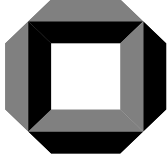

Universität Karlsruhe
Fakultät für Informatik
76128 Karlsruhe

Daten in verteilten Systemen

Seminar SS 1995 & WS 1995/96

Herausgeber:
Arnd G. Grosse
Dietmar A. Kottmann

Universität Karlsruhe
Institut für Telematik

Interner Bericht 38/96

Zusammenfassung

Verteilte Systeme gewinnen zunehmend an Bedeutung für moderne Anwendungen. Um sie geeignet unterstützen zu können, bedarf es innovativer Mechanismen zur Fehlertoleranz, zur Integration von Datenbanken und zur Verteilung von Daten, bei gleichzeitiger Bereitstellung von geeigneten Synchronisationsverfahren. Diese Mechanismen waren Gegenstand des Seminars "Daten in verteilten Systemen", das im Sommersemester 1995 sowie im Wintersemester 1995/96 am Institut für Telematik der Universität Karlsruhe abgehalten wurde.

Abstract

Distributed systems become more and more important for building modern applications. Integrating solutions for fault tolerance to database access, distributing of data and finding agreements in a distributed environment into solide mechanisms are henceforth the cornerstone for the efficient realization of distributed systems and applications. These mechanism have been studied in detail in the seminar "Data in Distributed Systems", which was held at the Institute of Telematics of the University of Karlsruhe in summer 1995 and winter 1995/96.

Vorwort

Das Anwendungsgebiet der Informationstechnik unterliegt einem stetigem und schnellen Fortschritt, welches sich durch die zunehmende informationstechnische Integration von Anwendungen und der globalen Vernetzung von Rechnersystemen bei gleichzeitiger, starker Tendenz zur Dezentralisierung von Anwendungsteilen ausdrückt. Damit einher geht die Verteilung der zur Bearbeitung benötigten Informationen und die Notwendigkeit zur Bereitstellung von Methoden und Mechanismen zu ihrer Unterstützung. Insbesondere dieses Spannungsfeld zwischen der integrierten Bearbeitung von Aufgaben einerseits bei gleichzeitiger Verteilung der Aufgabenträger anderseits, erfordert die Erarbeitung und Bereitstellung neuartiger Lösungskonzepte seitens der Informationstechnik.

Das Seminar setzt eine Veranstaltungsreihe des Instituts fort, die erstmals im Sommersemester 1992 unter dem Thema "Mechanismen für fehlertolerante verteilte Anwendungen" stattfand. Der damalige Schwerpunkt lag dabei auf dem speziellen Themengebiet der Fehlertoleranz. Dieses hat sich in den letzten Jahren hin zu einem breiten Spektrum aus dem Bereich der verteilten Systemen verschoben, da eine Vielzahl der gezeigten Thematiken stark miteinander verwoben sind. Aus diesem Grund wurden neben speziellen Fragestellungen der verteilten Systeme auch die Behandlung grundlegender, neuartiger Verteilungsansätze in das Seminar mitaufgenommen. Der Kristallisationskern der Fragen lag nun bei der gemeinsamen Verwendung von Daten, woraus auch die neue Namensgebung resultierte.

Die ersten beiden Beiträge befassen sich mit weitergehenden Ansätzen. Im Rahmen der Modellierung verteilter Anwendungen vollzieht sich ein Paradigmenwechsel, welcher ausgehend von einer getrennten Modellierung von Daten und Funktionen, über objektorientierte Modellierungsansätze hin zu einer geschäftsprozeßorientierten Ausrichtung führt. Die Steuerung und die Durchführung dieser Prozesse übernehmen die im ersten Beitrag vorgestellten Workflow-Managementsysteme. Neben der klassischen Implementierung verteilter Systeme basierend auf dem Client–Server–Ansatz in Verbindung mit dem Remote Procedure Call (RPC), werden neuerdings alternative Konzepte erforscht, welche unter dem Begriff der Agenten firmieren und dem Gebiet der künstlichen Intelligenz entstammen. Wesentliche Zielsetzung ist die Verringerung der Komplexität verteilter Systeme bei gleichzeitiger Erhöhung der Effizienz für den Anwender. Eine Einführung in diesen Themenkomplex behandelt der zweite Beitrag.

Die folgenden Beiträge drei und vier behandeln ausgewählte Kapitel aus dem Bereich der verteilten Systeme. Der dritte Beitrag aus dem Gebiet der erweiterten Namensverwaltung in verteilten Sytemen, dem Trading, stellt zwei Ansätze zur Kopplung von Namensbereichen vor und vergleicht diese miteinander. Der vierte Beitrag beschreibt das Konzept von TP-Monitoren zur Steuerung von verteilten Transaktionen sowie deren Vor– und Nachteile.

Neben den Grundlagen und Mechanismen zur Realisierung verteilter Systeme stellt der dritte Abschnitt dieses Berichts im fünften Beitrag eine konkrete Programmierumgebung namens Nexus vor, die es erlaubt, verschiedene Konzepte zur Fehlertoleranz wie geschachtelte Transaktionen experimentell zu erforschen.

Bevor nun die einzelnen Ausarbeitungen der Seminarbeiträge präsentiert werden, möchten

wir allen beteiligten Studenten für ihre engagierte Mitarbeit danken, ohne die weder der Erfolg der Seminare noch die Anfertigung des vorliegenden Berichts möglich gewesen wäre. Hierzu haben auch die nach den einzelnen Vorträgen stattfindenden Diskussionen maßgeblich beigetragen.

Karlsruhe, im Oktober 1996

Arnd G. Grosse Dietmar A. Kottmann

Inhaltsverzeichnis

Vorwort . . . iii

Urban Bettag:

Workflow Management . . . 1

1 Einleitung . . . 1

2 Ein Beispiel . . . 2

3 WFMS und ihre Eigenschaften . . . 4

4 Die Workflow Management Coalition (WMC) . . . 6

5 Daten im Workflow-Management . . . 7

6 Das Referenzmodell der WMC . . . 8

7 Workflow Management am Beispiel FlowMark/2 . . . 12

8 Ausblicke, Fragen und Probleme . . . 13

LEQUANG Pascal:

Agent Base Computing . . . 15

1 Einleitung . . . 15

2 Ansätze . . . 16

2.1 Artificial Intelligence Ansatz . . . 16

Society of Mind Theory. . . . 16

Adaptative Systeme. . . . 16

2.2 Watchdog Ansatz . . . 17

2.3 Desktop Ansatz . . . 17

2.4 Verteilte Systeme Ansatz . . . 18

3 Eigenschaften . . . 18

3.1 Allgemeine Einführung . . . 18

3.2 Abstraktion/Agentverhalten . . . 19

3.3 Prinzipien eines Agentenverhaltens . . . 20

4 Mobile Agenten . . . 20

4.1 Mobile Agent Software Engineering . . . 21

4.2 Agentensprache . . . 23

4.3 Agent Communication Language . . . 23

5 Anwendungsperspektive . . . 23

5.1 Mobile Client . . . 24

5.2 Semantic Information Retrieval . . . 25

5.3 Electronic Commerce 25
6 Technische Problemen: Sicherheit, Migration 26
6.1 Migration (Agentification) 26
6.2 Security 27
6.3 Virus Detection 28
7 Zusammenfassung und Ausblick 29

Teresa Ortega:

Zusammenarbeit beim Trading: ANSA vs ODP 31
1 Einleitung 31
2 Verwandte Standardisierungsprojekte in offenen verteilten Systemen 32
2.1 Entwicklungstand 32
2.2 ODP-RM: Open Distributed Processing Reference Model 32
2.3 Viewpointsdefinition 33
Trading-Szenario im ODP-Modell: 34
2.4 ANSA: Advanced Network Systems Architechture 35
ANSAs Computational Modell für das Trading: 35
3 Grundlegende Begriffe 36
3.1 Organisationseinheit 36
Organisationsmodelle von Organisationseinheiten: 36
3.2 Kontext 37
3.3 Was ist eine Föderation? 37
4 Zusammenarbeitsmodelle von ODP und ANSA 38
4.1 ODP Modell der Zusammenarbeit 38
Beschreibung der Föderation durch Viewpoints 39
ODP-Konfiguration der Föderation: 41
4.2 ANSA-Modell 42
Verwaltungsfunktionalität: 42
Beschreibung des ANSA-Zusammenarbeitsmodels durch Grenzen 42
Modell für das "cross boundary": 43
Information Viewpoint in ANSA: 44

Ein flexibler Trading-Mechanismus (Coercion): 44
5 ANSA vs ODP 46
6 Zusammenfassung 48

Christoph Schlenker:
TP-Monitore 49
1 Einführung 49
2 Einführendes Szenario 50
3 Grundlagen 51
3.1 ACID-Transaktionen 51
3.2 Transaktionsverwaltung 52
3.3 Transaktionsverwaltung in verteilten Systemen 52
3.4 Dienst zur Datenübermittlung 54
3.5 Aufgaben und Einsatzgebiete 55
3.6 Komponenten und Kontrollfluß 56
3.7 Realisierungsmöglichkeiten 59
4 Standards 61
4.1 X/Open 62
5 Zusammenfassung und Ausblick 63

Ralph Müller:
Ein Überblick über das verteilte Betriebssystem Nexus 65
1 Einleitung 65
2 Berechnungsmodell und Mechanismen 66
2.1 Übersicht 66
2.2 Die Nexus Objekte 66
2.3 Die Typ-Objekte 69
2.4 Systemdefinierte Typen 69
2.5 Konsistenz von Aktionen in Nexus 70
2.6 Transaktionsmodelle und Primitive 72
2.7 Aufruf von Operationen auf Objekten 72
2.8 Setzen von Sicherungspunkten und Wiederaufsetzen 73
3 Die Nexus Systemarchitektur 73
3.1 Übersicht 73
3.2 Klassenrepräsentanten 74
3.3 Objekt Manager Shell 74

3.4 Der Nexus-Kern 75
4 Zusammenfassung 76

Abbildungsverzeichnis 79

Literatur 81

Workflow Management

Urban Bettag

Kurzfassung

Workflow Management - Nur eine Sprechblase oder ein neuer Trend? Der vorliegende Bericht soll in dieses noch recht junge Themengebiet einführen und die grundlegenden Begriffe und Zusammenhänge klären. Unter einem Workflow-Management-System versteht man dabei ein System zur rechnerunterstützten Verarbeitung von Geschäftsprozessen. Hierbei wird nicht nur der Weg von der Prozeßdefinition bis hin zum Workflow-Management-System skizziert, sondern es wird auch auf die Architektur von Workflow-Management-Systemen eingegangen. Dies wird exemplarisch am Beispiel des Referenzmodells der Workflow Management Coalition, eine Organisation aus Herstellern und Anwendern von Workflow-Management-Systemen dargestellt. Darüberhinaus werden am Beispiel von FlowMark/2, ein kommerzielles Workflow-Management-System der Firma IBM die charakteristischen Architekturmerkmale des Referenzmodells erläutert. Abschließend werden noch offene Probleme angesprochen, die gerade beim Einsatz von Workflow-Management-Systemen anfallen.

1 Einleitung

Die aktuelle Wirtschaftslage erzwingt nicht nur einen immer härteren Konkurrenzkampf zwischen einzelnen Unternehmen, sondern sie wirkt sich auch auf den Arbeitsplatz jedes einzelnen aus. Eine Unternehmung, die heute auf dem Markt bestehen will, muß hohe Qualitätsanforderungen an ihre Produkte stellen. Effiziente Nutzung externer Dienstleistungen und kurze Durchlaufzeiten firmeninterner Arbeitsabläufe sind die Voraussetzung für die schnellstmögliche Einführung neuer Produkte auf dem Markt.

Gerade der Prozeß von der Herstellung bis zum Verkauf ist eine lange Kette aus einzelnen Vorgängen. Diese werden oftmals räumlich getrennt bearbeitet, so daß das Produkt von einer zur anderen Abteilung weitergereicht wird. Solche Vorgänge können teilweise automatisiert werden, sowohl durch Hardware als auch durch Software. Hier setzt Workflow Management (WFM) an.

Workflow Management, auch oft als Vorgangsverarbeitung bezeichnet, ist eine kontrollierte, systemgesteuerte Ausführung von Geschäftsprozessen. *Geschäftprozesse* sind hierbei abteilungsübergreifende, aber fachlich zusammenhängende Aktivitäten, die in logischen und zeitlichen Abhängigkeiten zueinander stehen. Ein *Workflow* ist somit eine computergestützte Vereinfachung oder Automatisierung eines gesamten Geschäftprozesses oder eines Teils davon. Letztere sind wesentlicher Bestandteil des Workflow-Management-Systems (WFMS). Ein *Workflow-Management-System (WFMS)* ist ein System, das somit vollständig Workflows definiert, managt und ausführt. Dabei bedient es sich einer Software, deren Ausführungsreihenfolge von einer computergestützten Workflow-Logik bestimmt wird [Ver95].

Workflow Management unterstützt Workgroup Computing. *Workgroup Computing* ist kooperatives rechnerunterstütztes Zusammenarbeiten verschiedener Teilnehmer im Team.

Die räumliche Entfernung der Teilnehmer spielt dabei keine Rolle. Das WFMS koppelt alle Teilnehmer und ermöglicht nicht nur einen kontrollierten Datenaustausch sondern kommuniziert (falls notwendig) auch mit anderen WFMS und Applikationen [Wag95].

Die Einführung eines WFMS ist jedoch nur sinnvoll für ein Unternehmen, wenn interne Arbeitsabläufe erheblich verkürzt werden können. Die Voraussetzung ist folglich ein Business Process Reengineering, d.h. eine völlige Überarbeitung der Geschäftsprozesse. Business Process Reengineering gepaart mit WFMS ist zur Zeit die Formel, mit der man Geschäftsprozesse optimieren kann.

2 Ein Beispiel

In diesem Abschnitt soll die prinzipielle Modellierung eines Geschäftsprozesses vorgestellt werden. Am Beispiel einer Reisekostenabrechnung [Jab95a] lassen sich einige typische Eigenschaften von Workflows erkennen. Die Umsetzung eines Geschäftsprozesses, der zunächst in natürlicher Sprache formuliert ist, in ein semantisches Modell kann dabei sehr komplex werden. Dieses ist notwendig um den gesamten Geschäftsprozeß zu modularisieren und zu gliedern. Hier erweisen sich objektorientierte Modellierungstechniken als geeignet z.B. [Rum91].

Für die Abrechnung der Reisekosten muß zunächst der Mitarbeiter, der die Dienstreise durchführt ein Antrag ausfüllen. Mit Sicherheit fallen unterschiedliche Formulare an, z.B. müssen Angaben zur Person gemacht werden und die Dienstreise muß begründet und anschließend genehmigt werden. Wurde die Genehmigung nicht erteilt, so muß entweder ein neuer Mitarbeiter ausgewählt werden oder der Vorgang muß abgebrochen werden. Wenn wir davon ausgehen, daß die Reise genehmigt wurde, muß eine Reiseroute festgelegt werden. Übersteigen die Reisekosten ein bestimmtes Budget so kann die Reise storniert werden. Falls das nicht der Fall ist, können die Reisekosten von dem zugehörigen Konto für Dienstreisen abgebucht werden.

In Abbildung 1 sind die Vorgänge stark vereinfacht dargestellt. Die Pfeile stellen den Kontrollfluß zwischen den Workflows dar. Mit den Verbindungen wurde der Daten- und Informationsfluß modelliert. Die gestrichelten Verbindungen referenzieren Applikationen, in denen Teile der Workflows ausgeführt werden. Mit der gepunkteten Verbindung wurde die Beziehung von Workflows zu Rollen dargestellt. *Rollen* sind hierbei Kontrollinstanzen.

Der Geschäftsprozess Reisekostenabrechnung wird in Unterprozesse (Ausfüllen, Genehmigen, Bearbeiten, Auszahlen und Ablegen) zerlegt. Diese funktionalen Einheiten, die Workflows (Vorgänge), spezifizieren, was ausgeführt wird. Darüber hinaus werden mehrere Arten von Workflows unterschieden. Workflows können verschachtelt sein und deshalb aus mehreren *Subworkflows* bestehen. Umgekehrt können Subworkflows übergeordneten *Superworkflows* zugeordnet werden. Ein Workflow, dem keine weiteren Superworkflows übergeordnet sind, heißt *Top Level Workflow*. *Elementare Workflows* besitzen keine Subworkflows sondern referenzieren eine Applikation. Die *kompositen Workflows* besitzen Subworkflows, die nicht elementar sind.

Im obigen Beispiel gibt es somit den Top Level Workflow Reisekostenabrechnung, fünf Subworkflows, die gleichzeitig alle elementar sind. D.h. sie referenzieren Funktionen, die

Abbildung1. Reisekostenabrechnung

entweder im Reisekostenabrechnungs-, Archivierungs- oder Buchungssystem realisiert sind. Solche Systeme können recht umfangreich sein und eigene WFMS darstellen. (Man stelle sich nur die Größenordnung eines Buchungssystems wie z.B. START vor [LKK93])

Die vorgestellten Arten von Workflows lassen sich in zwei Gruppen einteilen [Jab95b]. Komposite und elementare Workflows haben aufgrund ihrer Modellierung einen statischen Charakter und werden schon beim Entwurf fest spezifiziert. Dagegen sind Top Level-, Super- und Subworkflows von dynamischer Natur. Diese Arten von Workflows können verschiedene Rollen annehmen, die vom Kontext abhängig sind, in dem sie verwendet werden, d.h. Subworkflows können durchaus auch als Superworkflows verwendet werden.

Anhand der verschiedenen Arten und Rollen von Workflows sehen wir, daß Modularität ein wichtiger Aspekt bei der Modellierung ist. Neben dem bisher erläuterten funktionalen Aspekt spielt auch der verhaltensbezogene Aspekt eine Rolle. Aus der Modellierung muß somit ersichtlich sein, wann welcher Workflow abläuft.

Es bleibt noch zu klären, wer einen Workflow ausführt. Welche Personen bearbeiten die weitergereichten Daten? Können alle Manager von Person x die Reise genehmigen? Was passiert, wenn der Finanzsachbearbeiter von x gerade an einem Auftrag arbeitet, bzw. er mit der Auszahlung beschäftigt ist? Aufgrund dieser Schwierigkeiten wird es unausweichlich sein, die organisatorische Struktur in einem separaten Diagramm zu modellieren oder Mechanismen zu entwickeln, die die Korrektheit der Abarbeitung gewährleisten.

Abschließend ist noch zu klären, wie ein Workflow ausgeführt wird. Dieser Aspekt ist rein operational zu sehen.

Zusammenfassend lassen sich folgende Schritte der Modellierung erkennen:

- Vorgänge gliedern und modularisieren
- Kontroll- und Datenfluß modellieren
- Organisationsstruktur modellieren
- Vorgänge auf Workflows abbilden
- Workflows Ressourcen zuordnen
- elementare Workflows Applikationen zuordnen

Zur Vereinfachung der Diagramme können Kontroll- und Informationsfluß in getrennten Diagrammen modelliert werden. Generell sind mehrere Verfahren zur Modellierung möglich (z.B. Object Modelling Technique [Rum91]).

3 WFMS und ihre Eigenschaften

Bereits in der Einführung und am Beispiel Reisekostenabrechnung haben wir gesehen, daß WFMS umfangreiche Softwaresysteme sind, die verschiedene Eigenschaften besitzen. Im weiteren Verlauf werden nun einige typische Anforderungen an Workflow-Management-Systeme vorgestellt.

- Skalierbarkeit

  Schon beim Grobentwurf eines Workflow-Management-Systems fällt auf, daß man generell keine genaue Aussage über die Anzahl der Workflows und die damit verbundenen Ressourcen machen kann. Daher muß ein WFMS skalierbar sein und das Entfernen und Hinzufügen von Workflows sollte unterstützt werden. Auch die Anzahl der Benutzer eines WFMS kann sich ständig ändern.

- Integration

  Eine der wichtigsten Eigenschaften ist die Integration. Um den Arbeitsaufwand und damit auch die Kosten zu reduzieren, müssen alte Datenbestände und bestehende Applikationen integriert werden. Die Integration von Altsoftware kann bereits erhebliche Probleme bereiten. Hier stellt sich die Frage nach einer Neuentwicklung oder einer Überarbeitung der alten Anwendungsprogramme (Reverse Engineering). Beide Verfahren, die sehr viel Zeit und Personal beanspruchen, wird man beim Entwurf von WFMS finden.

- Interoperabilität

  Ein WFMS sollte Interoperabilität gewährleisten, d.h. es sollte nicht nur Kommunikation möglich sein, sondern auch eine Befehlssequenz an andere WFMS geschickt und dort ausgeführt werden können. Hierbei kann es zu verschiedenen Formen der Interoperabilität kommen. Diese kann sich auf der Protokollebene oder Anwendungsebene abspielen.

– Heterogenität

  Bei der Integration und Interoperabilität wird man zwangsläufig auf unterschiedliche Datenformate und Betriebssysteme stoßen. Es muß daher ein Netz-Betriebssystem vorhanden sein, das alle verschiedenen Dialekte der einzelnen Betriebssysteme vereinbaren kann. Die Common Object Request Broker Architecture (CORBA) [Gro93] scheint hier ein gelungener Ansatz zu sein, der sich durchsetzen könnte.

– Transparenz

  Für den Benutzer eines WFMS sollten die Daten lokalisationstransparent gehalten werden, d.h. wenn ein Benutzer eines WFMS ein bestimmtes Dokument einsehen möchte, das auf einem anderen Server abgelegt ist, sollte er ohne weiteres darauf zugreifen können. Desweiteren sollten dem Benutzer Ressourcenengpässe verdeckt werden. Durch einen Lastenausgleich können hier recht wirksame Techniken eingesetzt werden, die auch für die Effizienz des Gesamtsystems förderlich sind.

– Benutzerfreundlich

  Diese Eigenschaft sollte jedes gute Programm besitzen. Mit Hilfe von Graphical User Interfaces (GUIs), die sehr gut entwickelt sind, können hier Benutzeroberflächen geschaffen werden, die auch bei Laien auf hohe Akzeptanz stoßen.

– Flexibilität

  Ein WFMS sollte flexibel sein, d.h. es sollte sich den anfallenden Aufgaben dynamisch anpassen. Eine dynamische Konfigurationsverwaltung [MS92] wäre hier ein sinnvoller Ansatz.

– Effizienz

  Die Workflows in einem WFMS sollten kurze Durchlaufzeiten haben. Neben dem erwähnten Lastenausgleich sollten die Workflows sich selbst, in Form eines Agenten eine freie Ressource suchen auf der sie abgearbeitet werden. Ein Agent ist eine Ressource, welche einen Workflow oder eine Applikation ausführen kann. Beispiele für Agenten sind Mitarbeiter eines Unternehmens, Maschinen oder auch Serverprozesse [Jab95b].

Neben diesen Qualitätsmerkmalen sollten WFMS auch über eine Komponente verfügen, die den historischen und transaktionellen Bereich abdeckt.

Im historischen Bereich sollten aktuelle und vergangene Abläufe in einem Logbuch protokolliert sein. Im Falle einer Störung ist es dann möglich, auf Synchronisationspunkte zurückzugreifen. Ferner kann die historische Komponente zur Entscheidungsfindung beitragen, da im Logbuch vergangene Abläufe registriert sind.

Der transaktionelle Teil [Jab95b] sollte in jedem WFMS enthalten sein. Gehen wir davon aus, daß ein WFMS parallel arbeitet, kann es vorkommen, daß Artefakte ständig modifiziert werden oder sich gar überholen können. Scheduling- und Sperrverfahren sorgen hier für einen korrekten Ablauf.

4 Die Workflow Management Coalition (WMC)

Im letzten Kapitel wurden einige Attribute aufgeführt, die in kommerziellen WFMS realisiert sein sollten. Zur Zeit gibt es über 100 verschiedene WFMS, die in den unterschiedlichsten Bereichen eingesetzt werden, z.B. in der Bildverarbeitung, CIM, Büroautomatisierung und Dokumentenverarbeitung. Die im letzten Abschnitt vorgestellten Charakteristika sind in den aktuellen WFMS unterschiedlich stark ausgeprägt bzw. sind teilweise nicht vorhanden. Jedes System besitzt seine eigene Architektur und vereinbart seine Schnittstellen für externe Applikationen und Datenformate selbst. Kommunikation mit anderen WFMS ist hierbei nur sehr schwer möglich.

Die Workflow Management Coalition(WMC) versucht, diesem Prozeß entgegenzuwirken. Die WMC ist eine nicht kommerzielle Organisation, die 1993 als Zusammenschluß führender Softwarehersteller und Benutzer von WFMS gegründet wurde. Zur Zeit zählt sie über 100 Mitglieder(z.B. IBM, Digital Equipment, Hewlett Packard, Siemens,...). Die Ziele der WMC sind neben der Verbreitung von WFMS und Schaffung einer einheitlichen Terminologie auch die Entwicklung von Standards und Schnittstellen für den Datenaustausch mit anderen WFMS. Dadurch wird das Risiko für den Einsatz von WFMS reduziert und WFMS können sich auf dem Softwaremarkt schneller etablieren.

Diese Zusammenhänge gehen bereits aus dem Beispiel Reisekostenabrechnung hervor. Wird der Antrag auf Dienstreise bearbeitet, so muß mit Sicherheit ein Buchungssystem referenziert werden. Solche externen Applikationen oder gar WFMS müssen über einheitliche Schnittstellen verfügen, damit eine korrekte Kommunikation und Abarbeitung des Workflows gewährleistet ist. Desweiteren kann auch ein Electronic Mail System referenziert werden, das dem Mitarbeiter eine Nachricht über den aktuellen Status seines Antrags zukommen läßt.

Die WMC ist in zwei große Gremien aufgeteilt, das technische Gremium und das Koordinierungs-Gremium. In jedem dieser Gremien existieren kleine Arbeitsgruppen, die gemäß der Zielsetzung der WMC arbeiten. Beide Gremien treffen sich viermal im Jahr, drei Tage lang, abwechselnd in USA und Europa und stellen ihre Ergebnisse vor bzw. diskutieren offene Probleme.

Bisher hat die WMC eine einheitliche Terminologie entwickelt. Im Januar 1995 wurde erstmals ein Glossar vorgestellt, in dem über 50 Begriffe definiert wurden, die im Zusammenhang mit WFM auftreten. Neben einem Anwendungsbeispiel werden auch Synonyme für manche Begriffe angegeben, die sich seit dem Aufkommen von WFMS in der Industrie etabliert haben.

Aufbauend auf der Terminologie konnte anschließend ein abstraktes Referenzmodell entwickelt werden, in dem sich alle gegenwärtigen WFMS (noch) einordnen lassen.

5 Daten im Workflow-Management

In einem1 WFMS gibt es verschiedene Typen von Daten. Diese lassen sich nach ihrer Entstehung grob in zwei Phasen einteilen.

Die erste Phase beschäftigt sich mit der Prozeßdefinition und wird auch als *Build-Time* bezeichnet. In der Build-Time entstehen Daten, die während dem Prozeßdesign und der Prozeßdefinition anfallen. Die WMC schreibt hier keine speziellen Datenstrukturen oder Schemata für die Daten vor. Es ist daher durchaus sinnvoll, Werkzeuge für die Prozeßmodellierung, z.B. CASE, OOA-Tools [Rum91] oder Skriptsprachen einzusetzen.

Die zweite Phase wird als *Run-Time* bezeichnet. Während der Run-Time entstehen Daten, die nicht nur zwischen den einzelnen Workflows ausgetauscht werden, sondern auch Kontrolldaten, die den korrekten Ablauf des Workflows garantieren.

Desweiteren werden referenzierte Applikationen oft mit Parameterdaten versorgt. Die WMC gliedert diese verschiedenen Daten in drei Typen auf.

- *Workflow Process Controll Data* sind Kontrolldaten, die nur vom WFMS definiert und manipuliert werden können.
- *Workflow Process Relevant Data* sind Daten, die sowohl von externen Applikationen als auch vom WFMS benutzt werden.
- *Application Data* sind Daten, die nur von externen Applikationen verwaltet werden. Das WFMS kann auf diese Daten nicht zugreifen.

In Abbildung 2 werden die beiden Phasen und die verschiedenen Daten nochmals graphisch veranschaulicht. Der Benutzer des WFMS interagiert über eine Client Applikation. Hierbei manipuliert er Daten, die entweder nur die Applikation betreffen oder für den Workflow relevant sind. Diese werden nach Abarbeitung des Workflows dem Workflow Ausführungsservice (Workflow Enactment Service siehe Abschnitt 6) übergeben. Dieser interpretiert die Daten und kann dabei auch die Prozeßdefinition ändern. Nachdem ein Workflow abgearbeitet wurde führt der Workflow Ausführungsservice einen neuen Workflow aus. Hierbei bedient er sich der Workflow Process Definition Data.

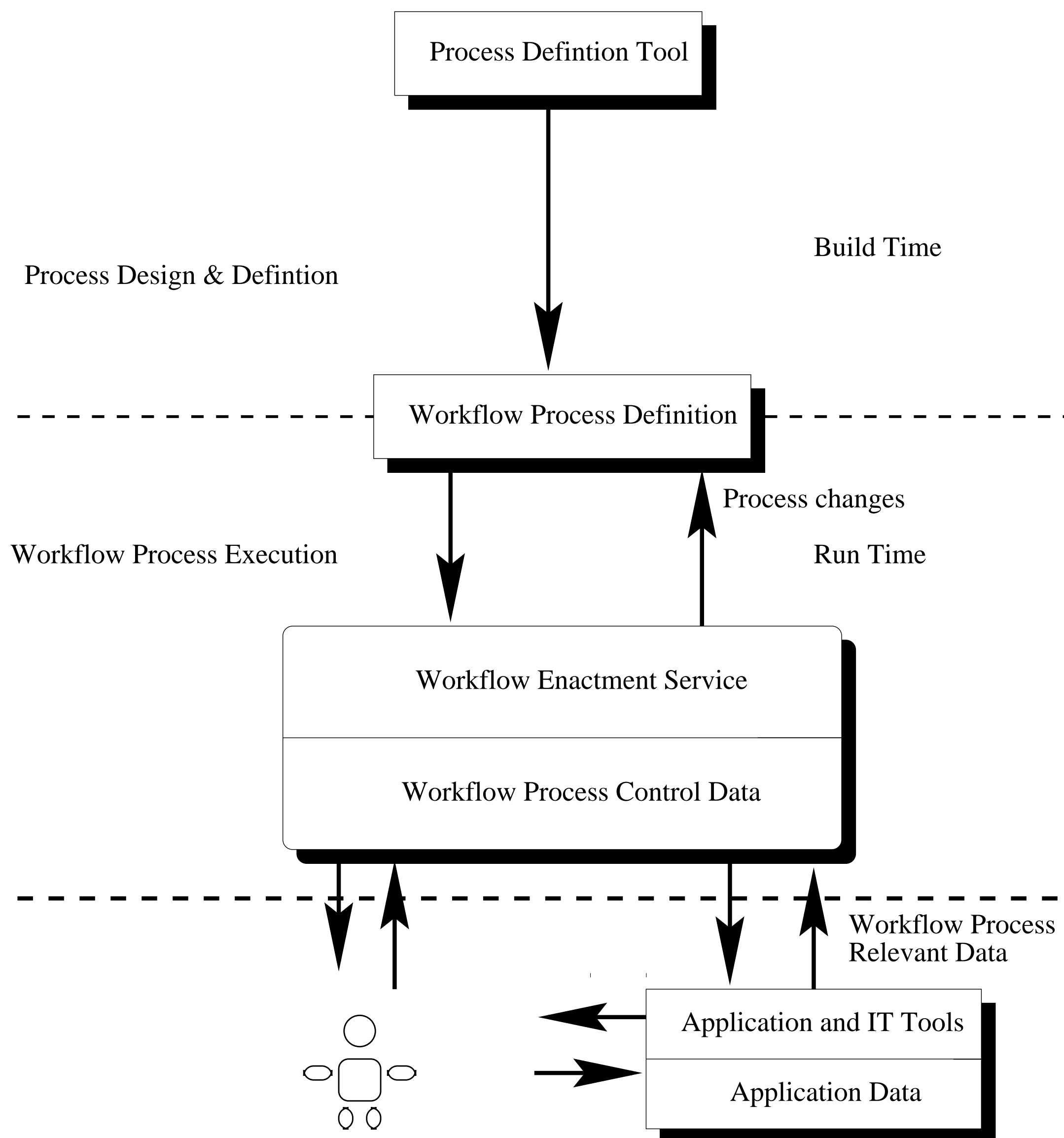

Abbildung2. Daten im WFMS

6 Das Referenzmodell der WMC

Die WMC hat neben dem Glossar auch ein Referenzmodell für WFMS erarbeitet. In diese Beispielarchitektur lassen sich alle auf dem Softwaremarkt befindlichen WFMS einordnen. Es bleibt zu hoffen, daß die Mitglieder der WMC und andere Softwarehersteller sich an diesem Modell orientieren. Prognosen, ob sich das Referenzmodell durchsetzen wird, sind zu diesem Zeitpunkt verfrüht.

Damit eine Kommunikation zwischen den Komponenten eines WFMS zustande kommt, werden Schnittstellen und ein standardisiertes Datenformat benötigt. Das bedeutet am Beispiel der Reisekostenabrechnung, daß die verschiedensten Applikationen Daten verändern. Damit diese die Daten korrekt verarbeiten werden können, müssen Schnittstellen vorhanden sein.

Folgende Übersicht zeigt den Vorschlag eines Referenzmodells der WMC.

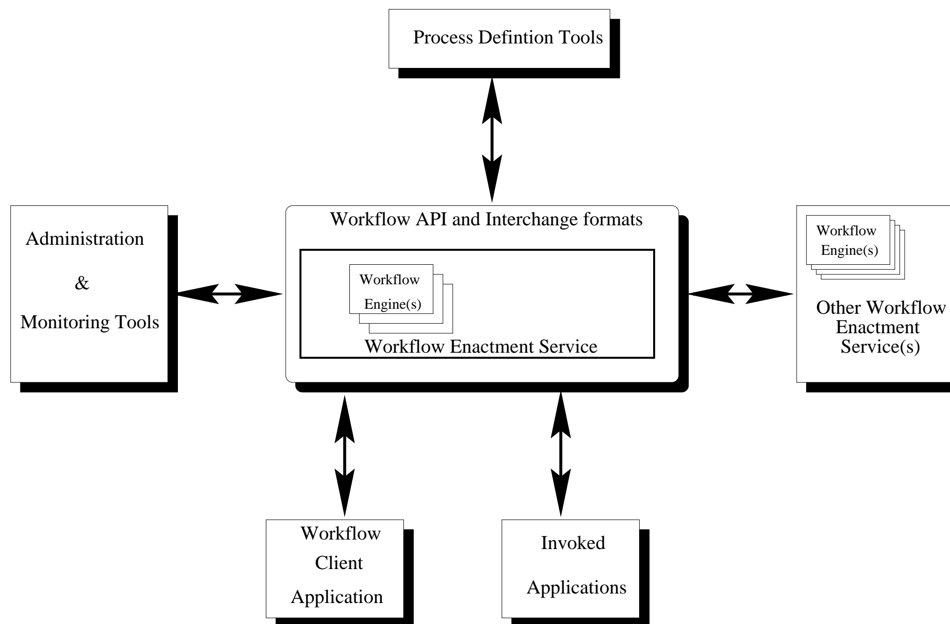

Abbildung3. Referenzarchitektur der WMC

– *Process Definition Tools* (Interface 1)

  Zur Zeit gibt es zahlreiche Werkzeuge, die es ermöglichen, Geschäftsprozesse zu analysieren und zu modellieren. Die WMC schreibt hier keine speziellen Verfahren vor. Sie arbeitet aber intensiv mit Firmen zusammen, die solche Werkzeuge entwickeln. Zweck dieser Zusammenarbeit soll eine einheitliche Beschreibungssprache für Workflows sein.

  Je nach Anwendungsgebiet können hier Werkzeuge aus der Systemanalyse z.B. CASE- oder OOA-Tools eingesetzt werden. Wichtig ist dabei, daß das Laufzeitsystem des WFMS, die Workflow Engine, über eine standardisierte Schnittstelle mit dem Process Definition Tool kommunzieren kann. Die Existenz der Schnittstelle ist von grundlegender Bedeutung, da ein Geschäftsprozeß ständig erweitert oder optimiert werden kann.

  Die Import/Export-Schnittstelle des Process Definition Tools sollte über Funktionen verfügen, die es ermöglichen, Prozesse in das Prozessmodell ein- bzw. auszugliedern. Weiterhin müssen die unterschiedlichen Datentypen gekennzeichnet werden. Darüberhinaus müssen auch Zugriffspfade ausgetauscht werden, um die Daten referenzieren zu können. Dabei spielt die Resourcenverwaltung eine wichtige Rolle.

– *Workflow Enactment Service*

  Der Workflow Enactment Service wird auch als Ausführungsservice bezeichnet. Er ist die Zentrale in jedem WFMS und stellt dem Laufzeitsystem ein oder mehrere Workflows bereit, die ausgeführt werden können. In diesem Vorgang können mehrere Workflows Engines beteiligt sein. Mehrere verteilte Workflow Engines, die auf verschiedenen Plattformen realisiert sind, verkürzen die Bearbeitungszeiten der Workflows. Wichtig ist, daß alle beteiligten Workflows Engines auf die gleichen Kontrolldaten zugreifen können.

Das Anwendungsprogramm, mit dem der Benutzer einen Workflow erstellt oder abschickt, ist vom Workflow Ausführungsservice getrennt.

Die Workflow Engine kann über Invoked Applications externe Applikationen ausführen(z.B. eine Textverarbeitung)

– *Workflow Client Applications* (Interface 2)

  Interface beschreibt die Schnittstelle zwischen Workflow Enactment Service und Workflow Client Application. Die Workflow Client Application ist hierbei eine Software, mit der der Endbenutzer seine Workflows in eine Arbeitsliste einträgt. Bevor er diese an den Workflow Enactment Service sendet, kann er die Workflow-relevanten Daten, falls er dazu berechtigt ist modifizieren. Aufgrund dieser Tatsache müssen hier Operationen zur Prozeßverwaltung(Prozesse starten und beenden) vorhanden sein, die wiederum in Form von APIs spezifiziert werden.

  Ein breites Spektrum von Anwendungssystemen, die in den unterschiedlichsten Industriezweigen eingesetzt werden, kann hier integriert werden, um ein komplettes WFMS zu bilden.

– *Invoked Applications* (Interface 3)

  Mit Hilfe des dritten Interfaces kann der Workflow Enactment Service externe Applikationen aufrufen. Hierbei kann es sich um ein Electronic Mail System, eine Textverarbeitung oder Datenbank handeln. Mit dem umfangreichen Angebot an Programmen treten jedoch auch Probleme auf. Jede referenzierte Applikation muß mit Parametern versorgt werden. Hierfür sind sogenannte Application Programming Interfaces (APIs) notwendig, die die Softwarehersteller in ihre Anwendungen integrieren müssen, damit eine korrekte Ausführung gewährleistet wird. Darüberhinaus soll das WFMS durch einen Tool-Agenten sich selbstständig die zugehörige Applikation aussuchen. Anschließend wird diese geprüft, ob sie über das gleiche API bzw. gleiche Datenformat verfügt.

  In diesem Bereich wird zur Zeit sehr intensiv in den einzelnen Gremien gearbeitet. Ein Standard wurde noch nicht veröffentlicht, jedoch existieren schon konkrete Vorschläge über das Aussehen dieser Schnittstelle in Form von technischen Berichten.

– *Andere Workflow Enactment Services* (Interface 4)

  Eine der wichtigsten Aufgaben der WMC ist es, standardisierte Schnittstellen für verschiedene WFMS der unterschiedlichsten Hersteller zu entwickeln. Damit ein Workflow auf einem anderen WFMS bearbeitet werden kann, müssen zahlreiche Anforderungen erfüllt werden.

  Zunächst müssen nicht nur die Prozeßdefinitionsdaten sondern auch die Workflow- und Applikations-relevanten Daten ausgetauscht werden. Insbesondere müssen sich die beteiligten WFMS gegenseitig synchronisieren, da die Daten in jedem WFMS gehalten werden. Um die korrekte Synchronisation hierbei zu gewährleisten sind umfangreiche Transaktionsverwaltungssysteme notwendig.

Wir haben bereits in Abschnitt 3 einige Eigenschaften von WFMS vorgstellt. Die Interoperabilität ist im Zusammenhang mit Schnittstelle 4 grundlegend. Die WMC unterscheidet mehrere Ebene(Levels) der Interoperabiltät.

- Level 1 Coexistence

  Die verschiedenen interoperierenden WFMS benutzen die gleiche Hard- und Software.

- Level 2 Unique Gateways

  Spezielle WFMS tauschen Workflows untereinander aus, z.B. nur solche im Bereich CIM oder Dokumentenverarbeitung.

- Level 2a Common Gateway API

  Hier handelt es sich um eine Erweiterung von Level 2. Es können auch andere WFMS aus verschiedenen Bereichen interoperieren.

- Level 3 Limited Common API1

  Die in diesem Level definierte Interoperabilität befaßt sich mit einer Teilmenge von Funktionen z.B. für Verbindungsaufbau, die sich auf eine offene API beschränkt.

- Level 4 Complete Workflow API

  Alle Aspekte eines WFMS werden in Level 4 über ein offenes API unterstützt.

- Level 5 Shared definition format

  Alle beteiligten WFMS können bereits zur Laufzeit die gleiche Prozeßdefinition benutzen.

- Level 6 Protocol Compatability

  Alle APIs sind standardisiert. Darüberhinaus werden auch Schnittstellen zu Überwachungs- und Verwaltungswerkzeugen unterstützt.

- Level 7 Common Look and Feel

  Alle Komponenten operieren und erscheinen einheitlich.

– *Administration and Monitoring Tools* (Interface 5)

  Die Administrations- und Überwachungsschnittstelle gibt Auskunft über den aktuellen Status eines Workflows. Lastenverteilung und Auslastung der Systemkomponenten sind weitere Funktionen, die im Administrations Tool zu finden sind. Eine Benutzerverwaltung mit zugehörigem Berechtigungsmanagement gehört ebenfalls zur Funktionalität.

7 Workflow Management am Beispiel FlowMark/2

In diesem Kapitel werden die wichtigsten Architekturmerkmale von FlowMark/2 vorgestellt und untersucht, in welcher Form sich diese im Referenzmodell der WMC wiederspiegeln.

FlowMark/2 ist ein kommerzielles Client/Server-WFMS der Firma IBM [OH92]. FlowMark/2 läuft unter dem Betriebssystem OS/2 und ist seit September 1993 auf dem Markt erhältlich. Die Architektur von FlowMark/2 orientiert sich am Referenzmodell der WMC, das jedoch erst im November 1994 veröffentlicht wurde [Coa94]. Aufgrund des hohen Marktanteils der Firma IBM und der Akzeptanz beim Einsatz von FlowMark/2 ist anzunehmen, daß IBM seine Terminologie in der WMC durchsetzen konnte.

FlowMark/2 läßt sich in zwei Phasen aufteilen. In der Build-Time kann der Entwickler mit Hilfe eines Design-Tools Geschäftsprozesse direkt in Workflows umsetzen. Mit Hilfe eines Simulations- und Animations-Tools kann er sich durch das Workflow Modell navigieren und bei Bedarf das Modell erweitern und verfeinern. Das so erstellte Workflow Szenario wird in einer objektorientierten Datenbank (ObjectStore) verwaltet. Dies macht die Prozeßdefintion sehr flexibel, da hier die Vorteile der Objektorientierung (Vererbung, Wiederverwendbarkeit und Kapselung) zum Tragen kommen.

Nachdem das Workflow-Modell entworfen wurde kann es in der Run-Time interpretiert und die einzelnen Workflows können abgearbeitet werden. FlowMark/2 arbeitet hier nach dem Client/Server Prinzip [LKK93]. Der Run-Time-Client fordert vom Run-Time-Server zunächst eine Liste der Aktivitäten an, die von verschiedenen Applikationen und Benutzern ausgeführt werden. Diese sind in FlowMark/2 durch Prozedurgraphen und Arbeitslisten realisiert. Danach führt der Client die Aktivität aus. Dies geschiet durch Starten und Beenden von Applikationen. Handelt es sich um eine Tätigkeit, die der Benutzer ausführen soll, so übersendet der Server dem Client eine Arbeitsvorschrift, nach der der Vorgang bearbeitet werden soll. Wann der Benutzer diesen Vorgang erledigt bleibt dabei ihm überlassen. Ist der Vorgang erledigt sendet der Client dem Server eine Meldung und fordert den nächsten Vorgang.

Der Run-Time-Server interpretiert das Workflow-Modell und managt den Workflow. Darüberhinaus übernimmt der Server das Scheduling und gewährleistet damit den korrekten Ablauf der Workflows. Der Server kann jedoch nur 20 Clients verwalten, was für sehr große Anwendungen eine erhebliche Einschränkung darstellt.

Alle Komponenten von FlowMark/2 lassen sich in das Refernzmodell der WMC einordnen. Auch die verwendete Terminologie ist identisch, mit der der WMC [Coa94]. FlowMark/2 unterstützt jedoch nicht die Interoperabilität mit anderen WFMS. Am Beispiel von FlowMark/2 haben wir gesehen, daß die Referenzarchitektur der WMC keineswegs abstrakt gehalten ist sondern sich an realen WFMS orientiert.

8 Ausblicke, Fragen und Probleme

Abschließend werden noch einige Probleme erläutert, die im Zusammenhang mit WFMS und den Vorschlägen der WMC auftreten. WFMS sind sicherlich innovative Systeme, die sich bewährt haben. Mit ihrem Aufkommen und der Verbreitung sind jedoch auch Probleme verbunden.

Im universitären Bereich ist zu beobachten, daß sich wenige Wissenschaftler mit der Thematik befassen. Dieser Aspekt spiegelt sich besonders im Nichtvorhandensein eines theoretischen Ansatzes, gar in Form eines Lehrbuches, wider. Neben den offiziellen Dokumenten der WMC gibt es einige Tagungsberichte oder Artikel, die jedoch nur konzeptionelle Ansätze oder aktuelle WFMS vorstellen.

Aktuelle WFMS besitzen nicht alle Komponenten wie sie in der Referenzarchitektur der WMC vorgeschlagen werden. Manche WFMS konzentrieren sich ausschließlich auf die Prozeßdefinition oder das Koordinieren der Workflows. Eine historische oder transaktionelle Komponente ist oft nicht vorhanden. Dieser Sachverhalt wirkt sich natürlich auf die Qualitätsmerkmale (siehe Kapitel 3) aus.

Weiterhin sind auch die Auswirkungen von WFMS auf die Arbeitswelt zu beachten. Lean Managment und Outsourcing sind zwei Begriffe, die oft im Zusammenhang von WFMS genannt werden. Abbau von Hierachien und Automatisierung sind die Folgen von WFM. Nachdem die Geschäftsprozesse eines Unternehmens optimiert wurden, ergeben sich nicht nur schnellere Durchlaufzeiten, sondern es wird auch weniger Personal benötigt. Aus Bereichen, die vorher Engpässe waren, kann Personal abgezogen werden und in anderen Abteilungen eingesetzt werden. Dies funktioniert in der Praxis leider nicht immer, da die notwendige Qualifikation der Mitarbeiter nicht immer vorhanden ist. Die Konsequenz wird Stellenabbau oder die Inanspruchnahme von Fremddienstleistungen sein.

Die Abbildung von Geschäftsprozessen aus Workflows ist noch nicht sehr weit entwickelt. CIM und Dokumentenverarbeitung sind mit Sicherheit zwei verschiedene Dinge, die auf unterschiedliche Art und Weise modelliert werden. Es dürfte daher sehr schwierig sein, ein standardisiertes Datenformat zu entwickeln, das beide Bereiche abdeckt. Hier müssen entweder Kompromisse gemacht oder die Forderungen der WMC ignoriert werden. Es stellt sich daher die Frage, inwieweit sich die Mitglieder, Hersteller und Industrie einigen können. Diese Entscheidung sollte schnellstmöglich getroffen werden, da sonst die WMC aufgrund ihrer Starrheit an Akzeptanz verlieren oder die führenden Softwarehersteller eigene Standards setzen würden.

[Ver95] berichtet über Standardisierungsbestrebungen im Bereich CASE-Tools. So ist zwar ein CASE Data Interchange Format (CDIF), das eine einheitliche Schnittstelle zwischen verschiedenen CASE Produkten darstellt, vorhanden. Die Standards reduzieren sich jedoch nur auf eine sehr kleine Menge, d.h. die Standardisierung wurde hier nicht komplett durchgeführt und ist daher fast wertlos. Somit bleibt der WMC zu wünschen, daß sie aus den Fehlern der Vegangenheit lernt und defacto bzw. dejure Standards verbinden kann.

Die Ausführung von Workflows kann mitunter sehr lange dauern, Tage und Wochen sind keine Seltenheit. Stellt man sich ein WFMS vor, das die Dokumentenverarbeitung im

Finanz- und Steuerbereich koordinert, so müssen die Artefakte ständig überwacht werden, da in diesem Bereich Gesetzesänderungen an der Tagesordnung sind. Das bedeutet, es muß eine Instanz vorhanden sein, die nicht nur das Dokument überwacht und koordiniert, sondern auch ständig verifiziert, z.B. Ist mein Dokument noch rechtskräftig? Ist der Bewilligungszeitraum abgelaufen? Ist mein Antrag verjährt?

Neue Aspekte tun sich auch im Bereich Mobile Computing auf. Das WFMS kann über Funk oder Modem mit einem Laptop kommunizieren. Der Mitarbeiter läßt sich dabei z.B. seine Arbeitsunterlagen übertragen und bearbeitet sie zu Hause. Nachdem er seine Arbeit beendet hat, sollte es möglich sein, einen Workflow in den bisherigen Ablauf zu integrieren. Hierbei entstehen Konsistenzprobleme.

Agent Base Computing

LEQUANG Pascal

Kurzfassung

Software Agenten - oder intelligente Agenten - sind ein neues Konzept im Bereich des Software Engineering. Agenten sind unabhängige Software-Komponenten, die als Vertreter eines Benutzers arbeiten können. Sie erlauben dem Benutzer, komplexe Aufgaben zu delegieren (*Delegation Model*). Die Übertragung von Agenten durch ein Netzwerk (*mobile Agenten*) bietet eine neue Lösung für den Entwurf von verteilten Anwendungen, bzw mobilen Code. Mobile Agenten sind Agenten, die durch ein Netzwerk wandern können, um Zugriff an entfernten Ressourcen zu erhalten. Hauptziel der Agentenentwicklung ist der Entwurf einer universellen, plattformunabhängigen Sprache, so daß Agenten durch ein heterogenes Netzwerk wandern können. Mobile Agenten bringen Anwendungsperpektiven mit sich, die die Arbeit der Benutzern vereinfachen wird, besonders was das Suchen und das Filtern von Informationen betrifft. Ausgehend von Agenten können auch *elektronische Marktplätze* entwickelt werden. Jedoch gibt es für mobile Agenten das Problem der Sicherheit, welches noch gelöst werden muß.

1 Einleitung

> "Agent base computing (ABC) is likely to be the next significant breakthrough in software developpment"

The Guardian, March 12th 1992 (siehe [WJ95])

Diese Behauptung ist sicherlich ein bißchen übertrieben, aber Tatsache ist, daß sich viele Informatiker für Agenten interessieren.
Das Webster's New Encyclopedic Dictionary" [Web96] gibt 3 Definitionen für das Wort *Agent*:

1. etwas das eine Wirkung produziert oder produzieren kann,
2. etwas/jemand der handelt oder der eine Arbeit übernimmt,
3. etwas/jemand der anstelle von jemandem anderen handelt.

In der Informatik ist ein Agent eine Softwarekomponente, die mit einem Assistenten vergleichbar ist. Ein Agent tut etwas anstelle des Benutzers oder hilft dem Benutzer bei seiner Arbeit auf einem Computer. Man spricht dabei auch vom *Delegation Model*.
Zur Zeit entwickeln sich Agenten nach 2 Schwerpunkten: Benutzerunterstützung und verteilte Anwendungen.
Benutzerunterstützung bedeutet, daß das Agentsystem den Benutzer unterstützt, um den Computer bedienungsfreundlicher zu machen. Dies entspricht der Rolle eines Assistenten, jemandem zu helfen.
Die aktuelle Entwicklung von Agenten wird hauptsächlich durch die Entwicklung von

Netzwerken, bzw. von verteilten Anwendungen bestimmt. Die Übertragung von Agenten durch ein Netzwerk bietet dabei nicht nur Alternativen zu existierenden verteilten Anwendungen, sondern auch neue Möglichkeiten.
Um sich vorzustellen, was ein Softwareagent ist, kann man zunächst gewisse Systeme, die Agentfunktionalität haben, untersuchen. Diese werden im Kapitel 2 beschrieben.
Im Kapitel 3 werden Eigenschaften dieser Systemen detailliert, sowie eine abstraktere Definition eines Agenten gegeben. Kapitel 4 handelt über die Implementierung von mobilen Agenten, bzw. vom sogenannten agenten-basierten Software Engineering.
Kapitel 5 präsentiert die Anwendungsperspektiven von mobilen Agenten. Die technischen Probleme, die durch die Benutzung von Agenten entstehen, werden in Kapitel 6 besprochen.

2 Ansätze

Agenten existieren schon seit einer gewissen Zeit. Der Schwerpunkt liegt dabei in der künstlichen Intelligenz (KI). In [WJ95] ist beispielsweise die künstliche Intelligenz als die Kunst der Herstellung von Agenten, die intelligentes Verhalten vorzeigen, definiert. Praktisch sind auch einige Systeme implementiert. Manche können als Vorgänger von Agenten betrachtet werden: es sind Systeme, die sogenannte "daemon"-Agenten umfassen, d.h. Programme, die resident in einem Computer bleiben, um zum Beispiel eine sich wiederholende Aufgabe zu bearbeiten. Im Bereich verteilter Anwendungen exitieren auch einige Systeme, die eine vollständige Agentenarchitektur integrieren.

2.1 Artificial Intelligence Ansatz

Society of Mind Theory. In seinem Society of Mind Theory (in [PY92] zitiert) hat Marvin Minsky den menschlichen Geist mit Agenten erklärt.
Der Geist ist seiner Meinung nach aus einer Menge Agenten zusammengesetzt. Alle Agenten arbeiten unabhängig und können miteinander kommunizieren. Diese Agenten sind hierarchisch organisiert, und können zusammenarbeiten, um eine schwierige Aufgabe zu lösen (die sie nicht individuell bewältigen konnten). Praktisch ist diese Theorie nicht einfach zu benutzen, weil sie eine große Komplexität aufweist.

Adaptative Systeme. Hier ist es die Rede von Agenten, die aus Erfahrung lernen können. Diese Agenten passen ihr Verhalten an Gewohnheiten, Bedürfnisse, Vorlieben, etc. des Benutzers an. Die Schwierigkeiten für ein solches System liegen bei der Definition von Lernregeln.
Solche Agenten, die durch Beispiele oder Erfahrungen lernen, werden *learning agents* genannt.
Das Verhalten dieser Agenten ist durch das Verhalten des Benutzers bestimmt.
Bsp.: Maxim, ein Electronic Mail Agent (Macintosh) (siehe [Mae94]).
Maxim ist ein Agent, der dem Benutzer bei der Bewältigung seiner Email hilft. Er lernt, Mails zu löschen, nachzusenden, zu sortieren und zu archivieren. Er beobachtet die Aktionen des Benutzers und speichert die zugehörigen Konfigurationen und die entspre-

chenden Parameter (memory based reasoning). Maxim kann die Aktionen des Benutzers vorhersagen, indem er die aktuelle Situation mit den in seinem Gedächtnis gespeicherten Beispielen vergleicht.

Dabei geht er nach der folgenden Entscheidungsmethode vor: Er rechnet die Distanz zwischen dem aktuellen Konfiguration und denen, die er gespeichert hat. Diese Distanz entspricht dem Entscheidungsniveau von Maxim. Daher kann der Benutzer spezifizieren, ab welchem Niveau Maxim Vorschläge generieren kann oder gar selbst handeln kann.

2.2 Watchdog Ansatz

Gewisse Systemen benutzen *Agenten* um dem Benutzer bei folgenden Aufgaben zu helfen:

- monitoring der Umgebung: Verwaltung und Beaufsichtigung des Systemzustandes,
- Benachrichtigung des Benutzers über verschiedene Ereignisse.

Bsp: Unix Crontab, Sun's Calendar Manager, Sun's SunNetManager (alle in [PY92] zitiert).

- Unix Crontab ist ein *system scheduling tool*. Der Benutzer kann in einer Crontab-Datei definieren, wann und wie oft er bestimmte Kommandos ausführen möchte. Das System führt dann diese Kommandos durch die Systemuhr gesteuert aus.
- SunNetManager: monitoring und managing eines Netzwerks.
  SunNetManager besteht aus einer Menge von kleinen Anwendungen oder Agenten, deren Ziel es ist, Netzwerkstatistiken zu erstellen. Der Benutzer kommuniziert mit diesen Agenten durch eine einzige Schnittstelle. Er kann über sie spezifizieren:
  - wann er Ergebnisse will,
  - welche Daten berechnet werden sollen.

2.3 Desktop Ansatz

In dem Desktop Ansatz ist der Agent in einer Menge von End-User-Anwendungen integriert, d.h. daß das Agenten-System im Main Operating Environment oder Desktop integriert ist. Die Agenten werden dabei durch benutzergesteuerte Anwendungen des Desktops erzeugt.

Beispiel: HP NewWave Agent (siehe [PY92])
Im dem New Wave Agent existieren Protokolle, die dazu dienen, benutzereigene Scripte zu erzeugen, die den New Wave Agent bestimmte Aktionen ausführen lassen. Das Hauptziel des New Wave Agent ist die Automatisierung von Aufgaben, die der Benutzer oft ausführt. Ein Beispiel einer Benutzung des New Wave Agent ist ein Datenbankzugriff.
Der Benutzer beginnt, indem er ein sogenanntes Aufnahme-Feature startet und danach seine Aufgabe ausführt (Sequenz von Kommandos). Dies produziert ein Script, das die Aufgabe repräsentiert. Die Wiederholung dieser Aufgabe kann mit einem Kalender (ähnlich wie Crontab) gesteuert werden.

Die Integration von Agenten im Desktop hat den Vorteil, daß der Benutzer keine Scriptsprache zu lernen braucht, da Aufgaben durch ein Beispiel definiert sind.

Der Nachteil des New Wave Agent ist, daß er als *single task* alle Prozessorressourcen belegt, sobald eine Agentaufgabe ausgelöst wurde.

2.4 Verteilte Systeme Ansatz

Die vorhergehenden Ansätze implementieren Agenten, die lokal auf einem Rechner arbeiten. Ein andere Möglichkeit ist es, Agenten zu entfernen Rechner zu schicken. Die Motivationen dazu sind der Zugriff auf entfernte Daten oder die Benutzung der Rechenkapazität eines anderen Rechner.

Beispiel: Telescript
Telescript ist eine *Softwaretechnologie*, die von General Magic entwickelt wurde ([Whi94]). Telescript wurde hergestellt, um kommunizierende Anwendungen zu ermöglichen. Es implementiert vollständige Agenten, die die Möglichkeit haben, sich über ein Netzwerk zu bewegen.
Diese Implementierung besteht aus einer Sprache und einer sogenannten *Engine*, d.h. eine Softwareschnittstelle zwischen Agenten auf der einen Seite und Non-Telescript-Programmen, Kommunikationsmittel und Betriebsystem auf der anderen Seite. Ein Telescript-Agent kann auf einen entfernten Computer zugreifen, um einen Dienst zu benutzen. Das Hauptziel von Telescript ist die Schaffung eines elektronischen Marktplatzes (siehe Kapitel 5).

3 Eigenschaften

Aus den beschriebenen Systemen kann man verschiedene Eigenschaften eines Agenten herausziehen. Diese können einen Agenten praktisch beschrieben, d.h. was ein Agent machen kann oder zu was eine Softwarekomponente fähig sein muß, damit sie Software-Agent genannt werden kann. Da Agenten noch nicht sehr verbreitet sind, besteht die Frage, ob diese Eigenschaften nicht zu begrenzt sind oder ob sie sich in Zukunft weiter entwickeln werden (KI kann viel zum Agentenkonzept beitragen).

3.1 Allgemeine Einführung

Die folgenden Eigenschaften können aus den in Kapitel 2 beschriebenen Systemen entnommen werden.

- Diese Systeme können durch ein Ereignis oder die Zeit ausgelöst werden.

- Sie haben ein anpaßbares Verhalten, d.h. sie ändern ihr Vorgehen, indem sie ihr Verhalten an die Antworten des Benutzers während eines Dialogs oder an das Verhalten des Benutzers anpassen.

– Rationalität: dies entspricht der Fähigkeit eines Agenten, seine Aufgabe unter optimalen Bedingungen auszuführen. Er kann sich beispielswiese auf einen anderen Rechner übertragen, wenn der Rechner, auf dem er sich befindet, nicht genug Informationen besitzt.

– Kooperation: darunter versteht man die Möglichkeit, daß Agenten zusammenarbeiten können. Ein Agent kann z.B. einen anderen Agenten bitten, eine Aufgabe für ihn zu erledigen (oder einfach Informationen nachfragen). In diesem Fall muß der zweite Agent den entsprechenden Dienst anbieten.

– Verteilte Architektur: die Möglichkeit, daß ein Agent über das Netzwerk wandern kann, ist wahrscheinlich die vielversprechendeste. Ein solcher Agent kann zu einem Computer geschickt werden, um dort Zugriff auf gewisse Dienste oder Daten durchzuführen (z.B. auf eine Datenbank, die sich auf einem Server befindet). Er kann sich selbst auch auf dem Server ausführen, da der Client-Computer nicht leistungsfähig genug ist.

– Für die besten Systeme kann auch eine sogenannte *resilient to failure* Charakteristik implementiert werden. Dies entspricht der Zuverlässigkeit eines Agenten im Falle eines Hardwarefehlers. Nach einem Ausfall kann der Agent seinen Zustand, den er vor dem Ausfall hatte, wiederherstellen. Dann kann er mit seiner Arbeit fortfahren.

3.2 Abstraktion/Agentverhalten

Agenten als intentionale Systeme:
Man betrachtet dabei nur die typischen Merkmale von Agenten und nicht ihre praktischen Eigenschaften, die in 3.1 beschrieben sind. Diese sind nur praktische Anwendungen der Agententheorie und können von einem System zum anderen verschieden sein.

Ein erster Schritt in der Abstraktion ist es, einen Agenten als etwas das handelt zu betrachten. Dies ist unzulänglich, weil es nicht die Unabhängigkeit eines Agenten in Betracht zieht. Der weitere Schritt ist einen Agenten mit einem Menschen zu vergleichen ([WJ95]).

Menschliche Handlungen können durch Beweggründe wie Hoffnung, Glaube, etc. erklärt (und vorhergesagt) werden. Man nennt diese Beweggründe die intentionale Vorstellung (intentionnal notions).

Beispiel [WJ95]: "Janine hat ihren Regenschirm mitgenommen, weil sie dachte, daß es regen würde"

Um dennoch einen greifbaren Begriff zu erhalten, können diese Beweggründe in 2 Kategorien klassifiziert werden:

– Informationseinstellungen (Glaube, Kenntnis, . . .): sie beinhalten Auskünfte über die Umgebung des Systems.

– Pro-Einstellungen (Wünsche, Absicht, Pflicht, ...): sie bedingen die Aktionen des Systems.

Nach [WJ95] ist ein Agentenverhalten durch eine bestimmte Kombination dieser Einstellungen repräsentiert:
"...ein Agent soll durch mindestens eine Informationseinstellung und mindestens einer Pro-Einstellung repräsentiert werden. ..."
Dies kann mit einem PROLOG Programm verglichen werden: Ein Prolog Programm besteht aus 3 Typen von Klauseln ([Bra87]):

die Fakten entsprechen den Kenntnissen des Programms.

die Reglen ziehen Folgerungen aus erfüllten Bedingungen (Bedingungsteil :- Folgerungsteil).

Die Anfragen erlauben es dem Benutzer, Fragen an ein Programm zu stellen.

Das Programm beanwortet die Anfragen, indem es die Regeln auf die Fakten angewendet. Die Fakten entsprechen somit den Informationseinstellungen des Programms und die Regeln seinen Pro-Einstellungen.

3.3 Prinzipien eines Agentenverhaltens

Die Benutzung von Agenten im Software Eingineering wirft einige Probleme auf. Grundsätzlich sind Agenten unabhängige Softwarekomponenten: wenn der Benutzer einen Agenten hergestellt hat, kann dieser Agent ohne Hilfe des Benutzers arbeiten. Agenten können auch im Netz wandern, d.h. daß Server- und Client- Computer Agenten akzeptieren müssen. Deshalb muß der Client (und der Server) sicher sein, daß ein Agent seinen Auftrag erfüllen wird und daß er keine anderen Agenten zerstören oder schädigen wird. Diese Probleme entsprechen der Möglichkeit unredliche Agenten zu schaffen. In [GK94] sind Prinzipien eines Agentenverhaltens vorgeschlagen, die ein Agent aufweisen muß, um unredliche Agenten zu vermeiden.

Veracity: Ein Agent muß die Wahrheit sagen. Obwohl dieses Konzept ganz abstrakt ist (wie kann ein Software Produkt die Wahrheit sagen? Hat das einen Sinn?), kann man es so verstehen: die Tätigkeiten eines Agenten müssen in Übereinstimmung mit seinem *Glauben* (d.h. seiner Informationseinstellung) sein.

Autonomie: Ein Agent darf nicht einen anderen Agenten zwingen, etwas zu tun und kann ferner alleine arbeiten, d.h. er braucht keine (direkte) menschliche Hilfe um seine Aufgabe zu erfüllen.

Commitment (Pflicht, Vertrag): Ein Agent muß die Arbeit, für die er programmiert wurde, machen. Er muß den Vertrag, den er mit dem Benutzer abgeschlossen hat, erfüllen.

4 Mobile Agenten

Moderne Informationssyteme entwickeln sich in Richtung verteilter Anwendungen. Diese Anwendungen überwinden Adreßräume, Hardwarearchitekturn und auch Administrationsgrenzen.

Mehrere Techniken stehen dafür zur Verfügung. Die bekannteste ist wahrscheinlich der Remote Procedure Call. Andere Möglichkeiten sind zum Beispiel verteilte Objekte (wie CORBA) oder dynamisch installierbarer Code. Bei letzterem handelt es sich um Programme, die über ein Netzwerk gesendet werden, um auf einem anderen Computer ausgeführt zu werden. Dabei spricht man auch von *mobilen Agenten.*

In einem Forschungsbericht [HCK95] hat ein Team von IBM definiert, wie ein mobiler Agent aussehen sollte. Ihre Definition eines Agenten war:

> "Mobile Agenten sind Programme, die typischerweise in einer Skript-Sprache geschrieben sind. Sie können von einem Client-Computer verteilt und zu einem entfernten Server-Computer übertragen werden, um dort ausgeführt zu werden."

Diese Definition ist nicht vollständig, weil sie nicht alle Eigenschaften von Agenten betrachtet. Eine bessere Definition wäre:

> "Mobile Agenten sind Softwarekomponenten, die als Vertreter eines Benutzers eine Aufgabe bearbeiten können. Zu diesem Zweck können sie über ein heterogenes Rechnernetzwerk wandern, um entfernte Ressourcen oder Dienste zu erreichen.

Das Heterogenitätsproblem ist nicht einfach zu lösen, aber für die Entwicklung von Agenten zentral. Sobald ein Standard für Agenten verabschiedet ist, werden alle Anwendungen dank Agenten miteinander kommunizieren können.

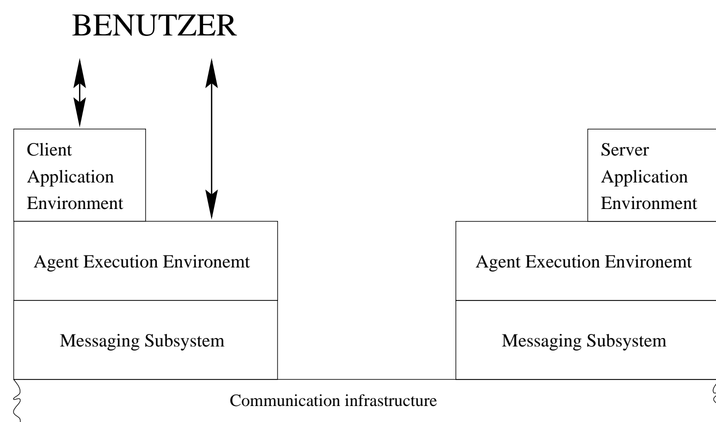

Abbildung4. Konzeptuelles Modell von mobilen Agenten Computing

4.1 Mobile Agent Software Engineering

Die folgende Beschreibung eines Agentensystems wurde in [HCK95] vorgeschlagen (siehe auch Abbildung 4). Sie beschreibt, wie Agenten in eine verteilte Architektur integriert werden können.

Konzept:
Auf einem Client-Computer laufen Anwendungen, die mit einem Server kommunizieren (z.B. Mail, Web).

Diese Anwendungen sind an ein Agent Execution Environment gebunden, d.h. Agenten und Anwendungen können Daten durch ein oder mehrere APIs austauschen.
Das Agent Execution Environment enthält mehrere Agentenprogramme, die zu der Basisumgebung gehören oder von einem anderen Rechner stammen. Die verschiedenen Agenten haben ihre eigene Funktionalität; zum Beispiel ein Agent für Electronic Mail, einer für Datenbankanfragen, einer für Web Browser, etc. .
Der Benutzer kann einen Agenten erzeugen

- entweder mittels seiner üblichen Anwendungen
- oder mittels einer Schnittstelle der Agentenumgebung (in diesem Fall müssen Agenten Zugriff auf *libraries of the device* haben).

Agenten müssen auch Zugriff zu Kommunikationsdiensten haben, damit sie über das Netz wandern können.
Der Prozess zur Erzeugung eines Agenten vollzieht sich dabei in folgenden Schritten:

1. Initialisierung: der Agent bekommt alle Informationen, die für seinen Auftrag notwendig sind. In dieser Stufe hat der Agent seinen eigenen Prozeß.
2. Der Agentenprozeß stoppt, bricht aber nicht ab. Der Prozeß, sein Prozeß-Status, Stack, Heap und externe Referenzen, d.h. alle Daten die für seine weitere Ausführung erforderlich sind, werden in einem Datenpuffer gespeichert. Das Format des Datenpuffers muß aus Gründen der Heterogenität plattformunabhängig sein. Dafür ist nach [HCK95] eine interpretierbare Sprache vorzuziehen. Es ist ebenfalls besser, wenn der Agent aus Objektklassen gebildet ist.
3. Diese Daten werden dann über das Netzwerk an den Zielrechner geschickt.
4. Auf dem Server wird aus den Daten wieder ein ausführbarer Agent erzeugt. Diese Operation findet in dem Agent Execution Environment des Servers statt.
5. Ausführung: der Agent startet mit seiner Abarbeitung.
6. Der Agent hat in diesem Zustand verschiedene Möglichkeiten:
   (a) Er kann seine Aufgabe zu Ende führen und dann zu dem Client zurückkehren.
   (b) Er kann eine bestimmte Zeit oder auf ein Ereignis warten. Der Agent wird zwischenzeitlich *resident* auf dem Server verbleiben.
   (c) Er kann zu einem anderen Server gehen.

4.2 Agentensprache

Warum sind interpretierte Sprachen für Agenten besser geeignet?
Verteilte Anwendung und Software-Interoperabilität sind ein wichtiges Ziel für Agenten. Dabei stellt sich das Problem der Heterogenität, weil Agenten auf verschiedenen Computern erzeugt werden und auf verschiedenen Computern ausgeführt werden. Interpretierte Sprachen haben den Vorteil, daß sie Heterogenität besser unterstützen. Außerdem haben diese Sprachen den Vorteil des *late-binding*. Dies erlaubt einem Agenten Funktionen, die nicht auf dem Client-Rechner sondern nur auf dem Server verfügbar sind zu referenzieren. Wenn der Agent eine interpretierte Sprache und zusätzlich Objektklassen benutzt, kann er auch Referenzen auf verteilte Objekte haben. Damit kann man die Menge an Information, die der Agent enthält, reduzieren.

4.3 Agent Communication Language

Die Benutzung einer interpretierten Sprache und von Objektklassen löst das Problem der Heterogenität nicht ganz. Ein Problem stellt sich bei der Kommunikation zwischen Agenten (und auch zwischen Agenten und Anwendungen). Programme sind nicht alle in derselben Sprache geschrieben. Deswegen können sie auch nicht unbedingt Daten direkt austauschen. Das gilt sowohl für die Syntax als auch für das Vokabular. Sprachen können dieselben Wörter in ihrem Vokabular haben, ohne daß sie dieselbe Bedeutung haben.
Man kann dieses Problem durch die Entwicklung einer allgemeinen Kommunikationssprache [GK94] lösen. Mit einer solchen Sprache können Agenten Daten, logische Informationen, Scripts (Programme), d.h Informationen und Ziele, austauschen.

Agent Communication Language (ACL) ist ein Beispiel für eine solche Sprache, welche im Rahmen des ARPA Knowledge Sharing Effort (zitiert in [GK94]) definiert wurde.
Die ACL ist in 3 Teilen gegliedert:

- ein Vokabular,
- eine interne Sprache, die KIF (Knowledge Interchange Format) heißt,
- eine externe Sprache, die KQML (Knowledge Query and Manipulation Language) heißt.

Programme die miteinander arbeiten möchten, tauschen ACL Nachrichten aus. Diese Nachrichten bestehen aus einem KQML Ausdruck, der selbst aus KIF-Sätzen besteht. Nach [GK94] muß ein Software-Agent fähig sein, ACL-Nachrichten zu lesen und zu schreiben. Die Verhaltenszwänge (siehe Kapitel 3.3), die ein Agent beachten muß, werden von ACL-Nachrichten vorausgestzt.

Diese Kommunikationsprache führt zu dem Problem, daß bereits existierende Anwendungen, die mit Agenten arbeiten wollen, sie nicht direkt verstehen.

5 Anwendungsperspektive

Die Benutzung von Agenten ist nicht nur als Alternativen zu exitierenden Anwendungen zu sehen, vielmehr weist sie auch neue Perspektiven für die zukünftige Entwicklung des

Netzwerks auf. In General Magic's Telescript Home Page kann man lesen, daß Telescript das aktuelle passive Netzwerk in ein aktives Netzwerk verwandelt. Das bedeutet, daß viele Aufgaben die heute manuell ausgeführt werden, wie z.B. die Suche im Web, mit der Benutzung von Agenten automatisiert werden können.
Die interessantere Perspektive für Agenten ist wahrscheinlich der elektronische Handel und die damit verbundene Entwicklung von Marktplätzen im Inneren des Netzwerks.

5.1 Mobile Client

Der Einsatz im Bereich mobiler Systeme weißt die folgenden Besonderheiten auf:

- drahtlose Kommunikation
  Mobile Systeme leiden unter den Schwächen drahtloser Kommunikation. Diese Kommunikationsmittel haben geringe Bandbreiten und höhere Kosten. Deshalb sind sie nicht gut geeignet für die Übertragung von Informationen, die immer zahlreicher werden. Zum Beispiel nehmen die Such- und Filteraufgaben viel Zeit in Anspruch.

- geringe Speicherkapazität und Rechenleistung
  Obwohl viel Fortschritt in diesem Gebiet gemacht wurde, ist Leistungskapazität von tragbaren Computern schwächer als die von Festnetzrechnern.

Einsatzperspektiven für Agenten:

- Der mobile Client kann einen Agenten lokal erzeugen, wenn er mit einem Server nicht verbunden ist. Dieser Agent wird dann während einer kurzen Verbindung zu einem Server geschickt. Das Ergbnis wird während einer späteren Verbindung zurückgesandt.

- Die Agentlösung kann die Aufgaben des Suchens und Filterns von Informationen besser lösen, indem der Agent die Daten auf dem Server bearbeitet. Daher erhält der Benutzer nur die relevanten Informationen. So ist die Menge der übertragenen Informationen stark reduziert, was folglich auch für die Kommunikationskosten gilt.

- Der Benutzer kann schwierige Aufgaben an einen Agenten delegieren. Dieser bearbeitet die Aufgabe auf einem Server und kehrt nur mit den wichtigsten Ergebnissen zum Client zurück.

Diese Anwendung der Agenten weist die folgenden Vorteile auf:

- Verkleinerung des Netzwerkverkehrs: für dieselbe Transaktion braucht ein RPC mehrere Informationsflüsse zwischen dem Client und dem Server. Diese Flüsse könnten auf die Übertragung eines mobilen Agenten reduziert werden.

- Remote searching and filtering: siehe 5.2

5.2 Semantic Information Retrieval

Heute wird viel Zeit für Informationsuche verwendet, da Informationen immer größer und verteilter werden. Mit einer semantischen Suche anstelle der gewöhnlichen Stichwortsuche wird die Arbeit des Suchens und Filterns vereinfacht. Indem die Arbeit an einen Agenten delegiert wird, wie man es mit einem Assistenten tun würde, wird dem Benutzer ein großer Teil der Arbeit erspart, die außerdem nicht sehr interessant ist.
Ein Semantic Information Retrieval Agent arbeitet wie folgt: Der Benutzer gibt dem Agenten eine Anfrage ein. Der Agent interpretiert diese Anfrage semantisch. Dafür kann er auch Fragen an den Benutzer stellen, damit die Anfrage klarer wird. Diese Anfrage wird dann mittels eines Agenten an einen oder mehrere Server geschickt. Auf dem Server sucht und filtert der Agent die Informationen, die die Anfrage betreffen.

Ein Problem liegt hier bei der Kommunikationen zwischen dem Client-Agenten und dem Server. Die Anfrage muß sehr deutlich sein, sonst werden die Informationsflüsse zwischen dem Client-Agenten und dem Server gewaltig. Das wichtigste ist, daß der Agent nicht mit einer großen Menge von Informationen zurückkommt.
Eine Möglichkeit wäre es, auf mehrere Agenten, die jeder für sich auf einen Bereich spezialisiert sind, zurückzugreifen.

Da die klassiche Methode daraus besteht, eine große Menge Daten zu holen und sie manuell oder mit Hilfe eines Programms zu bearbeiten, scheint es viel einfacher zu sein, ein Programm, das die Informationen bearbeitet, direkt an die Quelle zu schicken und nur die relevanten Informationen zurückzubekommen.

Die Vereinigung eines Semantic-Retrieval-System mit einem Multiagenten System kann die Leistungen der entfernten Suche und Filterung verbessern. Ein Multiagent besteht aus mehreren spezialisierten Agenten. Jeder Agent besitzt mehrere Indizes oder Referenzen für die Datenquellen, die jeweils seiner Spezialisierung entsprechen. Zusätzlich kann der Agent selbst seine Referenzen auf den neusten Stand bringen, indem er selbst zu den Server geht und abfrägt, welche Daten neu oder geändert sind (Im Vergleich hierzu ist eine Schlüsselwortsuche statischer; ein Schlüsselwort kann mehrere Bereiche betreffen, die nichts mit der aktuellen Suche zu tun haben).

5.3 Electronic Commerce

Eine wichtige Anwendung für Agenten ist wahrscheinlich der *electronic commerce*. Unter Electronic Commerce versteht man Handelsaustausch mittels des Netzwerks. Ein Beispiel wäre der Versandhandel, wobei ein elektronischer Katalog das Anwendungsprogramm bildet.
Die Händler haben Anwendungen, in denen sie ihre Produkte präsentieren. Der Kunde kann diese Kataloge über das Netzwerk einsehen. Wenn er etwas kaufen will, muß er eine bestimmte Transaktion mit dieser Anwendung ausführen. Electronic Commerce kann sich auf viele Produkte beziehen: Eintrittskarten für Konzerte oder Veranstaltungen, Flug- oder Zugtickets (und Reservierungen), etc. .
Man erkennt dabei ein interessantes Anwendungsgebiet für Agenten: statt sich selbst den Katalog anzusehen, kann der Kunde diese Arbeit an einen Agenten delegieren. Mit ei-

nem höher entwickelten Agenten können kompliziertere Aufgabe von Agenten ausgeführt werden.

- Ein Agent kann mit der Händleranwendung kommunizieren, um ein Produkt zu wählen. Er kann auch selbst ein Produkt einkaufen, wenn es den Kriterien des Kunden entspricht (Preise, technische Kriterien, etc.).
- Ein Vertreiber kann Agenten mit seinen Angeboten an mehrere Kunden verteilen.
- Ein Kunde kann seinen Wunsch an einen Agenten geben. Der Agent kann dann zu verschiedenen Händlern gehen, um Produkte zu vergleichen. Er kann das Produkt bei dem Händler kaufen, bei dem es beispielsweise am billigsten ist.

Das General Magic's White Paper über den Telescript Agenten ([Whi94]) schlägt noch mehr Szenarien vor, die die Möglichkeiten von Agenten im elektronischen Handel vorstellen.

6 Technische Problemen: Sicherheit, Migration

Abschließend seien die technischen Probleme betrachten, die von Agenten aufgeworfen werden.
Vorausgesetzt, daß eine universelle Kommunikationsprache für Agenten verabschiedet wird und daß Agenten sich verbreiten werden, stellt sich die Frage nach der Migration existierender Anwendungen auf diese Technologie. Wird es nötig sein, alle Anwendungen wieder neu zu entwickeln?
Ein anderes Problem, das nicht zu vernachlässigen ist, ist die Sicherheit von Agentensystemen. Sicherheitsaspekte sind problematisch, da Agenten Zugriff zu Ressourcen haben werden, die nicht dem Agenten (bzw. Client) gehören. Der Server muß unbedingt wissen, wem der Agent gehört, damit er bestimmen kann, welche Rechte der Agent hat.

6.1 Migration (Agentification)

Drei Möglichkeiten stehen zur Verfügung, damit eine konventionelle Anwendung mit Agenten arbeiten kann:

Transducer: Ein Transducer versteht zwei "Sprachen", bzw. hat er 2 Schnittstellen (siehe Abbildung 5):

- eine zum Agenten: sie versteht Nachrichten in ACL
- eine zu dem existierenden Programm: sie kennt die Kommunikationsprotokolle dieses Programms.

Der Transducer übersetzt Nachrichten in ACL und liefert diese an das Programm und umgekehrt.
Für dieses Verfahren sind keine Kenntnisse über die innere Struktur des Programms notwendig. Es ist vorteilhaft, wenn man kein Quellcode des Programms besitzt, sondern nur eine Beschreibung der Kommunikationsmittel (d.h. eine externe Spezifikation) des Programms.

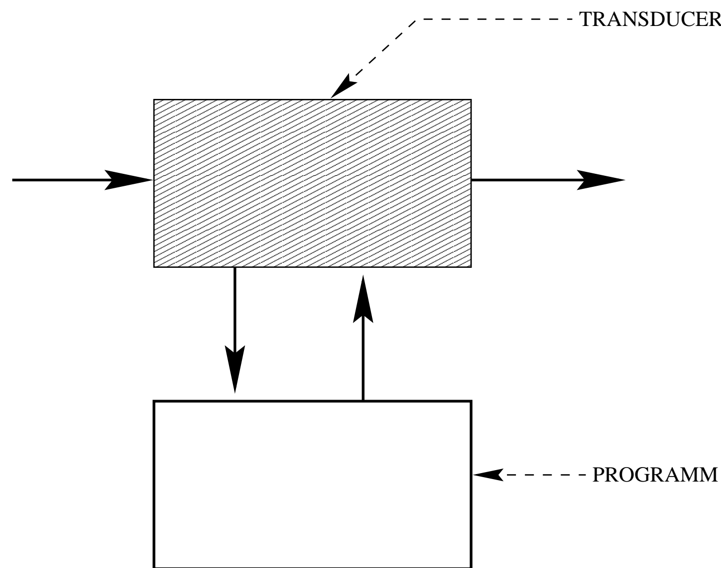

Abbildung5. Transducer

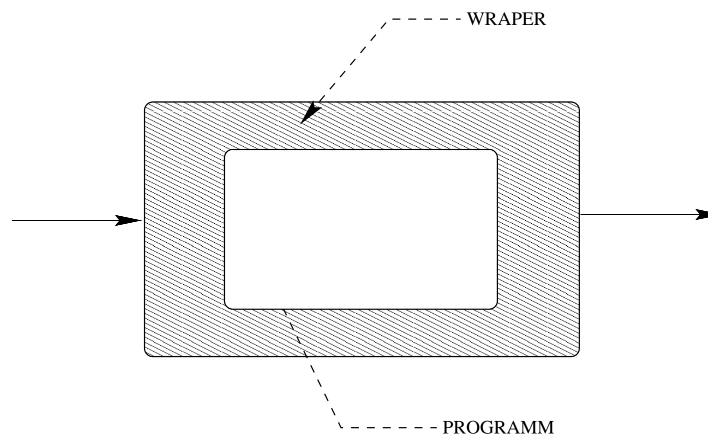

Abbildung6. Wrapper

Wrapper: Diese Technik besteht daraus, in einem Programm zusätzlichen Code einzufügen, damit das Programm ACL Nachrichten verstehen kann (siehe Abbildung 6).

Rewrite: Obwohl diese Methode drastisch ist, hat sie einen Vorteil: man kann das Programm verbessern, d.h. nicht nur das Programm ändern, damit es selbst agentenbewußt wird, sondern auch seine Leistungen erhöhen.

Die Transducer- und Wrapperlösung haben den Vorteil, daß man das Programm nicht ganz verändern muß. Aber ist es oft so, daß die Änderung eines Programms mehr Zeit erfordert, als eine neue Entwicklung dieses Programms. Man muß also gut abwägen, ob die Programmänderung nicht teurer als eine Neuentwicklung ist.

6.2 Security

Damit keinem Server durch einen Agenten geschadet wird, brauchen agenten-basierte Systeme Verfahren, um für Zuverlässigkeit in diesen Systemen zu sorgen. Drei Punkte

sind dabei zu betrachten:

User Authentisierung: Der Server muß wissen, woher der Agent kommt. Er kann entweder den Benutzer (Client) selbst identifizieren, oder der Benutzer kann einer zulässigen Gruppe angehören. Diese Authentisierung ist notwendig, wenn der Server geschützte Dateien enthählt.

Autorisierung: Darf der Benutzer einen Agenten auf dem Server ausführen? Welche Funktionen darf der Agent benutzen? Wird der Agent dem Server oder anderen Agenten schaden? Der Server muß hier wissen, welches die Absichten des Agenten sind. Wenn der Server den Benutzer authentisiert hat, wandelt er den Agenten in eine ausführbare Form. Bevor der Server den Agenten seine Arbeit machen läßt, inspiziert er den Agentencode, um zu sehen, zu welchen Ressourcen der Agent Zugriff haben möchte. Wenn der Server bestimmt hat, daß der Agent wohlgesinnt ist, läßt er den Agent sich ausführen.

Zahlung: Obwohl heute viele Dienste kostenlos sind, wird es vielleicht in Zukunft anders sein. Für verteilte Anwendungen muß der Server wissen, ob der Client für die Dienste, die er benutzen wird, zahlen kann.
Beispiel: Telescript, der General Magic's Agent (siehe [HCK95])
Die Telescript-Sprache besitzt eine elektronische Währung, die Teleclick heißt. Ein Agent hat eine bestimmte Menge von Teleclicks. Während der Laufzeit werden Teleclicks von dem Agenten zu dem Server überwiesen.

Diese Verfahren sind notwendig, aber man weiß noch nicht genau, wie sie implementiert werden sollen. Agentensysteme sind auch nicht gegen eventuelle unredlichen Agenten geschützt. Beim elektronischen Handel kann es auch Opfer diebischer Agenten geben. Deswegen müssen Agentensysteme vor allem das Zuverlässigkeitsproblem lösen.

6.3 Virus Detection

Leider sind Agenten auch gut zur Übertragung von Viren geeignet. Während der Authorisierung muß der Server bestimmen, ob der Agent einen Virus enthält. Diese Aufgabe ist nicht einfach und hängt von der Sprache des Agenten ab. Da eine Sprache für Agenten noch nicht voll definiert ist, ist es möglich, eine Sprache zu entwickeln, die keinen Virus enthalten kann (möglich in der Theorie). Mit dieser Sprache würde es für einen Agenten nicht möglich sein, einem anderen Agenten oder einer Anwendung zu schaden. Agenten wären dabei auf eine gewissen Menge von Funktionen begrenzt:

- Modifizierung der eigenen Variablen,
- Anfrage an Datenbanken auf dem laufenden Server,
- Gang zu einem anderen Server,
- Sendung von Textnachrichten an den Benutzer.

7 Zusammenfassung und Ausblick

Agenten selbst bieten keine besseren Möglichkeiten als die existierenden Systeme.
Im Gebiet verteilter Anwendungen haben sie keine Funktionalitäten, die andere Systeme (RPC, messaging) nicht auch bieten.
Der Vorteil der Agenten ist, daß sie mehrere Funktionalitäten, die bis jetzt von verschiedenen Systemen unterstützt werden, in einem einzigen System vereinigen. Beispielsweise können Agenten für die Ausführung eines Programms auf einem entfernten Rechner oder für die Sendung einer Mail dienen. Ebenfalls ist die Delegierung von Aufgaben, die bis jetzt manuell ausgeführt wurden, möglich.
Intelligente Agenten können sehr wichtig werden, wenn sie den Benutzer besser unterstützen, ohne daß es für ihn nötig sein wird, neue Methoden zu erlernen. Aber die Entwicklung von Agenten braucht noch Forschung in verschiedenen Gebieten der Informatik ((V)KI, verteilte Anwendungen).
Ein Problem von Agenten ist die Sicherheit, aber dieses Problem stellt sich nicht nur für Agenten, sondern auch für alle verteilte Anwendungen.
Die zukünftige Aufgabe für Agenten wäre, eine Lösung für Problemen, die die aktuellen Anwendungen haben, vorzuschlagen.

Zusammenarbeit beim Trading: ANSA vs ODP

Teresa Ortega

Kurzfassung

Erste Ansätze zur Unterstützung der Verwaltung und Vermittlung von Diensten (sog. *Trading*) in offenen verteilten Systemem wurden in Rahmen des Open Distributed Processing Modells (ODP) und des Advanced Network System Architektur Projektes (ANSA) entwickelt. Das Trading-Konzept ergibt sich als eine wichtige strukturierende Technik bei der Unterstützung von Kooperationen von Dienstbringer und Dienstnehmer. Folgende Probleme treten dabei auf: Das ständige Wachstum des Marktes (Skalierung), die transparente Bereitstellung der Dienste bei den heterogenen autonomen Trading-Teilnehmern, nämlich den Tradern (Dienstvermittler), der Dienstbringer und der Dienstnehmer und einer größtmöglichen effizienten Suche von Diensten von dem Dienstnehmer. Um mit diesen Problemen zurechtzukommen, sind mehrere organisatorische Modelle für die Zusammenarbeit zwischen Tradern möglich. In dieser Ausarbeitung werden die Modelle zur Zusammenarbeit zwischen Tradern des ANSA-Projektes und des ODP-Modells dargestellt und schließlich gegenübergestellt. Der Vergleich wird durch die fünf Sichtweisen (*Viewpoints*) des ODP durchgeführt, welche als Referenzpunkte zur Untersuchung und Beschreibung von verteilten Systemen dienen.

1 Einleitung

Ein Trader ist ein Objekt, dem ein anderes Objekt (auch *Dienstgeber* oder Exporteur genannt) seine Dienste anbieten (exportieren) kann, und von dem ein anderes Objekt (auch *Dienstnehmer* oder Importeur bezeichnet) Dienste in einem verteilten System importieren kann [BR91].

Die Komplexität der dabei anfallenden Probleme beruht u.a. auf der Heterogenität und Offenheit der verwendeten Netze und Dienstbringer, sowie der Diskrepanz zwischen möglichst parallel zu unterstützenden Integrations- und Autonomieanforderungen der beteiligten Knoten.

Die Vorstellung und der Vergleich beider Modelle wird mit Hilfe der fünf Viewpoints des ODP Modells durchgeführt und ist in dieser Ausarbeitung wie folgt zusammengestellt: Abschnitt 2 stellt die beiden Organisationen, ihre Ziele und bisherige Arbeiten bei der Entwicklung der verteilten Systeme dar, und es werden ihre jeweiligen einfachen Trading-Szenarios als Grundlage ihrer Trading-Modelle erläutert.

Abschnitt 3 führt einige grundlegende Konzepte ein, die zum Verständnis der Modelle für die Zusammenarbeit zwischen Tradern benötigt werden.

Abschnitt 4 beschreibt die beiden Modelle, wobei bei der ODP-Beschreibung diejenigen Aspekte wo beide übereinstimmen ausgeführt werden. Bei der ANSA Beschreibung wer-

den nur ergänzende Aspekte angegeben. Schließlich werden im Abschnitt 5 die beiden Modelle verglichen und im Abschnitt 6 die wichtigsten Aspekte zusammengefaßt.

2 Verwandte Standardisierungsprojekte in offenen verteilten Systemen

2.1 Entwicklungstand

Mit dem generellen Ziel einer weitgehenden Systemintegration (schrittweise und zunächst in Teilaspekten) werden international zur Zeit verschiedene Standardisierungsbemühungen verteilter Anwendungen parallel zueinander verfolgt.

Zur Erläuterung dieser Entwicklungen werden beispielweise die folgenden zwei Projekte im Bereich von Vermittlung und Verwaltung von Diensten (Trading) erwähnt und später näher erklärt:

- Das ISO/CCITT Open Distributed Processing Reference Model (ODP-RM) als Versuch, einen sehr abstrakten, generellen Rahmen zur Einordnung aller unterschiedlichen, vergangenen und zukünftigen Standardisierungsprojekte im Bereich offener verteilter Systeme zu schaffen.
- Das Advanced Network System Architechtur Project, ursprünglich ein vom U.K. ALVEY-Programm für fortgeschrittene Forschung und Entwicklung in Informationstechnologien gegründetes Projekt zur Erzeugung eines kohärenten Modells für verteilte Systeme.

2.2 ODP-RM: Open Distributed Processing Reference Model

Im Rahmen der internationalen Standardisierung hat sich die ISO als Teil ihrer Aktivitäten im Bereich des ODP damit befaßt, einen Dienst zur Vermittlung zwischen Dienstnehmer und Dienstgeber unter die Bezeichnung ODP-Trader zu entwerfen und weltweit zu standardisieren.

Die Arbeit im Bereich des Open Distributed Processing der ISO wurde 1987 [Her89] gestartet. Das ODP-RM besteht aus vier verschiedenen Teilen: Der erste ('Overview and Guide to Use of the Reference Model') führt in die verwendeten Konzepte ein. Dort wird versucht, ein verteiltes System aus fünf unterschiedlichen Sichtweisen *(Viewpoints)* oder *Projections*, wie sie dort genannt werden zu betrachten und diese im Einzelnen zu beschreiben.

Der zweite Teil des ODP-RM (Descriptive Model) definiert Konzepte, Rahmen und Notationen zur Beschreibung verteilter Systeme. Er stellt die wesentlichen Grundbegriffe zusammen, die zur Beschreibung allgemeiner verteilter Systeme verwendet werden, um eine einheitliche Verwendung dieser Begriffe bei der Beschreibung unterschiedlicher verteilter Systeme zu ermöglichen. Sie ist informell und soll lediglich als Ausgangspunkt für nachfolgende (auch formale) Spezifikationen dienen.

Der dritte Teil enthält die Spezifikation der notwendigen Eigenschaften, die eine verteilte Verarbeitung als "offen" charakterisieren (wie z. B. der unterschiedlichen "Transparenz" Eigenschaften). Er gibt damit normierend die Randbedingungen und Einschränkungen vor, die alle ODP-konformen Standards einzuhalten haben, und benutzt dazu die Beschreibungstechniken aus dem zweiten Teil des ODP-RM.

Der vierte Teil (architectural semantics) soll schließlich eine vollständige formale Beschreibung der ODP-Modellierungskonzepte durch die Spezifikation jedes der dabei verwendeten Konzepte mit Hilfe von verschiedenen standardisierten formalen Beschreibungsverfahren geben.

Während die hier genannten Aktivitäten von OSI und CCITT noch einen ganz allgemeinen, eher konzeptionellen Rahmen zur einheitlichen Beschreibung verteilter Systeme anstreben, zielen die Standardisierungsaktivitäten verschiedener industrieller Gruppierungen darauf ab (darunter ANSA), ganz konkrete Software-Umgebungen zur Förderung der Interoperabilität von Anwendungen und Diensten in offenen verteilten Systemen zu vereinbaren und zu realisieren. [Lam94]

2.3 Viewpointsdefinition

Da die Vorstellung und Vergleich der beiden Modellen durch die Viewpoints durchgeführt werden, sollen diese hier eingeführt werden:

Abbildung 1 stellt die Viewpoints dar.

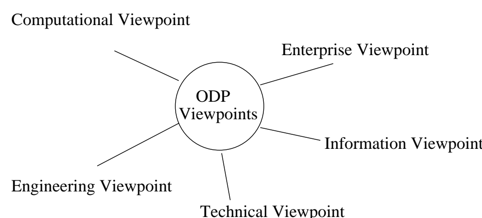

Abbildung7. Viewpoints des ODP-RM

- *Der Enterprise Viewpoint* stellt im einzelnen die organisatorischen Strukturen im System, ablaufende Prozeduren, etc. auf der Basis einer vollständigen Systemanalyse dar.

- *Der Information Viewpoint* beschreibt die Informationstypen und die Relationen zwischen denen, die für die Definition der Trading-Funktion benötigt werden, z.B. Schnittstelle (Identifikation, Typ, usw.), Eigenschaften, Dienstypen.

- *Der Computational Viewpoint* definiert ein Trader-Object-Template (Format). Dieses enthält die Templates für alle beim Trader generierten Schnittstellen und die Beschreibung des Trader-Verhaltens.

- *Der Engeneering Viewpoint* beschreibt (z.B. für den Systementwickler) das verteilte System als Modulbaukasten mit einzelnen Systemkomponenten wie Programmier-

sprachen, Datenbanken, sonstige Soft-und Hardware-Ausstattung (inklusive Aspekte wie Perfomance, Transparenz, 'Quality of Service', etc).

– *Der Technological Viewpoint* stellt schließlich -z.B. für Hard und-Softwarehersteller -ganz konkret die dem verteilten System zugrundeliegende Hardware und Betriebsystembasis dar (inlusive Anforderungen wie etwa für 'Multi Threading' etc).

Trading-Szenario im ODP-Modell: Um das zu einer Dienstanforderung zusammenpassende Dienstangebot zu finden, ruft der Trader die Type-Repository-Funktion (Typenbehälter) auf, die bei der ODP-Infrastruktur bereitgestellt wird. Die Menge der bei einem Trader registrierten Diensttypen ist bei der Type-Repository-Funktion bekannt, ebenso die assoziierten Schnittstellentypen für jeden Diensttyp. Sie enthält sowohl eine Menge von Relationen für die Schnittstellentypen (Äquivalenzrelationen, Ober-/Unterrelationen) als Ober-/Unterrelationen der Dienstypen. Ein Trader ruft folgende Operationen auf:

Type Check - zur Überprüfung der Typengültigkeit im Diensangebot und in der Dienstanforderung.

Compatibility Check - um die Kompatibilität zwischen zwei Dienst/Schnittstellen-Typen zu überprüfen.

List Type Relationship - Auflisten der Untertypen bestimmter Dienst/Schnittstellen-Typen

Fig. 2 stellt das Szenario dar. Der Trader bekommt eine Anforderung eines Klienten und überprüft mit Hilfe der Type-Repository-Funktion, ob die Anforderung einen gültigen Schnittstellentyp oder Dienstyp enthält. Die Type-Repository-Funktion gibt eine Liste der Typen zurück, die (gleiche oder Ober-/Untertypen) mit dem angefordeten Typ kompatibel sind.

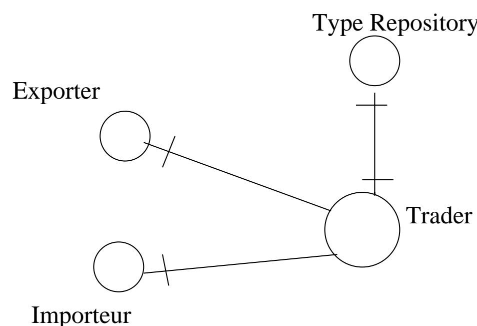

Abbildung8. ODP Trading-Szenario

Der Auswahl kann noch verfeinert werden, indem man Diensttypeigenschaften verwendet, um zusammenpassende Dienste zu finden.

2.4 ANSA: Advanced Network Systems Architechture

Mit dem Ziel, eine Architektur für verteilte Systeme zu schaffen bzw. diese Architektur als einem industrieweiten Standard zu etablieren, haben sich einige größere Firmen zusammengeschlossen. Die ANSA Sponsoren sind: Britische Telekom, Digital Equipment Corporation, General Electric Company/Marconi, Hewlet Packard, International Computers Limited, Information Technology Limited, Olivetti Research Ltd, Plessey Office & Networked Systems und Racal.

Man wollte die beträchtlichen Forschritte in der Forschung auf dem Gebiet der verteilten Systeme des letzten Jahrzehntes bzw. deren gegenwärtigen Stand für die Schaffung eines einheitlichen und modularen Rahmens zur Untersuchung und Konstruktion verteilter Systeme von verschiedenen *Viewpoints* (dieselbe wie bei ODP) ausnutzen. Stichworte dieses Rahmens sind: Portabilität, Erweiterung und Entwicklung aktueller industrieller Standards wie UNIX und OSI. Das Projekt war im Standardisierunsbereich aktiv. -Mehr als die Hälfte der aktuellen ISO-ODP Dokumente stammen von ANSA. [Her89]

Die fünf *Viewpoints* schließen alle unterschiedliche Zwecke und Bedürfnisse verteilter Systeme ein, ausgehend von den Anforderungen von Industrie- und Handelsfirmen, bis zu Hard- und Softwareüberlegungen [MB92].

Eine größere Leistung des ANSA Teams ist das ANSA Testbench. Das ANSA Testbench ist ein Pilotprojekt, das Einrichtungen zum Importieren und Exportieren von Services enthält, und das eine Implementierungsreferenz bereitstellt, um die Verwendung der ANSA Prinzipien zu beweisen. Das Testbench hat zwei Hauptbereiche : Die *Konstruktion* von Modulen (Objekten) und ihre Zusammenarbeit oder *Manipulation.* Abstrakte Schnittstellen sind durch die Verwendung einer Schnittstellenbeschreibungssprache (Interface Definition Language -IDL-) konstruiert. Diese Tools sollen die Arbeit zur Entwicklung verteilter Anwendungen vereinfachen [MB92].

Zunächst wird das ANSA Computational Model dargestellt, weil seine Komponenten zum Verständnis des Zusammenarbeitsmodelles in ANSA benötigt werden.

ANSAs Computational Modell für das Trading: Das Modell für eine einfache Zusammenarbeit zwischen einem Klienten und den Trader-Diensten besitzt drei Dienstobjekte, wie in Abbildung 3 dargestellt. Ein Klient kann ein Dienstnehmer oder ein Dientgeber sein. Folgende drei Dienste wurden als Teile des Trading-Dienstes identifiziert:

TyDS *Type Description Service* (Typenbeschreibungdienst), speichert Schnittstellendefinitionen.

TyCD *Type Conformance Service* (Typenkonformitätdienst), zur Überprüfung der Typkonformität.

TCtxt *Trading Context* (Tradingkontext), enthält alle Dienstangebote, die zu diesem Kontext gehören.

Das Importszenario ist einfach: Ein Klient instanziert zuerst einen TyDS-Dienst mit einer Schnittstellentypbeschreibung des nachgefragten Dienstes. Danach ruft er den Trading-Kontext TCtxt und gibt ihm die Referenz der TyDS und Information über den Dienst

(eine Eigenschaftsbeschränkung). Das TCtxt sucht dann in seiner Angebotsdatenbank und fragt den TyDS nach Typkonformität. Um das auf diese Weise zu machen, muß dem TyDS eine andere TyDSReferenz von einem anderen Dienst bereitstellen. Die Abbildung 3 stellt dieses Modell dar.

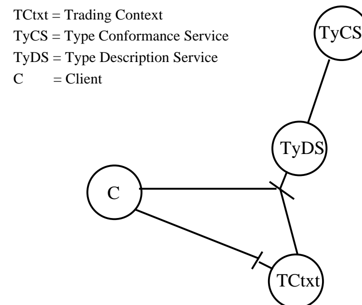

Abbildung9. Computational-Modell in ANSA

3 Grundlegende Begriffe

Die Zusammenarbeit zwischen Tradern wird bei Verwendung von organisatorischen Strukturen der Trader, die kooperieren wollen, und die Klassifikation der bei den Tradern registrierten Diensten in Kontexte, realisiert. An dieser Stelle werden wir die Organisationseinheitsdefinition, Kontextdefinition und die ursprüngliche Bezeichnung der Zusammenarbeit, nämlich Föderation näher betrachten, weil sie für das Verständnis der Zusammenarbeitsmodelle von Interesse sind. Die folgenden Konzepte gelten für beide Modelle.

3.1 Organisationseinheit

Eine Organisationseinheit (Administration) ist eine Menge von Gemeinschaftsgliedern eines verteilten Systems, die weil sie Objektive teilen unter einer gemeinsamen Ordnungsstruktur (Instanz besonderer Eigenschaft) stehen. Um ihre Verpflichtungen zu erfüllen und ihre Ziele zu erreichen, können die Mitglieder Dienste benutzen, die bei anderen Gemeinschaftmitgliedern derselben Organisationseinheit und/oder einer anderen Organisationseinheit(en) angeboten werden.

Organisationsmodelle von Organisationseinheiten: Die Organisationseinheiten können in unterschiedlicher Weise verbunden sein: hierarchische Organisationen oder föderative (zusammenarbeitende) Organisationen.

– *Hierarchie*: Eine Organisationseinheit kann in Unterorganisationseinheiten unterteilt werden, die wiederum unterteilt sein können, und daher eine Baumstruktur bilden.

– *Föderation*: Ist der Versuch der Zusammenarbeit zwischen den verteilten Systemen, ohne sich unter eine zentrale Autorität zu stellen. Jedes System in einer Föderation

ist für sich selbst verantwortlich. Die Abbildung 4 zeigt diese Verbindungsmodelle [ANS93].

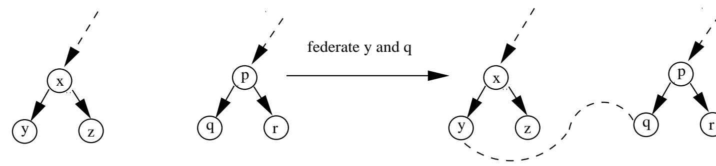

Abbildung10. Verbindungsmodelle der Organisationseinheit

3.2 Kontext

Mit jeder Organisationseinheit ist ein Default-Kontext assoziert. Ein Kontext ist eine Menge von Dienstangeboten, die zu den Dienstbringern einer bestimmten Organisationseinheit gehört.

Jeder Kontext wird mit dem Namen der Organisationseinheit, auf die er sich bezieht, bezeichnet. Zwei Arten von Verbindungen unter Kontexten können erzeugt werden: hierarchische und föderative (zusammenarbeitende) Verbindungen. Jeder Kontext bezeichnet eine Menge Information, die er bereitstellt und nach der er die Suche erlaubt [ANS93, OD94].

3.3 Was ist eine Föderation?

Das Konzept wurde zuerst bei Hiembigner und McLeod in ihrer Arbeit über Informationsdatenbanken eingeführt [Her89]. Eine Föderation ist eine organisatorische Struktur, in der die Mitglieder (die Organisationseinheiten) verhandeln, in wie weit sie ihre Dienste zur Verfügung stellen wollen. Eine Föderation scheint eine Lösung für die Zusammenarbeit von verteilten, autonomen Systemen zu sein.

Eine Föderation von Tradern erlaubt den Mitgliedern eine gesteuerte und partielle Informationsteilung. Die Menge der geteilten Informationen zwischen Tradern, hängt von der Menge der Zusammenarbeit zwischen den Komponenten ab. Jede einzelne Komponente der Föderation muß fähig sein, seine normalen, lokalen Operationen, ohne äußere Störungen, ausführen zu können [BR91].

Verschiedene Trader-Gemeinschaften haben verschiedene Authoritäten und verschiedene Trading-Strategien. Ein **Domain** ist der logische Verwaltungsbereich eines Traders. Wenn ein Trader beim Zugriff auf einen anderen Trader eine **Domain-Grenze** (boundary) überquert, benötigt es einen Interceptor. Ein **Interceptor-Objekt** ist ein Objekt zwischen Tradern, welches die Unterschiede zwischen ihnen überbrückt. [ANS93]

Die Möglichkeiten zum Überschreiten von Domain-Grenzen werden durch Föderationsverträge (zwischen den betroffenen Organisationen) geregelt. Jeder **Vertrag** (contract) legt die im entfernten Trader verfügbaren Typen und Typenstrukturen bzw. die Regeln

und Funktionen zur Abbildung sowohl der lokalen Traderanforderung auf eine verständliche Form für den entfernten Trader als auch die Ergebnisse fest. Man wird später die Verträge näher erläutern [BR91].

Die grundlegende Merkmale von föderativen Traders sind: Trennung, Hetereogeneität, Autonomie und Transparenz.

4 Zusammenarbeitsmodelle von ODP und ANSA

4.1 ODP Modell der Zusammenarbeit

In den ODP-Dokumenten wird statt des Wortes Föderation das Wort "Interworking" verwendet. Das Föderationsmodell ist ein dezentrales Modell sowohl in ODP als auch in ANSA. Um einer Föderation beitreten zu können, muß der Trader mindestens an einen anderen Trader in der Föderation Importieren und Exportieren können.

Für das Exportieren erlauben entfernte Trader die Angabe (innerhalb einer Traderföderation) von Diensttypen, die als lokaler Teil der Dienstangebote erscheinen. Es ist nicht notwendig, daß ein Benutzer Kenntnisse über Einzelheiten eines entfernten Traders oder seines Dienstangebotes besitzt, jedoch könnte der Benutzer diese Kenntnisse (falls er diese besitzt) als Teil seiner Suchstrategie benutzen [OD94].

Ein Trader, der von einem anderen importiert bzw exportiert, hat einen Vertrag mit diesem abgeschlossen. Für jeden Exportvertrag gibt es einen entsprechenden Importvertrag beim entfernten Trader.

Ein Föderationsvertrag dokumentiert die Vereinbarung zur Zusammenarbeit. Er muß ausgehandelt werden, um festzulegen, wo, und wie die Dienstangebote verteilt werden können. Ein Föderationvertrag enthält u.a.: eine globale, eindeutige Vertragsbezeichnung, die Identifikation der beiden Trader, eine hierarchische Struktur des Exporteurs, eine Liste der zur Zeit beim Exporteur angebotenen Dienstypennamen, eine Liste der zur Zeit beim Importeur angefordeten Dienstypennamen. Tatsächlich besteht der Föderationsvertrag aus zwei Teilen: dem Importvertrag, der beim Importeur bleibt und dem Exportvertrag, der beim Exporteur bleibt, wie dies in der Fig 5 veranschaulicht wird.

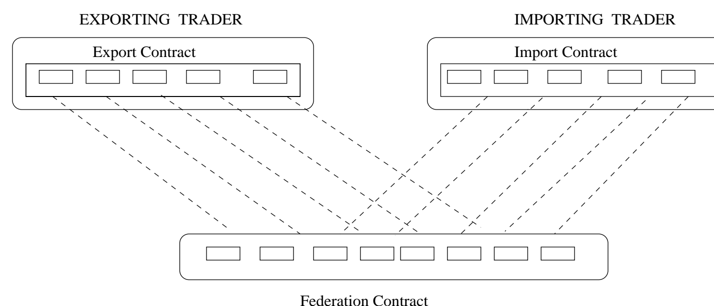

Abbildung11. Föderationsvertrag

Die Sicht eines Export-Traders einer Föderation ist also die Menge von Exportverträgen mit verschiedenen Tradern und die Sicht eines Import-Traders einer Föderation ist die

Menge von Importverträgen mit entfernten Tradern.

Ein Trader beginnt zuerst die Suche einer Benutzeranforderung in der lokalen Traderdatenbank. Ist der angefordete Dienst nicht auf dem lokalen Trader verfügbar, wird die Suche auf eigene Importverträge der entfernten Trader ausgedehnt. Ist der angefordete Dienst auf dem entfernten Trader verfügbar, wird die Anforderung übergeben. Der entfernte Trader beginnt gemäß des entsprechenden Exportvertrags mit der Suche in seiner Datenbank. Erst wenn die Suche vollkommen abgeschloßen ist, wird das Ergebnis -wenn nötig schon transformiert- zum Import-Trader zurückgesendet [BR91].

Beschreibung der Föderation durch Viewpoints

– **Enterprise Viewpoint**: Eine Föderation bildet mit ihren entsprechenden Import-Tradern und Export-Tradern eine Trader-Gemeinschaft. Die Anforderungen zum Importieren und Exportieren werden hier festgelegt. Ein Trader kann von mehreren Tradern importieren/exportieren.

  Es ist keine zentrale Instanz notwendig. Der föderative Trader muß dazu fähig sein, den Benutzern heterogener, verteilter und autonomer Trader, einen transparenten Zugriff bereitszustellen. Die Grundregeln einer Föderation sind:

  - Ein Trader ist nicht verpflichtet, eine Dienstvermittlung für einen anderen Trader auszuführen.
  - Jeder Trader muß die vollständige Kontrolle über seine eigene Handlungsstrategie haben.
  - Jeder Trader hat die Freiheit zum Beitreten bzw. Verlassen einer oder mehrerer Föderationen.
  - Jedes Mitglied bestimmt wie es die angebotene Dienste betrachtet und kombiniert.

  Jeder Trader bestimmt seine eigene Suchstrategie. Die folgenden zwei Strategien wirken sich auf das Importieren aus:

  Global search policy (globale Suchstrategie) führt die Suche von Importangeboten an einem Trader der gleichen Föderation durch.

  Domain crossing policy ist die Strategie, welche die Überquerung der Domaingrenzen steuert, z.B. Domain-Typen, Sicherheits-Domain, Technologie-Domain, etc.

– **Information Viewpoint**: Es enthält folgende Konzepte:

  - *Trader-Link* kann als ein Verweis von einem Quell-Trader (lokaler Trader) auf einen Ziel-Trader (Trader zu dem die Anforderung weitergeleitet wird) angesehen werden. Er identifiziert die Trading-Schnittstelle eines anderen Trader und eine Eigenschaftsmenge des Trader-Links. Beispiel einer Dienstangebotseigenschaft sind die Kosten der Dienste oder das Ablaufdatum der Verbindung.

- *Beschränkungen der Zusammenarbeit*: Die Suchstrategie und die Domain-Überquerungsstrategie sind im Information Viewpoint als **Scope-Beschränkungen** dargestellt.

- *Trader-Informationsobjekt*: Der Zustand eines Trader-Objektes [OD94] in einer Föderation wird durch die folgenden Punkte beschrieben:

  * Menge der Dienstangebote

  * Menge der Trader-Eigenschaften und Auswahlstrategien

  * Verbindungsmenge zu benachbarten Tradern

- *Trading-Graph* stellt das verteilte Wissen, das ein Trader von anderen Tradern in einer Föderation hat, dar. Die Knoten entsprechen Tradern und die Kanten den Verbindungen zu benachbarten Tradern, die von einem Quell-Trader wahrgenommen werden.

Außerdem muß man hier folgende Information beschreiben: Die Trader-Identifikation, vorhandene Dienste zum Importieren und Exportieren, Einschränkungen für den Zugriff auf die Datenbank eines Exporteurs. Diese Informationen werden durch Kataloge, Föderationsverträge und Export/Importverträge angegeben.

Ein Katalog enthält eine Beschreibung der Dienstobjekte (Directory), die zum Exportieren bei einem Trader zu Verfügung stehen, und die Beschreibung der assozierten Diensttypen. Der Katalog kann als ein standardisierter Diensttyp dargestellt werden. Jeder Eintrag im Katalog enthält u.a. folgende Informationen: Den Teil der Verzeichnishierarchie auf die der Importeur zugreifen kann, eine Liste der zur Zeit vorhandenen Namentypen zum Exportieren, die in kanonischer Form gegeben wird. Wenn ein Trader exportieren will, muß er einen Katalog vorbereiten und zu einem anderen Trader schicken. Wenn ein Trader importieren will, mußt er sich einen Katalog besorgen.

– **Computational Viewpoint**: Hier werden relevante Strategien zur Bestimmung der Aktivitätsmenge, die durch den Tradings-Graph unternommen werden können, vereinbart. D.h., bevor irgendeine Zusammenarbeit begonnen werden kann, müssen zuerst die Verträge zwischen den Tradern ausgehandelt werden, um festzulegen, welcher Dienst importiert bzw. exportiert werden kann.

Ein Trader kann sowohl die Rolle eines Klientes spielen als auch im Auftrag eines Benutzers arbeiten. Die Operationen zwischen zwei Tradern können in zwei Kategorien unterteilt werden.

- Aktivitäten für die Föderation, bei denen der Trader als Klient eines entfernten Traders die Export-Import-Verträge sowie die Kataloge aushandelt.

- Föderative Operationen, die der Trader in Auftrag seiner Klienten ausführt.

Zusätzlich werden die Operationen zur Verteilung von Katalogen, Verhandlung von Verträgen, Auflisten und Modifizieren von Verträgen beschrieben.

– **Engineering Viewpoint**: Hier müssen die Einzelheiten der Export- und Importverträge gespeichert und bereitgestellt werden. Alle Übersetzungen zwischen den lokalen Trader-Operationen und den föderativen Operationen zwischen Tradern sind hier für den Benutzer unsichtbar realisiert. In diesem Viewpoint kann man die Benutzung der ASN.1 (Anstract Sintax Notation 1) zur Definition von Typparametern und Operationen der föderativen Trader entscheiden.

  In beiden Modellen sind Vorschläge angegeben, wie ein Trader durch die Verwendung des X.500 OSI Standards (Directory Service) implementiert werden kann.

– **Technical Viewpoint**: Der technologische Viewpoint bezieht sich auf die grundlegende Hardware und Software, die ein implementierter Trader oder eine Gruppe von zusammenarbeitenden Tradern einschließt. Die verfügbaren Hard- und Softwaresysteme beschränken sich auf die Auswahl von:

  - Datenaustausch
  - Datenspeicherung
  - Implementierung der spezifizierten Strukturen und Funktionen

  Der Standard schreibt für den technologischen Viewpoint nichts vor, ermöglicht aber eine spezifische Auswahl von Datendarstellungen, z.B. ASN.1.

ODP-Konfiguration der Föderation: Eine Gruppe von Tradern kann in verschiedenen Formen miteinander in Beziehung treten. Beispielweise könnten sie sich dieselbe Verwaltung, denselben Speicher oder dieselbe Implementierung teilen. Solche in Beziehung stehenden Trader können eine gemeinsame Trading-Schnittstelle für die anderen "externen" Trader bereitstellen. Alle Anforderungen an die Gruppierung, sowie alle eigenen Anforderungen werden durch diese gemeinsame Schnittstelle kanalisiert sein. Nur eine Interceptor-Menge ist für die ganze Gruppe notwendig.

Man stellt sich die in Beziehung stehende Gruppe als einen "logischen" Trader vor. Um diese gemeinsame Schnittstelle zu erreichen, wird der Verwalter der Gruppe einen oder mehrere Trader bestimmen, die Verbindungen zu externen Tradern haben werden (für das "external" Trading). Dieser besondere "Schnittstelle-Trader" hat zwei Schnittstellen: eine "innere" für Trader der eigenen Gruppe und eine "äußere', die die Schnittstelle des logischen Traders ist. Trader-Gruppierungen werden die "inneren" Trader als "Black-Box" betrachten, entweder weil es die Gruppe so will oder weil außenstehende Trader mit den Einzelheiten des logischen Traders nichts zu tun haben wollen. Dies stellt jedoch kein Hindernis für einen inneren Trader zur Festlegung einer Verbindung nach außen dar. Abbildung 6 stellt die Zusammenarbeit zwischen logischen Tradern dar.

Die äußeren Links bestehen nur in einer Richtung. Wenn es einen anderen Link in der Gegenrichtung gibt, dann haben die betroffenen Trader eine wirkliche eins zu eins Beziehung. [OD94]

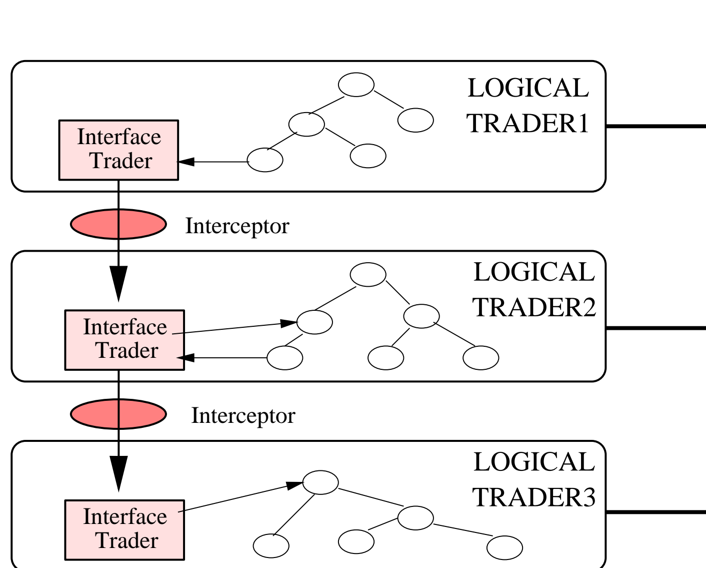

Abbildung12. Zusammenarbeit zwischen logischen Tradern in ODP

4.2 ANSA-Modell

Bisher findet man für die Begriffe, die vorgestellt wurden, außer logischen Trader und Verträge, ihre Parallelität zu ANSA. In Folgenden werden diejenige Aspekte, bei denen das ANSA-Projekt eine weitergehende Arbeit durchgeführt hat, vorgestellt. Zunächst werden die Schnittstelle der Verwaltung in ANSA kurz eingeführt, weil sie in ODP Modell keine parallelen Einrichtungen haben und zur Verwaltung der Zusammenarbeit von Tradern dienen. Die Merkmale des Information-Viewpoints in ANSA wird auch weiter behandelt, weil sie entscheidend für die Bereitstellung von Diensten über Domaingrenzen hinweg sind. Schließlich werden wir noch einen besonderen Trader kennenlernen, nämlich den prokustischen Trader, der ein flexibler Mechanismus für das Trading anwendet, welches die Zusammenarbeit flexibler macht.

Verwaltungsfunktionalität: Der Trader wird durch die folgenden vier Schnittstellen die Verwaltungsaspekte des Trader-Betriebs regeln:

TrType - Type management: Eintragen/Löschen von Typen

TrCtxt - Context management: Eintragen/Löschen/Zeigen/Schachteln von Kontexten

TrFed - Federation management: Binden/Unbinden von Kontextbäume

TrShut - Shutdown. Ermöglicht das Beenden des Traders

Beschreibung des ANSA-Zusammenarbeitsmodels durch Grenzen :

Die Zusammenarbeit zwischen Tradern wird in ANSA mit Hilfe von *Grenzen* (boundaries) erläutert. Eine Grenze vermerkt den Punkt, bis zu dem die technische Autonomie einer Organisationseinheit erlaubt sind. Überall wo es Grenzen gibt, werden Interceptor-Objekte gebraucht, um Zusammenarbeit zu ermöglichen oder unerwünschten Zugriff zu vermeiden.

Es ist wichtig, die Grenzen zu identifizieren, weil sie eine tiefe Auswirkung auf die Systemstruktur haben und die Darstellung der Werte im Computersystem begrenzen, welche bei dem Interceptor manipuliert werden können.

Die Grenzklassen von ANSA sind: Grenzen von Typsystemen, Grenzen von Bennenungssystemen, Eigenschaftsgrenzen, technische Grenzen. Außerdem werden Verwaltungs-Domains und Sicherheits-Domains modelliert.

Modell für das "cross boundary": ANSA modelliert die Überquerung (crossing) der Domain-Grenzen (boundaries) je nach Klasse der Domain-Grenze und den Objekten die den Trading-Dienst gestalten. Das Bereitstellen von Diensten über die Grenzen hinweg, erfordet, daß die beiden Trading-Modelle beider Seiten miteinander verbunden sind. Außerdem ist es erforderlich, daß die Tradingskontexte (TCtxt) ebenfalls miteinander verbunden sind, und daß das Type Conformance Service (TyCS) der einen Seite das Type Description Service (TyDS) der anderen Seite aufruft.

Ein Verbindungszenario auf der Basis des ANSAs Computational Models für die Grenzüberquerung ist in der Abbildung 7 dargestellt. Andere Konfigurationen sind möglich.

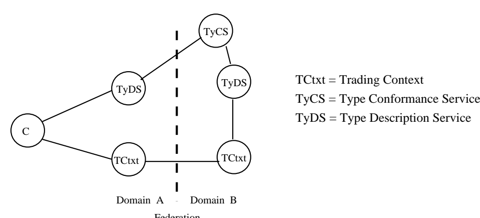

Abbildung13. Modell zur Grenzüberquerung

Die Abbildung 7 erläutert, daß das Bereitstellen eines Trading-Dienstes über Domain-Grenzen hinweg, zwei Aufrufe erfordert: Einen Aufruf für einen Trading-Kontext und eine Anforderung für eine Typenbeschreibung.

In einer ANSA-Strategie wird beispielweise das Überschreiten der Grenzen von Typensystemen (type system boundaries) näher erläutert. Bei der Überquerung dieser Grenzen muß man mit den folgenden Unterschieden von heterogenen Typensystemen zurecht kommen:

1. Die verwendete Sprache zur Schnittstellenbeschreibung
2. Die Datendarstellung
3. Die Feststellung der Konformität zwischen Schnittstellen
4. Die Vereinbarung des Schnittstellentyps der Anwendungen, die Domaingrenzen überqueren.

Die Strategie zu 1., d.h. die Übersetzung von IDLs könnte folgendermaßen aussehen: man kann zwei Schnittstellenbeschreibungsprachen (IDL) von verschiedenen Domains verbinden, in dem man einen Interceptor zwischen den TyDS der einen Seite und TyCS

der anderen Seite einsetzt. Dabei erzielt man die Übersetzung der Schnittstellenbeschreibungen der einen auf die andere Sprache.

Information Viewpoint in ANSA: Der Zugriff auf Dienste ist durch ihre Schnittstellenreferenzen gegeben. Diese Referenzen sind Namen. Die Zusammenarbeit hängt eben sowohl von der Bezeichnung der Objekte als auch vom Informationsfluß zwischen den Umgebungen ab. Von besonderer Bedeutung für die Bereitstellung des Tradingdienstes sind:

- Die Verbindung der Informationsmodelle
- Die Klassifizierungsschemata
- Benennung
- Sicherheitsarchitektur
- Grenzen des Benennungssystems

Das ANSA-Benennungsmodell enthält eine Beschreibung der verwendeten Benennungsysteme, darunter, das Invokations-Benennungsystem, das Attribut-Benennungsystem und das Typ-Benennungsystem. Es gibt mehrere Unterschiede bei der Benennung, z.B: unterschiedliche Domains oder verschiedene Benennungskonventionen.

Man kann das "cross boundary" Problem aus der Perspektive des Benennungsystems betrachten, so kann eine Verbindung zwischen unterschiedlichen Typensystemen wie eine Verbindung zwischen Typbenennungsystemen behandelt werden. Die Unterschiede könnten durch die Verwendung von "Brücken" transparent überwunden werden. Es müßten Konventionen zur Namensgestaltung in zwei Aspekten eingeführt werden: a) die verwendete Syntax zur Formulierung der Namen und b) die assoziierte Semantik.

Ein flexibler Trading-Mechanismus (Coercion): Viele der Dienste stehen, was die Typen betrifft, eng miteinander in Beziehung, dabei unterscheiden sich jedoch die Dienstqualitäten (Quality of Service, QoS). Um diese Dienste effektiver handhaben zu können, werden einige Mechanismen benötigt, die eine höhere Flexibilität beim Trading ermöglichen. Einige davon sind:

- Ausnutzung der Signaturen
- Verwendung von Untertypen
- Verwendung von Namen
- Ausnutzung der QoS
- Zwang (coercion)

Dienstanpassung an die Klientanforderung:

Mechanismen wie Polymorphie, Ausnutzung der Signaturen und QoS sind notwendig für eine flexible Trading-Umgebung. Ein Klient kann sowohl Operationen, Attribute als auch Intervalle auswählen, innerhalb deren die bestimmten QoS liegen müssen. Es gibt aber noch Probleme bei denen diese Mechanismen nicht ausreichen, z.B.: Es gibt Workstations mit verschiedene Audiokapazitäten: Einige haben HIFI Qualität und andere nur Telefonqualität. [MB92].

Um mit diesen Problemen zurechtzukommen, kann man Coercion anwenden. Coercion stammt aus dem Bereich der Typentheorie. Die Idee ist wie folgt: Ein Dienst kann leicht abgewandelt werden, so daß er nun als erweiterter Dienst benutzt werden kann. Zwei Arten von Coercion werden identifiziert:

- **Typcoercion**: durch die Verwendung einer geeigneten Transformation wird ein Typ in einen anderen Typ umgewandelt, z.B.: Eine real Zahl kann unter Verlust von Information auf eine Integerzahl abgebildet werden. Ein anderes Beispiel ist die Abbildung von Text auf Stimme.

- **Quality of Service (attribute) coercion**: Dieser Coercionmechanismus wird zum Wechsel der Objektdienstqualität angewendet. z.B.: Ein Quality of Service Coercion könnte ein färbiges Bild auf ein Schwarz-Weiß Bild abbilden oder eine HIFI-Audioquelle in ein Signal mit Telefonqualität umwandeln. Bei dieser Coercion wechselt der Typ des Dienstes nicht (die Typensignatur bleibt dieselbe).

Ein flexibler Trader, welcher Coercion verwendet, könnte Untertyprelationen sowie Attribute und Typencoercion benutzen, um einen benutzerpassenden Dienst zu gestalten.

Fazit: Bei der Benutzung dieser flexiblen Mechanismen wird es möglich sein, den Trader in folgenden Formen ansprechen zu können:

1. Ausdrücklich für den Dienst

2. Durch die Verwendung von Untertypen

3. Trading mit Hilfe von Signaturen (Operationen)

4. Trading mit Hilfe von erweiterten Signaturen (bei Operationen und Attributen)

5. Trading mit Hilfe von erweiterten Signaturen und Attributwerten

6. Trading mit Hilfe von erweiterten Signaturen, Attributwerten und Wertintervallen

7. Verwendung der Attribute-Coercion

8. Verwendung der Quality of Service-Coercion

Die Methoden (1) und (2) haben eine verbreitete Anwendung in ANSA. Methode (3) bis (8) sind Teile des sogenannten "Procustean[1] Trading", welche eine flexiblere Typenbehandlung ermöglicht.[MB92]

Wie diese Mechanismen in der ANSA Architektur integriert sind (oder werden) ist in der Abbildung 8 dargestellt:

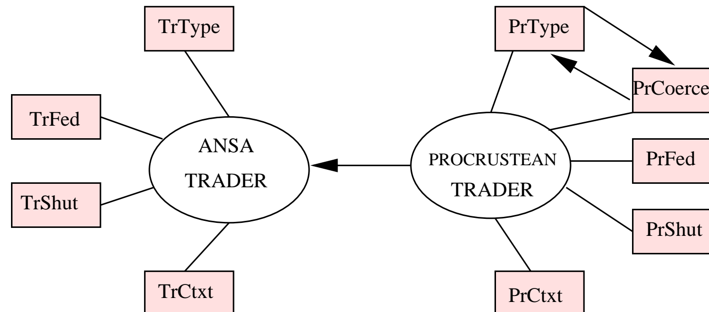

Abbildung14. Coercion in ANSA

Der neue prokustisches Trader wird als Dienst beim ANSA-Trader registriert. Die Schnittstellen des prokustischen Traders sind ähnlich wie die von einfachen Tradern des ANSA-Modelles. Sie unterscheiden sich nur durch:

PrCoerce - Coercion management (Coercionverwaltung) ist verantwortlich für die Absicherung der Dienste, die außerhalb der Klientanforderung fallen und umgeformt werden, um sie an die Klientanforderungen anzupassen.

Prtype - multimedia type management (Verwaltung der Multimediatypen) wendet die Untertypinformationen an, um diejenigen Dienste zu finden, welche ausreichende Funktionalitäten bereitstellen.

5 ANSA vs ODP

Ein Überblick der Gemeinsamkeiten und Unterschiede der beiden Modelle sind in der Abbildung 9 zusammengefaßt:

Der Vergleich wurde auf der Basis des Viewpoints (genauer Enterprise, Computational und Information Viewpoints) durchgeführt, und aufgrund derjenige relevanten Aspekten bei denen sich die beiden Modellen sie ausgedehnt haben (Strategien für die Grenzüberquerung, flexibler Trading-Mechanismus). Die Bennenung als Aspekt der Information Viewpoints und die Verträge als Aspekt der Computational Viewpoint sind jeweils Ausweitungen von ANSA und ODP.

Klassifikationschema: Beide Modelle klassifizieren die Diensangebote in Kontexte.

[1] geht auf den Procusteus, aus der griechischen Mythologie zurück. Seine Gäste, die bei ihm übernachteten, mußten ins Bett passen. Diejenigen Gäste, die für das Bett zu groß waren, wurden auf die richtige Länge zurechtgestutzt, diejenigen die zu klein waren, wurden solange gestreckt, bis sie genau ins Bett paßten

	ANSA	ODP
Dienstklassifikation	Kontexte	
Organisatorische Strukturen	Hierachische Strukturen von Kontexte/ Organisationseinheiten	
	Organisationseinheit	Logische Trader mit einer gemeinsamen Schnittstelle
	Zusammenarbeit	Zusammenarbeit Link nur in einer Richtung
Enterprise Viewpoint	Trader-Gemeinschaft mit jeweiligen Exporteuren, Importeuren	
Computational Viewpoint	TyCS TyDS TCtxt als Dienstobjekte bezeichnet	Type check Compatibility check Type-Repository Lyst type relationship
		konkrete Beschreibung der Vertraege
Informations Viewpoint	Referenzen	Links oder Verbindungen
	eindeutige Identifizierung des Kontext durch ihren Pfadnamen	
	mehrere Bennenungssysteme fuer Objekte	
	Strategien fuer die Grenzueberquerung je nach Grenzklasse	
Flexibler Trading Mechanismus	Coercion	

Abbildung15. Vergleich von ANSA und ODP

Organisatorische Strukturen: Es gibt zwei Formen: eine hierarschische und eine föderative (dezentrale Zusammenarbeit) Bei der hierarchische Struktur, sind die Kontexte hierarchisch strukturiert (Baum). Die Contexte sind auf eine Organisationseinheit zugeordnet. Die Verbindungen zwischen Kontexten spiegeln die Relationen unter Organisationseinheiten wieder.

Enterprise Viewpoint: Beide Modelle haben dieselbe Enterprise-Definition, nämlich, die Trading-Objekte (Trader, Importeur, Exporteur) und dieselbe Grundregeln der Föderation.

Information Viewpoint: Beide Modelle haben folgende parallele Informationsstruktur: Trading-Graph, Link (oder Referenz in ANSA). Die Strategien für die Grenzüberquerung und Suche entsprechen den Scopebeschränkungen. Dabei ist ANSA noch weit bei der Spezifikation bestimmter Grenzklassen und jeweiliger Überquerungsmodelle.

Beide Modelle verwenden den relativen Namenspfad der Kontexte, um auf diese zuzugreifen. Jedoch hat ANSA eine konkrete Beschreibung einiger Bennenungsysteme für Dienstobjekte zur Unterstützung der Überquerungstrategien.

Computational Viewpoint: wie wir in den Szenarien der einfachen Trader gesehen haben, ist grundsätzlich das Verhalten der Trader ähnlich: Beide Modelle rufen 3 Operationen auf, für die Überprüfung der Dienstypen, mit Hilfe der Kontexte. Jedoch ist bei der Szenarien der Zusammenarbeit von ODP-Tradern bemerkbar: Die Verwendung eines Traders als Schnittstelle für die Organisationseinheit, die ein 'logischer' Trader darstellt und die einseitige Richtung der äußeren Links (Verbindungen) dieser logischen Trader,

was nicht im ANSA-Modell gefunden wurde.

Eine Beschreibung der Gestaltung der Verträge ist in der ODP-Literatur vorhanden, wobei bei ANSA nur kurz ihre Notwendigkeit erkannt wird, um die Aktivität in der Enterprise Viewpoint zu regeln.

Schließlich wurde bei ANSA die *Coercion* als flexibler Mechanismus für die Bereitstellung von Diensten (eigentlich ursprünglich für Multimedia Dienste) über Domain-Grenzen hinweg eingeführt, was noch nicht im ODP-Draft behandelt wurde.

6 Zusammenfassung

In dieser Ausarbeitung liegt die Einführung der Modelle zur Zusammenarbeit zweier Standardisierungsinitiativen (ANSA und ODP) vor, mit ihren jeweiligen einfachen Computational Viewpoints, die als Grundlage der Untersuchung der Zusammenarbeit zwischen Tradern dienen. Es wurden einige Grundkonzepte eingeführt und die beiden Modelle wurden auch durch die Viewpoints zur Beschreibung verteilter Systeme verglichen. Zu einem gibt es überwiegende Gemeinsamkeiten zwischen beiden Modelle, und zum anderen, wie weit ANSA im Bereich der Implementierung von Dienstvermittlung, in der Beschreibung von Strategien zur Grenzüberquerungen und die Behandlung des Benennungsproblems, ist.

TP-Monitore

Christoph Schlenker

Kurzfassung

Dieses Kapitel beschreibt die Steuerung von verteilten Transaktionen durch Transaktions-Verwaltungs-Monitore.

An Hand eines einführenden Beispiels wird eine typische Arbeitsumgebung für einen TP-Monitor beschrieben und die Idee eines TP-Monitors motiviert. Es werden dann Grundlagen für das Verständis eines TP-Monitors erläutert: Zuerst werden ACID-Transaktionen beschrieben. Es wird die Transaktionsverwaltung von ACID-Transaktionen im lokalen Fall erläutert und dann auf den Fall der verteilten Transaktion, bei dem TP-Monitore benötigt werden, erweitert. Anschliesend werden Aufgaben und Einsatzgebiete des TP-Monitors erklärt und gezeigt wie der Fluß von Transaktionen durch einen TP-Monitor verwaltet wird und welche Komponenten er dazu benötigt. Es gibt verschiedene grundsätzliche Ansätze einen TP-Monitor zu realisieren. Auf diese Möglichkeiten wird kurz eingegangen und ihre Vor- und Nachteile beschrieben. Im letzten Kapitel werden existierende Standards und Systeme von Herstellern kurz beschrieben.

1 Einführung

Die ersten Transaktionen wurden vor allem bei Datenbank-Abfragen genutzt. Sie sind dafür zuständig Daten von einem Datenträger in den Hauptspeicher zu kopieren und u.U. nach dem Ändern wieder zurückzuschreiben. Über die Zeit entstand die Forderung nach Konsistenz, mehrere Transaktionen sollten sich nicht gegenseitig behindern, eine abgeschossene Transaktion soll nicht wieder (aus Versehen) gelöscht werden können. Diese Probleme wurden für Rechner mit lokalen Datenbanken gelöst. Computer entwickelten sich weiter, Computernetzwerke entstanden, doch die Forderungen blieben die gleichen. Wenn Daten über ein Netzwerk gelesen oder geschrieben werden, so sind die gleichen Bedingungen gefordert wie auch bei Transaktionen in einer monolitischen Datenbank. Gleichzeitig wurden die Programme komplexer und es war nicht nur eine Transaktion auszuführen, sondern mehrere, auf verschiedenen Rechnern und auf verschiedenen Plattformen, z.B. einer Datenbank, einer gewöhnlichen Datei, einem Bildschirm, o.a. Diese Transaktionen sollen parallel und im Wechselspiel untereinander ablaufen. Es entstand der Wunsch nicht nur eine einzelne Transaktion mit den sogenannten ACID-Bedingungen zu versehen, sondern eine Gruppe von Transaktionen. Um all diese Forderungen zu erfüllen wurden Transaktions-Verwaltungs-Monitore (Transaction Proccessing Monitors, TP-Monitore) geschaffen.

Im nächsten Kapitel wird ein Szenario aufgebaut, in dem ein TP-Monitor Einsatz kommen kann. Im Kapitel 'Grundlagen' werden zunächst ACID-Funktionen erklärt, dann auf Transaktionsverwaltungen im lokalen und im verteilten Fall eingegangen und dann auf den Dienst der Datenübermittlung und speziell des Remote Procedure Calls eingegangen. Das folgende Kapitel beschreibt die Einsatzumgebung eines TP-Monitors und seine Aufgaben. Es wird der Kontrollfluß einer Transaktion durch einen TP-Monitor erläutert

und die einzelnen Komponenten beschrieben. Zum Schluß des Kapitels wird auf verschiedene Architekturen von Transaktionsverwaltungen eingegangen. Das nun folgende Kapitel beschäftigt sich mit Standards und beschreibt 'X/Open' genauer.

2 Einführendes Szenario

TP-Monitore arbeiten mit verteilten Transaktionen, die sich aus einzelnen lokalen Transaktionen zusammensetzen, die u.U. voneinander abhängen.

Als Beispiel sei hier ein großer Handelsmarkt gegeben, der zwei Lagerverwaltungen besitzt. In der ersten Lagerhalle werden alle Artikel gelagert, die in Stück gerechnet werden, in der zweiten Halle alle die Artikel in in Gewicht gemessen werden. Da es sich bei Stückzahlen um diskrete, bei Gewichten um kontinuierliche Werte handelt, werden hierfür zwei völlig verschiedene Datenbanken benutzt, um die Lager zu verwalten. Man nehme nun an, ein Kunde, Mitarbeiter einer Firma A, möchte einen Blumentopf aus Lager 1 und 10kg Gartenerde aus Lager 2. Die Kosten sollen von dem Kundenkonto seiner Firma bei diesem Handelsmarkt abgebucht werden.

Im normalen, 'handgesteuerten' Betrieb, wird nun als erstes der Mitarbeiter überprüfen ob sich genügend Geld auf dem Konto befindet. Dazu werden die Preise für die beiden Artikel in der jeweiligen Datenbank nachgeschlagen, addiert und mit dem Kundenkonto in einer dritten Datenbank verglichen. Danach wird der Blumentopf aus der Lagerverwaltung abgezogen, dann die Gartenerde aus der zweiten Lagerverwaltung, dann wird der Betrag vom Konto abgebucht.

Wenn all diese Schritte keine Probleme machen und nicht verzahnt mit weiteren Zugriffen auf die Datenbanken ablaufen, so ist in dieser verteilten Transaktion kein großes Problem zu sehen. Sollte jedoch z.B. die Gartenerde ausgegangen sein, so muß der Blumentopf wieder in die erste Lagerverwaltung zurückgeschrieben werden. Hier kann es passieren, daß z.B. der letzte Blumentopf aus dem Lager geholt wurde und ein Lagerarbeiter beschließt, diesen Topf nachzubestellen. Er merkt nun nicht mehr, daß die Buchung rückgängig gemacht wurde. Es wird in diesem Falle zu einer Überbelegung der Regalplätze kommen. Wenn ein weiterer Mitarbeiter der Firma A zum gleichen Zeitpunkt an einem anderen Schalter einkauft, so könnte es passieren, daß bei beiden die Kontoüberprüfung positiv ausfällt, einer von beiden am Ende der Transaktion aber nicht mehr bezahlen kann, weil der andere bereits das Konto geleert hat.

Um diesen gesamten Vorgang zu automatisieren und gegen oben beschriebene Ausfälle zu sichern, setzt man TP-Monitore ein, die überwachen, wie eine verteilte Transaktion abgearbeitet wird. Zusätzlich muß ein TP-Monitor mit Problemen umgehen können, die mit einem Rechnerabsturz, Stromausfall, Softwarefehler, usw. zusammenhängen und nicht direkt aus den verschiedenen lokalen Transaktionen und deren Wechselspiel hervorgehen.

3 Grundlagen

Es wird nun erklärt, warum ACID-Transaktionen eingesetzt werden und was ACID-Transaktionen sind. Danach wird auf Transaktionsverwaltungen eingegangen. Hier sind zwei Fälle zu unterscheiden. Die Transaktionsverwaltungen für lokale Datenbasen und Transaktionsverwaltungen für verteilte Transaktionen. Es wird dann noch auf einen Dienst zum Prozeduraufruf über ein Netzwerk eingegangen.

3.1 ACID-Transaktionen

Ganz allgemein ist eine Transaktion ein informationsverarbeitender Prozeß, der für seine Wirkungen nach innen und außen Qualitätszusicherungen gibt. Beispiele hierfür sind Flugbuchungen, Hotelreservierungen, Kassenverkehr. Es wird stillschweigend angenommen, daß es sich bei Transaktionen immer um ACID-Transaktionen handelt. Diese setzen die Qualitätszusicherungen konkret in vier Zusagen um: "Atomar", "Konsistent" (Consitency), "Isolation" und "Dauerhaftigkeit". Eine atomare Transaktion garantiert, daß entweder alle Schritte ihrer Aufgabe ausgeführt werden oder überhaupt keiner. Wenn eine Transaktion verkündet, daß Werte geschrieben wurden, dann kann man dies als sicher annehmen, da eine Transaktion "nach außen" unteilbar (atomar) ist. Die Isolationsbedingung sorgt dafür, daß es zu keinen unerwünschten Seiteneffekten beim Schreiben oder Lesen von Daten kommt. Ist dies nicht der Fall, könnte es z.B. vorkommen, daß eine Transaktion einen Wert ausliest, ihn verarbeitet und zurückschreibt (z.B. um 1 dekrementiert) und eine zweite Transaktion dies, um einen Zeittakt versetzt, ebenfalls tut. So lesen beide den alten Wert und überschreiben sich gegenseitig. Das hätte zur Folge, daß eine falsche Zahl in der Datenbank steht, ohne daß man eine von beiden Transaktionen dafür direkt verantwortlich machen könnte. Die vierte Bedingung, die Dauerhaftigkeit, stellt klar, daß ein einmal geändertes Datum nicht durch ein Versehen, Programm- oder Rechnerausfall wieder rückgesetzt wird, obwohl die Transaktion den Vorgang bereits bestätigt hat.

Die Konsistenz ist gegeben, wenn eine Transaktion eine konstistente Datenbasis wieder in eine konsistente Datenbasis überführt. Für einen Benutzer ist damit sicher, daß er mit Transaktionen kein (formelles) Chaos in seiner Datenbank erzeugt. Dies ist die einzige der vier Bedingungen, die nicht (alleine) vom System garantiert werden kann. Denn eine fehlerhaft programmierte Transaktion kann eine Datenbasis natürlich inkonsistent werden lassen. Hier muß der Programmierer selber darauf achten, daß die Schritte einer Transaktion keine Fehler erzeugen.

Stillschweigend wird hier angenommen, daß es sich bei ACID-Transaktionen um kurze Aktionen handelt, die nicht länger als ein paar Sekunden oder Minuten dauern. Sie verlagern alle Probleme der Datensicherung und des parallelen Zugriffs hinter eine Transaktions-Schnittstelle. Ein Programmierer hat so die Gewissheit, auf Transaktionen zuzugreifen, die ihm eine solide Plattform geben, um komplizierte Aufbauten zu entwerfen. Weitere Informationen über ACID-Transaktionen findet man in [Loc95].

In unserem Beispiel bedeutet dies: keine weiteren Gedanken mehr an lokale Datenbasen. Ist z.B. ein gewisser Geldbetrag von einem Konto-Datensatz abgebucht, so kann niemand

anderes den selben Betrag nocheinmal abbuchen.

3.2 Transaktionsverwaltung

Um viele Transaktionen von verschiedenen Prozessen auf einer Datenbank gleichzeitig abzuarbeiten, benötigt man eine Verwaltung, die dafür Sorge trägt, daß alle Transaktionen in einer gewissen Ordnung ausgeführt werden, um zu vermeiden, daß sie sich gegenseitig behindern. Dabei ist es durchaus erwünscht, daß Transaktionen parallel ausgeführt werden, um einen möglichst hohen Durchsatz zu erreichen. Des weiteren müssen Transaktionen verhindert werden, die die Datenbasis inkonsistent werden lassen, sofern das vom Transaktionsverwalter erkannt werden kann. Transaktionen, die "auf halbem Wege" abgebrochen werden, müssen wieder zurückgesetzt werden ohne Änderungen in der Datenbasis hervorzurufen. Da sich Transaktionen von einem System zum nächsten portieren lassen sollen, muß eine gewisse Hardware-Unabhängigkeit erzeugt werden. Die Transaktions-Schnittstelle muß die hardware-spezifischen Darstellung und Lagerung der Daten verdecken.

Um all diesen Aufgaben gerecht zu werden, besteht eine Transaktionsverwaltung aus einer Transaktions-Schnittstelle (Transaktion-Manager), einem Scheduler um den Ablauf der verschiedenen Transaktionen konsistenz zu halten und einem Daten-Manager, der Anfragen für die entsprechende Hardware aufbereitet (siehe Abbildung 16).

Transaktionen kommen normalerweise nicht als Ganzes beim Transaktions-Manager an, sondern stückchenweise. Z.B. wird zuerst gelesen, ob sich noch ein Blumentopf vorhanden ist. Diese Information wird sofort benötigt, also kann der Transaktions-Mangager nicht warten bis die komplette Transaktion eingelaufen ist, sondern er muß diesen Lese-Befehl weiter an den Scheduler geben. Dieser prüft die Anfrage. Je nach verwendetem Schema (Sperrenverfahren, Zeitstempel, Ein- oder Mehrversionshistorien, usw.) wird nun diese Anfrage möglichst schnell bearbeitet und an den Datenbank-Manager weitergegeben. Ist dem Scheduler dies nicht möglich, so muß er sie entweder in eine Warteschlange einreihen oder ganz zurückweisen. In diesem Falle muß der Transaktions-Manager wiederum den Abbruch der Transaktion nach außen weitergeben, während der Datenbank-Manager bereits geänderte Daten wegen der Atomizität einer Transaktion wieder zurücksetzen muß. Tritt also ein Fehler bei einer Transaktion auf, so muß die Transaktions-Verwaltung zusätzlich dafür sorgen, daß ein sogenanntes Fehler-Recovery (-Wiederanlaufen) eingeleitet wird.

Eine Transaktion bestätigt an ihrem Ende die erfolgreiche Durchführung mit einem 'Commit'. Tritt während der Transaktion ein Fehler auf, so wird die Transaktion mit einem 'Abort' abgebrochen. Das Fehler-Recovery hat hier die Aufgabe, die neuen Werte zu löschen. Je nach Art des Daten-Managers müssen diese Werte aus einem Puffer gelöscht werden oder die alten Werte wieder in eine Datei zurückgeschrieben werden.

3.3 Transaktionsverwaltung in verteilten Systemen

Transaktionsverwaltungen in verteilten Systemen unterscheiden sich von der Komplexität grundsätzlich von Transaktionsverwaltungen in lokalen Systemen. Eine Transaktion

schreibe y (T2)
schreibe x (T1)
lese Wert x
(Transaktion 1)
lese x (T3)
Transaktions-Manager
zurückweisen
Scheduler
warten
T2
T1
T3
T4
weiterreichen
Datenbank-Manager
Datenbasis

Abbildung16. Innere Struktur der Transaktions-Verwaltung

kann nun von irgendeinem Client im Netz abgesetzt werden. Die Transaktionsverwaltung selbst muß nun entscheiden, welchem Server sie die Transaktion übergibt. Dazu muß sie die Informationen über alle verfügbaren Server halten, sie muß das Format der Anfrage dem jeweiligen Server anpassen, sie muß ein Rücksetzen einer verteilten Transaktion an alle lokal beteiligten Ressourcen melden und dafür Sorge tragen, daß keine der lokalen Datenbanken (allgemeiner: Ressourcen) in einen inkonsistenten Zustand gerät, wenn die verteilte Transaktion wieder zurückgesetzt wird. Eine solch komplexe Transaktionsverwaltung nennt man einen Transaktions-Monitor.

Das Fehler-Recovery im verteilten Fall ist komplizierter als im lokalen Fall. Das 2-Phasen-Commit-Protokoll (2PC) ist eine Möglichkeit, eine Transaktion über ein Netzwerk zu beenden. Beim 2-Phasen-Commit-Protokoll senden im Normalfall alle beteiligten Transaktionen ein 'Ready to Commit'. Wenn sie alle eingetroffen sind, sendet der Transaktion-Manager ein 'Commit' an alle Transaktionen. Sollte jedoch eine einzige Transaktion ein 'Abort' senden, so geht sofort an alle anderen Beteiligten ein 'Abort' und die Transaktion wird abgebrochen. Ein 'Abort' wird nichtmehr bestätigt. (Es gibt im 2PC-Protokoll die Möglichkeit der Blockierung, die durch ein 3-Phasen-Commit umgangen wird, wenn Netzpartitionen ausgeschlossen werden können. Darauf wird hier jedoch nicht näher eingegangen (siehe [Loc95])).

3.4 Dienst zur Datenübermittlung

Damit eine lokale Transaktion auf einem entfernten Server über das Netz ausgeführt werden kann, muß der Server von seiner Netzsoftware über diesen Wunsch informiert werden. Am Ende dieser Transaktion wird das Ergebnis vom Server an den Anfragenden (Client) zurückgeschickt.

Damit dieser Dienst nicht für jedes Netzwerk komplett neu programmiert werden muß, bietet fast jedes Betriebssystem für das Ausführen von Programmen bzw. Routinen über das Netz einen netzunabhängigen Dienst an.

Es kommt auf die Art des Netzwerkes und auf die Software an, wie dieses Verfahren genau aussieht. So kann ein Netzwerkdienst verbindungsorientiert arbeiten und eine direkte Verbindung vom Server zum Client halten oder es kann sich um einen paketvermittelten Dienst handeln, bei dem Datenpakete vom Client bis zum Zielrechner durchgereicht werden. Es gibt hier sehr viele unterschiedliche Spielarten mit den verschiedensten Netzwerken. Das ISO/OSI-Referenzmodell bietet auf der Anwendungsschicht den Dienst Remote Procedure Call (RPC) an. Seine Aufgabe ist es, eine angegebene Routine auf dem angegebenen Server zu starten. Der Aufruf eines RPCs erfolgt in einem einheitlichen Format, was die Programme netzunabhängig macht und leichter portieren läßt. Durch das ISO/OSI-Schichten-Modell können kleinere Fehler und kurzfristige Ausfälle des Netzwerkes von den unteren Schichten abgefangen werden. Während des RPC-Aufrufs wartet der Client auf die Antwort des Servers (siehe [LKK93]).

Ein RPC ist ein Basisdienst zur Arbeit mit Netzwerken. Er kann nicht nur von Transaktionen genutzt werden, sondern z.B. auch von verteilten Anwendungen oder von anderen Programmen zum Datenaustausch bzw. zum Aufruf von Subroutinen auf fremden Rechnern.

Was die Grundversion des RPCs nicht kann, ist das Aussuchen eines beliebigen Servers, auf dem die geforderte Subroutine ablaufen soll. Im nichttolerierbaren Fehlerfalle bricht ein RPC die Verbindung einfach ab, ohne sich darum zu kümmern was diese Routine auf dem gestarteten Server angerichtet hat. Des weiteren speichert ein Remote Procedure Call keine Kontexte ab; ist die gleiche Arbeitsumgebung bei mehreren RPCs erwünscht, muß der Client für den entsprechend gleichen Rahmen selber sorgen. Das heißt z.B. bei Zugriffen auf eine bestimmte Datei, daß diese Datei jedes mal mitangegeben werden muß und nicht auf frühere Daten zurückgegriffen werden kann.

Es gibt mehrere RPC-Varianten und Erweiterungen, die auf der Grundversion dieses RPCs aufsetzen. Weiter unten wird noch genauer der Transactional RPC besprochen (siehe Kapitel 3.6) und das Produkt TxRPC (siehe Kapitel 4.1) beschrieben. Diese Erweiterungen eignen sich für verteilte Transaktionen.

Der Vorteil eines RPCs ist die große Verbreitung und Akzeptanz in vielen verschiedenartigen Netzwerken. So bietet dies einen robusten Dienst an, auf die der Programmierer wiederum komplexere Strukturen und Aufrufe aufbauen kann.

Der lokale Rechner am Schalter des Verkaufs in unserem Beispiel könnte an die lokalen Datenbanken für Lager 1, Lager 2 und Kundendaten einen RPC absenden, um die gewünschten Werte auszulesen.

In den folgenden Unterkapiteln werden die Einsatzgebiete und Aufgaben eines TP-Monitors beschrieben. Dann wird auf die einzelnen Komponenten eines TP-Monitors eingegangen, die die Verwaltung und Durchführung von Transaktionen unterstützen. Es wird gezeigt, welchen Weg eine Transaktion durch einen TP-Monitor nimmt. Zum Schluß wird beschrieben, wie sich eine verteilte Transaktionsverwaltung auf Prozesse abbilden läßt und welche Variante welche Vor- und Nachteile hat.

3.5 Aufgaben und Einsatzgebiete

Für unser Beispiel sieht die komplette verteilte Transaktion grob vereinfacht wie in Bild 17 aus. Das markierte Gebiet liegt innerhalb der Reichweite des TP-Monitors und garantiert als Ganzes die ACID-Eigenschaften. Die lokalen Transaktionen selber laufen bereits in einer ACID-Umgebung während die globalen Bedingungen vom TP-Monitor geschaffen werden müssen. Es eignen sich nicht alle lokalen Transaktions-Verwaltungen zur Zusammenarbeit mit einem Transaktions-Monitor. Insbesondere gibt es Transaktions-Verwaltungen die nicht mit anderen zusammenarbeiten (siehe Kapitel 3.6).

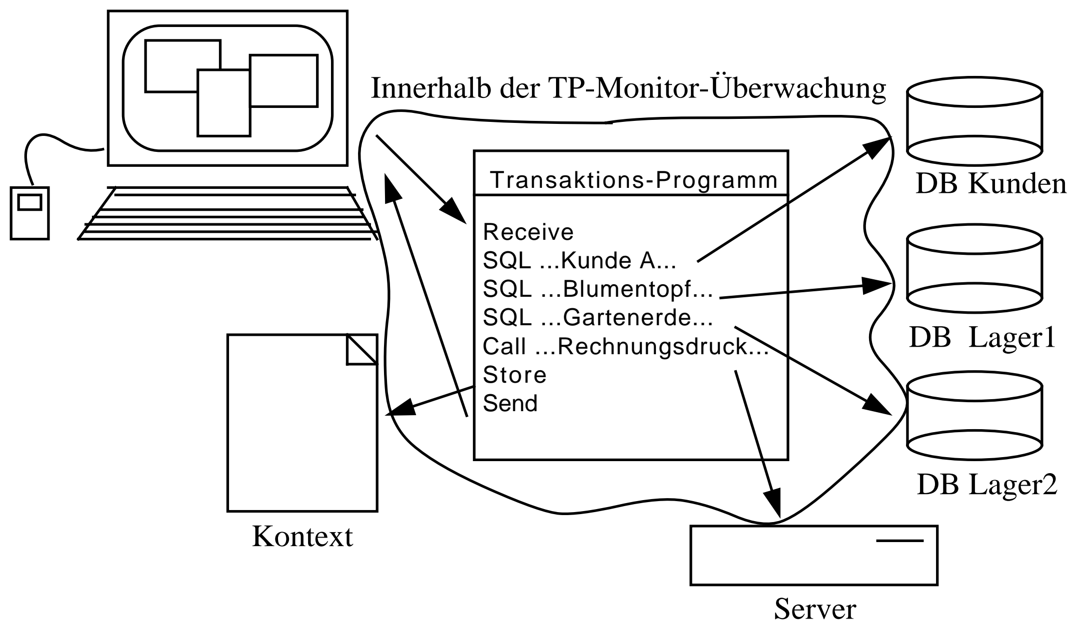

Abbildung17. Ausdehnung einer verteilten Transaktion

Ein TP-Monitor soll dem Programmierer einer Anwendung eine Schnittstelle anbieten, die ihm die Möglichkeit gibt, hardware- und netzartunabhängig zu programmieren. Er muß also nicht mehr berücksichtigen, daß sich ein gewünschter Dienst nicht mehr auf dem lokalen Rechner findet sondern in einem angeschlossenen Netzknoten. Der Diensteanbieter hingegen kann von einem einheitlichen Aufruf durch den TP-Monitor ausgehen, und muß nicht auf Spezialaufrufe verschiedener Applikationen Rücksicht nehmen. Diesen Vorteil kann auch der Programmierer nutzen; er hat eine vorgegebene Spezifikation für alle Serveraufrufe über den TP-Monitor.

Zudem kann ein Programm nicht nur einen einzigen Serveraufruf mit Hilfe des TP-Monitors absetzen, sondern eine Liste aller gewünschten Transaktionen, die dann alle

zusammen die ACID-Bedingungen erfüllen. Es ist damit möglich z.B. die Abfrage einer Datenbank und das Ausgeben der Ergebnisse auf dem Bildschirm als eine "virtuelle" Transaktion ablaufen zu lassen.

Der TP-Monitor verwaltet aus diesem Grunde nicht nur einzelne Server, sondern einzelne Serverklassen. Jede Serverklasse bietet einen bestimmten Dienst an. Ein Server ist Element einer Klasse, wenn er den für diese Klasse definierten Dienst anbietet. Da es nun egal ist, welcher dieser Server aufgerufen wird, kann der TP-Monitor die gesamte Netzlast und die Lasten der einzelnen Rechner beachten und je nach Belastung auf anderer Server der gleichen Klasse ausweichen. In unserem Beispiel könnte es sein, daß mehrere Rechner existieren, die Produktbeschreibungen in einer Datenbank gespeichert hätten. Nun kann der TP-Monitor den Server aus einer Liste heraussuchen, die den Dienst "Produktbeschreibung" anbietet. Es kann sich hierbei um denjenigen Server handeln, der als Kriterium die kürzeste Antwortzeit hat oder der die geringste Netzlast erzeugt oder irgend ein anderes Kriterium erfüllt. Das Kriterium kann entweder beim Aufruf mitgegeben werden oder der TP-Monitor versucht automatisch nach eigener Strategie zu optimieren.

Unter einem Server wurde hier stillschweigend ein Diensteanbieter in Form einer Applikation angenommen. Da ein Server immer als Applikation auf einem Rechner gestartet wird, muß davor Betriebsmittel (Ressourcen) wie Speicherplatz, CPU-Leistung, u.a. reserviert werden. Da ein Server u.U. weitere Server benötigt um die gewünschte Funktion zu bearbeiten, werden auch Server selbst als Ressourcen bezeichnet. Um eine Transaktion ausführen zu können, werden von einem TP-Monitor verschiedene Ressourcen belegt. Dabei kann es sich im einzelnen um Speicherplatz (Adreßraum), Applikationen, Datenbanken, Dateien, Bildschirme (bzw. Fenster), Rechenleistung, uva. handeln.

An Ressourcen, die mit einem TP-Monitor zusammenarbeiten sollen, werden Bedingungen gestellt: Sie müssen in der Lage sein, alle Änderungen, die eine verteilte Transaktion hervorruft, wieder rückgängig zu machen (oder wenigstens in geeigneter Weise zu kompensieren), falls die Transaktion abgebrochen wird. Dabei kann die Ressource Hilfe vom Transaktions-Manager anfordern. Er muß auch eine Liste von Ressourcen anlegen, die bei einer Transaktion benutzt werden, um bei einem 'Abort' oder 'Commit' alle Ressourcen über das Ende der Transaktion verständigen zu können.

Da Ressourcen u.U. auch mehrfach gleichzeitig genutzt werden können, müssen entweder die Ressourcen selber oder aber der Log-Manager den Kontext einer jeden Transaktion zwischenspeichern um bei nochmaligem Aufruf von einer Transaktion nicht mit dem falschen Kontext zu arbeiten. Unter Kontext ist hier alles zu verstehen, was in irgendeiner Weise von der Transaktion betroffen sein könnte. Nehmen wir an, es handelt sich um einen technische Anlage, die verschiedene Produkte gleichzeitig bearbeiten kann. Dann muß sie beim Aufruf einer Transaktion immer darüber informiert sein, um welches der Produkte es sich handelt.

3.6 Komponenten und Kontrollfluß

Wenn eine Anforderung an einen TP-Monitor gesendet wurde, dann muß der TP-Monitor einen entsprechenden Server suchen, der diesen Aufruf ausführt. Dazu muß zuerst ge-

prüft werden, ob der Aufrufer berechtigt ist, diese Serverklasse zu benutzen. Wenn diese Prüfung erfolgreich verläuft, wird ein Server belegt und das entsprechende Programm aufgerufen. Danach wird dieses Programm abgearbeitet und das Ergebnis wieder an den TP-Monitor zurückgegeben. Dieser sendet dann eine Nachricht an den Anfordernden.

Was nun hier so einfach klingt, ist deshalb komplizierter, weil die ACID-Bedingungen bei diesem ganzen Aufruf gelten sollen, weil auf mehreren Netzknoten mehrere Aufrufe gleichzeitig stattfinden können und weil eine Transaktion parallel auf mehreren Netzknoten gleichzeitig abgearbeitet wird. Um dies richtig zu koordinieren bedient sich der TP-Monitor mehrerer Datenbanken in denen er speichert, welche Serverklasse wo Vertretungen hat, welcher Benutzer wo Zugriffsrechte hat und wohin eine Rückmeldung geschickt werden muß. Um ein 'Commit' oder 'Abort' einer Transaktion richtig auszuführen, müssen bei der Ausführung der Recovery-Manager und der Log-Manager unterhalb der Applikation arbeiten.

Im Unterschied zum lokalen Fall kann es hier z.B. passieren, daß ein Netzknoten ausfällt. Transaktionen, die davon betroffen sind, können nicht mehr fortgeführt werden, und müssen abgebrochen werden. Da aber die lokale Transaktion auf dem vom Netz abgeschnittenen Server weiterläuft, muß sie, sobald der Netzknoten wieder hochgefahren wird, ebenfalls abgebrochen werden. Um ein verteiltes 'Commit' zu versenden, muß man sicher sein, daß alle Knoten auch in der Lage sind, ein 'Commit' zu empfangen. Es könnte sonst passieren, daß ein paar Server ein 'Commit' erreicht (und damit die Transaktion dauerhaft festschreiben) und ein Knoten z.B. durch einen Fehler ein 'Abort' zurücksendet. Ein mögliches Protokoll für dieses Verfahren ist das 2-Phasen-Commit-Protokoll (siehe 3.3).

Sollten verschiedene Server verschiedene Protokolle fahren (z.B. verschiedene, inkompatible Time-Out-Verfahren), so kann es zu Inkonsistenzen kommen, die weder der TP-Monitor noch der einzelne Server erkennen bzw. abfangen kann. Um dies zu Vermeiden, muß vor der Transaktion vereinbart werden, welches einheitliche Protokoll verwendet wird. Es hat sich hier das 2-Phasen-Commit-Protokoll durchgesetzt.

Die einzelnen Transaktionen werden mit einem Transaktional Remote Procedure Call (TRPC) versandt. Dieser TRPC verwendet gewöhnliche RPCs um seine Aufrufe an den jeweiligen Server abzusenden. Während ein RPC nur dafür sorgt, daß eine Anfrage den richtigen Server erreicht, speichert ein TRPC den Kontext zu einer Transaktion, um bei einem zweiten Aufruf im gleichen Kontext weiterarbeiten zu können. Ein TRPC muß dafür sorgen, daß der Transaktions-Manager erfährt, welche Ressourcen angesprochen werden um diese bei einem Abschluß der globalen Transaktion (positiv oder negativ) zu informieren. Zu alledem muß ein Transaktional RPC das ACID-Transaktionsprotokoll unterstützen. Im Falle eines Abbruchs muß ein TRPC eine lokale Transaktion wieder zurücksetzen können (siehe auch [GR93], Kap. 5.4).

Der Kontrollfluß einer Transaktion durch einen TP-Monitor läßt sich in Abbildung 18 verfolgen.

Die Präsentationsschicht dient dazu, alle Eingaben in ein Format zu wandeln, die völlig netz- und aufrufer-unabhängig sind. Die Ausgaben werden entsprechend von einem internen Format auf den jeweiligen Client angepasst. Bei Änderungen am Netz muß dann

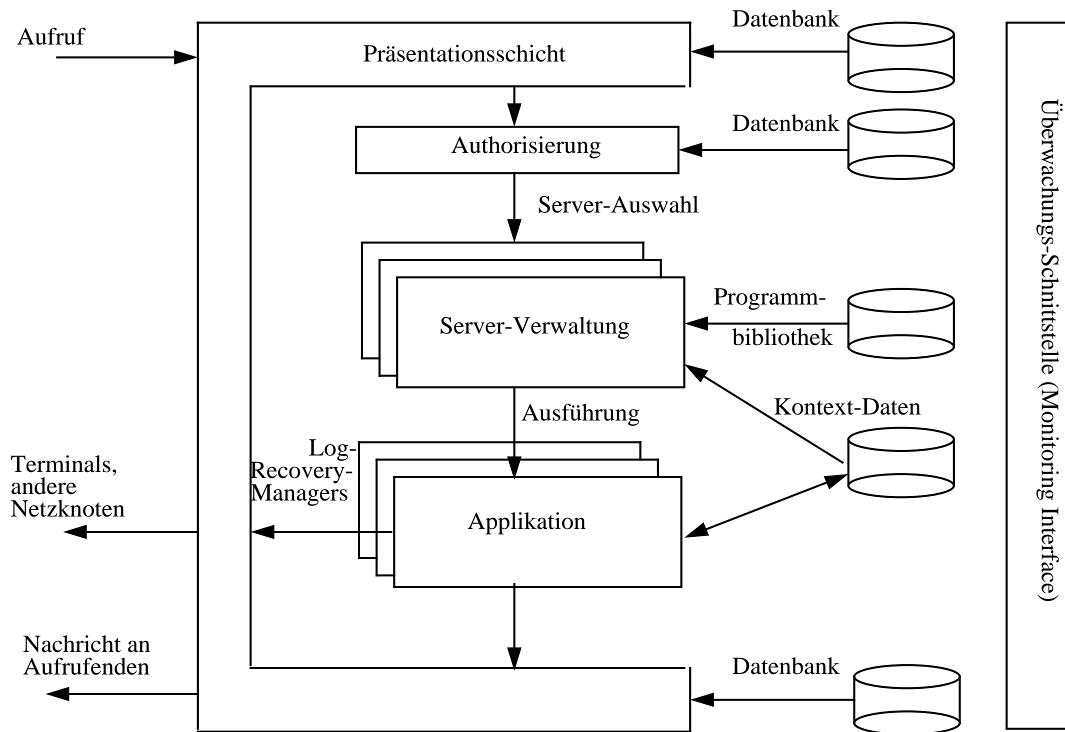

Abbildung18. Kontrollfluß durch einen TP-Monitor

nur die Präsentationsschicht geändert werden. Der Aufruf wird auf seine Zulässigkeit für den Aufrufer geprüft. Ist er authorisiert, die verlangte Transaktion durchzuführen wird ein Server ausgewählt, der die Aufgabe erfüllen kann. Die Ausführung findet dann auf einem gewählten Server statt. Das Ergebnis wird durch die Präsentationsschicht zum Aufrufer geschickt.

Von außen betrachtet sieht ein Aufruf über einen TP-Monitor wie in Abbildung 19 aus.

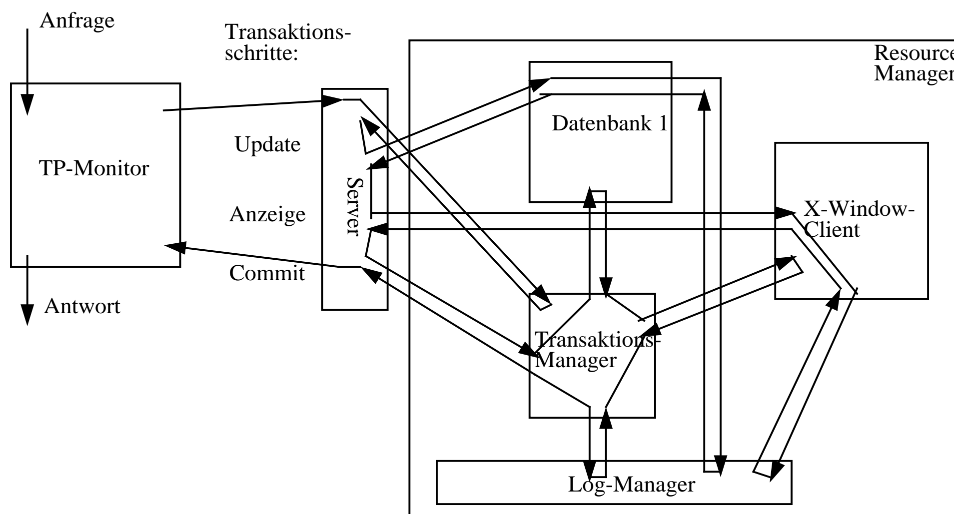

Abbildung19. Ablauf einer verteilten Transaktion mit Hilfe eines TP-Monitors

Der TP-Monitor erhält eine Anfrage und gibt sie an den entsprechenden Server weiter.

Nach einer Intitialisierung beim Transaktions-Manager schickt dieser einen Update-Befehl an eine Datenbank, diese wiederum meldet diese Änderung dem Log-Manager, der die Kontrolle über alle angesprochenen Ressourcen während einer Transaktion hält. Als zweites schickt der Server Daten an ein X-Windows-Fenster. In diesem Fall scheint es nicht sinnvoll, Änderungen sofort auf dem Bildschirm anzuzeigen, da sie ja u.U. wieder zurückgesetzt werden könnten, wenn die Transaktion abgebrochen wird. Die Daten müssen jetzt solange zurückgehalten werden, bis sicher ist, daß sie nicht mehr zurückgesetzt werden könnten. Als drittes sendet der Server ein 'Commit' an den Transaktions-Manager, der alle angesprochenen Ressourcen über den erfolgreichen Abschluß einer Transaktion informiert. Nachdem alle positive Rückantwort erteilen, wird im Log-Manager ebenfalls festgeschrieben, daß die Transaktion beendet wurde. Alle lokalen Teil-Transaktionen werden nun festgeschrieben. Erst danach schickt der Transaktions-Manager eine Bestätigung an den TP-Monitor. Dieser gibt das Resultat an den Anfragenden weiter.

Vom Zeitpunkt des Beginns bis zum Ende der Transaktion muß es möglich sein die Transaktion rückgängig zu machen. Da ein Server u.U. abstürzen kann oder aus irgend einem Grund nicht mehr ansprechbar sein kann, muß ein TP-Monitor (mit Hilfe der Transaktions-Verwaltung) auch Transaktionen rücksetzen können, bei denen es keinen Server mehr gibt. Ein TP-Monitor muß also von Anfang bis Ende über den Stand einer Transaktion informiert sein. Da für die gesamte verteilte Transaktion gelten muß, daß sie atomar ist, dürfen Änderungen auf Datenbanken, in Programmen, auf Bildschirmen erst dann sichtbar gemacht werden, wenn die Transaktion nicht mehr abbruchgefährdet ist.

Wie wir gesehen haben, benötigt der TP-Monitor dazu weitere Komponenten und Manager. Da diese von manchen Betriebssystemen zur Verfügung gestellt werden und von anderen nicht, gibt es nun eine Vielzahl von Realiserungsmöglichkeiten und eine Vielzahl von tatsächlichen Realisierungen. Das folgende Kapitel geht auf verschiedene Implementierungsmöglichkeiten ein.

3.7 Realisierungsmöglichkeiten

Hier wird nun beschrieben, wie Transaktionsverwaltungen auf Prozesse abgebildet werden können.

Eine Primitivlösung ist die folgende: Ein Terminal ruft seinen eigenen Server auf, auf dem es exclusiv arbeiten kann und läßt dort die gewünschte Applikation laden und ausführen. Der Nachteil ist natürlich, daß hier eine viel zu große Menge von Ressourcen verschwendet wird. Denn wird ein Terminal für eine Zeit nicht benutzt, so wartet der Server ohne daß er sich sinnvollen Aufgaben annimmt (siehe Abbildung 20a). Andererseits hat hier ein TP-Monitor praktisch keine Aufgaben und ist sehr leicht zu implementieren.

Um diese Ressourcen-Verschwendung zu beseitigen, wird pro Terminal ein Prozeß auf einem Server gestartet (siehe Abbildung 20b). Von dort aus können dann Applikationen gestartet werden, die wiederum auf Datenbanken zugreifen. Es laufen dann also bei 1000 Terminals 1000 Prozesse von denen viele die gleichen Programme verwenden, parallel nebeneinander. Dies ist wieder eine Verschwendung von Ressourcen. Jeder Prozeß kann

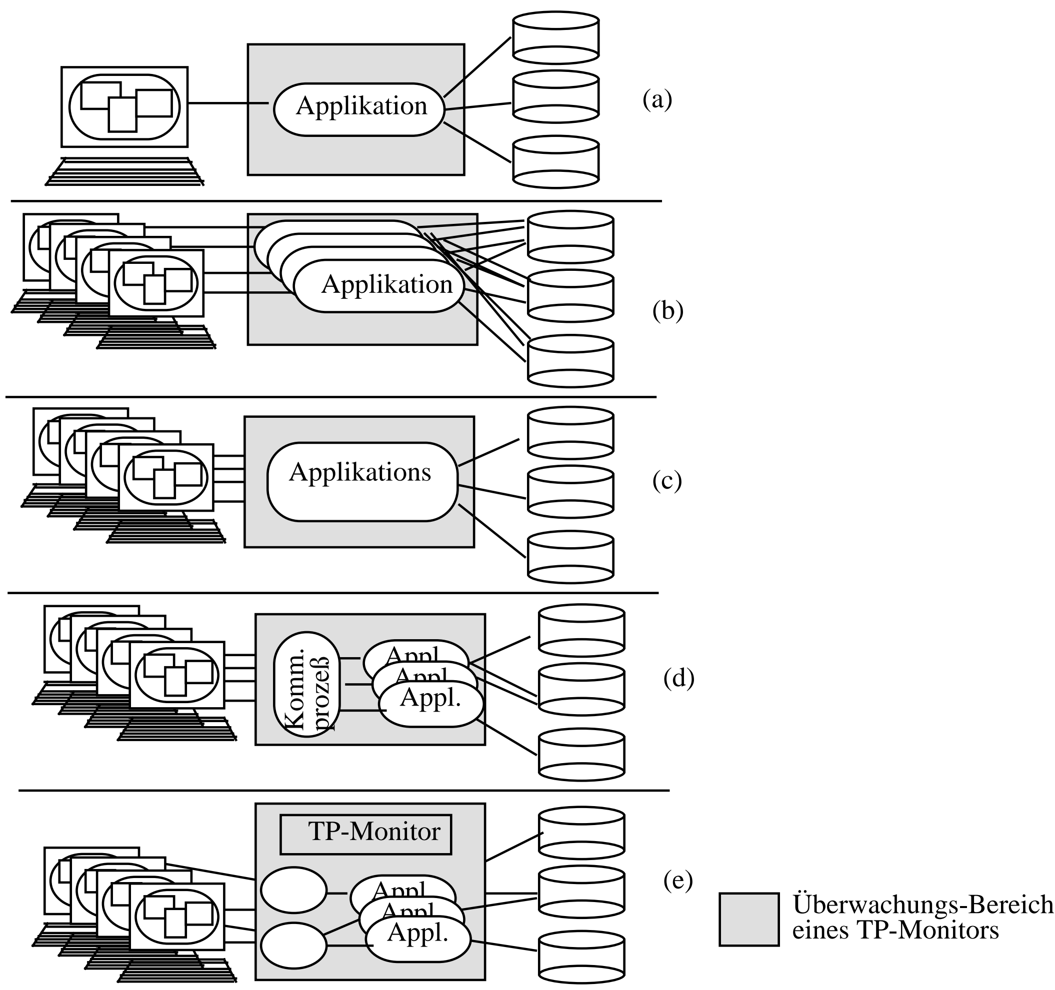

Abbildung20. Verschiedene Realisierungsmöglichkeiten eines TP-Monitors

potentiell jedes verfügbare Programm starten, alle Programme können wiederum alle Datenbanken öffnen. Es entsteht so ein viel zu große Überlast an Kontrollblöcken für geöffnete Files, Applikationen und Prozesse. Da Prozesse sich eine geringe Anzahl von Prozessoren teilen müssen (in den meisten Computern genau einen), wird jeder einzelne Prozeß sehr langsam, da mit einem Prozeßwechsel auch die gesamte Umgebung eines Prozesses ausgelagert werden muß. Zudem kommt eine u.U. ungerechte Verteilung der Rechnerverteilung. Es kann sein, daß 500 Terminals an einem Rechner sehr beschäftigt sind, während an einem anderen Rechner Totzeiten entstehen, weil er nicht benutzt wird. Zudem benötigt in Terminal überhaupt keinen ganzen Prozeß für sich, denn eine transaktionsorientierte Anwendung muß keine Dateien anlegen, Programme compilieren oder eine eigene Shell unterhalten.

Aus diesen Überlegungen heraus ergibt sich die Idee '1000 Terminals, 1 Prozeß' (siehe Abbildung 20c). Hier läuft nun ein einziger Prozeß, der alle Terminals verwaltet. Er steuert die Ein- und Ausgabe an die verschiedenen Applikationen (die alle in diesem Prozeß laufen). Aus der Sicht eines TP-Monitors wird die Aufgabe damit sehr einfach. Der TP-Monitor kann eine Anfrage auswerten und sie nach eigenen Kriterien weiterverarbeiten ohne daß Seiteneffekte auftreten, die von ihm abgefangen werden müßten. Fast alle oben genannten Nachteile sind hiermit ausgeschaltet, das Problem ist jetzt jedoch ein anderes:

Wenn 1000 Terminals gleichzeitig Aufträge eingeben oder grössere Ausgaben erwarten, wird dieser eine Prozeß in Schwierigkeiten kommen, seinen Anforderungen an Zeitlimits Folge zu leisten. Bei solch großen Anforderungen an die Bearbeitungszeit hat ein TP-Monitor keine Zeit komplexe Überlegungen zur Ausführung verschiedener Anfragen auf verschiedenen Rechnern zu machen. Da ein Prozeß nur auf einer CPU zur selben Zeit ausgeführt werden kann, werden alle möglichen Vorteile der Parallelverarbeitung zunichte gemacht. Stürzt nur eine einzige dieser vielen Applikationen ab, so fällt die gesamte Terminalverwaltung aus.

Die oben genannten Probleme wiederum lassen sich durch einen Teilung des einen Prozesses in viele Applikations-Prozesse und in einen Kommunikations-Prozeß entschärfen. Der Kommunikationsprozeß, dessen Aufgabe der TP-Monitor übernimmt, muß die Anfragen der Terminals auf verschiedene Applikationen verteilen. Dabei kann der Prozeß entscheiden, welchem Programm er welche Aufgabe zuteilt, um eine gleichmäßige Belastung zu sichern (siehe Abbildung 20d).

Ein einzelner Kommunikationsprozeß ist ein Flaschenhals. Hier müssen alle Daten von den Terminals auf die Applikationsprozesse verteilt werden. So wäre es besser, für viele Terminals ein paar Kommunikations-Prozesse zu öffnen, die dann wiederum auf verschiedene Applikationen zugreifen können (siehe Abbildung 20e). Dieser komplizierte Fall, bei dem mehrere Kommunikationsprozesse auf mehrere Applikationsprozesse zugreifen können, die wiederum mehrere Dateien/Datenbasen öffnen, benötigt den komplexesten TP-Monitor. Er hat die Aufgabe, die Verteilung der einzelnen Clients (Terminals) auf verschiedene Server (Applikationen) zu überwachen. Diese Technik ist mächtiger als ein gewöhnlicher Subroutine-Aufruf, da Server in ihrer eigenen Umgebung und mit eigenen Tasks laufen. Gleichzeitig muß der TP-Monitor den Überblick aller existierender Serverklassen und deren Elemente behalten und auswählen, welche der Server als nächstes angesprochen werden dürfen, um eine gleichmäßige Last auf dem ganzen Netz zu erzeugen.

Nach einem Crash ist der TP-Monitor in allen Fällen für das Wiederherstellen der Verbindungen und Kontexte zuständig. Unfertige Transaktionen müssen entweder zu Ende geführt oder abgebrochen werden. (Im besten Fall geschieht dies alles transparent für einen Benutzer, der dann nur eine Verzögerung spürt, bis alle Systeme wieder auf dem Stand der Dinge sind.)

4 Standards

Wie oben beschrieben, gibt es viele verschiedene Möglichkeiten, einen TP-Monitor in ganz verschiedenen Umgebungen zu realisieren. Dazu kommt noch, daß manche Betriebssysteme Dienste anbieten, die von anderen nicht bereitgestellt werden. Grundsätzlich erwarten alle existierenden TP-Monitore, daß sie auf einem RPC aufsetzen können. Manche Systeme bieten ein Threading oder Lightwight-Prozesse an (z.B. SUN Solaris, OSF/1). Diese ermöglichen innerhalb eines Prozesses verschiedene Aktivitätspfade zu verfolgen, ohne die komplette Prozeß-Umgebung immer wieder auszulagern und einzulagern.

TP-Monitore bieten als Zusatz zum normalen Betriebssystem (Operation System (OS)) das sogenannte Transaktional Processing Operating System (TPOS) und die darauf aufbauenden Transaction Processing Services (TRAPS) an.

Ein TPOS baut auf dem gewöhnlichen Betriebssystem auf und bietet u.a. Dienste zum Einrichten und Abfragen von Server-Klassen und der Anfrageverarbeitung (Request Scheduling) an. Darauf aufbauend bietet ein TRAPS ein Operator Interface an, um den TP-Monitor zu verwalten. Von hier aus kann die Netzbelastung gesteuert werden und die Algorithmen zur Lastverteilung angegeben werden. Für die Speicher- und Hardwareverwaltung bietet ein TPOS einen Transaktions- und Log-Manager, darauf bauen die TRAPS mit Ressource-Manager und Flußkontrollen auf; für die Datenübertragung bietet das TPOS einen TRPC an. Die TRAPS wiederum stützen sich darauf, um Dienste an Hand ihrer (global eindeutigen) Namen zu finden.

Die Aufzählung der angebotenen Elemente von TP-Monitoren ist unvollständig. Für nähere Informationen siehe [GR93], Kap. 6, Abb. 6.1.

All dies ist in der Theorie vorgegeben und geplant. Existierende Systeme implementieren nur manche dieser Dienste, manche auf völlig anderem Wege und manche überhaupt nicht.

Es gibt verschiedene Stadardisierungen im TP-Bereich. Besonders zu erwähnen sind hier die ISO und X/Open. (Es gibt noch weitere, z.B. die DIN 66 265, sie sollen jedoch hier nicht weiter betrachtet werden.)

4.1 X/Open

Von X/Open werden mehrere TP-Monitor-Spezifikationen angegeben. Eine davon ist die Software-Architektur DTP (Distributed Transaction Processing). Hier wird ein Prozeß in drei Teilen unterschieden: dem Applikation Program, das sowohl auf die Resouce Manager bzw. die Kommunikations-Ressource-Manger (CRM), wie sie in X/Open genannt werden, als auch auf den Transaktions-Manager zugreifen kann. Die beidem Manager-Typen kommunizieren untereinander über die Schnittstellen XA bzw. XA+ (für CRMs) (siehe Abbildung 21).

Die Schnittstelle zwischen der Anwendung und dem Transaktions-Manager ist die TX (Transaction Demarcation) Schnittstelle. Diese Schnittstelle bietet Dienste zum Beginnen und Beenden einer Transaktion an. Ferner gibt es unter anderem Dienste zum Setzen von Zuständen, die den Transaktions-Manager beeinflussen.

Es gibt in X/Open eine Spezifikation, die eine Abbildung von einem CRM auf das OSI Transport Protokoll definiert. Zudem werden drei verschiedene Schnittstellen zwischen den Kommunikations-Ressource-Managern und der Applikation definiert: Peer-to-Peer, XATMI und TxRPC.

XATMI, Tuxedo: Es werden von X/Open hier zwei Schnittstellen beschrieben, eine vom Request/Reply-Typ, die andere um dialogorientierte Kommunikation zu ermöglichen. Diese Definition geht zurück auf das Produkt “Tuxedo” von AT&T (das inzwischen von USL (UNIX System Laboratiories) und dann von Novell übernommen wurde. Dieses

Abbildung21. Komponenten und Schnittstellen der DTP-Architektur

Produkt ist als Quell-Code verfügbar und wurde von verschiedenen Herstellen auf verschiedene Unix-Plattformen portiert und teilweise vom jeweiligen Produzenten erweitert.

TxRPC, Encina: Wie der Name schon vermuten läßt, handelt es sich hier um eine Spezifikation zum Absetzen von Transaktional Remote Procedure Calls. Dabei orientiert sich das Protokoll sehr stark an der Form eines Unterprogrammaufrufs. Diese Spezifikation wurde aus dem Produkt Encina entnommen und dieses wiederum ist aus der RPC-Schnittstelle des DCE (Distributed Computing Environment) der OSF (Open Software Foundation) entwickelt worden. Es wird vor allem von IBM und HP angeboten und ermöglicht es Großrechner über das Protokoll LU 6.2 anzubinden (siehe [Bü95]).

Es gibt noch viele weitere Spezifikationen und Definitionen von Diensten und Schnittstellen (z.B. CPI-C, KDCS, UTM, TOP END, CICS [Bü95]). Die meisten dieser Spezifikationen sind aus Routinen entstanden, die verschiedene Hersteller auf ihren Plattformen anbieten. Da der TP-Monitor-Markt noch relativ jung ist, gibt es noch sehr viele verschiedene Systeme die untereinander konkurrieren und auch sehr wenig gemeinsame Schnittstellen anbieten. Einzig die XA-Schnittstelle die die Kommunikation zwischen Nicht-Kommunikations-Ressourcen und Applikationen definiert, scheint sich zu etablieren. Dies erlaubt den Austausch von TP-Monitoren bei Beibehaltung der Datenbanken und umgekehrt. X/Open versucht all diese Quasi-Standards unter einen Hut zu bringen. Leider stehen dem viele technische Probleme wie auch die Notwendigkeit, viele unterschiedliche Kommunikationsmodelle zu integieren, im Wege.

5 Zusammenfassung und Ausblick

Um verteilte Transaktionen über ein Netz auszuführen, ist ein TP-Monitor unumgänglich. Denn ohne eine Kontrolle und Steuerung dieser Transaktion können keine Garantien mehr über ihre Dienst-Qualitäten gegeben werden. Die Herausforderung am Bau von TP-Monitoren liegt darin, alle Fehlermöglichkeiten abfangen zu können und so einen vollautomatischen Wiederanlauf zu ermöglichen.

Dazu wird vorrausgesetzt, daß es bereits einen Unterbau gibt, auf dem ein TP-Monitor aufsetzen kann. So muß es auf jedem lokalen Rechnerknoten möglich sein, die jeweiligen Aufrufe zu empfangen und zu verarbeiten. Die Rechnerknoten müssen in der Lage sein, ihre lokale Transaktion im Bedarfsfall wieder rückgängig zu machen. Es muß sicher gemacht werden, daß die verschiedenen Knoten einer Transaktion miteinander arbeiten, ohne das Inkonstistenzen entstehen können. Dies kann weder vom TP-Monitor noch von den einzelnen Knoten garantiert werden, sondern muß als Vereinbarung vor der Transaktion gesichert sein.

Die Standardisierung der TP-Monitore ermöglicht es, ein modulares System von Servern und Clients aufzubauen, die sowohl austauschbar, als auch unabhängig voneinander wart- und verwaltbar sind.

Ein TP-Monitor hat seine Tauglichkeitsgrenzen in den Tauglichkeitsgrenzen der ACID-Forderungen. Eine Transaktion die viele Netzknoten belegt und Datenbanken anfrägt, blockiert diese u.U. für eine recht lange Zeit (Sekunden, Minuten), da geschriebene Daten erst dann freigegeben werden können, wenn alle Knoten ihr 'OK' zu dieser Transaktion gegeben haben.

Komplexere Transaktionen können sich über Tage oder Wochen hinziehen. Man nehme in unserem Beispiel an, man würde auch überwachen, daß ein Paket mit der Post abgeschickt würde. Eine Transaktion könnte z.B. erst dann geschlossen werden, wenn das Paket bei dem Empfänger eintrifft und die Post dies meldet. In diesem Fall sind die ACID-Bedindungen unsinnig, denn ein verschicktes Paket kann nicht wieder durch einen einfachen Rücksetz-Befehl ins Lager geschickt werden, die Atomizität ist somit nicht gesichert. Transaktionen, die weitere Transaktionen aufrufen, müssen die ACID-Bedingungen verletzen um die Daten ihrer Wurzel-Transaktion zugänglich zu machen und gleichzeitig noch zurücksetzbar zu sein. Dies läßt sich mit verschiedenen Modellen lösen, ist jedoch außerhalb des ACID-Schemas. Es gibt weitere Transaktionen, die z.B. Statistiken führen und deshalb über Monate hinweg arbeiten. Sie sind nicht mit den gewöhnlichen ACID-Transaktionen zu vergleichen, da sie ja ausnutzen wollen, daß weitere Transaktionen zur gleichen Zeit ausgeführt werden.

Diese Überlegungen lassen zwei Schlüsse zu, erstens daß solch komplexe Transaktionen, die anderen als den ACID-Bedingungen genügen sollen, nicht als ein Muster, sondern als Spezialfälle betrachtet werden müssen, für die kein einheitliches Schema entwickelt werden kann. Für diese Art der Transaktionen wird es wohl nie einheitliche TP-Monitore geben, sondern Programmier-Werkzeuge, die es ermöglichen, TP-Monitore für die gewünschten Bedingungen zu schreiben. Zweitens, daß komplexere Transaktionen auf ACID-Transaktionen aufbauen und sie als Bausteine verwenden um ihre Aufgaben ausführen.

Ein Überblick über das verteilte Betriebssystem Nexus

Ralph Müller

Kurzfassung

Mit **Nexus** wurde eine flexible objektorientierte verteilte Programmierumgebung geschaffen, die es einem erlaubt mit verschiedenen Konzepten der Fehlertoleranz zu experimentieren, basierend auf atomaren Aktionen, dem Setzen von Sicherungspunkten und Wiederaufsetzen innerhalb von Aktionen. Das Modell der atomaren Aktionen in **Nexus** wurde mit dem Ziel entwickelt, Implementierungen für verschiedene Strategien zur Kontrolle der Nebenläufigkeit und des Wiederaufsetzens zu ermöglichen. Die Standardbibliotheken für die Entwicklung von Objektmanagern unterstützen geschachtelte Transaktionen basierend auf einem 2-Phasen-Sperrprotokoll und einem 2-Phasen-Commitprotokoll. **Nexus** unterstützt die Vererbung von Typen und erlaubt das Experimentieren mit verschiedenen Implementierungen für einen Typ. Es ist möglich sich eine eigene Objektmanagershell mit speziell zugeschnittenen Mechanismen für die Nebenläufigkeitskontrolle und für das Transaktionsmanagement zu bauen.

1 Einleitung

Nexus ist ein allgemein einsetzbares verteiltes Betriebssystem für lokale Netzwerke von Workstations. Das Design von **Nexus** stellt eine Synthese von mehreren bereits bekannten Konzepten dar mit der primären Zielsetzung eine verteilte Programmierumgebung zu schaffen, welche die experimentelle Forschung in den Gebieten der fehlertoleranten Techniken und der objektorientierten Programmierung in verteilten Systemen unterstützt. Dies wird durch das Modell der sogenannten **atomaren Aktionen** erreicht, durch deren Flexibilität ein hohes Maß an Nebenläufigkeit erzielt wird. Dieses Modell erlaubt selbst nichtserialisierbare Schedules. Weitere Zielsetzungen beim Design von **Nexus** sind die Gewährleistung von Ortstransparenz im Bezug auf Operationsaufrufe bei Objekten, sowie die Koexistenz von **Nexus** mit einem der üblichen kommerziellen Betriebssysteme beim jeweiligen Wirtsknoten. Durch die Koexistenz kann das Nexus-System außer den eigenen auch die Funktionen des Wirtsbetriebssystemes nutzen. Mit diesem Design gehört **Nexus** zur Gruppe der sogenannten **Netzbetriebssysteme** [MS92]

Das Betriebsystem besteht aus einer Sammlung von systemdefinierten Objekten und einem minimalen verteilten Kern für die Kommunikation zwischen diesen Objekten. Der Kern und andere Hauptkomponenten des Systems wurden bereits implementiert und sind derzeit in einem Netzwerk von **SUN** Rechnern mit dem Betriebssystem **BERKLEY UNIX 4.2** einsatzfähig (siehe Abbildung 22).

Bei der derzeitigen Implementierung des Systems lagen die Bestrebungen weniger darin eine hohe Performance zu erreichen, als vielmehr einen Prototyp zur Abschätzung der Systemstruktur und der Funktionalität des Systems zu bauen. Aus diesem Grund sind

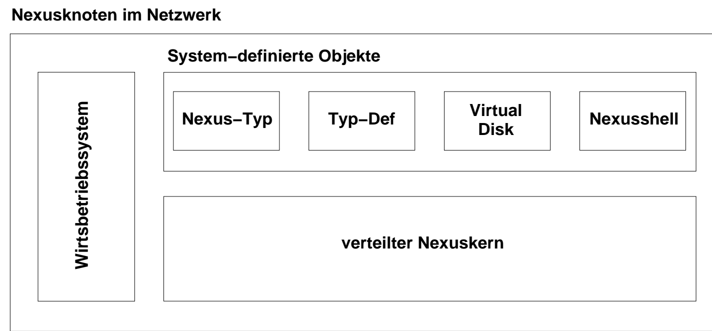

Abbildung22. Nexusarchitektur

der Nexus-Kern und die Objekte derzeit als Unix-Prozesse realisiert worden, und die aktuelle Basis für die Implementierung von **Nexus** ist eine Umgebung mit Workstations, die alle dasselbe Betriebssystem besitzen. Im Design selbst ist aber die Einbindung von Rechnern mit unterschiedlichen Betriebssystemen bereits vorgesehen.

Das **Nexus** Betriebssystem dient als eine abstrakte Maschine für die Implementierung von verschiedenen objektorientierten Anwendungssystemen und Programmiersprachen. Aus diesem Grund wurde auch nicht Wert auf eine bestimmte Programmiersprache für die Definition von Objekttypen gelegt. Die Sprache, in welcher der Code von **Nexus** geschrieben wurde und mit der derzeit die verschiedenen Objekttypen definiert werden, ist C. Viele der Grundkonzepte die, im Nexusdesign verwendet wurden, sind bereits in anderen Systemen wie z.B. **Eden** [AGJ85] und **Argus** [B.84] zum Einsatz gekommen. Im Design von **Eden** wurde bereits das Konzept der Netztransparenz angewendet und sowohl **Argus** als auch **Eden** verwendeten das objektorientierte Modell für verteilte Berechnungen.

2 Berechnungsmodell und Mechanismen

2.1 Übersicht

In diesem Kapitel wird das objektorientierte Konzept von **Nexus** sowie die wichtigsten vorkommenden Objekte beschrieben. Im Anschluß daran wird das Konsistenzmodell der sogenannten schwachen Konsistenz von **Nexus** erläutert, sowie die Mechanismen beim Aufruf von Operationen erklärt. Schließlich wird noch auf das Setzen von Sicherungspunkten eingegangen.

2.2 Die Nexus Objekte

Das Konzept von **Nexus** entspricht dem gängigen Konzept objektorientierter Modelle wie es beispielsweise in [MS92] beschrieben werden. Alle Entitäten in **Nexus** sind Objekte und alle Objekte sind Instanzen eines abstrakten Datentypes. Sie bestehen aus einer Reihe von Zustandsvariablen und besitzen eine Menge von außen sichtbarer Operationen. Diese von außen sichtbaren Operationen bieten die einzige Möglichkeit für andere Objekte, die Zustandsvariablen eines Objektes zu verändern. Eine Operation auf einem

Objekt wird durch das Empfangen einer sogenannten Aufforderungsnachricht (Request) ausgelöst. Nach Beendigung der geforderten Operation wird eine Antwort an die aufrufende Instanz zurückschickt. Die Operationsaufrufe auf den Objekten basieren damit auf dem Konzept des **Remote Procedure Call** (RPC) [MS92].

Jedes Objekt ist Instanz eines benutzer- oder systemdefinierten Typs. Jeder Typ im System ist wiederum ein Objekt, welches Instanz eines systemdefinierten Typs, dem **Nexus-Typ**, ist (siehe Abbildung 23). Solche Objekte werden hier **Typ-Objekte** genannt.

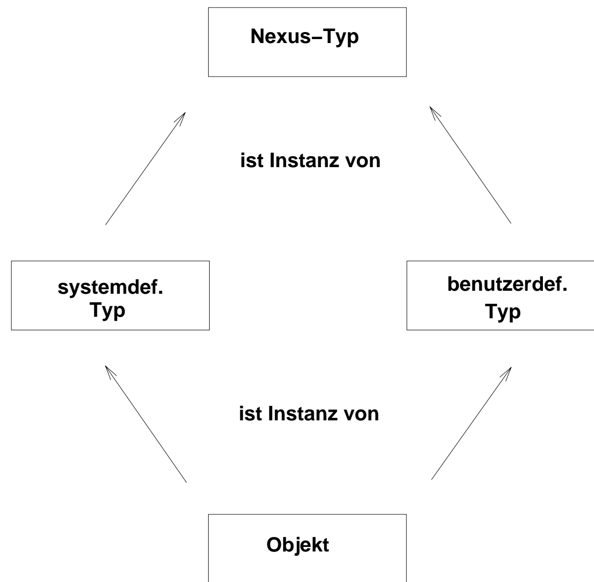

Abbildung23. Nexusobjekte

Die Menge von Objekten, die Instanzen ein und deselben Typs sind, bilden eine **Klasse**. Eine Klasse wird mit dem Namen ihres Typ-Objektes benannt. Teilklassen- sowie Oberklassenbeziehungen basieren auf der einfachen Vererbung von Typen, wie dies z.B. in **Smalltalk** der Fall ist. Jedem Objekt wird ein 64-bit Identifikator (UID) zugeteilt, welcher als globaler Name für das Objekt im Netzwerk dient (siehe Abbildung 24).

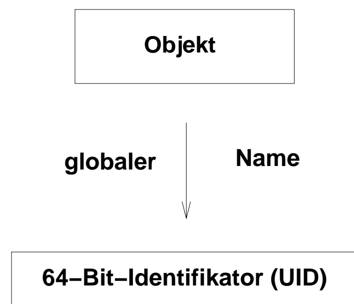

Abbildung24. Objektidentifikator

Ein Nexustyp wird derzeit unter der Verwendung der Sprache C definiert. Die Typdefinition besteht dabei aus mehreren Teilen:

- der Definition von **Interfaceprozeduren**, welche die Menge der nach außen sichtbaren Operationen auf dem Objekt repräsentieren,
- der Definition der Zustandsvariablen, die den konkreten sowie den abstrakten Zustand des Objektes beschreiben,
- dem Code für jede einzelne Interfaceprozedur und
- der Definition der Instanzenerzeugungsprozeduren, die angeben, wie neue Instanzen dieses Typs erzeugt und initialisiert.

Es können auch noch eine Reihe von lokale Prozeduren existieren, die nur aus dem Inneren des Objektes heraus aufgerufen werden können und die nach außen nicht sichtbar sind. Die Interfaceprozeduren besitzen die 3 Parameter **in**, **out** und **in-out** in direkter Analogie zum Syntax beim RPC [MS92]. Die Zustandsvariablen eines Objektes werden in einem sekundären Speicher abgelegt und definieren den Zustand eines Objektes, auf welchen nach einem Crash wieder aufgesetzt werden kann. Die Bezeichnung **Aktion** wird hier benützt, um eine Sequenz von Operationen zu beschreiben; diese Operationen werden dabei mit einer lokalen Prozedur oder einer Interfaceprozedur assoziiert. Der Code einer lokalen Prozedur oder einer Interfaceprozedur besteht aus einer Reihe von Operationen auf den lokalen Zustandsvariablen des Objektes oder denen eines anderen Nexus-Objektes, dessen UID bekannt ist.

Für die Ausführung einer Operation auf einem Objekt wird ein leichtgewichtiger Prozeß erzeugt. Unter einem leichtgewichtigen Prozeß versteht man laut [MS92] einen Prozeß der sich mit anderen leichtgewichtigen Prozessen einen gemeinsamen Adreßraum teilt. Diese Prozesse laufen innerhalb eines Betriebssystemprozesses ab und umfassen nur recht wenige eigene Zustandsinformationen. Der leichtgewichtige Prozeß agiert als Server und führt die Aktionen der Operation aus. Die Zustandsvariablen des Objektes werden dabei aus dem sekundären Speicher in den Hauptspeicher gelesen. Mehrere solcher Serverprozesse können nebenläufig Operationen auf einem Objekt ausführen. Dabei teilen sich die Prozesse einen gemeinsamen Adressraum. Nach der Beendigung einer Aktion wird der neue Zustand des Objektes wieder in den sekundären Speicher zurückgeschrieben.

Jedem Objekt wird ein eigener Objektmanager zugeteilt. Dieser empfängt die Aufrufnachrichten von den anderen Objektmanagern und erzeugt die leichtgewichtigen Prozesse zum Ausführen der Operationen (siehe Abbildung 25).
Die Beziehung zwischen einem Objekt und seinem Objektmanager ist dabei vergleichbar mit der Beziehung zwischen einem Benutzer-Prozeß und dem Kern in einem herkömmlichen Betriebssystem. Der Objektmanager stellt für sein Objekt eine Reihe von Funktionen für den parallelen Aufruf von Operationen, für die Nebenläufigkeitskontrolle und das Wiederaufsetzen zur Verfügung. Er ist auch für das Bereithalten eines wiederaufsetzbaren Zustandes des Objektes im sekundären Speicher verantwortlich. Alle diese Funktionen werden durch eine Reihe von systemdefinierten Bibliotheksroutinen erbracht.

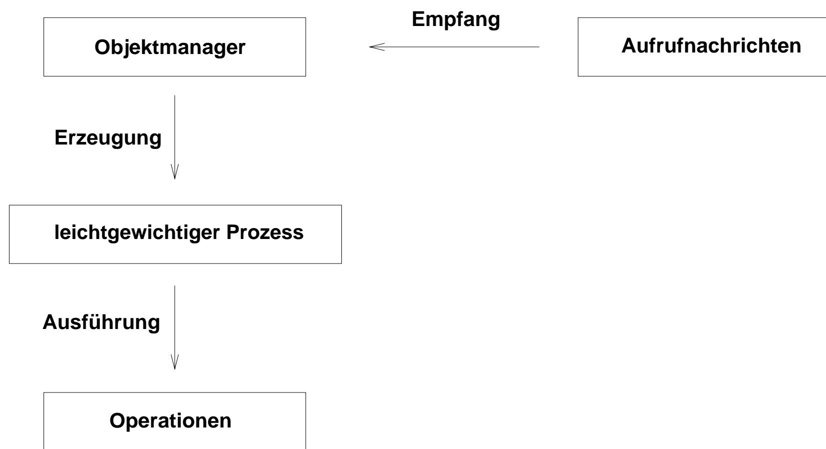

Abbildung25. Funktionen des Objektmanager

2.3 Die Typ-Objekte

Ein Typ-Objekt im Nexus-System hat die Aufgabe, Instanzen seines Typs im Netz zu verwalten. Zu diesem Zweck hält es Informationen über ihren gegenwärtigen Sitz im Netzwerk. Ein Typ-Objekt wird normalerweise durch eine Menge von kooperierenden, vervielfältigten Entitäten implementiert, die als **Klassenrepräsentanten** bezeichnet werden. Jeder dieser Klassenrepräsentanten betreut eine Teilmenge der Objekte seiner Klasse und alle zusammen agieren als eine Art Serverpool, der die Funktionen dieses Objekttyps im System erbringt.

Für alle Typ-Objekte gibt es eine Operation **New**, die noch nicht initialisierte Instanzen dieses Typs erzeugt. Die Typdefinition läßt aber auch eigene Instanzenerzeugungsprozeduren zu. Jedes Typ-Objekt unterstützt auch eine Operation **Delete** zum Löschen von bestimmten Objekten dieser Klasse. Es ist auch möglich verschiedene Implementierungen für ein Typ-Objekt zu unterstützen. Dies geschieht durch den Aufruf der Operation **AddVersion** auf einem Typ-Objekt. Die Operation erfordert die Spezifizierung einer neuen Typdefinition durch den Benutzer. Alle neuen Instanzen dieses Typs werden dann in Übereinstimmung mit der neuen Version erzeugt und verwaltet. **Nexus** unterstützt einen optionalen Parameter, der bei den Operationen zur Erzeugung von Instanzen die gewünschte Version des Typs spezifiziert. Die zuvor erzeugten Instanzen werden weiterhin in Übereinstimmung mit der bei ihrer Erzeugung angegebenen Version verwaltet. Die Funktionen, die eine Instanz von einer Version in eine andere transformieren werden vom Benutzer definiert und sind Bestandteil der Typdefinition.

2.4 Systemdefinierte Typen

Die Programmierumgebung von **Nexus** enthält einige systemdefinierte Objekte. Die zwei Objekte, die eine wichtige Rolle bei der Erzeugung von neuen benutzer-definierten Typ-Objekten spielen, sind **Nexus-Typ** und **Typ-Def**. Beides sind Typ-Objekte und gehören zur Klasse **Nexus-Typ**. Ein Typdefinitions-Objekt, welches eine Instanz von **Typ-Def** ist, dient als Speicher- und Rückholobjekt für die Quell- und Binärfiles einer Typdefinition. **Typ-Def** erledigt die notwendigen Übersetzungs- und Bindeoperationen die erforderlich sind, um ein neues Typdefinitions-Objekt zu erzeugen. Das Objekt

Nexus-Typ ist verantwortlich für die Verwaltung aller Typ-Objekten und der zugehörigen Repräsentanten in der Umgebung von **Nexus** und unterstützt die verschiedenen Versionen für einen Typ. Es unterstützt die Erzeugung und das Löschen sowie die Aktivierung und die Deaktivierung von Repräsentanten für einen Typ an einem bestimmten Nexus-Knoten. Außerdem ist es ein Bestandteil des Objektlokalisierungsalgorithmus, der vom Kern ausgeführt wird.

Durch die Einführung der Typdefinitions-Objekte in das System wurden die Übersetzungs- und Bindefunktionen von **Nexus-Typ** abgetrennt. Dadurch konnte das Design von **Nexus-Typ** einfach gehalten werden und es ermöglicht eine bessere Performance für den Objektlokalisierungsalgorithmus. Die Trennung der quellcoderelevnanten Informationen eines Typdefinitions-Objektes von seinen Verwaltung läßt es zu, dasselbe Typdefinition-Objekt für verschiedene Typ-Objekte zu verwenden.

Um einen neuen Typ im System zu erzeugen muß zuerst ein Typdefinitions-Objekt vom Benutzer erzeugt werden. Dazu gibt der Benutzer ein File an, daß den Code für den neuen Typ sowie die Vererbungsspezifikationen enthält. Anschließend wird die Operation **New** auf **Typ-Def** ausgeführt. **Typ-Def** führt dann die folgenden Schritte aus:

– Zuerst wird der Code des neuen Typs vorverarbeitet und die Vererbungsinformationen integriert.

– Anschließend werden die RPC Stubroutinen [BA84] generiert, der Quellcode übersetzt und zur zugehörigen Bibliothek gebunden, um ein ausführbares Binarfile zu erzeugen.

Eine Klasse von Objekten die vom Nexus-System unterstützt wird, ist die **Nexus-Shell**. Objekte dieser Klasse werden für den interaktiven Zugang zur Nexusumgebung verwendet und stellen dem Benutzer Schnittstellen zum System zur Verfügung. Ist der Benutzer erst einmal in das System eingeloggt, assoziert der Kern eine Instanz dieser Klasse mit dem Terminal, von dem aus der Einlogvorgang stattgefunden hat. Die **Nexus-Shell** interpretiert die Benutzerbefehle und übersetzt sie in die erforderlichen Aufrufe des Nexus-Systems.

2.5 Konsistenz von Aktionen in Nexus

Das Konzept **atomarer Aktionen** wurde in der Vergangenheit als Werkzeug zum Bau fehlertoleranter Systeme und zur Systemstrukturierung eingesetzt. Eine **atomare Aktion** genügt den Bedingungen der Unteilbarkeit und der Serialisierbarkeit. Unteilbarkeit bedeutet hier, daß die Aktion die "Alles oder Nichts" Bedingung für den Fehlerfall erfüllt, d.h., daß entweder alle Operationen der atomaren Aktion ausgeführt werden, oder keine. Die Definition der Serialisierbarkeit besagt, daß der Endzustand nach der Ausführung einer Menge von nebenläufigen Aktionen derselbe ist, wie jener, der durch die Ausführung dieser Aktionen in einer beliebigen anderen Reihenfolge entstehen würde. Das System **Argus** stellte dieses Konzept atomarer Aktionen (auch Transaktionen genannt) und geschachtelter atomarer Aktionen in seinem objektorientierten, verteilten Berechnungsmodell vor. In **Argus** kann eine Aktion andere Aktionen auf dem eigenen Objekt oder

auf anderen entfernten Objekten auslösen. Solche Aktionen werden als **geschachtelte Aktionen** bezeichnet. Eine **Top-level Aktion** ist eine Aktion, die unabhängig von der Terminierung der Aktion des Aufrufers terminieren kann. Ist sie nicht unabhängig von Aufrufer so nennt man sie **geschachtelte Transaktion** [Tan95] . Das Modell der geschachtelten Transaktionen erfordert, daß:

- alle Top-level Aktionen untereinander serialisierbar sind,
- daß alle geschachtelten Transaktionen derselben aufrufenden Aktion untereinander serialisierbar sind und
- daß die Terminierung einer geschachtelten Transaktion von der Terminierung der aufrufenden Aktion abhängig ist.

Das Konzept des globalen "Alles oder nichts" im Bezug auf Terminierung und Serialisierbarkeit schränkt die Nebenläufigkeit in einem System sehr stark ein und stellt zudem sehr große Anforderungen an die Durchführung der erforderlichen Protokolle. In manchen Fällen ist es wünschenswert, Ausnahmen von diesem Konzept zu machen, um die Effizienz zu steigern. Es gibt einige Vorteile, die durch die Unterstützung von nichtserialisierten Ausführungen von Aktionen entstehen. Durch die Reduzierung von Verzögerungen, die bei Konflikten beim Zugriff auf diesselben Objekte zwischen nebenläufigen Aktionen entstehen, kann die Performance verbessert werden. Wenn eine Aktion aufgrund eines Crashes weder terminieren noch abbrechen kann, werden die von ihr benutzten Objekte nicht mehr für unabsehbare Zeit blockiert und dadurch die Verfügbarkeit der Objekte im System erhöht. Außerdem sollten gewisse Objekte, die sehr oft benützt werden und die kritisch für die Aufrechterhaltung der normalen Systemoperationen sind, nicht für die gesamte Dauer einer langen Transaktion gesperrt werden. Beispiele für solche Objekte sind Filesystemverzeichnisse, Datenbankkataloge und Namenserververzeichnisse in verteilten Betriebssystemen.

Aufgrund der obigen Beobachtungen wurde im Nexusdesign das Konzept der schwachen Konsistenz für Aktionen eingeführt. Demnach genügt die Ausführung einer Aktion den folgenden Bedingungen:

- alle Top-level Aktionen sind untereinander serialisierbar,
- alle geschachtelten Aktionen dersselben aufrufenden Aktion sind untereinander serialisierbar,
- ein entfernter Aufruf durch eine Aktion wird wieder als eine Top-level Aktion ausgeführt und
- ein lokaler Aufruf wird immer als geschachtelte Transaktion seiner aufrufenden Aktion behandelt und hängt von der Terminierung der aufrufenden Aktion ab.

Per default terminiert eine Aktion in **Nexus** mit der schwachen Konsistenz. Der Transaktionsmechanismus kann vom Benutzer eingesetzt werden, um stärkere und kostspieligere Anforderungen an Konsistenz und Serialisierbarkeit durchzusetzen. Jede Anwendung

kann diesen kostspieligeren Mechanismus benutzen, wenn es nötig ist. Per default genügt eine Aktion den Bedingungen der Unteilbarkeit und Serialisierbarkeit nur im Bezug auf die Zustandsvariablen des Objektes.

2.6 Transaktionsmodelle und Primitive

Eine Aktion kann eine Reihe von Operationen im Transaktionsmodus ausführen, Dies wird durch das Setzen des Befehls **BeginTrans** zu Beginn der Aktion und des Befehls **EndTrans** am Ende der Aktion erreicht. Alle lokalen und entfernten, geschachtelten Aktionen die innerhalb eines Transaktionsmodus aufgerufen werden, befinden sich ebenfalls im Transaktionsmodus und werden wie geschachtelte Transaktionen behandelt. Eine Transaktion die von einem Paar von **BeginTrans** und **EndTrans** Befehlen eingeschlossen ist, terminiert genau dann, wenn die sie aufrufende Aktion terminiert. Eine Transaktion terminiert dann, wenn sie entweder das Primitiv **EndTrans** oder **AbortTrans** ausführt. Falls keine Fehler aufgetreten sind, wird die Aktion durch den Befehl **EndTrans** festgeschrieben, anderenfalls wird sie abgebrochen. Bei den dabei verwendeten Commitprotokoll handelt es sich um das Zwei-Phasen-Commit Protokoll (siehe auch [Tan95].

Durch Ausführung des Primitivs **AbortTrans** kann eine Transaktion sich selbst abbrechen. Dabei werden alle Änderungen, die durch die Transaktion bisher hervorgerufen wurden, rückgängig gemacht. Die übergeordnete Aktion setzt dann ihre Ausführung unmittelbar hinter dem zugehörigen **EndTrans** Primitiv fort.

2.7 Aufruf von Operationen auf Objekten

Auf der Objektebene unterstützt **Nexus** sowohl den synchronen als auch den asysnchronen Aufruf von Operationen auf einem Objekt. Dieser Aufruf kann entweder durch das Objekt selbst (lokaler Aufruf) oder durch ein anderes Objekt (entfernter Aufruf) erfolgen.

Das synchrone Aufrufsmodell basiert auf dem Konzept des **RPC** [MS92]. Dabei bleibt der Aufrufer solange blockiert bis eine Antwort empfangen wird oder ein Timout auftritt. Im Falle des asynchronen Aufrufs wird dem Aufrufer ein Aufrufsidentifikator zurückgegeben, mit dem es ihm später möglich ist, sich mit der Antwort zu synchronisieren. Das Ziel bei der Unterstützung von asynchronen Aufrufen ist es, einen parallelen Aufruf von Operationen auf Objekten möglich zu machen. Es gab im Nexus-Design zwei Gründe die zur Unterstützung des asynchronen Aufrufes führten:

- Die Primitive, die für die Synchronisierung von synchronen Aufrufen notwendig sind, unterscheiden sich nicht so sehr von den Primitiven, die hier für die Synchronisierung von asynchronen Aufrufen mit deren Antworten eingesetzt werden.
- Der zweite Grund basiert auf dem bereits beschriebenen Konsistenzmodell von **Nexus**. Dieses sieht vor, daß ein Prozeß, der asynchrone Aufrufe (geschachtelte Transaktionen) gestartet hat, keine weiteren Berechnungen mehr durchführen darf, bis alle von ihm initiierten Aufrufe beendet worden sind. Lokale Aufrufe auf dem Objekt,

welches durch den Prozess realisiert wird, sind aber weiterhin erlaubt. Der Prozeß ist dadurch selbst für die Koordination aller seiner asynchronen Aufrufe selbst verantwortlich, und dadurch wird ein unnötiger Overhead an zusätzlichen Prozessen für die Steuerung der Aufrufe vermieden. Zusätzlich reduziert sich daruch die Komplexität in der Behandlung von geschachtelten Transaktionen.

Die Standard RPC Bibliothek unterstützt wenn gewünscht für einen entfernten Aufruf eine exaclty-once Fehlersemantik [MS92]. Das bedeutet, daß eine Prozedur entweder ganz ausgeführt wird oder im Fehlerfall keine Auswirkungen hinterläßt und nur den Fehler zurückmeldet. In diesem Fall müssen die Transaktionprimitive **BeginTrans** und **EndTrans** verwendet werden, da **Nexus** per default mit der schwachen Konsistenz arbeitet.

2.8 Setzen von Sicherungspunkten und Wiederaufsetzen

Primitive zum Setzen von Sicherungspunkten und zum Wiederaufsetzen werden aus 2 Gründen unterstützt. Zum einen stellen sie ein Mittel zur Sicherung von Zwischenergebnissen innerhalb einer langen Aktion dar, so daß nach einem Crash die Aktion nicht noch einmal von vorne beginnen muß. Zum anderen bieten sie die auf der Anwenderebene die Möglichkeit, eine Aktion von einer gewissen Stelle an noch einmal ablaufen zu lassen, wenn die bisherige Berechnung zu einem ungewollten Ergebnis geführt hat. Desweiteren verhindert dieses Konzept, daß durch das Zurücksetzen einer Aktion, nicht automatisch alle bereits vor der Aktion gestarteten Aktionen auch zurückgesetzt werden müssen.

Die Funktion **Checkpoint(N)** speichert den gegenwärtigen Zustand einer aufrufenden Aktion im nichttemporären Speicher und gibt an die Aktion einen Parameter **N** zurück. Bei dem Paramter handelt es sich um einen Integer, der mit dem Sicherungspunkt assoziiert wird. Alle Sicherungspunkte, die innerhalb einer Transaktion gesetzt wurden, werden nach der erfolgreichen Beendigung der Transaktion gelöscht. Der Aufruf der Funktion **Restart(N)** durch eine Aktion setzt die Ausführung der Aktion in dem Zustand fort, der mit dem Sicherungspunkt N bezeichnet wurde. Ein Prozeß der eine Transaktion ausführt kann ein mögliches Wiederaufsetzen nur an einem der innerhalb der Transaktion gesetzten Sicherungspunkte anfordern.

3 Die Nexus Systemarchitektur

3.1 Übersicht

Der Nexus-Kern ermöglicht die Kommunikation zwischen Objekten durch den Transport von Nachrichten. Zu diesem Zweck ermittelt er den Standort der Objekte mit Hilfe von Lokalisierungsfunktionen. Objekte derselben Klasse an einem Nexusknoten werden durch einen oder mehrere Unix-Prozesse, den **Klassenrepräsentanten** verwaltet. Die Objekte treten mittels dieser Klassenrepräsentanten mit dem Kern in Kontakt.

3.2 Klassenrepräsentanten

Ein Klassenrepräsentant ist ein Unix-Prozeß an einem Nexusknoten, der die folgenden 2 Funktionen erbringt: Zum einen implementiert er mit Hilfe anderer Repräsentanten seiner Klasse die Funktionen des zugehörigen Typ-Objektes. Zum anderen agiert er als Objektmanager für die Teilmenge der Objekte seiner Klasse, die sich an dem Nexusknoten befinden. Durch das Konzept der Klassenrepräsentanten kann ein einzelner Unix-Prozeß eine Vielzahl von Objekten einer bestimmten Klasse verwalten. Der Repräsentant selbst besteht wiederum aus einer Reihe von nebenläufigen Prozessen, die in einen gemeinsamen Adressraum ablaufen. Jeder Repräsentant besitzt eine eigene UID mittels der die ID des Wirtsknoten, auf dem er sich befindet, ermittelt werden kann. Diese ID ist für die vom Kern durchgeführten Lokalisierungsfunktionen zum Auffinden von Objekten wichtig. Ein Repräsentant kann nicht von einem Wirtsknoten zu einem anderen migrieren.

Es können unterschiedlich viele Repräsentanten für eine bestimmte Klasse existieren, aber nur einer pro Klasse pro Nexusknoten. Ein Nexusknoten kann nur dann Sitz für Objekte eines bestimmten Typs sein, wenn sich ein Klassenrepräsentant für diesen Objekttyp am Knoten befindet. Die Repräsentanten für die Klasse **Nexus-Typ** verwalten alle Typ-Objekte und deren Repräsentanten im Netzwerk. Dabei unterhält jeder Repräsentant dieser Klasse ein Verzeichnis, in dem steht, welche Typ-Objekte im Netzwerk existieren und wer ihre zugehörigen Repräsentante sind.

Jedem Repräsentanten stehen zur Speicherung seiner lokalen Objekte ein oder mehrere **virtuelle Festplattenobjekte** (**virtual Disks**) zur Verfügung. Die Objekte dieses Typs realisieren den Sekundärspeicher für Objekte. Der Grund für ihre Einführung liegt darin begründet, daß das Unix File System nicht für das synchrone Schreiben von Datenblöcken auf Festplatten ausgelegt ist. Aus diesem Grund mußte eine Speichereinheit entworfen werden, die das Unix File System umgeht. Die Funktionen der **virtual Disks** erlauben sowohl das synchrone wie das asynchrone Schreiben von Speicherblöcken auf die virtuelle Festplatte. Zusätzlich realisieren die **virtual Disks** eine Reihe von Funktionen zum Wiederaufsetzen und zur Verwaltung von Transaktionen. Sie dienen damit als Basis für die Implementierung von verschiedenen Transaktions Commit-Protokollen. Durch die Integration dieser Funktionen in die **virtual Disks** werden Updates an Objekten oder ein Wiederaufsetzen nach einem Crash nur lokal durch die Festplattenobjekte ausgeführt, ohne daß die Repräsentanten, welche die Objekte verwalten, davon betroffen werden. Dies vermeidet einen Overhead in der Kommunikation beim Wiederaufsetzen und beim Aktualisieren. Die **virtual Disks** können wie jedes andere Objekt, ohne Wissen ihres genauen Standortes im Netzwerk, angesprochen werden. Ihr Standort kann während der Laufzeit variieren, was aber keine Auswirkungen auf die von ihnen erbrachten Funktionen hat. Gesteuert werden die **virtual Disks** durch einen Unix-Prozeß.

3.3 Objekt Manager Shell

Die Hauptkomponente eines Repräsentanten ist die **Objekt Manager Shell**. Sie unterstützt eine Menge von nebenläufig arbeiteten, leichtgewichtigen Serverprozessen und versorgt sie mit Objektverwaltungsfunktionen. Dies beinhaltet den parallelen Aufruf

von Operationen, Nebenläufigkeitskontrolle, Transaktionsverwaltung, das Setzen von Sicherungspunkten und das Wiederaufsetzen und die Speicherverwaltung des sekundären Speichers für lokale Objekte. Die Shell besteht aus einem Kontrollprozeß, einigen Datenstrukturen und Routinen welche die oben beschriebenen Funktionen realisieren.

Die Kontrolle der Nebenläufigkeit basiert in **Nexus** auf dem **2-Phasen-Sperrprotokoll** [Tan95] aufgrund seiner Einfachheit. Bevor eine nichttemporäre Zustandsvariable angesprochen werden kann, wird der Speicherblock in dem sie sich befindet durch die betreffende Aktion gesperrt. Diese Sperre werden wieder aufgehoben sobald die Aktion terminiert oder abbricht. Deadlocks zwischen Objekten sollen im Nexus-Design unter anderem durch Timoutmechanismen vermieden werden.

Die nichttemporären Zustandsvariablen eines Objektes werden in aufeinanderfolgenden Speicherblöcken auf einem virutal Disk-Objekt gespeichert. Die Abbildung dieser Zustandsvariablen auf die Speicherblöcke wird während der Übersetzung der Typdefinition des Objektes durchgeführt.

Alle Aufrufe von Operationen eines Objektes auf sich selbst, werden nach dem Konzept der schwachen Konsistenz als geschachtelte Transaktionen behandelt. Damit sie den Synchronisationsanforderungen für geschachtelte Transaktionen gerecht werden, müssen die Anforderungen für Sperren von nebenläufigen, miteinander konkurrierenden Aktionen mit Hilfe von Semaphoren synchronisiert werden [RU92]. Die Implementierung der Semaphore wird hier durch die Tatsache erschwert, daß eine Aktion, die gegenwärtig eine Lesesperre für ein Objekt hält, später eine Schreibsperre für das gleiche Objekt beantragen kann. Um Verklemmungen zwischen Objekten zu verhindern werden Verklemmungsvermeidungsschemata (basierend auf wound-wait oder wait-die [Tan95]) oder ein Verklemmungserkennungsschema (basierend auf wait-for-cycles [Tan95]) eingesetzt.

3.4 Der Nexus-Kern

Der Nexus-Kern ist eine verteilte Einheit die an jedem Nexusknoten ausgeführt wird. Auf der Kernebene sind die Funktionsaufrufe asynchron. Der Nexus-Kern realisiert die Kommunikation zwischen den Objekten und aktiviert bzw. deaktiviert die Klassenrepräsentanten. Wenn ein Request für einen Repräsentanten eintrifft, der zur Zeit inaktiv ist, beauftragt der Nexus-Kern einen Unix-Prozeß mit der Ausführung des Codes dieses Repräsentanten. Die Requests und die zugehörigen Antworten werden in Mailboxen zwischengespeichert.

Das Hauptproblem, das bei der Kommunikation zwischen Objekten auftritt ist die Lokalisierung der betreffenden Objekte. Da der Kern hauptsächlich mit Repräsentanten zu tun hat, sind seine Informationen über die einzelnen Objekte begrenzt. Die Nachrichten werden daher an die Repräsentanten geschickt, die das betreffene Objekt verwalten. Theoretisch müßte somit der Kern für jedes Objekt, Informationen über den zugehörigen Repräsentanten verwalten. Dies ist aus Effizienzgründen abzulehnen. In der Praxis langt bereits eine gute Schätzung für den Repräsentanten, denn wenn die Nachricht an einen falschen Repräsentanten schickt, kann dieser sie an die richtige Adresse weiterleiten (forwarden). Für die Schätzung ist der sogenannte **Objektlokalisator** im Kern zuständig. Seine Aufgabe erfüllt dieser mittels zweier Cachetabellen. Ein Repräsentant

kann einen Aufruf auf einem Objekt entweder durch die Angabe der UID des betreffenden Repräsentanten oder der UID des Objektes selbst anfordern. Gibt er die UID des zugehärigen Repräsentanten an, so wird der Request direkt an den Repräsentanten weitergeleitet, da die ID des Wirtsknotens leicht aus der UID abgeleitet werden kann. Gibt er die UID des Objektes an, so sucht der Kern zuerst in den Cacheseiten des **Objektcache** nach der betreffenden UID. Dieser Cache bildet die UID eines Objektes auf die UID eines Repräsentanten ab. Wird dort kein Eintrag gefunden so sucht der Kern im **Typcache**, der für einen gewissen Typ eine Liste mit der Teilmenge der zugehörigen Repräsentanten angibt. Wenn dort ein Eintrag gefunden wird, so wird die Nachricht an den ersten Repräsentanten in der Liste weitergeleitet. Wurde auch dort kein Eintrag gefunden, so fordert der Kern den Standort für diesen Typ von einem der Repräsentanten des Typs **Nexus-Typ** an. Ein solcher Repräsentant kann immer entweder im **Typ-Cache** oder durch eine Netzwerksuche gefunden werden. Im Falle einer Netzwerksuche wird der Kern nicht blockiert und steht anderen Instanzen zur Verfügung. Dieses Vorgehen wird in Abbildung 26 veranschaulicht.

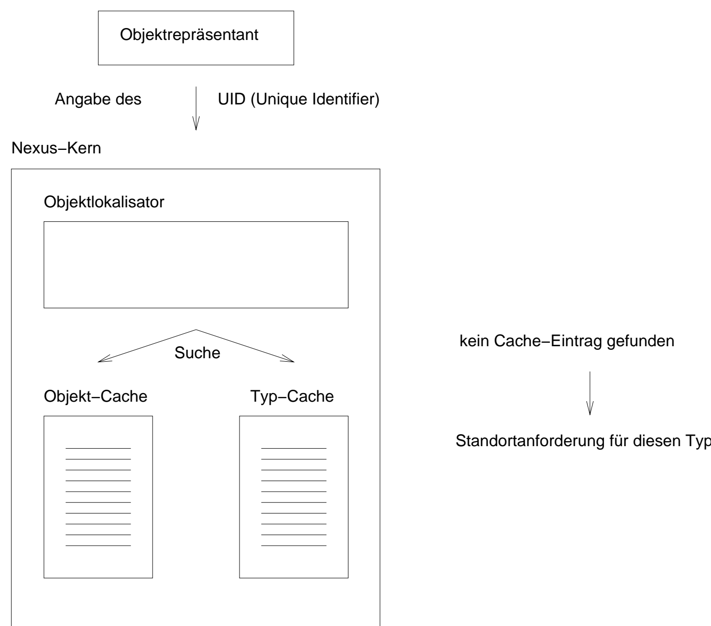

Abbildung26. Lokalisierung von Objekten

Beim Design der beiden Caches wurden Funktionen berücksichtigt, mit deren Hilfe sich die Größe der Caches dynamisch zur Laufzeit ändern lassen. In der gegenwärtigen Ausführung werden die Cache-Einträge periodisch aktualisiert.

4 Zusammenfassung

Nexus vereint mehere bereits bekannte Konzepte wie Netzwerktransparenz und objektorientierte Modelle für verteiltes Programmieren, die bereits in Systemen wie **Eden** und **Argus** eingesetzt wurden. Hauptziel von **Nexus** ist es, eine flexible Umgebung

für die experimentelle Forschung in den Gebieten der verteilten, objektorientierten Programmierung und der fehlertoleranten Techniken zu schaffen. Zu diesem Zweck erlaubt **Nexus** z.B. die Implementierung maßgeschneiderte Objektmanager mit denen verschiedenen Mechanismen für das Wiederaufsetzen und die Nebenläufigkeitskontrolle realisiert werden können. Um eine stärkere Nebenläufigkeit zu erzielen, unterstützt **Nexus** auch nichtserialisierbare Schedules. Möglicht wird dies durch das Modell der schwachen Konsistenz, das ermöglicht, daß eine Aktion selbst dann terminieren kann, wenn die sie aufrufende Aktion abbricht. Dabei wird dem Faktor Sicherheit aber keine Priorität eingeräumt. Der Benutzer muß sich also darüber im klaren sein, was er mit seiner Freiheit anfängt bzw. im System anrichtet. Es besteht zwar die Möglichkeit, auch starke Konsistenz und geschachtelte Transaktionen zu unterstützen, dies geht aber auf Kosten der Performance des Systems. Die Argumentation, daß durch die schwache Konsistenz die Verfügbarkeit von systemkritischen Objekten wie z.B. Filesystemverzeichnissen erhöht wird, läßt sich nicht nachvollziehen. Es existieren in anderen Systemen wie z.B. **CODA** [KS92] Spezialbehandlungen für solche systemkritischen Objekte, die auch ohne das Modell der schwachen Konsistenz die Verfügbarkeit garantieren. Der systemdefinierte Typ der virtuellen ermöglicht eine Umgehung des Unix File Systems, daß für synchrones Schreiben von Speicherblöcken auf die Festplatte nicht ausgelegt ist. Das Setzen von Sicherungspunkten und Funktionen zum Wiederaufsetzen entspricht den gängigen Sicherheitstechniken. Der Benutzer kann eigene Typen von Objekten definieren aber in der gegenwärtigen Version des Systems dauert das Compilieren und Installieren eines neuen Typs im System, noch zu lange (10-20 Minuten). Ein Vorteil des Netzbetriebssystemansatzes ist, daß auf der Anwenderebene alle Unix-Funktionen sichtbar sind.

Abbildungsverzeichnis

1 Reisekostenabrechnung 3
2 Daten im WFMS 8
3 Referenzarchitektur der WMC 9
4 Konzeptuelles Modell von mobilen Agenten Computing 21
5 Transducer 27
6 Wrapper 27
7 Viewpoints des ODP-RM 33
8 ODP Trading-Szenario 34
9 Computational-Modell in ANSA 36
10 Verbindungsmodelle der Organisationseinheit 37
11 Föderationsvertrag 38
12 Zusammenarbeit zwischen logischen Tradern in ODP 42
13 Modell zur Grenzüberquerung 43
14 Coercion in ANSA 46
15 Vergleich von ANSA und ODP 47
16 Innere Struktur der Transaktions-Verwaltung 53
17 Ausdehnung einer verteilten Transaktion 55
18 Kontrollfluß durch einen TP-Monitor 58
19 Ablauf einer verteilten Transaktion mit Hilfe eines TP-Monitors 58
20 Verschiedene Realisierungsmöglichkeiten eines TP-Monitors 60
21 Komponenten und Schnittstellen der DTP-Architektur 63
22 Nexusarchitektur 66
23 Nexusobjekte 67
24 Objektidentifikator 67
25 Funktionen des Objektmanager 69
26 Lokalisierung von Objekten 76

Literatur

[AGJ85] Lazowska E.D. Almes G.T., Black P. und Noe J.D. The Eden System, a technical review. *IEEE Transactions on Software Engineering* Band SE-11, Jan 1985, Seite 43–59.

[ANS93] Architechture Projects Management Limited ANSA. The ANSA Trading and Federation model. *Architechture Report*, Feb 1993, Seite 1–58.

[A.R89] Tripathi A.R. An Overview of the Nexus Operating System Design. *IEEE Transactions on Software Engineering* **15**(6), June 1989, Seite 687–695.

[B.84] Liskov B. *Overview of the Argus Language and System*. Feb 1984.

[BA84] Nelson B.J. Birrell A. Implementing remote procedure calls. *ACM Trans. Comp. Syst.* **2**(1), Feb 1984, Seite 39–59.

[Bor90] Uwe M. Borghoff. *Catalogue of Distributed File/Operating Systems*. Springer. 1990.

[BR91] Mirion Bearman und Kerry Raymond. Federating Traders: an ODP Adventure. *Int. IFIP Workshop on ODP, Berlin*, Oct 1991.

[Bra87] Ivan Bratko. *PROLOG, Programming für Künstliche Intelligenz*. Addison-Wesley. 1987.

[Bü95] U. Bürger. Aufgaben und Schnittstellen von TP-Monitoren in Verteilten Systemen. *Kommunikation in verteilten Systemen*, Feb 1995, Seite Seite 330–344.

[Coa94] Workflow Management Coalition. *Glossary - A Workflow Management Coalition Specification*. Workflow Management Coalition. 1994.

[DGMH+93] U. Dayal, M. Garcia-Molina, M. Hsu, B. Kao und M.-C. Shan. Third Generation TP-Monitors: A Database Challange. *Proc. 1993 ACM SIGMOD*, May 1993.

[GK94] M. R. Genesereth und S.P. Ketchpel. Software Agents. *Communication of the ACM* **37**(7), July 1994, Seite 48.

[GR93] Jim Gray und Andreas Reuter. *Transaction Processing*. Morgan Kaufmann Publishers, San Mateo, California. 1993.

[Gro93] Object Management Group. The Common Object Request Broker Architecture and Specification. *OMG Document Number 931243* (Draft 29), 1993.

[HCK95] Colin G. Harrisson, David M. Chess und Aaron Kershenbaum. Mobile Agents: Are they a good idea? Technischer Bericht, IBM Research Division, March 1995.

[Her89] A. J. Herbert. The ANSA Project and Standards. *ACN Press,Frontier Series, Addison and Wesley*, Dec 1989, Seite 458.

[Jab95a] Stefan Jablonski. Workflow-Management-Systeme: Motivation, Modellierung, Architektur. *Informatik-Spektrum*, 1995, Seite pp. 13 – 24.

[Jab95b] Stefan Jablonski. *Workflow-Management-Systeme: Motivation, Modellierung, Architektur*. International Thomson Publishing. 1995.

[KS92] Kistler und Mahadev Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems* **10**(1), Feb 1992, Seite 3–25.

[Lam94] Winfried Lamersdorf. *Datenbanken in verteilten Systemen:Konzepte, Lösungen, Standards*. Vieweg. 1994.

[LKK93] Peter Claus Lockemann, Gerhard Krüger und Heiko Krumm. *Telekommunikation und Datenhaltung*. Hanser Verlag. 1993.

[Loc95] Prof. Peter Lockemann. *Transaktionsverwaltung*. Vorlesungsskript. 1995.

[Mae94] Patti Maes. Agents that Reduces Work and Information Overload. *Communications of the ACM* **37**(7), Juli 1994, Seite 31–40.

[MB92] Alistair J. Macartney und Gordon S. Blair. Flexible trading in distributed multimedia systems. *Computer Networks and ISDN Systems 25* **Vol 5**(Nr 2), 1992, Seite 145–157.

[MS92] Max Mühlhäuser und Alexander Schill. *Software Engineering für verteilte Anwendungen*. Springer Verlag. 1992.

[OD94] ISO/IEC 13235:1994 ODP-Draft. ODP Annex A: tutorial of the Trading function. *Draft ODP Trading function :1994*, Dec 1994, Seite A1–A27.

[OH92] Robert Orfali und Dan Harkey. *Client/Server Survival Guide*. International Thomson Publishing. 1992.

[PY92] M. Palaniappan und N. Yankelovitch. The Envoy Framework : An Open Architecture for Agents. *ACM Transactions on Informations Systems* **10**(3), July 1992, Seite 233–264.

[RU92] Levi P. Rembold U. *Realzeitsysteme zur Prozessautomatisierung*. 1992.

[Rum91] James Rumbaugh. *Object-oriented modeling and design*. Prentice Hall. 1991.

[Tan92] Andrew S. Tanenbaum. *Computer-Netzwerke*. Wolfram s Verlag. 1992.

[Tan95] Andrew S. Tanenbaum. *Distributed Operating Systems*. Prentice Hall. 1995.

[Ver95] Gerhard Versteegen. Alles im Fluß. *iX*, März 1995, Seite pp. 152 – 160.

[Wag95] Michael Peter Wagner. Elektronische Debatte. *iX*, Februar 1995, Seite pp. 112 – 116.

[Web96] Webster's New Encyclopedic Dictionary. New Revised Edition 1996. Black Dog &Leventhal Publishers Inc., New York, 1996.

[Whi94] James E. White. Telescript Technology: Scenes from the Electronic Market Place. Technischer Bericht, General Magic White Paper, 1994.

[WJ95] Michael J. Wooldridge und Nicholas R. Jennings. *Agent Theories, Architectures, and Languages: A Survey*. Springer Verlag. January 1995.