komponentenorientiert*

Aus der Sicht einer ConTract-Engine besteht zwischen einem Automated Step Server und einem Benutzer Agenten kein Unterschied. Beide Komponenten werden durch Orbix-Server implementiert. Im Hinblick auf den System Monitor besteht ebenfalls kein Grund, die Zustandsermittlung der beiden Komponenten zu unterscheiden. Step Server werden nur dann vom System Monitor überwacht, wenn der Soll-Zustand „*launched*“ ist.

Die Verfügbarkeit einer Komponente ist zunächst durch das „Binden“ (Orbix-Call *\_bind()*) überprüfbar. Nach [IONA95a] wird beim Binden an ein Orbix-Server-Objekt im Clienten ein sogenanntes „Proxy-Objekt“ angelegt. Das Proxy-Objekt stellt die Referenz des Server-Objekts im Clienten dar. Der *\_bind()*-Call entspricht in gewisser Weise dem Erzeugen eines neuen Objekts, welches durch einen Orbix-Server implementiert wird. Methodenaufrufe des Clients werden dann über das Proxy-Objekt direkt zum Server weitergeleitet und umgekehrt. Durch den Vorgang des Bindens an ein Server-Objekt wird der Serverprozeß nach [IONA95a] nicht angestartet. Erst beim Aufruf von Methoden des Servers wird dieser, falls notwendig, angestartet. Durch das Binden an eine Komponente kann daher nur untersucht werden, ob die Komponente im System vorhanden ist oder nicht. Über die Funktionsfähigkeit der Komponente kann nichts ausgesagt werden.

Aus diesem Grund wurde beschlossen, daß jeder Benutzer Agent und jeder Step Server im IDL-Interface eine Methode anbieten soll, die durch den System Monitor aufgerufen werden kann. Dadurch kann das Anstarten des Serverprozesses ausgelöst und überprüft werden. Ein weiterer Vorteil dieses Ansatzes liegt darin, daß die Komponente in der Methode die Verfügbarkeit der von ihr benötigten Ressourcen überprüfen kann (z.B. Resource Manager). So kann die Verfügbarkeit der Komponente relativ sicher ermittelt werden.

Aus diesen Überlegungen ergeben sich folgende mögliche Ist-Zustände von Komponenten:

Zustand	Erläuterung
PRESENT	Komponente ist vorhanden. <code>_bind()</code> erfolgreich.
UNDISCOVERABLE	Komponente ist nicht auffindbar. <code>_bind()</code> erfolglos.
AVAILABLE	Komponente ist PRESENT und ist aus ihrer Sicht verfügbar.
DOWN	Komponente ist PRESENT, jedoch aus ihrer Sicht nicht verfügbar.

Tabelle 6.2: Mögliche Ist-Zustände einer Komponente

## 6.4 Anforderungen an Benutzer Agenten bzw. Step Server

In diesem Abschnitt werden die Anforderungen an Benutzer Agenten bzw. Step Server im Detail dargestellt. Die System Administration gibt die Möglichkeit, daß ein anderer Benutzer Agent, als der „generische“ von APRICOTS bereitgestellte verwendet werden kann. Die Anforderungen an einen eigens implementierten Benutzer Agenten bzw. Step Server werden hier festgelegt.

### 6.4.1 Aufruf-Syntax

Wie oben beschrieben, startet die System Administration eine neu einzubringende Komponente erstmalig an. Java bietet durch die Methode `exec()` auf einem Runtime-Objekt die Möglichkeit, einen Java-Interpreter-externen Betriebssystemprozeß zu starten. Dabei wird ein „*Process*“-Objekt erzeugt, über welches der Exit-Status der Operation ermittelt werden kann. Der `exec()`-Methode kann das Kommando in Form eines Strings übergeben werden. Nach [Stein96] muß beim erstmaligen Anstarten eines Benutzer Agenten folgender Parameter übergeben werden:

```
-username <UserName>
```

Step Server müssen analog mit folgendem Parameter angestartet werden können:

```
-name <StepServerName>
```

Um den Benutzer Agenten auf dem gewünschten Rechnerknoten anzustarten, wird das UNIX-Shell-Kommando `rsh` benutzt. Dies setzt voraus, daß der Systemadministrator eine Zugangsberechtigung unter dem gleichen Benutzernamen auf der Zielmaschine besitzt. Der

Vorteil von *rsh* liegt darin, daß auf dem entfernten Rechnerknoten eine Shell erzeugt wird ohne die Eingabe eines Passworts zu verlangen.

#### 6.4.2 Aktionen beim erstmaligen Anstarten

Wie oben bereits erwähnt, muß der Benutzer Agent bzw. Step Server beim erstmaligen Anstarten einige Arbeitsschritte durchführen. Die Arbeitsschritte sind das Registrieren des Server-Objekts im entsprechenden ORB und das Eintragen des Namens und der Objekt-Referenz im Naming Service unterhalb von „*APRICOTS.Admin.UserAgents*“ bzw. „*APRICOTS.Admin.StepServers*“.

### 6.5 Die graphische Benutzeroberfläche

Das administrative Werkzeug hat die Aufgabe, einen System Monitor bereitzustellen. Außerdem sollen neue Komponenten in das System eingebracht werden können. Grundsätzlich kann das graphische Userinterface (GUI) funktional- oder objektorientiert entworfen werden. Den Ausschlag für den funktional-orientierten Entwurf gaben folgende Überlegungen:

- Das System sollte überwacht werden können, während beispielsweise ein neuer Benutzer in das System eingebracht wird. Dies kann durch die Verwendung von mehreren verschiedenen Fenstern auf übersichtliche Art und Weise realisiert werden (vgl. Abschnitt 4.2).
- Die Aufgaben des Werkzeugs sind grundsätzlich verschieden. Monitoring ist ein Vorgang von rein informativer Natur, während die Administration das „Eingreifen“ in das System bedeutet. Diese Trennung soll klar in der Benutzeroberfläche ersichtlich sein. Damit ist dem Benutzer jederzeit bewußt, was er gerade tut.
- Intuitiv will der menschliche Benutzer zunächst „etwas tun“, bevor er sich überlegt, womit er das tun will (vgl. [Somm92]).

Das Design der graphischen Benutzeroberfläche orientiert sich daher an den Funktionen, die das Werkzeug zur Verfügung stellt.

#### 6.5.1 Zugang zum administrativen Werkzeug

Das Benutzungskonzept des administrativen Werkzeugs fordert den Zugangsschutz zum Werkzeug. Nur berechtigte Personen sollen das Werkzeug verwenden können. Üblicherweise werden solche Anforderungen durch einen Zugangsdialog verwirklicht. Dort wird der Benutzername und ein Passwort eingegeben. Anhand dieser Identifikation wird die

Berechtigung überprüft. Die Abb. 6.5 zeigt den Zugangsdialog des administrativen Werkzeugs.

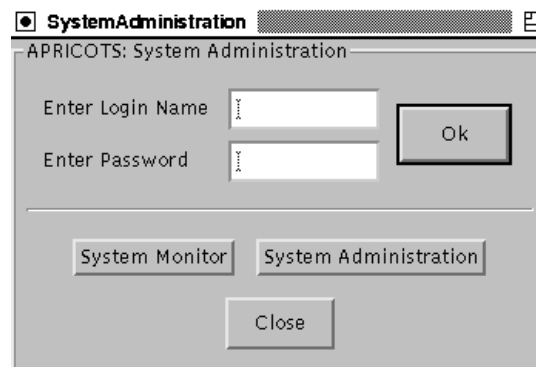


Abb. 6.5: Der Zugang zum administrativen Werkzeug

Bei der Eingabe des Passworts eines Benutzers erscheinen im entsprechenden Feld nur „\*“-Zeichen. Dadurch wird das Passwort eines Benutzers geheim gehalten. Nach der Eingabe der Benutzer-Identifikation überprüft das System die Berechtigungen des Benutzers. Die möglichen Berechtigungen werden in Tabelle 6.3 spezifiziert.

AdminRights	Erläuterung
NONE	Der Benutzer hat keine Zugangsberechtigung zum administrativen Werkzeug
ALL	Der Benutzer darf alle Funktionen des administrativen Werkzeugs nutzen
ADMIN	Der Benutzer darf ausschließlich die System Administration verwenden
MONITOR	Der Benutzer darf ausschließlich den System Monitor verwenden

Tabelle 6.3: Administrationsberechtigungen

Die Granularität der Administrationsberechtigungen könnte weiter verfeinert werden. Denkbar wäre z.B. die weitere Verfeinerung im Bezug auf die System Administration. Beispielsweise könnte das Einbringen eines neuen Rechnerknotens in den Soll-Zustand des Systems unterschieden werden vom Einbringen von neuen Benutzern oder Step Servern. Auf eine derartige, weitere Verfeinerung wird jedoch verzichtet. Die oben gezeigte Aufteilung der Berechtigungen ist durch die folgende Überlegung motiviert: der System

Monitor wirkt nur „lesend“. Er ist daher im Hinblick auf unsachgemäße Verwendung unsensibel im Gegensatz zu der System Administration.

Nachdem anhand der Identifikation des Benutzers dessen Berechtigungen bekannt sind, werden die Schaltflächen „System Monitor“ und „System Administration“ entsprechend „freigegeben“ (*enabled*).

### 6.5.2 Der System Monitor

Der System Monitor visualisiert den gegenwärtigen Zustand der zu überwachenden Komponenten. Nach Abschnitt 4.1 soll er insbesondere in der Lage sein, folgende Fragen zu beantworten:

- Welche Rechnerknoten sind ausgefallen?
- Welche applikationsspezifischen Komponenten sind ausgefallen?
- Welche applikationsspezifischen Komponenten sind auf Rechnerknoten X lokalisiert?
- Welche applikationsspezifischen Komponenten, die auf Rechnerknoten X lokalisiert sind, sind ausgefallen?

Um diese Fragen übersichtlich beantworten zu können, empfiehlt sich die Trennung des System Monitors in eine knotenorientierte und eine komponentenorientierte Sichtweise. Ein Rechnerknoten wird dabei als Komponente angesehen. Die Abb. 6.6 zeigt das Hauptfenster des System Monitors.

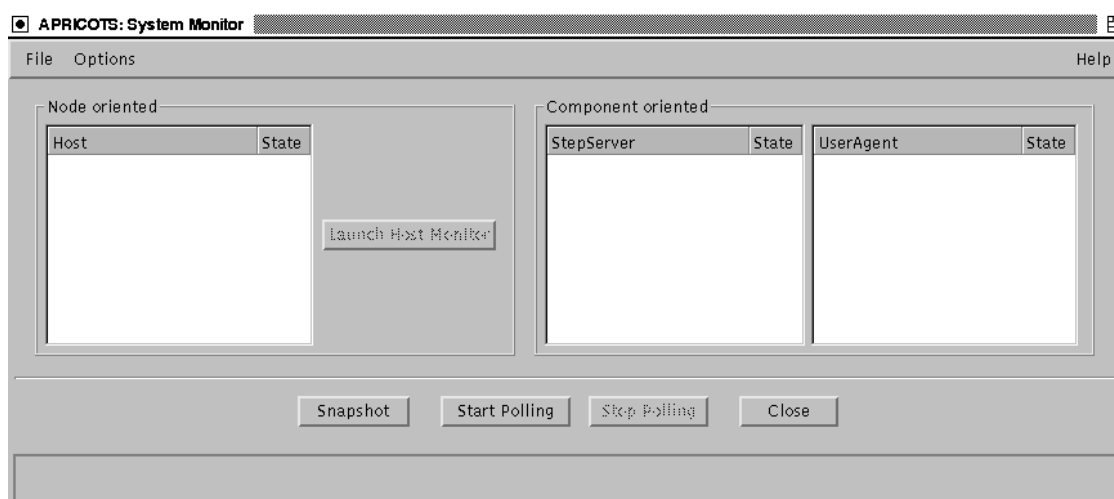


Abb. 6.6: Der System Monitor

Links werden alle am System teilnehmenden Rechnerknoten dargestellt. Der komponentenorientierte Teil (rechts) beinhaltet alle Step Server und Benutzer Agenten, die im System vorhanden sind. Unter dem Menüpunkt „*Options*“ können Rechnerknoten, Step Server und Benutzer Agenten durch jeweils einen „Wildcard-String“ gefiltert werden. So lassen sich gezielt nur diejenigen Komponenten überwachen, die gerade gewünscht werden. Außerdem kann dort das Polling-Intervall eingestellt werden.

Damit können die ersten beiden der oben genannten Fragen beantwortet werden. Um die letzten beiden zu beantworten existieren mehrere Möglichkeiten. Denkbar wäre das Selektieren eines Rechnerknotens. Daraufhin werden im komponentenorientierten Teil nur diejenigen Komponenten angezeigt, welche auf dem selektierten Rechnerknoten lokalisiert sind. Diese Methode hat zur Folge, daß zu einem Zeitpunkt nur ein Rechnerknoten angezeigt werden kann. Das Problem könnte gelöst werden, indem in der Liste der Rechnerknoten mehrere Selektionen zugelassen werden. Im komponentenorientierten Teil müßten dann die Komponenten jeweils mit dem Rechnerknoten markiert werden, auf dem sie lokalisiert sind. So wird die Zuordnung zwischen einer Komponente und einem Rechnerknoten sichtbar. Diese Methode führt zu einer sehr unübersichtlichen Darstellung und scheidet deshalb aus.

Die zweite Möglichkeit besteht im Entwerfen eines „Host-Monitors“ in einem eigenen Fenster. Der Host-Monitor ist genau einem Rechnerknoten zugeordnet und visualisiert die Zustände aller Komponenten, die auf diesem Host lokalisiert sind. Es ist leicht möglich, mehrere Host-Monitore für verschiedene Rechnerknoten zu starten. Die Informationen werden dadurch auf übersichtliche Art und Weise dargestellt. Dieser Ansatz wurde hier realisiert. Nachdem in der Liste der Rechnerknoten ein Host selektiert ist, kann über die Schaltfläche „Launch Host Monitor“ der zugehörige Host-Monitor gestartet werden. Der Host-Monitor arbeitet im Polling-Modus. Damit können die letzten beiden Fragen auf übersichtlicher Weise beantwortet werden. Die Abb. 6.7 zeigt das Layout des Host-Monitors.

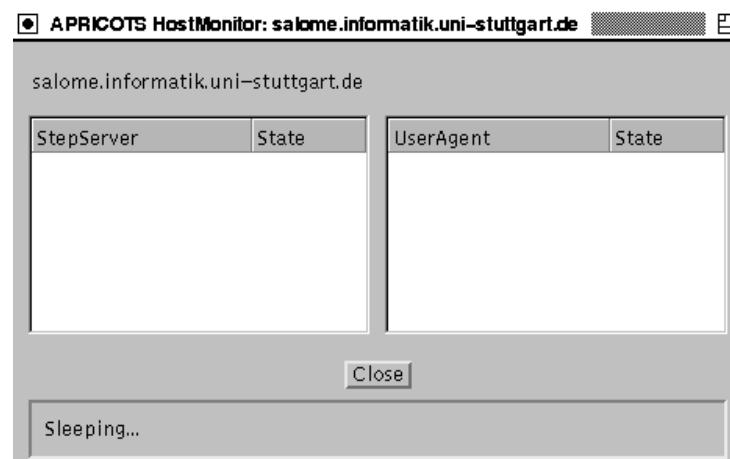


Abb. 6.7: Der Host Monitor

### 6.5.3 Die System Administration

In der System Administration können Benutzer Agenten und Step Server in das System eingebracht werden. Außerdem wird eine Liste von Rechnerknoten für den System Monitor verwaltet. Aus der Benutzersicht wird in der System Administration für jede zu verwaltende Komponente die gleiche Funktionalität benötigt. Die Objekte, die verwaltet werden, unterscheiden sich jedoch in einigen Attributen. Diese Eigenschaften können gut durch ein „Karteikarten“-Layout dargestellt werden. Die Ähnlichkeiten werden dadurch angedeutet, daß nur ein Fenster verwendet wird, in dem sich die einzelnen „Karteikarten“ befinden. Die spezifischen Eigenschaften werden dann beim „Aufklappen“ der jeweiligen Karteikarte sichtbar. Mit diesem Ansatz wird der Benutzer klar geführt, da er nur diejenigen Eingaben machen kann, die für das gewählte Objekt sinnvoll sind. Außerdem wird die sehr übersichtliche Darstellung gewährleistet.

Die Abb. 6.8 zeigt stellvertretend für die gesamte System Administration das Layout der Benutzer Administration.

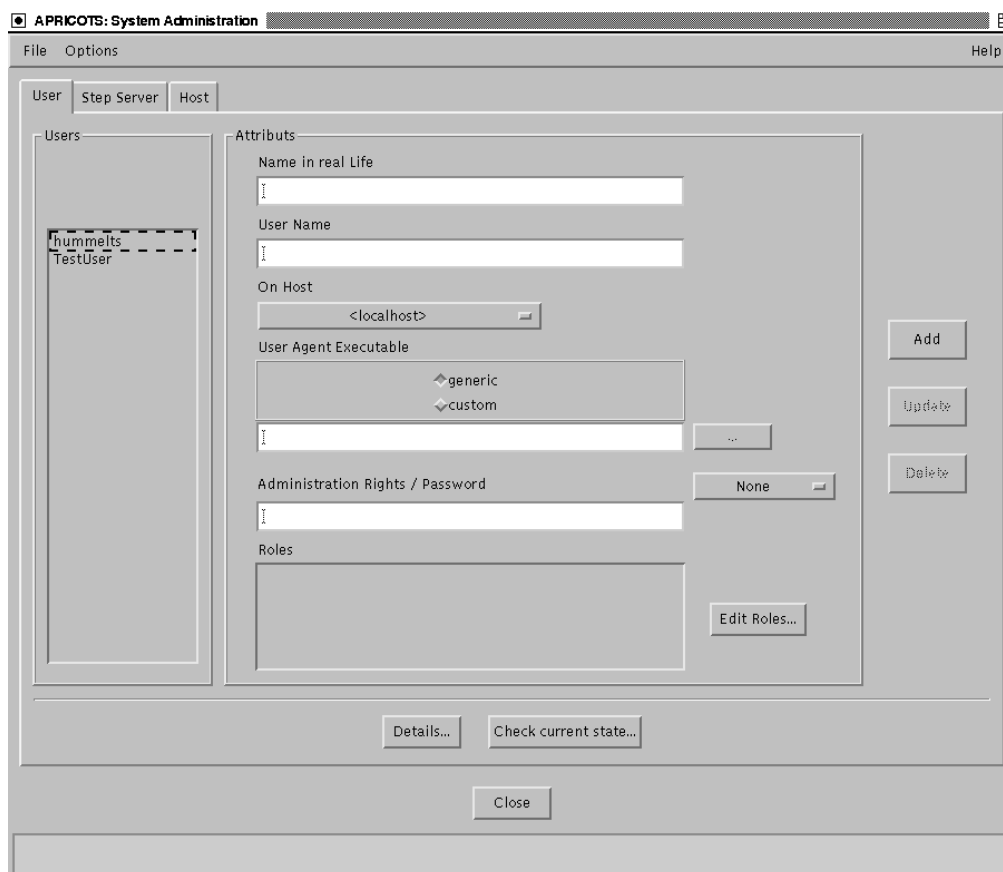


Abb. 6.8: Die Benutzer Administration

In der Benutzer Administration befindet sich links eine Liste aller in das System eingebrachter Benutzer. Jeder Benutzer wird durch einen Eintrag repräsentiert. Das in der Liste dargestellte Attribut ist der eindeutige Name des Benutzers. Unter dem Menüpunkt „Options“ kann auch hier die Liste durch einen „Wildcard-String“ gefiltert werden.

Bei der Selektion eines Benutzers werden die Attribut-Felder mit den entsprechenden Daten ausgefüllt und können bearbeitet werden. Jeweils mögliche Funktionen werden durch „enabled buttons“ angeboten. Momentan nicht verfügbare Funktionen werden schattiert dargestellt. So wird der Benutzer geführt und Fehler werden vermieden.

Jedem Benutzer können mehrere Rollen zugeordnet werden. Die Rollen eines Benutzers werden in der Liste „Roles“ dargestellt. Die Editierung von Rollen kann durch die Betätigung der Schaltfläche „Edit Roles“ initiiert werden. Aus Übersichtlichkeitsgründen wurde dafür ein eigenes Fenster entworfen, das in Abb. 6.9 dargestellt ist. Während des Editierens von Rollen muß verhindert werden, daß der Administrator z.B. einen anderen Benutzer wählt. Um den klaren Ablauf sicherzustellen, werden während des Editierens von Rollen alle anderen Funktionen gesperrt.

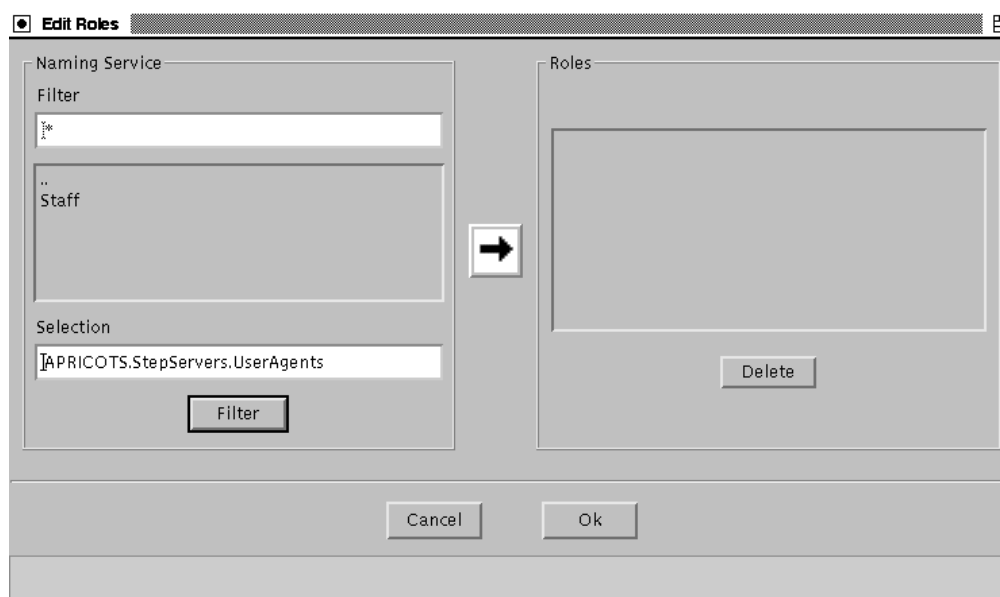


Abb. 6.9: Das Editieren der Rollen

Das Zuordnen von Rollen an einen Benutzer oder Step Server besteht aus dem Auswählen von Kontext-Knoten aus dem Naming Service. Es ist sehr mühsam und fehlerträchtig, diese Namen „von Hand“ eintragen zu müssen. Aus diesem Grund wird im linken Teil des Fensters die Möglichkeit gegeben, sich durch den Namensraum per Mausklick zu bewegen. Das Layout dieses Teils ist dem einer gängigen „File Selection Box“ angelehnt. Das Feld „Selection“ zeigt den gegenwärtig gewählten Namen. In der Liste darüber werden alle Verzweigungsmöglichkeiten von der aktuellen Selektion aus angezeigt. Es kann so per Maus durch den Namensraum von APRICOTS „navigiert“ werden. Wurde der Name einer

Rolle gefunden, die dem Benutzer zugeordnet werden soll, so kann diese durch die Schaltfläche „→“ der Liste rechts hinzugefügt werden. Soll eine neue Rolle im Namensraum von APRICOTS erzeugt werden, so kann dies durch die Eingabe des Namens in das Feld „*Selection*“ und dem anschließenden Betätigen der Schaltfläche „→“ erfolgen. Das System erkennt, daß dieser Name noch nicht existiert und erzeugt diesen nach der Bestätigung durch den Administrator.

Die Administration von Step Servern wird analog zu Benutzer Agenten durchgeführt und wird deshalb hier nicht explizit dargestellt. Auch die Liste von Rechnerknoten läßt sich ähnlich verwalten. Ein Unterschied bei den Rechnerknoten besteht allerdings im Fehlen der Rollenbearbeitung.

## **6.6 Der Einsatz von Multi-Threading im administrativen Werkzeug**

Threads sind Bestandteil der Programmiersprache Java. Es stellt sich die Frage, in welchen Belangen des administrativen Werkzeugs diese Fähigkeit genutzt werden könnte.

### **6.6.1 Administration**

Bei dem Einbringen von Komponenten muß - wie oben gesehen - eine bestimmte Reihenfolge eingehalten werden. Dies macht den Einsatz von Multi-Threading sinnlos. Der einzige Bereich innerhalb der System Administration, in dem Multi-Threading denkbar wäre, ist der Zugriff auf persistenten Speicher. Der Datenzugriff könnte in einem eigenen Thread ablaufen, so daß der Benutzer während des Zugriffs weiterhin mit dem System interagieren könnte. Auch hier ist der Sinn in Frage gestellt. Zum einen kann erwartet werden, daß die Antwortzeit des Systems auf Datenzugriffe relativ klein ist, so daß die Parallelisierung kaum einen Vorteil bringt. Zum anderen läßt sich während des Datenzugriffs keine Aufgabe identifizieren, die sinnvollerweise parallel dazu ablaufen könnte. Dem sehr geringen Nutzen steht der relativ große Aufwand der Realisierung gegenüber. Das Hauptproblem stellt dabei die Synchronisation dar. Aus diesen Gründen kommt in der System Administration kein Multi-Threading zum Einsatz.

### **6.6.2 Monitoring**

Im Bereich des System Monitors kommt Multi-Threading an zwei Stellen zum Einsatz.

Die Realisierung des Polling-Betriebs kann auf zwei Arten erfolgen. Beim Eintreten in den Polling-Betrieb wird eine Endlosschleife betreten. Innerhalb der Schleife werden folgende Schritte durchgeführt: Zustand ermitteln, Zustand anzeigen, warten, bis Pollingintervall abgelaufen ist, Zustand ermitteln etc.. Interaktionen des Benutzers können so nicht mehr vom System angenommen und behandelt werden. Eine Lösung des Problems besteht in der Abfrage von Benutzerinteraktionen als zusätzlicher Schritt in der Endlosschleife. Dies hat jedoch den Nachteil, daß Benutzerinteraktionen bis zum gesamten Pollingintervall verzögert werden können. Dies ist nicht tragbar.

Die zweite Möglichkeit, den Polling-Betrieb zu realisieren, liegt in der Implementierung eines „Timers“. Der Timer läuft in einem eigenen Thread und sendet in regelmäßigen Zeitabständen ein Signal an die Oberfläche. Daraufhin ermittelt die Oberfläche den Zustand des Systems, zeigt diesen an und wartet auf Benutzerinteraktionen oder das nächste Timer-Signal. Der Nachteil der oben genannten Lösung ist damit aufgehoben.

Dieser Ansatz ist bereits deutlich besser, als die oben gezeigte Lösung, da während des Wartens auf das nächste Signal Benutzerinteraktionen behandelt werden können. Diese Lösung ist jedoch noch immer nicht optimal. Beispielsweise kann die Zustandsermittlung von Rechnerknoten mittels „ping <host>“ im Fehlerfall relativ lange dauern. Insbesondere wenn mehrere Rechnerknoten nicht verfügbar sind, kann die Benutzerinteraktion längere Zeit blockiert sein. Dieses Problem kann gelöst werden, indem die Zustandsermittlung in einem eigenen Thread abläuft. Diese Methode löst die genannten Probleme. Sie kommt deshalb im System Monitor zum Einsatz.

Multi-Threading kommt an einer weiteren Stelle im System Monitor zum Einsatz. In Abschnitt 6.5 wurde der Host Monitor dargestellt. Der Host Monitor visualisiert die Zustände von denjenigen Benutzer Agenten und Step Server, welche auf einem bestimmten Rechnerknoten lokalisiert sind. Zu einem Zeitpunkt können mehrere Host Monitore gestartet sein. Um die oben genannten Probleme zu umgehen, läuft jeder Host Monitor in einem eigenen Thread ab.

Synchronisationsprobleme treten bei dieser Art der Thread-Verwendung nicht auf, da die hier verwandten Threads nur lesend agieren. Änderungsanomalien sind damit ausgeschlossen.

## 6.7 Recovery-Fähigkeiten

Das administrative Werkzeug benötigt stabilen Speicher, um den Soll-Zustand des Systems wiederherstellen zu können. Zu Beginn dieser Arbeit gab es für das administrative Werkzeug nur die Möglichkeit, Daten über den Speicher Agenten persistent zu speichern (vgl. Abschnitt 6.2). Inzwischen ist JDBC verfügbar. JDBC ist eine API, die den Datenbankzugriff für Java-Applikationen ermöglicht.

### 6.7.1 Speicher Agent (Storage Agent)

Für die Verwendung des Speicher Agenten muß eine Schnittstelle in IDL definiert werden. Die exakte Schnittstellendefinition, die für das administrative Werkzeug entworfen wurde, findet sich im Anhang dieser Arbeit. Auf der anderen Seite muß der Speicher Agent die IDL-Schnittstelle implementieren. Dies bedeutet Anpassungsarbeit durch den Programmierer des Speicher Agenten.

### 6.7.2 Direkte Datenbankanbindung mit JDBC

Die Verwendung des Speicher Agenten ist nicht sehr sinnvoll für das Speichern des Soll-Zustands des Systems, da keine andere APRICOTS-Komponente die Daten des administrativen Werkzeugs nutzt. Der durch den Speicher Agenten erzeugte „Overhead“ kann durch direkte Datenbankanfragen umgangen werden. Allerdings muß die Lokationstransparenz der Datenbank gewährleistet sein.

JDBC ist ein SQL-Interface für Java. Der Schwerpunkt liegt in der Ausführung von SQL-Anfragen und dem Erhalten von Ergebnissen. Auch bei JDBC wird der Ansatz der Plattformunabhängigkeit verwirklicht. JDBC bildet das „*call-level*“ - Interface zu einem darunter liegenden DBMS-Treiber (Data Base Management Systems). D.h. JDBC stellt ein standardisiertes SQL-Interface für die Verwendung von DBMS zur Verfügung. Diese Schnittstelle ist unabhängig vom tatsächlich verwandten DBMS. Diese Unabhängigkeit wird durch das Zwischenschieben des sogenannten „DBMS-Driver“ zwischen das API und das DBMS erreicht.

Der JDBC-Manager ermöglicht die Verwendung von „entfernten“ Datenbanken. Bevor SQL-Anfragen an ein DBMS abgesetzt werden können, muß eine Verbindung zu diesem hergestellt werden. Der JDBC-Manager hat Treiber zur Verfügung, die verschiedene Protokolle implementieren (z.B. ODBC). Dadurch können Verbindungen zu entfernten DBMS hergestellt werden.

JDBC erfüllt daher alle Anforderungen, die an den stabilen Speicher gestellt werden. Der Entwurf des Datenbankschemas für das administrative Werkzeug befindet sich im Anhang dieser Arbeit.

## 7 Abschließende Diskussion

### 7.1 *Praktikabilität des Werkzeugs*

Mit Hilfe des administrativen Werkzeugs lassen sich auf einfache Art und Weise individuelle Applikationen unter APRICOTS konfigurieren. Das administrative Werkzeug übernimmt alle Aufgabenschritte, die beim Einbringen von individuellen Komponenten in das System notwendig sind. Diese Schritte müßten ohne das Werkzeug an der Kommandozeile erledigt werden. Attribute von Komponenten, die einer relativ hohen Fluktuationsrate unterliegen (z.B. die Rollen eines Benutzers), lassen sich leicht anpassen.

Der System Monitor des administrativen Werkzeugs ermöglicht einen umfassenden Überblick über den gegenwärtigen Systemzustand. Sowohl die knoten- als auch die komponentenorientierte Betrachtungsweise ist möglich. So können Ausfälle von Komponenten schnell erkannt und beseitigt werden. Damit läßt sich die Verfügbarkeit des Gesamtsystems erhöhen. Der System Monitor kann einen „Snapshot“ des Systemzustands erstellen. Für die kontinuierliche Zustandsermittlung steht ein „Polling“-Mechanismus zur Verfügung. Das Zeitintervall kann frei gewählt werden, um individuellen Bedürfnissen gerecht zu werden.

### 7.2 *Ausblick*

#### 7.2.1 *Lastbalancierung und Migration*

APRICOTS ist als „Middleware“ für ein verteiltes System zu sehen. Auf APRICOTS können große, verteilte Applikationen aufgebaut werden. Der Aspekt der Lastbalancierung kann im Namensraum von APRICOTS durch die Rollenverteilung berücksichtigt werden. Beliebigen vielen Step Servern kann eine einzige Rolle zugeordnet werden, so daß die ConTract-Engine einen geeigneten Server auswählen kann. Um eine gute Performance des Systems erreichen zu können, ist die Information der Lastverteilung für den administrativen Benutzer ebenfalls von Bedeutung. Die Ermittlung und Visualisierung der Systemlast könnte deshalb eine weitere Aufgabe des administrativen Werkzeugs sein. Sehr stark frequentierte Komponenten („Hotspots“) ließen sich so identifizieren und beseitigen.

Mit der Lastbalancierung in engem Zusammenhang steht die Migration. Migration ist eine Methode in verteilten Systemen, die Last auszugleichen. Es gibt jedoch andere Aspekte, die zur Migration führen. Beispielsweise ist die räumliche Verteilung der Daten in Richtung ihres Bedarfsorts ein solcher Aspekt. Im Bezug auf APRICOTS könnte daher eine weitere Aufgabe des administrativen Werkzeugs die Verteilung des Namensraums auf verschiedene Rechnerknoten sein. Beispielsweise könnten die Rollen einer Abteilung auf einen Abteilungsrechner ausgelagert werden, anstatt zentral im Naming Service gehalten

zu werden. Nach [IONA96] kann bereits in der vorliegenden Version des Naming Services der Namensraum auf verschiedene Rechnerknoten verteilt werden. Der Vorteil der Verteilung des Namensraums liegt im Verteilen des „*single-point-of-failure*“ auf mehrere Rechnerknoten. Die Verfügbarkeit des Systems kann damit erhöht werden. Außerdem dürfte sich die Verteilung positiv auf die Antwortzeit des Naming Services auswirken, da nicht das vollständige System auf einen einzigen Naming Service zurückgreift. Die vorliegende (Beta-) Version des Naming Services hat nach [IONA96] allerdings noch einen Fehler im Bezug auf die Verteilung: der Versuch, einen Namen aufzulösen (Methode „*resolve()*“) führt im Falle, daß der Name in einem Kontext eines Naming Servers beginnt, dann auf einem anderen Server fortgesetzt wird und von dort wieder zum ursprünglichen Server zurückkehrt, zu einem Deadlock. Diese Einschränkung soll in der Vollversion des Orbix Naming Services behoben sein.

### 7.2.2 System Monitor Agent

Der Benutzer Agent in APRICOTS dient in gewisser Weise als Mailbox für den menschlichen Benutzer. Der Benutzer Agent ist im System auch vorhanden, wenn der Benutzer nicht arbeitet. Aufträge werden während der Abwesenheit des Benutzers in dessen Benutzer Agenten gesammelt. Dadurch kann die Abarbeitung einer ConTract-Instanz sofort fortgesetzt werden, wenn der Benutzer Agent asynchron in Anspruch genommen wird. Diese Idee läßt sich ebenfalls auf das Monitoring des Systems übertragen. Es könnte wünschenswert sein, das System zu überwachen, während das administrative Werkzeug nicht gestartet ist. Ein „System Monitor Agent“ könnte diese Aufgabe übernehmen. Dabei werden im Falle der Abwesenheit des Administrators bzw. des administrativen Werkzeugs die Zustandsdaten auf stabilem Speicher protokolliert. Der Administrator kann dann jederzeit den Verlauf des Systemzustands nachvollziehen.

Die Realisierung dieser Idee bedeutet, daß sich die Architektur des administrativen Werkzeugs grundlegend ändert. Von den drei Basis-Verteilungsmethoden nach [Clie95] entstünde die Version „kooperative Verarbeitung („distributed processing“), da sowohl der Agent, als auch die Benutzeroberfläche Funktionalität enthält (vgl. Abschnitt 6.2). Die Realisierung in der Programmiersprache Java ist zu Anfang dieser Arbeit nicht möglich gewesen, da OrbixWeb nur als Client agieren konnte. Der „System Monitor Agent“ muß jedoch als Server implementiert werden.

## 8 Literatur

### *World Wide Web:*

Aufgrund der relativ hohen Fluktuationsrate von WWW-Adressen werden hier nur die wichtigsten Einstiegspunkte genannt.

Java: <http://www.javasoft.com>

Java WorkShop: <http://www.sun.com/developer-products/java>

CORBA: <http://www.omg.org>

IONA Technologies Ltd.: <http://www.iona.com>

### *Newsgroups:*

Java: [comp.lang.java.programmer](mailto:comp.lang.java.programmer)  
[comp.lang.java.api](mailto:comp.lang.java.api)  
[comp.lang.java.advocacy](mailto:comp.lang.java.advocacy)  
[comp.lang.java.misc](mailto:comp.lang.java.misc)

### *Literaturstellen*

[Bay96] Bayer, H.: Ein elektronischer Eingangskorb in Java, Studienarbeit Nr. 1559, Fakultät Informatik, Universität Stuttgart, 1996

[Clie95] Das Client / Server Planungshandbuch, Ausgabe 1995, Jenz & Partner GmbH, Erlensee, 1995

[Flan96] Flanagan, D.: Java in a Nutshell, O'Reilly & Associates, 1996

[Glass93] Glass, G.: UNIX for Programmers and Users - A Complete Guide, Prentice Hall, 1993

[GosGil96] Gosling, J., McGilton, H.: The Java Language Environment: A white paper, Sun Microsystems, Inc., Mai 1996

- 
- [GrRe93] Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers, 1993
- [Gunz91] Gunzenhäuser, R.: Interaktive Systeme, Skriptum zur Vorlesung Interaktive und Intelligente Systeme, Universität Stuttgart, WS 91/92
- [HaCa96] Hamilton, G., Cattell, R.: JDBC: A Java SQL API, Version 1.00, Sun Microsystems, Inc., Juni 1996
- [IONA95a] Orbix 2: Programming Guide, Release 2.0, IONA Technologies Ltd., November 1995
- [IONA95b] Orbix 2: Reference Guide, Release 2.0, IONA Technologies Ltd., November 1995
- [IONA96] Naming Service, Release 1.0 (beta), IONA Technologies Ltd., März 1996
- [Jab195] Jablonski, S.: Workflow-Management-Systeme: Modellierung und Architektur, Thomson Publ., 1995
- [McB191] McCarthy, J.C., Bluestein, W.M.: The Computing Strategy Report: Workflow's Progress, Forrester Research Inc., Cambridge, Mass., Oct. 1991
- [OMG94] Object Transaction Service, Object Management Group, Dokument 94.8.4, August 1994
- [OMG95] The Common Object Request Broker: Architecture and Specification, Revision 2.0, Object Management Group, Juli 1995
- [ÖzDaVa94] Özsu, M.T., Dayal, U., Valduriez, P.: Distributed object management, Morgan Kaufmann Publishers, 1994

- 
- [ReScWä92] Reuter, A., Schwenkreis, F., Wächter, H.: Zuverlässige Abwicklung großer verteilter Anwendungen mit ConTracts - Architektur einer Prototyp-implementierung, Bayer, Härder, Lockemann (Hrsg.) in Objektbanken für Experten, Springer, 1992
- [Seif96] Seifert, J.: Zuverlässige Workflowbearbeitung auf der Basis von OTS, Diplomarbeit Nr. 1404, Universität Stuttgart, Fakultät für Informatik, 1996
- [Shnei92] Shneiderman, B.: Designing the user interface: strategies for effective human-computer-interaction, 2nd. ed., Addison-Wesley, 1992
- [Somm92] Sommerville, I.: Software Engineering, Addison-Wesley, 1992
- [Stein96] Steinmetz, M.: Ein Agent als Informationsschnittstelle für einen elektronischen Eingangskorb, Studenarbeit Nr. 1560, Fakultät Informatik, Universität Stuttgart, 1996
- [Wald94] Waldruff, A.: Lean Management, Neuer Wein oder neuer Schlauch?, in Zahn, E.: Skriptum zum Kolloquium Logistik und prozeßorientiertes Management, Universität Stuttgart, WS 1994/1995
- [WäRe91] Wächter, H., Reuter, A.: The ConTract Model, Computer Science Report No. 8/91, September 1991
- [Zahn91] Zahn, E.: Innovation als Strategie in turbulenter Zeit, in Zahn, E. (Hrsg.): Auf der Suche nach Erfolgspotentialen. Strategische Optionen in turbulenter Zeit, Stuttgart, 1991
- [Zahn96] Zahn, E.: Strategische Erneuerung für den globalen Wettbewerb, in Zahn, E. (Hrsg.): Strategische Erneuerung für den globalen Wettbewerb, Stuttgart 1996

## 9 Anhang

### 9.1 Die IDL-Schnittstelle zum Speicher Agenten

Zu Beginn dieser Arbeit wurde der Speicher Agent für die Speicherung des Soll-Zustands des Systems verwendet. Im folgenden wird der Entwurf der Schnittstelle zum Speicher Agenten dargestellt.

```
// IDL

#include dbagent.idl
interface SYSAdminStorageAgent : APRICOTS_DBAgent
{
    // type for the n roles of an UserAgent or StepServer
    typedef sequence<APRICOTS_Definitions::NameString > Roles;

    // type for a list of strings
    typedef sequence<string> NameList;

    // types for system settings
    struct MonitorSettings {
        boolean WatchHosts;           // flag whether hosts should be watched
        boolean WatchStepServers;     // analogous for StepServers
        boolean WatchUserAgents;     // analogous for UserAgents
        short PollingIntervall;       // Pollingintervall in msec
        string InitialNSHost;         // host where NamingService is located
        string HostFilter;           // watch hosts matching this wildcard-string
        string SSFilter;             // analogous for StepServers
        string UserFilter;           // analogous for UserAgents
    };

    struct AdminSettings {
        string InitialNSHost;         // host where NamingService is located
        string SSPutitCommand;       // Parameters for first StepServer launch
        string UserAgentGenericExe;  // path and name of generic UserAgent
        string UserAgentPutitCommand; // Parameters for first UserAgent launch
        string HostFilter;           // show only hosts at Sys.Admin. matching that
                                    // wildcard-string
        string SSFilter;             // analogous for StepServers
        string UserFilter;           // analogous for UserAgents
    };

    struct SystemSettings {
        MonitorSettings monitor;
        AdminSettings admin;
    };

    // types for Users
    enum Admin_Rights {
        AR_NONE,           // no rights
        AR_ALL,            // all rights
        AR_MONITOR,       // only System Monitor
        AR_ADMIN          // only System Administration
    };

    struct User {
        string Username;    // Username (key!)
```

```

    string      NameInRealLife;
    boolean     GenericUserAgent; // flag: use generic UserAgent
    string      UserAgentExe;     // path+name of UserAgent if not generic
    string      AdminRights      rights; // rights to user Admin.Tool
    string      passwd;          // password for Admin.Tool
    Roles       roles;           // roles of the user
    string      onHost;          // UserAgent located on this host
};

// type for hosts
struct Host {
    string      name; // symbolic host name
    string      ip;   // ip-address of host
};

// types for StepServers
enum SState {
    SS_DEFINED, // ready to be launched
    SS_LAUNCHED, // available
    SS_STOPPED, // stopped
    SS_DELETED // removed from system
};

struct StepServer {
    string      Name; // name of StepServer (key!)
    SState      state; // shall-state of StepServer
    string      executable; // path+name of StepServer's executable
    Roles       roles; // roles of StepServer
    string      onHost; // located on host
};

// ***** METHODS *****

// methods for system settings
void SaveSystemSettings (in SessionId session, in SystemSettings settings)
    raises (APRICOTS_Definitions::DBError);

void GetSystemSettings (in SessionId session)
    raises (APRICOTS_Definitions::DBError);

// methods for users
void AddUser (in SessionId session, in User user)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::Reject);

void DeleteUser (in SessionId session, in string UserName)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

void UpdateUser (in SessionId session, in string UserName, in User UserData)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

NameList GetUserList (in SessionId session, in string NameFilter, in string HostFilter)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

User GetUser (in SessionId session, in string UserName)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

boolean UserExists (in SessionId session, in string UserName)
    raises (APRICOTS_Definitions::DBError);

// methods for StepServers
void AddStepServer (in SessionId session, in StepServer stepserver)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::Reject);

```

---

```
void DeleteStepServer (in SessionId session, in string StepServerName)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

void UpdateStepServer (in SessionId session, in string SSName, in StepServer SSData)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

NameList GetStepServerList (in SessionId session, in string NameFilter, in string HostFilter)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

StepServer GetStepServer (in SessionId session, in string SSName)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

boolean StepServerExists (in SessionId session, in string StepServerName)
    raises (APRICOTS_Definitions::DBError);

// methods for hosts
void AddHost (in SessionId session, in Host host)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::Reject);

void DeleteHost (in SessionId session, in string HostName)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

NameList GetHostList (in SessionId session, in string NameFilter)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

Host GetHost (in SessionId session, in string HostName)
    raises (APRICOTS_Definitions::DBError, APRICOTS_Definitions::NotFound);

boolean HostExists(in SessionId session, in string HostName)
    raises (APRICOTS_Definitions::DBError);

}
```

## 9.2 Der Entwurf der administrativen Datenbank

Mit der Verfügbarkeit von JDBC wurde der direkte Zugriff auf eine Datenbank möglich. Hier wird nun der Entwurf der administrativen Datenbank gezeigt. Da sich die Attribute von Step Servern und Benutzer Agenten nur geringfügig unterscheiden, wird für diese Komponenten die gemeinsame Relation *SysAdmin\_Components* verwendet.

Attribut	Typ	Erläuterung
<u>Name</u>	varchar	Eindeutiger Name der Komponente
<u>UserAgent</u>	boolean	Flag, ob BenutzerAgent oder StepServer
OnHost	varchar	Komponente ist lokalisiert auf diesem Rechnerknoten
Generic	boolean	Flag, ob generische Implementierung verwendet werden soll
Executable	string	Pfad+Dateiname der ausführbaren Programmdatei
SState	smallint	Soll-Zustand, falls StepServer
NameInRealLife	varchar	Benutzername, falls Benutzer Agent
Rights	smallint	Administrationsrechte, falls Benutzer Agent
Passwd	varchar	Passwort, falls Benutzer Agent

Tabelle 9.1: Die Relation *SysAdmin\_Components*

Die Rollen einer Komponente werden in der Relation *SysAdmin\_ComponentRoles* gespeichert.

Attribut	Typ	Erläuterung
<u>Name</u>	varchar	Eindeutiger Name der Komponente
<u>UserAgent</u>	boolean	Flag, ob Benutzer Agent oder Step Server
<u>Role</u>	varchar	Rollen-String

Tabelle 9.2: Die Relation *SysAdmin\_ComponentRoles*

Dieses Relationenschema könnte weiter normalisiert werden. Es wird an dieser Stelle jedoch darauf verzichtet, da der geforderte Zweck durch das Schema ausreichend erfüllt wird. Die Tabellen 9.3 (*SysAdmin\_Hosts*) und 9.4 (*SysAdmin\_Settings*) vervollständigen die administrative Datenbank.

Attribut	Typ	Erläuterung
HostName	varchar	Symbolischer Name des Rechnerknotens. Wenn unbekannt, IP-Adresse
IP_Address	char(15)	IP-Adresse des Rechnerknotens

Tabelle 9.3: Die Relation *SysAdmin\_Hosts*

Attribut	Typ	Erläuterung
WatchHosts	boolean	Flag, ob Rechnerknoten überwacht werden sollen
WatchStepServers	boolean	analog für Step Server
WatchUserAgents	boolean	analog für Benutzer Agenten
PollingIntervall	int	Polling-Intervall in msec.
InitialNSHost	varchar	Name des Naming Service Hosts
Monitor_HostFilter	varchar	Wildcard-String für zu überwachende Hosts
Monitor_SSFilter	varchar	analog für Step Server
Monitor_UserFilter	varchar	analog für Benutzer Agenten
SS_PutitCommand	varchar	Parameters für erstes Anstarten eines Step Servers
UA_PutitCommand	varchar	analog für Benutzer Agenten
UserAgentGenericExe	varchar	Pfad+Name der ausführbaren Programmdatei des generischen Benutzer Agenten
Admin_HostFilter	varchar	Wildcard-String für zu administrierende Hosts
Admin_SSFilter	varchar	analog für Step Server
Admin_UAFilter	varchar	analog für Benutzer Agenten

Tabelle 9.4: Die Relation *SysAdmin\_Settings*

---

Hiermit versichere ich, daß ich diese Arbeit selbständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.

Tobias Hummel

---