Kommunizierende Informationssysteme am Beispiel autonomer *Internet WebSiteProfiler*: Vergleich objekt- und agentenbasierter Ansätze

Diplomarbeit von Ingo Richtsmeier

Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Datenbanken und Informationssysteme (DBIS)
Betreuer: Prof. Dr. J.W. Schmidt

Universität Karlsruhe
Fakultät für Informatik, Forschungszentrum Informatik
Institut für Programmstrukturen und Datenorganisation
Betreuer: Dr. Ralf Kramer

verantwortlicher Betreuer: Prof. Dr. P.C. Lockemann

Januar 1997

Zusammenfassung

Ziel dieser Arbeit ist die Entwicklung eines kommunikationsorientierten Modells für kooperative Informationssysteme. Dabei wird die Kommunikation nach der Language/Action-Perspektive in Konversationen, die aus Dialogen bestehen, gegliedert. Die Konversationen werden anwendungsorientiert geführt. Im Modell werden maschinelle und menschliche Akteure gleich behandelt. So können verschiedene Kommunikationsformen abgewickelt werden: Zum Beispiel können mittels graphischer Benutzeroberflächen Menschen mit Maschinen kommunizieren, und die Kommunikation zwischen zwei Maschinen kann objekt- oder agentenbasiert geschehen. Außerdem soll die Kommunikation so weit wie möglich vom Medium abstrahieren.

Das Modell wird durch ein Metamodell in der Notation einer objektorientierten Entwurfsmethode beschrieben. In der Arbeit wurde die Infrastruktur für das Modell prototypisch im persistenten Objektsystem Tycoon mit der zugehörigen Programmiersprache TL implementiert. Das System und die Sprache wird in der Arbeit ausführlich vorgestellt und mit anderen kommerziellen Systemen für Objektverteilungen wie CORBA, DCE oder OLE verglichen.

Anhand eines konkreten Beispiels, das sich aus autonomen WebSiteProfilern und MultiSiteAnalyzern zusammensetzt, wird das System getestet. Die WebSiteProfiler erstellen Nutzungsstatistiken von WWW-Servern für jeden Server einzeln. Die MultiSiteAnalyzer aggregieren die Daten der WWW-Server, die inhaltlich zusammenhängen, und vergleichen die Server miteinander. Das Beispielszenario wird für einen Vergleich von objektbasierter und agentenbasierter Kommunikation herangezogen.

Inhaltsverzeichnis

1 Einführung und Motivation **7**
1.1 Ziel der Arbeit 8
1.2 Gliederung der Arbeit 8

2 Polymorphismus, orthogonale Persistenz und Mobilität **11**
2.1 Das Tycoon-System 11
2.2 Die Systemarchitektur von Tycoon 12
2.2.1 Das Tycoon Store Protocol (TSP) 12
2.2.2 Die Tycoon Virtual Machine (TVM) 15
2.2.3 Die Tycoon Machine Language (TML) 16
2.2.4 Die Tycoon-Hochsprachen 16
2.2.5 Erweiterbare Grammatiken 17
2.3 Die Programmiersprache TL 18
2.3.1 Die Eigenschaften von TL 18
2.3.2 Die Bedeutung von Typen und Subtypen in TL 20
2.3.3 Typen von Typen 21
2.4 Kommunikation zwischen Tycoon-Systemen 21
2.4.1 Typunsichere Kommunikation 22
2.4.2 Entfernte Prozeduraufrufe 22
2.4.3 Mobile Threads 23
2.5 Module und Bibliotheken in Tycoon 24
2.6 Ein praktisches Beispiel 24

3 Realisierung von Informationssystemen mit verteilten Objekten **31**
3.1 CORBA: Unabhängige Objektverteilungen 31
3.1.1 Das Objektmodell von CORBA 32
3.1.2 IDL-Schnittstellen 32
3.1.3 Der Object Request Broker 34
3.1.4 Die Object Management Architecture 35
3.1.5 Vergleich von CORBA und Tycoon 37
3.2 DCE: Umgebung für verteilte Systeme 39
3.2.1 Vergleich vom DCE und CORBA 41
3.2.2 Vergleich vom DCE und Tycoon 42

3.3 OLE: Dokumentenzentriertes Arbeiten 43
3.3.1 Vergleich von OLE und CORBA 46
3.3.2 Vergleich von OLE und Tycoon 46

4 Erfassung und Analyse von Internetnutzungsstatistiken 47
4.1 Problemstellung 47
4.2 Der WebSiteProfiler 49
4.2.1 Die Systemarchitektur des WebSiteProfilers 49
4.2.2 Das Datenbankschema des WebSiteProfilers 51
4.3 Der MultiSiteAnalyzer 53
4.3.1 Der inhaltliche Zusammenhang verschiedener WWW-Server 53
4.3.2 Die Systemübersicht 55
4.3.3 Kommunikation des MultiSiteAnalyzers mit den WebSiteProfilern . 55

5 Das Modell der Business Conversations 57
5.1 Die Motivation für das Modell 57
5.2 Die Grundlagen des Modells 59
5.3 Die Konzepte des Modells 61
5.3.1 Das Verarbeitungsmodell von Konversationen 62
5.3.2 Konversationsspezifikationen 63
5.3.3 Konversationsinstanzen 66
5.4 Invarianten des Modells 69
5.5 Ein Anwendungsbeispiel 69

6 Tycoon Business Conversations 73
6.1 Ein Systemüberblick 73
6.1.1 Die Kommunikationsmodule 74
6.1.2 Module für Konversationen 75
6.1.3 Module für die Akteure 76
6.2 Die Einbettung des WebSiteProfiler 78
6.3 Die Nutzung durch den MultiSiteAnalyzer 79
6.4 Vergleich der Tycoon Business Conversations mit anderen Systemen 79

7 Vergleich verschiedener Kommunikationsformen 81
7.1 Objektbasierte Kommunikation 81
7.2 Kommunikation mittels mobiler Agenten 82
7.2.1 Motivation für mobile Agenten 82
7.2.2 Ablauf von agentenbasierter Kommunikation 86
7.3 Ein Vergleich beider Arten 87

8 Zusammenfassung und Ausblick 89
8.1 Zusammenfassung 89
8.2 Praktische Einsatzmöglichkeiten 89

8.2.1 Integration bestehender Applikationen 90
8.2.2 Vorteile beim Einsatz der Tycoon Business Conversations 90
8.2.3 Meßbarkeit der WWW-Server-Reichweiten 90
8.3 Ausblick 90

A Überblick über die Methode der *Mainstream Objects* **93**
A.1 Das Objektstrukturmodell 93
A.2 Das Übersichtsmodell 96

B Glossar für Business Conversations **99**

Literaturverzeichnis **101**

Kapitel 1

Einführung und Motivation

Bisher wurde untersucht, wie *ein* Informationssystem modelliert und realisiert werden kann. Dabei wurden im Bereich der Datenbanken verschiedene Modelle zur Realisierung dieser Systeme und Sprachen zur Manipulation der Daten erstellt. Bei diesen Systemen wurden alle Sachverhalte einer Miniwelt in der Datenbasis aufrechterhalten. Dabei sorgten generische Operatoren des Datenmodells für die Einhaltung der Strukturierungs- und Sortenregeln [LKK93]. Eine Erweiterung der Systeme, in denen auch Operatoren auf Datentypen vorhanden sind, wurde mit den objektorientierten Datenbanksystemen eingeführt. In diesen Systemen werden auch die Operatoren an die Sachverhalte der Miniwelt angepaßt [ABD+90].

Die mächtigen, zentralisierten Informationssysteme fassen Daten, die häufig an unterschiedlichen Orten erzeugt werden, zusammen. Dabei kann die Pflege der Daten vor Ort einfacher sein als die Zusammenfassung aller Daten an einem zentralen Ort. Damit erhält man verteilte Datenbanken. Die Daten einer verteilten Datenbasis sind über mehrere autonom agierende und über ein Netzwerk verbundene Rechner verteilt. Sie hängen dabei logisch zusammen, und auf sie wird von jedem Rechner des Netzwerk als Gesamtheit zugegriffen. Verteilte Datenbanken werden zum Beispiel in Finanzsystemen oder Informationsystemen einer Fluggesellschaft eingesetzt [ÖV94].

Durch verteilte Objektsysteme wie zum Beispiel CORBA, DCE oder OLE wird eine höhere Flexibilität, als sie verteilte Datenbanken bieten, erreicht. In diesen Systemen werden die Daten in Objekten gekapselt. Dadurch wird wie in objektorientierten Datenbanken mehr Autonomie durch restriktiven Zugriff auf die Daten über wohldefinierte Schnittstellen gewährleistet. Benutzer der Objekte können hier jedoch nicht nur von internen Darstellungen abstrahieren, sondern auch eine offene Kommunikationsplattform benutzen, die eine beliebige Verteilung der Objekte im Netzwerk zuläßt [OHE96].

Jedoch wird in Zukunft der Trend in Richtung Aktivitätsorientierung gehen. Agenten, egal ob intelligente Agenten [Mae94], Software Roboter [EW94] oder mobile Agenten [Whi94, Hoh95] bieten trotz aller Unterschiede Autonomie, die es für die Zukunft anbietet, Informationssysteme aus einer beliebigen Mengen solcher Agenten zusammenzusetzen [HP94].

Insgesamt läßt sich für die zeitliche Entwicklung verteilter Informationssysteme eine

Kette von Generalisierungen bilden. Begonnen bei informationszentrierten Systemen, wo die Domäne der Daten, die verwendeten Datenstrukturen und die Integritätsbedingungen an die Daten im Mittelpunkt stehen, geht sie über objektzentrierte Systeme, in denen die Operationen auf den Datenstrukturen und erlaubte Zustandsübergänge eine wichtige Rolle spielen, bis hin zu aktivitätsorientierten Systemen, bei denen die Fragen nach der Zusammenarbeit der Systeme mit ihrer Umgebung und die Ausrichtung an Zielen von großer Bedeutung sind.

Am Arbeitsbereich Datenbanken und Informationssysteme (DBIS) des Fachbereiches Informatik der Universität Hamburg wurde das persistente Objektsystem Tycoon entwickelt. Es bietet eine leistungsfähige Umgebung, um damit ein Informationssystem zu erstellen [MS93, Mat93, MSS95]. In diesem System wurde durch die plattformunabhängige Entwicklung und die Integration von Kommunikationsmechanismen bis hin zu mobilen Threads bereits eine gute Grundlage für die Unterstützung verteilter Informationssysteme geschaffen [Mat96].

Das weitere Forschungsinteresse des Arbeitsbereiches geht von dieser Basis aus. Es soll untersucht werden, wie unternehmensübergreifende Anwendungen betriebswirtschaftlicher Art mit agenten- und kommunikationsbasierten Modellen entworfen und realisiert werden können.

1.1 Ziel der Arbeit

Am Beispiel von autonomen Anwendungen, dem WebSiteProfiler und MultiSiteAnalyzer, sollen verschiedene Aspekte von kommunizierenden Informationssystemen untersucht werden. Das Tycoon-System soll mit anderen Systemen für objektbasierte Verteilungen verglichen werden, um eine Standortbestimmung zu erhalten.

Es soll ein Prototyp für die autonomen Anwendungen und eine Einbindung in eine Umgebung für kooperative Informationssysteme erstellt werden. Dazu soll ein Modell für die Kooperation autonomer Anwendungen entworfen und beschrieben werden. Anschließend sollen in dieser Umgebung Ansätze für zwei verschiedene Kommunikationsformen verglichen werden.

Diese Arbeit soll damit ein Ausgangspunkt für weitere Projektarbeiten im Bereich der kooperativen Informationssysteme am Arbeitsbereich sein.

1.2 Gliederung der Arbeit

Im nächsten Kapitel wird das persistente Objektsystem Tycoon mit seiner Programmiersprache TL behandelt. Alle Programme, die in dieser Arbeit beschrieben werden, sind mit diesem System erstellt. Deshalb soll der Leser einen Überblick über das System und seine Möglichkeiten erhalten.

In Kapitel 3 wird ein Überblick über verschiedene Möglichkeiten zur Realisierung verteilter Informationsysteme gegeben. Dabei werden die Systeme CORBA, DCE und OLE

vorgestellt und miteinander verglichen. Ziel dieses Kapitels ist es, Tycoon in die anderen Systeme einzuordnen und aufzuzeigen, welche Möglichkeiten Tycoon zu Beginn der Arbeit für verteilte Informationsysteme geboten hat.

In Kapitel 4 wird die Beispielanwendung, die aus WebSiteProfilern und MultiSiteAnalyzern besteht, motiviert. Dabei wird die Funktionalität zur Kommunikation noch vernachlässigt und es wird nur auf die Basisfunktionalitäten und den Systemaufbau eingegangen.

In Kapitel 5 und 6 wird ein Modell für Business Conversations vorgestellt. Diese Darstellung erstreckt sich über zwei Kapitel, wobei sich das erste auf die Grundzüge des Modells bezieht und das zweite die Implementierung des Modells im Tycoon-System erläutert. Anschließend werden die Tycoon Business Conversations mit den Systemen aus Kapitel 3 verglichen.

Nachdem das Prinzip der Business Conversations dargelegt und ihre Implementierung mit dem Tycoon-System erklärt ist, werden in Kapitel 7 zwei verschiedene Kommunikationsansätze für das Anwendungsbeispiel verglichen. Der erste Ansatz beruht auf objektbasierter Kommunikation, im zweiten Ansatz wird die Kommunikation mittels mobiler Agenten durchgeführt.

In der Zusammenfassung in Kapitel 8 wird zusätzlich auf die praktischen Einsatzmöglichkeiten des Modells und der Implementierung eingegangen, um die kommerzielle Relevanz der Arbeit zu untersuchen.

Kapitel 2

Polymorphismus, orthogonale Persistenz und Mobilität

In diesem Kapitel wird ein Einblick in das Tycoon-System gegeben. Der Text soll Lesern, die das System noch nicht kennen, einen Überblick über die Leistungsfähigkeit und die Besonderheiten von Tycoon und seiner Programmiersprache TL verschaffen. Außerdem soll der Stand des Tycoon-Systems im Bezug auf aktivitätsorientierte kommunizierende Informationssysteme dargestellt werden.

Nach einer Zusammenfassung der Entwicklungsziele und der wichtigsten Eigenschaften wird die Systemarchitektur des persistenten Objektsystems vorgestellt. Alle Schichten des Systems werden dabei kurz angesprochen. Dann werden die Eigenschaften und Konzepte der Programmiersprache TL besprochen. Die Kommunikationskonzepte der einfachen Kommunikationsverbindungen, entfernten Prozeduraufrufe und mobilen Threads spielen in dieser Arbeit eine große Rolle. Schließlich wird der Leser in das Modul- und Bibliothekenkonzept von Tycoon eingeführt. Am Ende dieses Kapitels wird anhand eines Beispiels untersucht, wie sich die Programmiersprache TL für die Programmierung einer Datenbankanwendung eignet. Dabei soll der Leser durch vier Programmstücke einen ersten Eindruck von der Syntax und der Semantik der Sprache erhalten.

2.1 Das Tycoon-System

Das Tycoon-Projekt, das an der Universität Hamburg im Fachbereich Informatik seit 1992 läuft, soll die Produktivität des Programmierers bei der Entwicklung datenintensiver Anwendungen erhöhen und flexibles Arbeiten in offenen, heterogenen Umgebungen ermöglichen. Um dieses Ziel zu erreichen, wurde an zwei Punkten angesetzt: zum einen wurde die persistente und polymorphe Programmiersprache TL mit einem Typsystem höherer Ordnung entwickelt. Mit dieser Sprache lassen sich bestehende Applikationen leicht in neue Programme integrieren, indem ihre Dienste einheitlich in TL beschrieben werden. Der zweite Ansatzpunkt war die Entwicklung einer persistenten Systemumgebung, die TL unterstützt und bestehende Dienste nahtlos integriert. Das entstandene System zeichnet

sich durch seine Skalierbarkeit und die Fähigkeit zur Zusammenarbeit mit bestehenden und neuen Anwendungen und Diensten aus [MSS95].

Die Zusammenarbeit mit anderen Systemen wird durch drei Aspekte realisiert:

- Die existierenden Dienste werden zum einen einheitlich über polymorphe Tycoon-Bibliotheken integriert. Auf diese Art und Weise wurden bereits mehrere generische Dienste integriert. Beispiele dafür sind Fenster-Systeme wie OpenWindows oder Datenbanksysteme wie Oracle oder Ingres.

- Zum zweiten sind die eingefügten Dienste innerhalb des Systems nicht von Diensten zu unterscheiden, die in TL implementiert wurden. Diese Eigenschaft wird durch die Typsicherheit und Persistenzabstraktion erreicht. Persistenzabstraktion bedeutet, daß die Lebensdauer eines Objektes für den Programmierer nicht sichtbar ist und er alle Objekte in dieser Hinsicht gleich behandeln kann. Außerdem wird dadurch der Unterschied zwischen der programmiersprachlichen Sicht und der Datenbanksicht auf Informationen verkleinert [AM95]. Streng typisierte Dienste sind in der Benutzung sicherer und viele Programmierfehler können schon zur Übersetzungszeit entdeckt werden.

- Der dritte Aspekt zur Unterstützung der Zusammenarbeit ist die Eigenschaft von Tycoon, daß es auch als Subsystem in anderen Systemen eingesetzt werden kann. So können zum Beispiel Teile von Tycoon als Dienstgeber für persistente Speicherung eingesetzt werden. Außerdem können die Tycoon-Applikationen von C-, C++- oder FORTRAN-Programmen aufgerufen werden.

Die Offenheit und Integrationsfähigkeiten des Tycoon-Systems werden in der Abbildung 2.1 deutlich.

2.2 Die Systemarchitektur von Tycoon

Die Systemarchitektur von Tycoon zeichnet sich durch eine strikte Unterteilung in Schichten aus. Die Speicherung, Manipulation, Optimierung, Modellierung und Darstellung von Daten wurde in verschiedene Schichten gelegt. Für viele Aufgaben, die eine Schicht im Gesamtsystem hat, existieren bereits mehrere alternative Implementierungen, die unterschiedliche Anforderungen unterstützen. Die Bausteine der Schichten sind in der Abbildung 2.2 zu sehen.

Im folgenden werden die Schichten des Tycoon-Systems einzeln vorgestellt.

2.2.1 Das Tycoon Store Protocol (TSP)

Durch das Speicherprotokoll wird eine einheitliche Schnittstelle zu verschiedenen persistenten Objektspeichern definiert. Diese Schnittstelle ist vom Datenmodell des Objektspeichers unabhängig. Dabei kann die Implementierung des Objektspeichers jederzeit gewechselt werden, oder es können gleichzeitig mehrere Objektspeicher benutzt werden. Nach [MMS95]

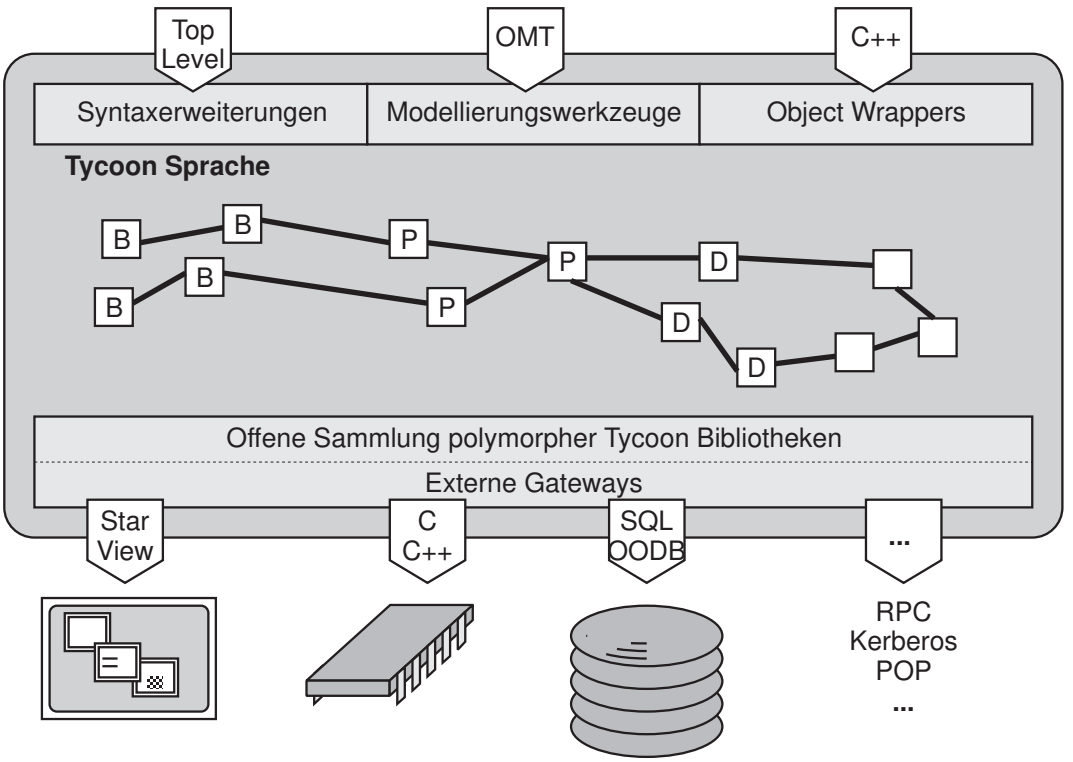

Abbildung 2.1: Die Integrationsfähigkeiten von Tycoon

soll mit diesem Protokoll ein Standard für die Schnittstelle zu persistenten Objektspeichern definiert werden.

In einem Objektspeicher werden nach [Mat93] die folgenden Dienste benötigt, um einer datenintensiven Anwendung gerecht zu werden:

Automatische Freispeicherverwaltung: Die Klienten des Objektspeichers dürfen ihre Plätze im Speicher nicht mehr selbst freigeben. Stattdessen soll eine Speicherbereinigung im Objektspeicher alle nicht mehr erreichbaren Speicherplätze wieder freigeben.

Persistenz: Objekte müssen als persistent gekennzeichnet werden können. Alle Objekte, die über das gekennzeichnete Objekt erreicht werden können, sind dann ebenfalls persistent.

Atomarität, Fehlererholung: Der Objektspeicher muß Sicherungspunkte definieren können. Alle Änderungen an einzelnen Objekten, die nach diesem Sicherungspunkt geschehen, müssen wieder zurückgenommen werden können.

Portabilität: Alle Daten aus einem Objektspeicher müssen in einen anderen Objektspeicher übertragen werden können.

Synchronisation: Es müssen Synchronisationsmechanismen wie das Sperren bestimmter Objekte und die Freischaltung gesperrter Objekte angeboten werden, um den

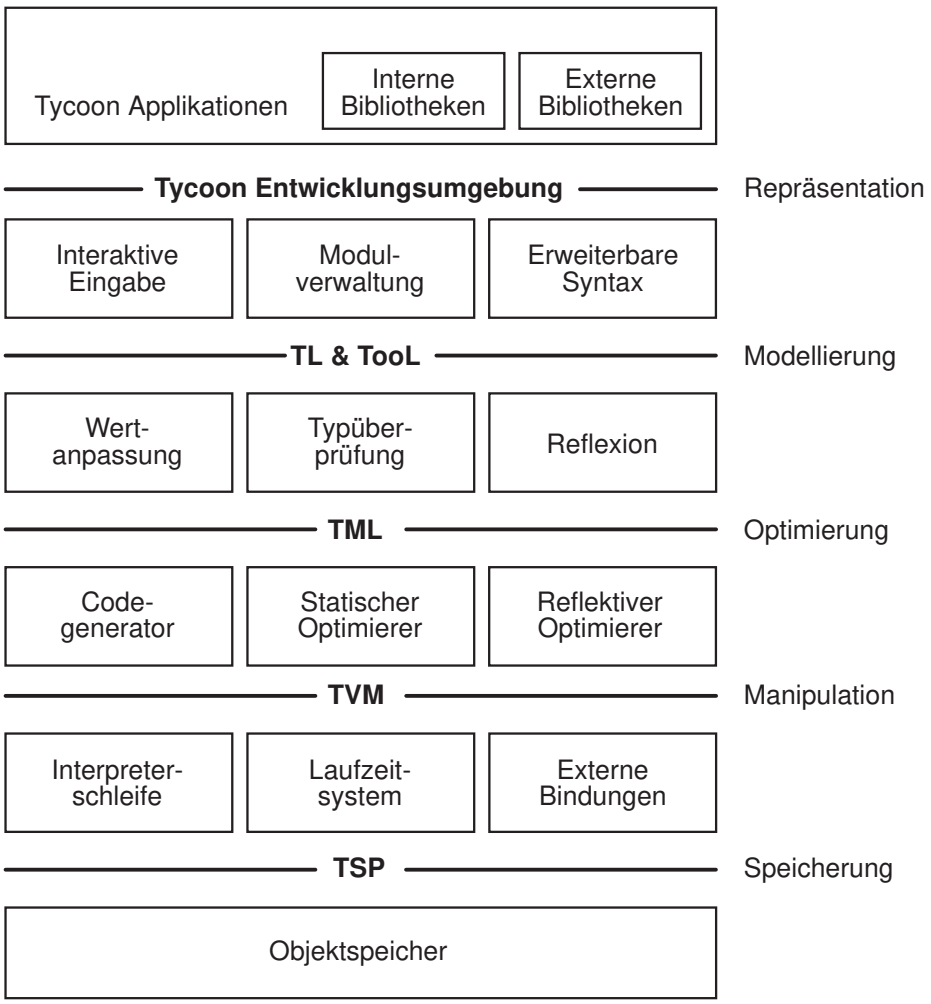

Abbildung 2.2: Übersicht der Schichten im Tycoon-System

Objektspeicher mehreren Benutzern gleichzeitig zugänglich zu machen.

Die Klienten greifen über das Protokoll auf den Speicher zu. Innerhalb des Speichers können neue Objekte angelegt, Objekte verändert und gelöscht werden. Verschiedene Protokollzustände sichern die Persistenz zu. Funktionen zum Commit oder Rollback sind möglich.

Über das Protokoll werden unterschiedliche Objektspeicher an das Tycoon-System angebunden. Die selbstentwickelten Speicher *Tymem* und *Tysin* binden Speichersysteme für Einzelbenutzer an. *Tymem* benutzt zur Speicherung den Hauptspeicher, *Tysin* wurde für Systeme entwickelt, die keine virtuellen Speicher besitzen. Außerdem wurden Schnittstellen zu ObjectStore und NapierStore implementiert. Diese Speichersysteme sind mehrbenutzerfähig. Innerhalb des Protokolls wurde eine lineare Repräsentation der Objektgraphen (TXR, Tycoon external data representation) definiert, mit der die Objekte von einem Speicher in einen anderen übertragen werden können.

Durch die folgenden Punkte kann der Objektspeicher den Anforderungen, die an ihn gestellt werden, angepaßt werden:

- Strategien zur Speicherbereinigung
- Mechanismen zur Behandlung von Fehlern im Objektspeicher
- Fehlererholung, Protokollierung und Mechanismen zur persistenten Speicherung
- Einzelbenutzer- oder Mehrbenutzerzugriff
- Protokolle zur Transaktionsbeendung
- Unterstützung der Zugriffssicherheit und Authentizität

Außerdem gibt das Protokoll Rückmeldungen des Objektspeichers an die Klienten weiter. Sie fördern die enge Zusammenarbeit von Dienstnehmern und -gebern.

2.2.2 Die Tycoon Virtual Machine (TVM)

Die virtuelle Tycoon-Maschine definiert einen Befehlssatz, der auf einem funktionalen Ausführungsmodell mit Funktionen höherer Ordnung basiert. Alle Befehle sind durch Bytes codiert. Die Ausführung des Bytecodes geschieht entweder durch Interpretation der Befehle oder durch eine Übersetzung in Maschinenbefehle. Weil die virtuelle Maschine plattformunabhängig entworfen und implemeniert ist, kann der Programmcode innerhalb eines heterogenen Rechnernetzes von einem Rechner auf einen anderen ohne Neuübersetzung übertragen werden. Zusammen mit der austauschbaren Darstellung der Daten im Objektspeicher können so persistente *Threads* (Bindungen zwischen Daten und Programm, das sich in der Ausführung befindet [MS94]) zwischen verschiedenen Rechnern wechseln.

Die Skalierbarkeit wird durch die Zulassung von einer beliebigen Anzahl Threads unterstützt. Für spezielle Aufgaben wie Fließkommaberechnungen können neue Befehle eingefügt werden.

Die virtuelle Maschine enthält einen portablen Mechanismus, um Bibliotheken, die in anderen Programmiersprachen entwickelt wurden, einzubinden. Über diesen Mechanismus werden viele Basistypen wie Fließkommazahlen mit unterschiedlichen Genauigkeiten oder Datums- und Uhrzeittypen angeboten.

Da in den Befehlen der virtuellen Maschine nicht zwischen statisch und dynamisch eingebundenen Bibliotheken unterschieden wird, ist die Maschine von Bindungsmechanismen der einzelnen Implementierungen unabhängig. Im Laufzeitsystem werden die Namen der externen Funktionen, die Liste der Parametertypen und die Bibliotheken, in denen sich die Funktionen befinden, gehalten. Der Interpretierer durchsucht zuerst eine Liste von Bibliotheken und Symbolen, die statisch gebunden sind. Wenn das Symbol nicht gefunden wird, versucht er über gemeinsame Bibliotheken auf Betriebssystemebene an das gesuchte Symbol zu kommen.

Durch die virtuelle Maschine wird der Zugriff auf flüchtige Daten in persistenten Systemen ermöglicht. Alle Konstruktionen und Destruktionen von Werten außerhalb des persistenten Objektspeichers werden dort protokolliert. So kann zum Beispiel ein Fenster einer grafischen Benutzeroberfläche nach dem Beenden und dem Neustart eines Programms automatisch wieder geöffnet werden. [MSS95]

2.2.3 Die Tycoon Machine Language (TML)

Für die Zwischensprache im Tycoon-System haben sich die folgenden Anforderungen ergeben: [Mat93]

- Sie soll als Zielcode für allgemeine Sprachen höherer Ordnung dienen und daher weitgehend unabhängig von speziellen Typaspekten dieser Sprachen sein.
- Sie wird zur Definition der dynamischen Semantik der Hochsprachen eingesetzt.
- Sie muß die Anwendung von Optimierungstechniken gestatten.
- Der Zwischensprachencode muß für reflektive Programmanalysen und -transformationen zur Verfügung stehen.

Bei einem Anwendungsprogramm, das mit dem Tycoon-System entwickelt wurde, wird viel Programmcode aus anderen Anwendungen wiederverwendet. Die Entscheidung, welche bereits übersetzten Codefragmente in das Programm eingebunden werden, wird oft bis zur Laufzeit verschoben. Dadurch kann das Programm zur Übersetzungszeit nicht effektiv optimiert werden. An diesem Punkt setzt die Zwischensprache ein, die für eine einheitliche Darstellung aller Programmkomponenten sorgt [GM96].

Dazu vereinigt die Zwischensprache im Tycoon-System sowohl die statische Analyse zur Übersetzungszeit als auch die Analyse zur Laufzeit mit der Optimierung. Als untypisierte Sprache, deren Semantik auf dem λ-Kalkül basiert, wird sie über die TML-Schnittstelle zur Implementierung mehrerer polymorph-typisierter Hochsprachen verwendet (siehe Abbildung 2.2).

Ein Übersetzer, der in TL implementiert ist, generiert die Bytecodesequenzen für die virtuelle Maschine. Durch die Darstellung von TML im Fortsetzungsübergabestil wird die Optimierung vereinfacht.

Die Optimierer und der Übersetzer von TML nach TVM überprüfen den Syntaxbaum des Programms nicht auf Korrektheit. Diese Aufgabe muß die Hochsprache erledigen.

2.2.4 Die Tycoon-Hochsprachen

Die Skalierung des Systems wird nach [MSS95] durch verschiedene Hochsprachen und eine feste Schnittstelle zur Zwischensprache unterstützt. Um eine weitere Datenbanksprache L in das System einzubauen, wird eine angepaßte Übersetzungsfunktion benutzt, die die

statische Semantik von L analysiert. Die dynamische Semantik wird dann durch eine strukturgerichtete Abbildung auf TML überprüft. Den TML-Kern kann man wiederum durch semantische Primitive, die für L benötigt werden, erweitern. Bisher wurden zwei Hochsprachen implementiert: TL und TooL. TL wird in Abschnitt 2.3 ausführlich beschrieben. Bei TooL handelt es sich um eine objektorientierte Sprache. Sie ist stark an Smalltalk angelehnt und benutzt statische Typisierung.

Die folgenden drei Aspekte, die für persistente Programmiersprachen wichtig sind, sind in den Hochsprachen implementiert.

Dynamische Typen: Falls die Typen von Objekten, die zum Beispiel über ein Netzwerk übertragen werden, zur Übersetzungszeit nicht bekannt sind, können sie dynamisch ermittelt werden. Solche dynamischen Werte bestehen aus einem Tupel mit dem Wert v und einer Laufzeitrepräsentation seines Typs t. Der Typ kann hier als Untertyp eines allgemeineren Obertyps oder durch Iteration über Wertattribute und Typattribute eines Tupels gewonnen werden. Es ist außerdem möglich, das Tupel aus einer Liste typisierter Bindungen zu gewinnen. Ein Einsatzbeispiel für dynamische Typen sind generische Funktionen, die ihr Verhalten von Typen abhängig machen. So kann ein Browser für den Objektspeicher alle Einträge, abhängig von ihren Typen, anzeigen.

Übersetzungszeit- und Laufzeitreflexion: Für die Implementierung der Sprachen TL und TooL wurde ein Übersetzer, der wiederum in der Spache TL entwickelt wurde, eingesetzt. Bei anderen Programmiersprachen wird ein Übersetzer als Werkzeug eingesetzt, um ein ablauffähiges Programm zu erhalten, das den Übersetzer anschließend nicht mehr benötigt. Im Tycoon-System können die Programme auf die Funktionalität des Übersetzers zurückgreifen. Zur Übersetzungszeit kann so auf Informationen aus dem Programmcode zurückgegriffen werden. Damit kann zum Beispiel generischer Code zum Durchlauf von rekursiven Datenstrukturen entwickelt werden.

Bei der Laufzeitreflexion werden vom Programm aus Komponenten des Übersetzers wie Zerteiler oder Codegeneratoren benutzt.

Typisierung externer Bindungen: Die Typen der Werte von externen Laufzeitbibliotheken liegen nur im Quellcode der Bibliotheken vor. Bei der Übersetzung der Bibliotheken gehen die Typen verloren. Wenn solche Bibliotheken an das System gebunden werden, sind sie nicht sicher typisiert. Um eine sichere Typisierung zu erreichen, müssen bei der Bindung externer Bibliotheksfunktionen die Typen der Funktionsparameter mit angegeben werden. Somit liegt die Typisierung in den Händen des Programmierers.

2.2.5 Erweiterbare Grammatiken

Die Abbildung von der konkreten auf die abstrakte Syntax des Tycoon-Systems ist nicht durch die Sprachdefinition vorgegeben. Durch dieses Konzept können die Sprachen leicht erweitert werden.

Erweiterbare Grammatiken sind durch eine Bibliothek von Übersetzerwerkzeugen in der Sprache TL implementiert. Durch die Angabe von Produktionen im Programmtext können so neue Sprachkonstrukte generiert werden, die dem Benutzer zum Beispiel als Abfragesprache für selbstentwickelte Datenbankanwendungen zur Verfügung stehen [MSW95].

2.3 Die Programmiersprache TL

Die Programmiersprache TL wird zunächst an Hand ihrer Eigenschaften beschrieben. Danach wird auf die Bedeutung von Typen und die Möglichkeiten der Subtypisierung eingegangen. Zum Schluß werden Typausdrücke und –operatoren vorgestellt.

2.3.1 Die Eigenschaften von TL

Das Modell, mit dem die Sprache TL definiert ist, beruht auf den Konzepten von Werten, Ausdrücken, Signaturen und Bindungen. Daher hat TL die folgenden Eigenschaften [Mat95, Mat93]:

TL ist algorithmisch vollständig: Zusätzlich zur Turing-Vollständigkeit bietet TL auch adäquate Kontroll- und Datenabstraktionen, um den hohen Anforderungen der Entwicklung von Datenbanksystemen gerecht zu werden.

TL ist funktional: Die Semantik von TL basiert auf dem λ-Kalkül. Jeder TL-Term wird zu einem Wert ausgewertet. Die zentralen Konzepte von TL sind Funktionsabstraktionen und Funktionsanwendungen. Funktionsabstraktionen bestehen aus einer Liste von Signaturen, die die Parameter für die Funktion bilden, und einem Ausdruck, der die Berechnungsvorschrift für den Funktionswert darstellt, und werden zu einem Funktionswert ausgewertet. Eine Funktionsanwendung definiert einen Funktionswert und eine geordnete Liste von Bindungen, die sich aus der Funktionssignatur ergeben. Der Funktionsterm wird in der Umgebung mit den Funktionsargumenten ausgewertet. Funktionen besitzen als Objekte *erster Klasse* eine Identität und können zum Beispiel in Datenbanken eingefügt werden.

TL ist imperativ: TL ist keine rein funktionale Sprache, da Objektidentifikatoren, destruktive Zuweisungen, Seiteneffekte in Funktionen, Schleifen und Ausnahmen erlaubt sind. Dadurch wird die zustandsbasierte transaktionale Datenbankprogrammierung unterstützt.

TL ist strikt evaluativ: Wenn Terme als Funktionsargumente übergeben werden, so werden diese Terme vor der Applikation ausgewertet. TL benutzt als Parameterübergabe die Call-By-Value-Strategie. Ausnahmen von dieser Regel bilden nur die bedingten Verzweigungen ***if***, ***orif***, ***andif*** und ***case***. Sie werden nach der Strategie der faulen Auswertung (engl.: lazy evaluation) evaluiert.

TL ist deterministisch: Unterterme von zusammengesetzten Termen werden in einer festen Reihenfolge ausgewertet. Es gibt auch keinen expliziten nil-Wert, der zu unerwartetem Programmverhalten führen kann. Mit der strikten und deterministischem Auswertung von Termen wird die Spezifikation und Verifikation der Programme vereinfacht.

TL besitzt eine Referenzsemantik: Mit Hilfe von statischen *let*-Bindungen, dynamischen Funktionsargumenten und destruktiven Zuweisungen von strukturierten Werten können mehrere Zugriffspfade auf den gleichen Wert eingerichtet werden. Mit dieser Referenzsemantik wird außerdem die Manipulation von rekursiven Datenstrukturen vereinfacht.

TL hat statische Sichtbarkeitsregeln: Durch die statischen und blockstrukturierten Sichtbarkeitsregeln kann die Deklaration eines Bezeichners mit einer textuellen Analyse des Quellprogramms gefunden werden. Mit Hilfe von Funktionsabstraktionen können Werte gekapselt werden.

TL ist nebenläufig: Mit dem Thread-Konzept, das in Abschnitt 2.4.3 vorgestellt wird, können mehrere Programme gleichzeitig denselben Objektspeicher benutzen. Threads werden durch Synchronisationsmechanismen unterstützt. Sie sind Objekte erster Klasse.

TL ist persistent: Die Lebensdauer eines TL-Threads ist nicht an die Lebensdauer des Prozesses vom Betriebssystem gekoppelt. Die Threads sind persistent und mobil. Alle Werte sind automatisch persistent. Die Auswertung eines TL-Terms kann sich auf Zustandsvariablen in einem prä-populierten Objektspeicher beziehen und Seiteneffekte im Objektspeicher erzeugen. TL-Programmierer können von der Speicherverwaltung vollständig abstrahieren. Speicherbereinigung, Datenübertragung auf Plattenspeicher und Zugriffskontrolle bei mehreren Benutzern werden vom Objektspeicher automatisch erledigt. Außerdem sind Commit- und Rollback-Aufrufe möglich. Die Persistenz der Objekte ist nicht von ihrem Typ abhängig, so daß man von orthogonaler Persistenz [AB87] sprechen kann.

TL ist strikt typisiert: TL definiert Konsistenzbedingungen an Terme, die durch Typregeln formalisiert sind. Die Einhaltung dieser Konsistenzbedingungen wird für beliebige Terme durch eine statische Programmanalyse erzwungen. Durch explizite Sprachkonstrukte wie dynamische Typen oder Module kann dieser Konsistenztest jedoch kontolliert auf wohldefinierte Zeitpunkte zur Programmlaufzeit verschoben werden.

TL ist explizit typisiert: In TL können Wert- und Typbezeichner jederzeit mit Typinformationen versehen werden. Bezeichner, die dynamischen oder rekursiven Bindungen unterliegen (zum Beispiel Funktionsparameter) müssen mit diesen Typinformationen versehen werden, bei anderen Bezeichnern werden sie automatisch vom Übersetzer inferiert.

TL ist typvollständig: Die Wertkonstruktoren von TL können einheitlich auf Werte aller Typen angewandt werden. Die Komponenten von Tupeln, Records oder Vektoren können über ihren Namen oder ihre Stelle selektiert werden.

TL ist polymorph: Konzepte zur existentiellen und universellen Typquantifizierung und eine induktiv definierte Subtypbeziehung auf Typen in TL ermöglichen parametrischen Polymorphismus, Subtyppolymorphismus und begrenzten parametrischen Polymorphismus (engl.: bounded parametric polymorphism). Benutzerdefinierte Typoperatoren höherer Ordnung ermöglichen neuartige Formen der polymorphen Programmierung.

TL ist interaktiv: Die Implementierung von TL-Programmen erlaubt einen fließenden Übergang zwischen stapelorientierter Übersetzung, Bindung und Ausführung von Modulen und Bibliotheken und der direkten Auswertung interaktiv eingegebener TL-Terme. So kann man TL sowohl zur Datenbankprogrammierung als auch für ad-hoc-Datenbankanfragen verwenden.

TL ist modular: Anwendungsprogramme sind in separat entwickelte Module und Bibliotheken gegliedert, deren Strukturen und Importabhängigkeiten explizit in TL dargestellt werden.

2.3.2 Die Bedeutung von Typen und Subtypen in TL

TL ist nach Abschnitt 2.3.1 eine explizit typisierte Sprache. Die Zusammenhänge zwischen Typkompatibilität und Subtypbeziehungen geben eine Antwort auf die Frage, wann ein Ausdruck a den Typ A besitzt.

Dabei ist es zunächst wichtig zu wissen, daß eine Typsignatur für einen Wert nur eine partielle Spezifikation darstellt. Der Wert hat mindestens den Typ A, er kann aber auch präziser sein. Die Ordnung von Typen wird durch die Subtypbeziehung beschrieben. Dabei ist ein Typ B präziser als ein Typ A, wenn B ein Subtyp von A ist.

Das Konzept der Subtypisierung leistet einen wesentlichen Beitrag zur Erweiterbarkeit von Softwaresystemen. Bestehende Komponenten können für Objekte mit präziseren Typen unverändert übernommen werden. Außerdem wird durch Subtypbeziehungen die Entwicklung generischer Algorithmen, die von irrelevanten Typdetails abstrahieren, vereinfacht. Die Subsumptionsregel sagt aus, daß jeder Wert eines Typs auch ein Wert des Supertyps ist. Der allgemeinste Typ ist ***Ok***.

Bei Tupeltypen besteht folgende Subtypbeziehung: Ein Tupeltyp A ist Subtyp eines Tupeltyps B wenn er mindestens die gleichen Komponenten an der gleichen Stelle besitzt. Es können weitere Komponenten hinzukommen. Außerdem müssen die Komponententypen von A Subtypen der Komponententypen von B sein. Wenn weiterhin Variablennamen für die Komponenten benutzt werden, so müssen die Variablen des Supertyps die gleichen Namen wie im Subtyp haben. Die Subtypisierung beruht also auf struktureller Kompatibiltät.

Bei Funktionstypen führt die Subtypisierung zur Kontravarianzregel: ein Funktionstyp A mit Parametertypen S und einem Ergebnistyp E ist Subtyp eines Funktionstyps A', wenn seine Parametertypen S' Supertypen von S sind und sein Ergebnistyp E' ein Subtyp von E ist. Damit können Funktionen typsicher spezialisiert werden.

2.3.3 Typen von Typen

In TL können die Typen wie Wertausdrücke und Wertvariablen verwendet werden. Dadurch wird eine größere sprachliche Mächtigkeit und Flexibilität erreicht. Mit diesem Ansatz werden Sprachkonzepte wie generische Datentypen, abstrakte Datentypen und polymorphe Funktionen erfaßt.

Wenn Typvariablen in Signaturen statisch gebunden werden, erhält man eine existentielle Quantifizierung von Typausdrücken. Mit dieser Konstruktion können abstrakte Datentypen leicht in eigenen Modulen implementiert werden, die erst bei der Benutzung durch den Aufruf mit einem konkreten Typ versehen werden [MS93].

Die universelle Quantifizierung von Typausdrücken führt zu polymorphen Funktionen, Typoperatoren oder dynamischen Typen, je nach Kontext. Sie kann zum Beispiel dazu eingesetzt werden, allgemeine Sortierfunktionen zu implementieren, die Werte aller Typen sortieren kann. Sie erhält beim Aufruf den Typ der zu sortierenden Werte und eine Vergleichsfunktion.

In polymorphen Funktionen werden Typvariablen als Parameter benutzt, um den Ergebnistyp abhängig davon zu berechnen. Wenn die Typvariable durch die Angabe eines speziellen Supertyps (spezieller als ***Ok***) eingeschränkt wird, spricht man vom Prinzip des begrenzten parametrischen Polymorphismus.

Typoperatoren sind Funktionen, die Typen auf Typen abbilden. Sie erlauben eine Anpassung der Sprache an verschiedene Konzepte zur Datenstrukturierung und -modellierung. Ein Beispiel für den Einsatz von Typoperatoren ist die Implementierung von Nullwerten.

Zu dynamischen Typen siehe Abschnitt 2.2.4.

2.4 Kommunikation zwischen Tycoon-Systemen

Die Kommunikationsbibliothek im Tycoon-System ist in Schichten implementiert. Dabei bestehen die unteren Schichten aus einfachen Kanälen für Byte-Ströme, auf denen typunsichere Kanäle für TL-Objekte aufbauen. Diese werden wiederum zu typsicheren entfernten Prozeduraufrufen (Remote Procedure Calls, RPC) erweitert. Diese Prozeduraufrufe beruhen auf einfacher Adressierung, wobei in der nächsten Schicht mobilitätsorientierte RPCs realisiert sind. Damit wird die Adressierung der Dienste für den Benutzer transparent. Aus der Orthogonalität der Typen im Tycoon-System können mit den mobilitätsorientierten RPCs auch ganze Programme in Ausführung (Threads) übertragen werden.

2.4.1 Typunsichere Kommunikation

Die Grundlage für Kommunikationsdienste im Tycoon-System bildet ein typunsicherer Dienst, mit dem beliebige Tycoon-Objekte aus dem Objektspeicher im Internet übertragen werden können. Für diesen Dienst, bei dem der Sender ein Objekt an eine Netzwerkadresse abschickt und der Empfänger bereits auf die Daten warten muß, wird die linearisierte Objektdarstellung TXR aus Abschnitt 2.2.1 benutzt. Es ist auch eine bidirektionale Übertragung möglich, bei der der Sender solange blockiert, bis die Antwort vom Empfänger wieder eintrifft. Dieser Dienst ist deshalb typunsicher, weil er sich eng an die Socket-Programmierung in C anlehnt und keine Typinformationen aus TL benutzen kann.

2.4.2 Entfernte Prozeduraufrufe

Um zu typsicheren Diensten zu gelangen, müssen die Operationen von oben in die Programmiersprache TL eingebettet werden. Um eine TL-Funktion als entfernten Dienst zur Verfügung zu stellen, muß sie in ein entsprechendes Tupel eingebunden werden. Auch auf Seite der Klienten muß die Funktion für die Benutzung eines entfernten Dienstes nicht neu übersetzt werden. Die Kommunikationsunterprogramme werden dabei durch Manipulation der Laufzeitwerte zusammengesetzt.

Ein Dispatcher verwaltet Funktionen, die als entfernte Dienste zur Verfügung stehen sollen. Wenn man einen Dienst einträgt, wird sein dynamischer Typ mit angegeben. Auf der Seite des Klienten muß für einen entfernten Aufruf die Funktion mit ihrem Typ an einen Namen gebunden werden. Bei der Bindung wird — für den Benutzer transparent — im Dispatcher nachgeschaut, ob ein entsprechender Funktionstyp vorhanden ist. Wenn die Anfrage erfolgreich ist, wird der untypisierte Klient nachträglich typisiert. Wenn die Anfrage keine Übereinstimmung ergibt, wird ein Fehler erzeugt.

Der nächste Schritt zu einer komfortablen Client/Server-Umgebung ist ein hierarchisches Adressierungsschema. Mit ihm können die angebotenen Dienste auch ihre Netzwerkadresse, die für die typsichere Kommunikation zwingend notwendig ist, wechseln. Eine übergeordnete Adressierungschicht teilt jetzt die Adressen dynamisch zu. In dieser Schicht müssen zwei Aufgaben getrennt voneinander behandelt werden: die Dienstauswahl und der entfernte Aufruf.

Der entfernte Aufruf wird mittels zweier Hilfsprogramme durchgeführt: Der Mapping Daemon (Mad) verwaltet die Kommunikationskanäle für die verschiedenen Dienste auf einem Host-Rechner. Wenn der Mad keinen passenden Dienst findet, wird die Anfrage an die nächsthöhere Instanz im Netzwerk weitergegeben, den Tycoon LAN Daemon (TyLANd). Er durchsucht alle anderen Mads im Netzwerk und gibt das Ergebnis der Suche an den anfragenden Mad zurück, der den anfragenden Klienten informiert. Die Suche nach migrierten Diensten funktioniert auch in einem entfernten LAN (Local Area Network), das über ein WAN (Wide Area Network) mit dem Klienten verbunden ist, weil nur der Ausgangspunkt für die Suche ausschlaggebend ist. Um alle denkbaren Fälle abzudecken, kann jedem Dienst eine eigene Suchfunktion zugeordnet werden. Diese Funktion wird im Dispatcher eingetragen.

Die Dienstauswahl, die dem entfernten Aufruf vorausgeht, beruht prinzipiell auf dem gleichen Mechanismus wie die Bindung typsicherer Dienste. Zusätzlich wird der *bind*-Funktion ein Parameter für die Suchtiefe mitgegeben. Damit kann bestimmt werden, ob die Suche auf den Host, das LAN oder größere Regionen des Internet beschränkt werden soll. Außerdem kann die Suche durch die Angabe von Attributen und Prädikaten für die Attribute geleitet werden.

2.4.3 Mobile Threads

Unter Multi-Threading versteht man die Möglichkeit, in einem Speicherbereich mehrere nebenläufige Programmausführungen zu halten. Sie unterscheiden sich von Betriebssystemprozessen durch den gemeinsamen Speicherbereich und einen geringeren Ressourcenverbrauch. Sie werden daher auch Leichtgewichtprozesse genannt.

Im Tycoon-System sind Threads orthogonal in das sprachliche Gefüge von TL integriert. Dadurch können sie ohne syntaktische Besonderheiten wie alle anderen Werte behandelt werden. Man kann sie an Funktionen übergeben und sie von ihnen zurückbekommen, sie können auf Identität verglichen werden, in komplexe Datentypen wie Tupel oder Arrays eingefügt werden oder Konstanten und Variablen zugewiesen werden.

Threads werden im Programmablauf mit einer vorgegeben Funktion gestartet, können gelöscht werden, sich auch selbst löschen oder mit anderen Threads synchronisieren. Bei der Synchronisation geben die Threads Werte an den wartenden Thread zurück. Wenn ein Thread auf einen anderen Thread wartet, wird seine Ausführung bis zum Eintreffen des Ergebnisses blockiert.

Mit diesem Konzept können zum Beispiel leicht Programme erstellt werden, die im Hintergrund verharren und bei bestimmten Anlässen aktiviert werden.

Durch die Einbettung in ein persistentes Objektsystem braucht sich der Programmierer oder Benutzer über die Dauerhaftigkeit seiner Anwendungen keine Gedanken zu machen. Wenn der Objektspeicher gesichert wird, werden auch alle Threads in ihm gesichert. Zusammen mit der Kommunikation von Tycoon-Systemen, die im Abschnitt 2.4.2 näher beschrieben ist, kommt man zu mobilen Threads, die sich für den Benutzer nicht von stationären Threads unterscheiden. Weil sich Threads in Tycoon reflexiv auf sich selbst beziehen können, können sie sich, wie im Agentenparadigma [Whi94] beschrieben, auch selbst verschicken. Mit der erweiterbaren Grammatik wurden dafür Befehle wie ***migrate to ... do*** realisiert.

Intern werden die Threads durch Objekte im Objektspeicher dargestellt. Dadurch sind sie von einer bestimmten Hardware-Umgebung abgekoppelt. Als untypisierte TSP-Objekte gleichen sie allen anderen TL-Werten und können damit unterschiedslos linearisiert, in Netzwerken übertragen und als RPC-Argumente akzeptiert werden (siehe Abschnitt 2.4.2). Damit erhält man mobile Threads. Bei der Entwicklung von Threads wurden möglichst wenige Objektreferenzen benutzt. Damit wird die Mobilität unterstützt, weil weniger Objekte übertragen werden müssen. [Mat96]

2.5 Module und Bibliotheken in Tycoon

Um die Skalierbarkeit von Programmen zu unterstützen, sind viele Dienste in Bibliotheken ausgelagert, die bei Bedarf wieder importiert werden können [MMM93]. In den Bibliotheken werden Module und Schnittstellen zusammengefaßt, die die geforderte Funktionalität liefern. Sie müssen in der Reihenfolge angegeben werden, in der sie sich aufeinander beziehen. Wenn ein Modul b auf einem Modul a aufbaut, so muß a in der Bibliothek vor b importiert werden. Bibliotheken können andere Bibliotheken, Schnittstellen und Module enthalten [MMS94]. Alle in einer Bibliothek deklarierten Komponenten werden exportiert und können vom Benutzer interaktiv aufgerufen werden.

Um ein Programm modular zu zerlegen, muß zunächst eine Hauptbibliothek erstellt werden. Sie wird anschließend übersetzt. Dadurch sind die Abhängigkeiten zwischen den Modulen und Schnittstellen sichtbar und für die weitere Übersetzung nutzbar. Dann kann das Hauptmodul übersetzt werden. Dabei werden alle Abhängigkeiten aufgelöst und die Module, die nach der letzten Übersetzung geändert wurden, neu übersetzt. Mit *import*-Befehlen von der interaktiven Benutzereingabe werden die exportierten Funktionen und Werte in die interaktive Benutzerumgebung geladen und können vom Benutzer ausgeführt werden.

In Schnittstellen werden die Signaturen der Werte und Typen eines Moduls, die nach außen sichtbar gemacht werden sollen, definiert. Im Modul befinden sich dann die Implementierungen zu diesen Signaturen. Man kann Module oder Schnittstellen, die bei der Definition benutzt werden, aus anderen Modulen oder Schnittstellen importieren.

2.6 Ein praktisches Beispiel

Atkinson und Buneman arbeiten in ihrem Artikel „Types and Persistence in Database Programming Languages“ [AB87] notwendige Eigenschaften von Datenbankprogrammiersprachen heraus. Sie diskutieren die Prinzipien *Persistenz*, *Typvollständigkeit* und *Ausdruckstärke* am Beispiel verschiedener Programmiersprachen. In jeder Sprache sollen verschiedene Programme entwickelt werden, die eine kleine Datenbankanwendung darstellen. In diesen Programmen geht es um die Datenbank einer Produktionsfirma, die alle von der Firma verwendeten Bauteile verwalten soll. Es gibt Grundbauteile, die von anderen Herstellern eingekauft werden, und zusammengesetzte Bauteile, die aus den Grundbauteilen oder anderen zusammengesetzten Bauteilen produziert werden. Für jedes Bauteil wird der Name und das Gewicht gespeichert, bei eingekauften Bauteilen der Einkaufspreis und der Lieferant, bei zusammengesetzten Bauteilen die Baukosten und alle benötigten Bauteile.

Die folgenden vier Aufgaben wurden mit allen Sprachen angegangen:

1. Beschreibung der Datenbank.

   In diesem Teil soll untersucht werden, welche zusätzlichen Informationen wie zum Beispiel Bedingungen an die Daten in das Schema der Datenbank gesteckt werden können.

2. Ausgabe der Namen, Preise und Gewichte aller eingekauften Bauteile, die mehr als 100 DM kosten.

   Mit dieser Aufgabe wird geprüft, ob sich einfache Anfragen an eine Datenbank auch einfach ausdrücken lassen.

3. Ausgabe des Gesamtgewichts und der Gesamtkosten eines zusammengesetzten Bauteils.

   Diese anspruchsvolle Anfrage erfordert Rekursionen, die in einfachen Abfragesprachen wie SQL nicht möglich sind.

4. Aufnahme eines neuen Produktionsprozesses in die Datenbank, d.h. Aufnahme eines neuen zusammengesetzten Bauteils, daß aus anderen Bauteilen hergestellt wird.

   Mit dieser Anfrage wird die Implementierung von Integritätsbedingungen gemessen.

Auf den Vergleich der Sprachen gehe ich hier nicht weiter ein, es soll lediglich anhand von Programmbeispielen gezeigt werden, wie diese Aufgabe mit dem Tycoon-System gelöst werden kann. Dabei wird der Programmcode ausführlich beschrieben, damit der Leser einen Eindruck von der Sprache TL bekommt.

Mit dem Programmfragment aus Abbildung 2.3 wird ein Datenbankschema definiert und die Datenbank aus *parts, supplier* und *components* erstellt. Alle Wörter im Programmtext, die mit Großbuchstaben beginnen, beziehen sich auf Typen. So werden in den ersten drei Abschnitten Datentypen für Bauteile, Komponenten und Hersteller definiert. Diese Typen sind mit ***Rec*** und ***and*** gekennzeichnet, so daß sie parallel gebunden werden. Alle Datentypen bestehen aus einem Tupeltyp, der verschiedene Attribute enthält. Der erste Typ ist ein varianter Tupel, der zum gemeinsamen Attribut *name* vom Basistyp String (der Typ zum Attribut wird syntaktisch mit dem Doppelpunkt eingeleitet) die Varianten Grundbauteil oder zusammengesetztes Bauteil enthält. Ein Grundbauteil hat die Attribute *cost, mass* und *supplier*. Ein zusammengesetztes Bauteil enthält die Attribute *assemblyCost, massIncrement* und *madeFrom* als Iteration von Komponenten, aus denen es zusammengesetzt wird.

Die Komponenten und Hersteller werden durch einfache Tupeltypen beschrieben. Ein Tupel für Komponenten enthält Attribute für die Menge und das Bauteil, das eingebaut wird. Der Hersteller wird bisher nur durch seinen Namen beschrieben.

Wegen der Rekursion der Datentypen müssen sie als Subtypen des allgemeinsten Datentyps ***Ok*** gekennzeichnet werden. Dies geschieht durch den Zusatz <:***Ok*** [Mat93].

Die Datenbank selbst besteht aus Iterationen von Variablen der eben definierten Typen. Dazu muß zuvor das Modul *iter* aus der Standardbibliothek in die interaktive Benutzerumgebung importiert werden. Die Funktionen können dann mit der Punktnotation *iter.new(...)* angesprochen werden. Mit dem Schlüsselwort ***var*** werden sie als veränderbar gekennzeichnet. Das Semikolon am Ende schließt die interaktive Eingabe ab und übergibt den Programmtext an den Übersetzer.

Im Programm aus Abbildung 2.4 steht der Programmcode für die zweite Aufgabe. Diese Aufgabe ist mit der mächtigen Iterations-Bibliothek des Tycoon-Systems leicht zu lösen. Da

```
(* Task 1: Describe the database *)

    Let Rec Part <:Ok =
      Tuple
        name :String
        case basePart with
          cost :Int
          mass :Real
          supplier :Supplier
        case compositePart with
          assemblyCost :Int
          massIncrement :Real
          madeFrom :iter.T(Component)
      end

    and Component <:Ok = Tuple quantity :Int subpart :Part end

    and Supplier <:Ok = Tuple name :String end;

    let var parts = iter.new(:Part);

    let var supplier = iter.new(:Supplier);

    let var components = iter.new(:Component);
```

Abbildung 2.3: TL-Code zur Beschreibung der Datenbank

alle Bauteile der Datenbank in einer Iteration vorliegen, kann man mit *iter.select(...)* leicht eine neue Iteration erstellen, in der sich nur ausgewählte Elemente der Ausgangsiteration befinden. In diesem Fall werden sie mit einer booleschen Funktion, die ***true*** zurückgibt, falls das Bauteil ein Grundbauteil ist und sein Preis mehr als 100 DM beträgt, bestimmt. Auf die Elemente des Tupels wird mit dem Variantentest *parts?basePart* und der Variantenprojektion *parts!basePart* zugegriffen. Wenn alle gesuchten Elemente in der neue Iteration enthalten sind, müssen sie ausgegeben werden. Dazu wird die Funktion *iter.forEach(...)* verwendet, die für jedes Element der Iteration die Ausgabefunktion aufruft.

Die Ausgabefunktion *printBaseNameCostMass(...)* kapselt die Ausgabe der Attribute des Grundbauteils. Hier wird die Funktion *print.string(...)* an einen neuen Namen mit einem Parameter gebunden. Der Funktion *print.string(...)* wird wiederum eine Zeichenkette übergeben, die sich aus mehreren Strings, die mit <> verkettet werden, zusammensetzt. Variablen, die nicht den Typ *String* haben, müssen mit Formatierungsfunktionen in diesen

```
(* Task 2: Print all the names, cost, an mass of all imported parts,
                that cost more than 100 Dollars *)

    let printBaseNameCostMass (part :Part) :Ok =
      print.string ("Part " <> part!basePart.name <> " "
        <> "costs " <> fmt.int(part!basePart.cost) <> " "
        <> "and its mass is " <> fmt.real(part!basePart.mass) <> "\ n")

    let expensiveBaseParts = iter.select(parts fun(p :Part) :Bool
      p?basePart andif p!basePart.cost > 100)

    iter.forEach(expensiveBaseParts fun(p :Part)
      printBaseNameCostMass(p));
```

Abbildung 2.4: Programmcode zur zweiten Aufgabe mit TL

Typ überführt werden.

In Abbildung 2.5 steht der Programmtext, um die Kosten und das Gewicht eines zusammengesetzen Bauteils zu berechnen. Kernfunktion ist die rekursive Funktion *addMassCost(...)*, die mit der Häufigkeit des Bauteils und dem Bauteil selbst aufgerufen wird und ein Tupel mit Preis und Gewicht zurückgibt. Falls es sich bei dem übergebenen Bauteil um ein Grundbauteil handelt, ist die Rekursion beendet. Der Preis und das Gewicht wird mit der Anzahl verrechnet und als Tupel zurückgegeben. Für ein zusammengesetztes Bauteil werden lokale Zähler für das Gewicht und den Preis angelegt und die Iteration von Bauteilen durchlaufen. Mit der Funktion *iter.forEach(...)* wird für jedes Element der Iteration eine namenlose Funktion aufgerufen. In dieser Funktion wird der Rekursionsschritt ausgeführt und die Zähler für Preis und Gewicht werden aktualisiert. Am Ende werden die Baukosten und die Gewichtsveränderung, die durch die Montage entsteht, dazu addiert. Das Tupel mit dem Preis und dem Gewicht an die aufrufende Umgebung zurückgegeben.

Damit der Benutzer das Bauteil nicht nur über die Stelle in der Iteration ansprechen kann, wird es vom Benutzer mit der Funktion *findPart(...)* anhand seines Namens gesucht. Zuerst wird eine globale Ausnahme definiert, die aktiviert wird, falls das Bauteil nicht in der Datenbank gefunden wird. Die Ausnahme ist global, weil die Funktion *findPart(...)* auch im Programm zur vierten Aufgabe verwendet wird. Innerhalb von *findPart(...)* wird mit der Funktion *iter.select(...)* eine lokale Iteration erzeugt, die nur die Elemente aus *parts* enthält, für die die Funktion *string.equal(...)* den Wert *true* zurückgibt. Aus dieser Iteration wird das erste Element an die aufrufende Umgebung zurückgegeben. Falls die Funktion *iter.head(...)* auf eine leere Iteration zugreift, wird eine Ausnahme ausgelöst, die mit dem ***try***-Konstrukt abgefangen wird. Wenn der Versuch ein Element auszuwählen fehlschlägt, wird die vom Programmierer bestimmte Ausnahme *notFound* ausgelöst, die den

```
(* Task 3: Print the total mass and total cost of a composite part *)

    Let MassCost = Tuple mass :Real cost :Int end;

    let rec addMassCost (quantity :Int part :Part) :MassCost =
      begin
        case of part
        when basePart with bp then
          let mass = real.mul(bp.mass real.val(quantity))
          let cost = bp.cost * quantity
          tuple mass cost end
        when compositePart with cp then
          let var mass = 0.0
          let var cost = 0
          iter.forEach(cp.madeFrom
            fun(sPart: Component)
              begin
                let mc = addMassCost(sPart.quantity sPart.subpart)
                mass := real.add(mass real.mul(real.val(quantity) mc.mass))
                cost := cost + mc.cost * quantity
              end)
          mass := real.add(mass real.mul(real.val(quantity) cp.massIncrement))
          cost := cost + cp.assemblyCost * quantity
          tuple mass cost end
        end
      end;

    let notFound = exception "Element not found\n" end;

    let findPart(name :String parts :iter.T(Part)) :Part =
      try
        iter.head(iter.select(parts fun(p :Part) string.equal(name p.name)))
      else
        raise notFound end
      end;
```

Abbildung 2.5: Das TL-Programm zur Untersuchung eines zusammengesetzten Bauteils

Benutzer darüber informiert, daß das gesuchte Element nicht in der Datenbank enthalten ist.

Durch Rekursion und komfortable Iterationsfunktionen, die alle Bauteile für ein zusammengesetztes Bauteil erfassen, läßt sich auch die dritte Aufgabe leicht bearbeiten. Weil aber in TL keine Überladung zugelassen ist, müssen die mathematischen Operatoren für Real-Werte umständlich ausformuliert werden. Jedoch können diese Formulierungen mit einer eigenen erweiterten Grammatik abgekürzt werden.

Im Programm zur letzten Aufgabe in Abbildung 2.6 wird ein neues zusammengesetztes Bauteil in die Datenbank eingefügt. Vor dem Aufruf der Funktion müssen alle verwendeten Bauteile mit ihrem Namen und ihren Anzahlen in einer Iteration festgelegt werden. Außerdem erwartet die Funktion *addCompositePart(...)* das einzufügende Bauteil ohne Komponentenliste und die Datenbank aller Bauteile. Sie gibt den Wert ***ok*** zurück, erzeugt also nur Seiteneffekte auf der Datenbank. Bevor das neue Bauteil eingefügt werden kann, muß die Komponenteniteration aus der Iteration über Namen und Mengen erzeugt werden. An dieser Stelle werden auch die Integritätsbedingungen der Datenbank überprüft. Alle angegebenen Bauteile müssen schon in der Datenbank enthalten sein. Wenn ein Bauteil fehlt, wird das neue Bauteil nicht eingefügt und eine Ausnahme erzeugt. Die Liste mit Namen wird wieder mit *iter.forEach(...)* durchlaufen. Dabei wird für jedes Element versucht, es in der Datenbank zu finden. Bei solchen unsicheren Funktionen wird das ***try***-Konstrukt verwendet. Falls in der *findPart(...)*-Funktion eine Ausnahme erzeugt wird, wird sie mit dem ***when***-Fall abgefangen und es wird eine Meldung an den Benutzer ausgegeben. Wenn *findPart(...)* ein Bauteil zurückgibt, wird es mit *iter.cons(...)* in die Komponenteniteration eingefügt. Am Ende wird die Komponenteniteration in das Tupel mit dem aktuellen Bauteil übernommen und dieses Bauteil in die Datenbank eingefügt.

Die Überprüfung der Integrität der Datenbank kostet an dieser Stelle einige Zeilen Programmtext, läßt sich aber mit der Ausnahmebehandlung leicht programmieren.

Insgesamt kann man an den vier Beispielen sehen, daß mit dem iter-Modul ein mächtiges Werkzeug für die Behandlung von Massendaten geschaffen wurde.

```
(* Task 4: Record a new manufacturing step in the database, that is, how a new
     composite part is manufactured from subparts. *)

   Let NameQuantity = Tuple name :String quantity :Int end

   let addCompositePart(part :Part
      componentNames :iter.T(NameQuantity)
      var parts :iter.T(Part)) :Ok =
         begin
            let var thisComponents = iter.new(:Component)
            iter.forEach(componentNames
               fun(component :NameQuantity)
                  begin
                     try
                        let subPart = findPart(component.name parts)
                        thisComponents := iter.cons(tuple
                              component.quantity
                              subPart
                           end
                           thisComponents)
                     when notFound then
                        print.string ("Part " <> component.name
                           <> " is not in the database\n")
                     end
                  end)
            parts := iter.cons(tuple_case
               compositePart of Part with
                  part.name
                  part!compositePart.assemblyCost
                  part!compositePart.massIncrement
                  thisComponents
               end
               parts)
         end;
```

Abbildung 2.6: Das TL-Programm zum Einfügen eines zusammengesetzen Bauteils

Kapitel 3

Realisierung von Informationssystemen mit verteilten Objekten

In diesem Kapitel sollen drei verschiedene, kommerziell relevante Ansätze zur Realisierung von Informationssystemen mit verteilten Objekten vorgestellt werden. Dabei gehen alle Ansätze von verschiedenen Ausgangspunkten aus und verfolgen unterschiedliche Strategien. CORBA will eine möglichst umfassende und allgemeine Vermittlung von Diensten in Netzen zur Verfügung stellen. Das DCE stellt eine Reihe von Softwarekomponenten bereit, um die Erstellung verteilter Anwendungsprogramme auf heterogenen Rechnernetzen zu erleichtern. OLE ist ein Ansatz, der bisher auf Betriebssystemebene implementiert ist und dokumentenzentriertes Arbeiten ermöglicht. Jedes System wird mit Tycoon verglichen.

3.1 CORBA: Unabhängige Objektverteilungen

Die Common Object Request Broker Architecture (CORBA) ist die Antwort der Object Management Group (OMG) auf die Frage nach der Zusammenarbeit einer ständig wachsenden Anzahl von Hardware- und Softwarekomponenten. Kurz zusammengefaßt soll CORBA die Kommunikation zwischen Applikationen, bei denen es keine Rolle spielt, wo sie stehen oder wer sie erstellt hat, erlauben. Dazu hat die OMG 1991 CORBA 1.1 als Standard definiert. In dieser Version wurden eine Interface Definition Laguage (IDL) und Application Programming Interfaces (API) definiert, die eine Zusammenarbeit von Objekten innerhalb einer speziellen Implementierung eines Object Request Broker (ORB) ermöglichen. Mit der Version 2.0 wurde die Zusammenarbeit verschiedener ORB-Implementierungen hinzugenommen.

Mit dem ORB werden Client-Server-Beziehungen zwischen Objekten etabliert. Mit Hilfe des ORB kann ein Klient transparent eine Methode eines anderen Objektes aktivieren, die entweder auf dem gleichen Rechner oder irgendwo im Netzwerk liegen kann. Der ORB fängt die Nachricht ab und ist für die Suche des Objektes, das die geforderten Dienste

erfüllt, für die Übertragung der Parameter, den Aufruf der Methode und die Rückgabe der Ergebnisse zuständig. Dabei braucht der Klient nichts über den Ort des Objektes, seine Programmiersprache, sein Betriebssystem oder andere Systemaspekte, die nicht Teil der Schnittstelle sind, zu wissen. Damit bewirkt der ORB die Zusammenarbeit von Applikationen auf verschiedenen Rechnern in heterogenen, verteilten Umgebungen und verbindet verschiedene Objektsysteme nahtlos.

In bisherigen Client-Server-Applikationen haben die Entwickler eigene Ansätze für die Kommunikation zwischen den Komponenten benutzt. Dabei hingen die Protokolldefinitionen von der benutzten Programmiersprache, dem Netzwerk und vielen anderen Faktoren ab. Dieser Enwurfsprozeß wird mit dem ORB einfacher. Das Übertragungsprotokoll zwischen den Schnittstellen der Applikationen wird mit nur einer Sprache, der IDL, beschrieben. Wenn die Entwickler den ORB benutzen, sind sie flexibler. Sie können die für die Anwendung am besten geeignete Programmiersprache und das beste Betriebssystem auswählen. Außerdem können mit dem ORB bestehende Komponenten benutzt werden. Mit Wrapper-Funktionen kann von der IDL zu bestehenden Schnittstellen übersetzt werden. [Obj96]

3.1.1 Das Objektmodell von CORBA

Drei wichtige Eigenschaften von CORBA-Objekten sind Kapselung, Vererbung und Polymorphismus. Kapselung ist die wichtigste Eigenschaft, weil sie die flexible Zusammenarbeit beliebiger Softwarekomponenten unterstützt und Ortstransparenz schafft. Außerdem können bestehende, gekapselte Applikationen mittels Wrapper-Funktionen mit einer IDL-Schnittstelle versehen werden und stehen für beliebige Klienten zur Verfügung. Im Gegensatz dazu unterstützen die beiden anderen Eigenschaften die Zusammenarbeit von CORBA mit objektorientierten Sprachen und Werkzeugen. Viele Analyse- und Entwurfswerkzeuge benutzen die Eigenschaften automatisch. Deshalb sind die Eigenschaften in die IDL integriert. Sie sind für brauchbare Softwarekomponenten aber optional.

3.1.2 IDL-Schnittstellen

Die Schnittstellen, über die die CORBA-Objekte nach außen sichtbar werden, sind in der IDL definiert. Die Schnittstellen enthalten die Operationen, die die Objekte durchführen können, die benötigten Eingabeparameter, die Ausgabeparameter, die zurückgegeben werden, und außerdem die Ausnahmen, die bei der Behandlung auftreten können. Alle Klienten können nur über die Schnittstelle auf die Objekte zugreifen, Zugriffe auf die Implementierung von Objekten sind nicht möglich.

Die IDL ist ähnlich wie die Programmiersprache C typisiert. Als zusätzlicher Basistyp existiert ein Typ *any*, mit dem alle Werte übergeben werden können. Konstruierte Typen dürfen nicht rekursiv benutzt werden. Dafür gibt es Folgen-Typen, die bei der Übersetzung eine maximale Länge bekommen, diese aber nicht ausnutzen müssen [Obj95].

In der CORBA-Architektur sind die Implementierungen von den Schnittstellen getrennt. Für einen Klienten stellt die Schnittstelle ein Versprechen dar. Wenn ein Objekt

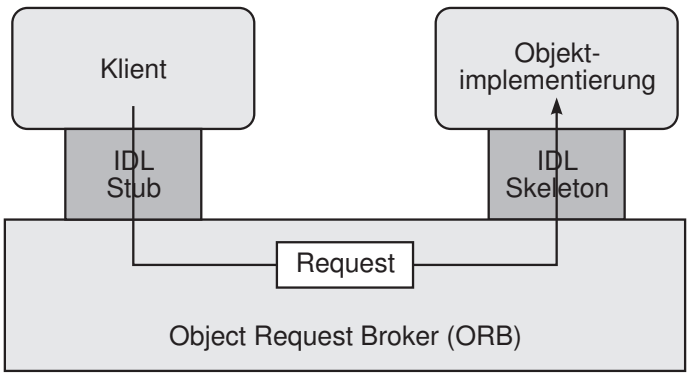

Abbildung 3.1: Der Weg einer Anfrage vom Klienten zur Objektimplementierung

mit einer geeigneten Nachricht aufgerufen wird, kommt auch eine Antwort zurück. Für den Programmierer ist die Schnittstelle eine Verpflichtung. Er muß in einer geeigneten Programmiersprache alle Operationen der Schnittstelle bereitstellen. Den Vertrag aufzustellen (die Schnittstelle in IDL definieren) und ihn einzuhalten (in einer Programmiersprache wie C++ oder Smalltalk) sind zwei verschiedene Schritte, in denen eine CORBA-Applikation erstellt wird.

Die Anfrage eines Klienten an einen Server nimmt den Weg, der aus der Abbildung 3.1 zu ersehen ist. Auf der Klientenseite wird der Aufruf durch den IDL-Stub an den ORB weitergereicht. Auf der Serverseite wird die Anfrage vom IDL-Skeleton aufgenommen und an die Implementierung übergeben.

Um ein CORBA-Objekt zu erstellen, muß man die folgenden Schritte durchführen:

1. Es muß eine genaue Spezifikation des Objektes erstellt werden.

2. Eine Schnittstelle wird mit der IDL definiert. Sie muß alle Operationen, die Eingabeparameter, die Rückgabewerte und die Ausnahmen enthalten.

3. Es ist eine geeignete Implementierungssprache für das Objekt auszuwählen. Dabei sollten zwei Aspekte berücksichtigt werden: Verfügbarkeit und Eignung. Die Schnittstelle muß dann auf entsprechende Konstrukte in der Programmiersprache abgebildet werden. Diese Abbildungen sind für die Programmiersprachen C, C++ und Smalltalk von der OMG vorgegeben. Der IDL-Übersetzer erzeugt Funktionsaufrufe aus den IDL-Operationen. Der Programmierer muß die Funktionsrümpfe bereitstellen.

4. Die Verbindung von der Schnittstelle zum ORB liegt in der Hand des ORB-Herstellers. Sie wird jedoch mit dem IDL-Übersetzer erzeugt, so daß sich der Programmierer nicht darum kümmern muß. Es müssen lediglich ORB und IDL-Übersetzer vom gleichen Hersteller kommen. Jedoch werden alle ORBs verschiedener Hersteller über die gleiche Schnittstelle angesprochen, so daß der Programmcode später leicht portiert werden kann.

Wenn das Serverobjekt und der Klient in der gleichen Programmiersprache erstellt wurden und über den gleichen ORB verbunden werden (dieser Spezialfall muß nicht immer erfüllt werden), kann aus der Schnittstellendefinition auch der Stub für den Klienten erzeugt werden. Er enthält Funktionsdeklarationen. Die Funktionsaufrufe müssen jetzt vom Programmierer erstellt werden. Sie passen direkt auf die Deklarationen im Server. Jedoch können die Verbindungen auch dynamisch erstellt werden.

3.1.3 Der Object Request Broker

Bisher sind gewöhnliche Client-Server-Verbindungen mittels entfernter Prozeduraufrufe vorgestellt worden. Das Ziel ist jedoch, mit ORB-zu-ORB-Verbindungen jedes Objekt mit jedem zu verbinden. Dazu müssen zwei Probleme gelöst werden: die Lokalisierung von Objekten und die Übersetzung in fremde ORB-Implementierungen.

Die Lokalisierung wird durch Objektreferenzen unterstützt. Sie spiegeln jedoch mehr als die reine Netzwerkadresse wider. Sie sollen gültig bleiben, wenn ein Objekt im Netz migriert. Dabei wird die Abbildung von der Referenz auf die Netzwerkadresse der Implementierung des ORB überlassen. Jede ORB-Implementierung muß jedoch alle Objektreferenzen entschlüsseln können.

Mit Hilfe der IDL wird das zweite Problem angegangen: Mit der IDL kann nicht nur das Skeleton und der Stub für die Kommunikation mit den Objekten erzeugt werden, sondern der ORB speichert zusätzlich die Schnittstellen aller Objekte in einem Interface Repository (IR). Diese Datenbank wird für die Zusammenarbeit der ORBs benötigt. Wenn die Typen und die Reihenfolge der Argumente einer Nachricht bekannt sind, können die ORBs Daten untereinander austauschen. Außerdem können mit Hilfe dieser Datenbank Schnittstellen auf Fehler untersucht oder alle Schnittstellen angezeigt werden.

Das Dynamic Invocation Interface (DII) sorgt für eine dynamische Verwaltung der Objekte und ist ein Teil des ORB. Damit können die Klienten auch neue Objekte benutzen, die später als der Klient erstellt wurden. Das DII stellt sich nach außen wie eine ORB-Schnittstelle dar, die auch in der IDL erstellt ist. Das DII enthält Routinen, mit denen der ORB und der Klient zusammenarbeiten können. Sie haben dabei die Möglichkeit, Objektschnittstellen aus dem IR zu benutzen, um alle Objekte anzusprechen.

Die Kommunikation zwischen ORBs bleibt für den Benutzer transparent. Falls der lokale ORB das aufgerufene Objekt nicht findet, gibt er den Aufruf an einen entfernten ORB weiter, um ihn dort an das gesuchte Objekt zuzustellen. Um diese Transparenz zu gewährleisten, müssen alle ORBs zwei Datenbanken verwalten: das IR mit den Schnittstellen und ein Implementation Repository, das Informationen über die verfügbaren Objektimplementierungen enthält. Außerdem müssen die Kommunikationsprotokolle der ORBs untereinander synchronisiert werden. Diese Probleme werden aber vor dem Benutzer und dem Applikationsprogrammierer verborgen. Mit der Version 2.0 ist die Kommunikation zwischen ORBs gesichert.

Um gekaufte Software integrieren zu können, wird sie in Zukunft mit einer IDL-Schnittstelle im Quellcode ausgeliefert, mit der auch Klienten eigene Schnittstellen erzeugen können. Bei der Übersetzung des Quellcodes in der IDL werden sowohl ein Stub für

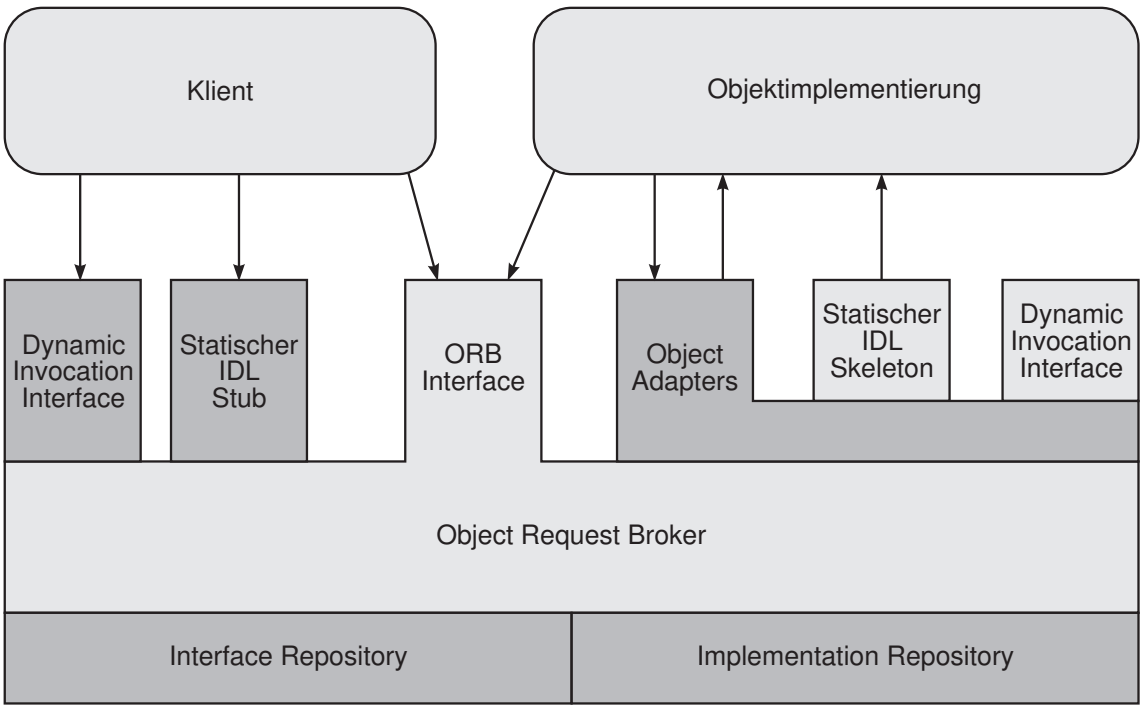

Abbildung 3.2: Der ORB mit Klient und Serverobjekt

den Klienten als auch ein Skeleton für der Server erzeugt. Nach der Übersetzung wird das Skeleton einfach gelöscht, weil der gekaufte Server ja ein passendes Skeleton mitbringt.

Insgesamt hat der ORB die Struktur und die Schnittstellen, die in der Abbildung 3.2 dargestellt sind. Klient und Implementierung können auf den ORB über eine Schnittstelle zugreifen. Der Objektadapter vermittelt zwischen dem ORB und der Objektimplemetierung.

3.1.4 Die Object Management Architecture

Die Object Management Architecture (OMA) aus Abbildung 3.3 bildet das Gerüst, das als Grundlage alle Applikationen im System zusammenhält. Die Transport- und Übertragungskomponente ORB wird durch verschiedene Standarddienste unterstützt, die von allen Applikationen genutzt werden können. Die Common Object services, die unterhalb des ORB liegen, standardisieren Grunddienste, die von fast allen Objekten gebraucht werden. Mit den Common Facilities werden den Applikationen Dienste der Mittelschicht angeboten.

Diese Dienste sind alle von der OMG spezifiziert, müssen jedoch nicht von jedem Hersteller mit dem ORB zusammen angeboten werden.

Bisher sind elf verschiedene Common Object services spezifiziert. Sie bieten alle eine systemnahe Unterstützung von objektorientierten Applikationen und ihren Komponenten. Die elf Dienste sind einzeln kurz beschrieben:

- Lifecycle Service: Erzeugen, Löschen Verschieben und Kopieren von Objekten.

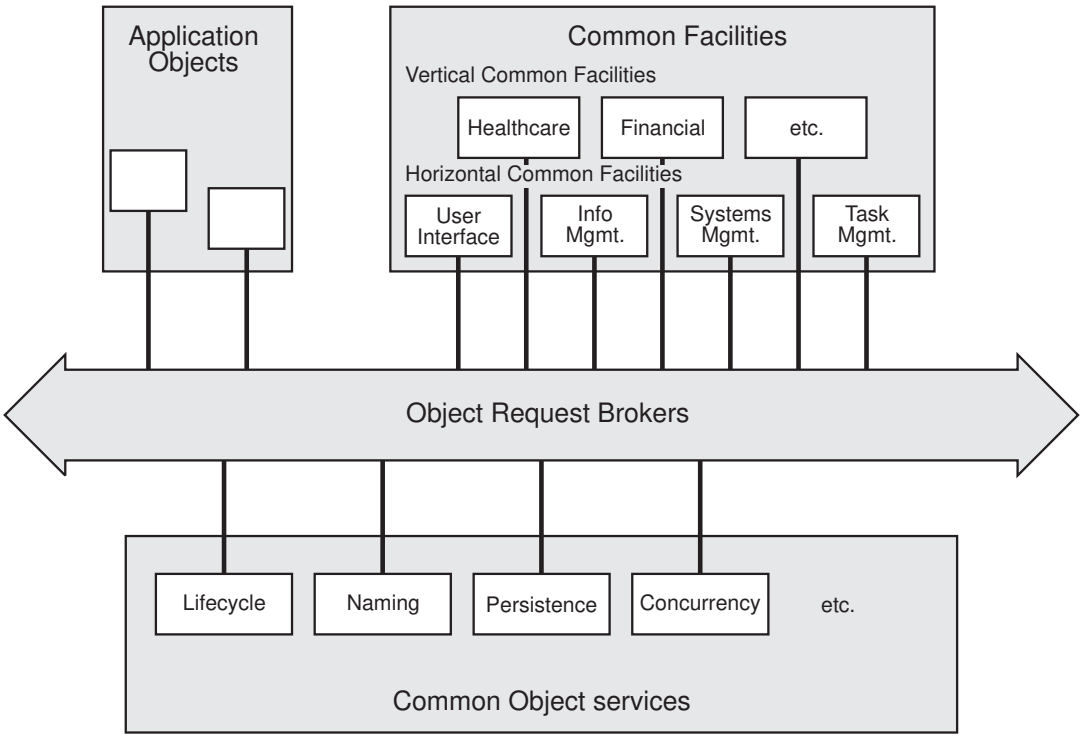

Abbildung 3.3: Die Object Management Architecture der OMG

- Relationship Service: Erzeugung und Verfolgung von dynamischen Verbindungen zwischen Komponenten, die nichts voneinander wissen.
- Naming Service: Abbildung von Objektnamen auf Objektreferenzen.
- Persistence Service: Speicherung von Objekten auf verschiedenen Medien.
- Externalisation Service: Datenaustausch von Komponenten über Byteströme.
- Event Service: Komponenten werden über bestimmte Ereignisse informiert.
- Query Service: Anfragen von Objekten an Datenbanken.
- Properties Service: Komponenten werden mit benannten Werten verbunden.
- Transaction Service: Zwei-Phasen-Commit-Protokoll für Transaktionen.
- Concurrency Service: Zusammen mit dem Object Transaction Service wird der Zugriff auf Business Accounting Systems unterstützt.
- Licensing Service: Alle einzelnen Aufrufe der Komponenten werden gezählt, um eine gerechte Abrechnung zu ermöglichen.

Fünf weitere Dienste sind noch nicht endgültig spezifiert: Trader Service, Security Service, Secure Time Service, Object Collection Service und Object Startup Service [OHE96].

Die Common Facilities liegen in einer Schichtenarchitektur des CORBA-Systems zwischen den Common Object services und der Applikationsschicht. Mit Hilfe dieser Komponenten können die Applikationen weiter vom Systemkern entfernt bleiben und schneller entwickelt werden. Die Bestandteile der Common Facilities werden in zwei Abschnitte unterteilt. Die horizontalen Komponenten sollen in allen Branchen benutzt werden. Es gibt vier Kategorien: user interface, information management, systems management und task management. Die vertikalen Komponenten sollen dagegen branchenspezifisch eingesetzt werden. Hierbei sollen zum Beispiel Komponenten für Banken oder das Gesundheitswesen erstellt werden.

Die Spezifikationen der Common Facilities sind noch nicht abgeschlossen. [Sie96]

3.1.5 Vergleich von CORBA und Tycoon

Bei der Konzeption und Realisierung des Tycoon-Systems und des CORBA-Systems wurden unterschiedliche Ansätze gewählt. Jedoch soll hier verglichen werden, welche Beiträge beide Systeme zu einem verteilten System leisten. Für einen Vergleich beider Systeme sollen die folgenden Kriterien benutzt werden.

In [LKK93] werden fünf Anforderungen an ein verteiltes System gestellt:

Kommunikationsmechanismen: Die Kommunikationsdienste sollen leicht zu handhaben sein und von den Eigenschaften des zugrundeliegenden Transportprotokolls möglichst weit abstrahieren.

Behandlung der Heterogenität: Die beteiligten Rechner des verteilten Systems stammen in der Regel von unterschiedlichen Herstellern und benutzen verschiedene Betriebssysteme und verschiedenartige Datenformate.

Zusätzliche Systemdienste: Neben Kommunikationsdiensten sind Systemdienste zur Unterstützung der verteilten Anwendung wünschenswert.

Einbettung von Diensten in Programmiersprachen: Verteilte Anwendungen werden meist in höheren Programmiersprachen entwickelt. Daher ist es wünschenswert, die Kommunikationsdienste und die zusätzlichen Systemdienste direkt in die jeweilige Sprache einzubetten.

Entwicklungsunterstützung: Neben den Diensten sollen auch Methoden und Werkzeuge zur Entwicklung verteilter Anwendungen vorhanden sein. Beispiele hierfür sind spezielle Entwurfssprachen und -editoren oder Testwerkzeuge.

Anhand dieser Kriterien sollen CORBA und Tycoon verglichen werden.

Während CORBA von Anfang als System für verteilte Objekte entworfen und entwickelt wurde, wurde die Verteilung bei Tycoon erst nach der Entwicklung eines lokalen

persistenten Objektsystems realisiert. Bei der Integration wurde aber die Orthogonalität beibehalten, so daß alle Objekte unabhängig von ihrem Typ mobil wurden. Entfernte Dienste können mit dem Remote Procedure Call genutzt werden, und Objekte können an andere Systeme verschickt werden. Diesem komfortablen Kommunikationsmechanismus steht bei CORBA der Object Request Broker gegenüber, der ebenfalls eine für den Benutzer transparente Ausführung entfernter Objektmethoden zuläßt. Der Dienst ist in Tycoon im System eingebaut, wobei mit CORBA eine weitere Softwarekomponente benutzt wird, die unter anderem zusätzliche Wartung benötigt.

Beide Systeme unterstützen heterogene Umgebungen. CORBA wird von mehreren Herstellern angeboten, die unterschiedliche Plattformen unterstützen. Über die IDL wird von Systeminterna wie dem Betriebssystem abstrahiert. Tycoon ist plattformunabhängig und damit in heterogenen Umgebungen einsetzbar.

CORBA unterstützt eine Vielzahl zusätzlicher Systemdienste als Common Object services. Diese Dienste gibt es bei Tycoon noch nicht. Jedoch wird der Benutzer durch die Adressierungstransparenz entlastet.

Tycoon besitzt mit TL eine eingebaute, mächtige Programmiersprache, mit der auch die Kommunikationsdienste benutzt werden können. CORBA hat keine eigene Programmiersprache, jedoch können mit der IDL Programme verschiedener Sprachen die Kommunikationsdienste nutzen.

Mit Blick auf die Entwicklungsunterstützung hat CORBA im Vergleich mit Tycoon den Vorteil der Objektorientierung und aller Erweiterungen, die aus diesem Ansatz entstehen. In der Programmiersprache TL kann lediglich ein objektorientierter Programmierstil benutzt werden, in dem Funktionen in Tupel gekapselt werden und von der Möglichkeit der Subtypisierung Gebrauch gemacht wird. Jedoch kennt TL keine Vererbung. Außerdem müssen die Funktionsnamen in TL aus Prinzip eindeutig vergeben werden, während die Objekte in CORBA ihre Methoden überladen können. Bei CORBA wird der Programmierer durch die Generierung der Schnittstellen entlastet [EH96].

Etwa gleichauf liegen beide Systeme bei der Integration bestehender Systeme, um sie verteilt einsetzen zu können. Die in Tycoon eingebundenen Applikationen können netzweit benutzt werden, nachdem sie exportiert wurden. Tycoon-Anwendungen, die in TL entwickelt sind, brauchen keine besondere Schnittstellendefinition mittels einer von der Programmiersprache unabhängigen IDL. Hier können entfernte Dienste direkt benutzt werden.

Außerdem bietet Tycoon mit dem Prinzip der mobilen Threads, das auf den gleichen Kommunikationsmachanismen wie der RPC basiert, eine Basis für mobile Agenten und Workflow-Umgebungen, die CORBA noch nicht hat. Innerhalb der OMG finden aber Diskussionen statt, wie Agenten in die Architektur aufgenommen werden können.

Tabelle 3.1 zeigt einen Überblick über den direkten Vergleich anhand der oben genannten Kriterien.

Kriterium	Tycoon	CORBA
Kommunikations-mechanismen	eingebaut, abstrakt	besondere Middleware nötig
Behandlung von Heterogenität	plattformunabhängig	verschiedene Hersteller
zusätzliche Systemdienste	Addressierungstransparenz	zahlreiche Dienste
Einbettung in Programmiersprachen	direkt in TL	IDL für verschiedene Programmiersprachen
Entwicklungs-unterstützung	wie TL-Programmierung	Generierung der Schnittstellen

Tabelle 3.1: Vergleich von CORBA und Tycoon

3.2 DCE: Umgebung für verteilte Systeme

Das Distributed Computing Environment (DCE) der Open Software Foundation (OSF) stellt eine Umgebung für verteilte Systeme zur Verfügung. Es unterstützt den Programmierer verteilter Systeme mit Diensten für die Kommunikation zwischen Anwendungen. Für den Benutzer ist die Verteilung des Systems transparent.

Eine DCE-Umgebung wird grundsätzlich in sogenannte Cells unterteilt. Jede Cell besteht aus einer Menge von Rechnerknoten und stellt eine organisatorische Einheit dar.

Die Abbildung 3.4 gibt einen Überblick über die Dienste, die im DCE zur Verfügung stehen. Die Basisdienste werden direkt von den Anwendungen angesprochen. Ihre Schnittstellen sind für den Programmierer wichtig. Die weitergehenden Dienste sind für die Anwendungsentwicklung nicht interessant, weil sie nur indirekt genutzt werden.

Alle Dienste beruhen auf Betriebssystem- und Transportdiensten. Der unterste Dienst in der DCE-Umgebung ist der Thread-Service. Er realisiert eine portable Implementierung von leichtgewichtigen Prozessen (Threads). Sie können sich einen Adressraum teilen und gemeinsam auf globale Daten zugreifen. Damit können nebenläufige, quasi-parallele Aktivitäten innerhalb eines Adreßraums flexibel programmiert werden. Ein Klient, der entfernte Funktionen aufruft, wird nicht blockiert, bis das Ergebnis eintrifft, ein Server kann nebenläufig auf Anfragen warten.

Der Remote Procedure Call (RPC) ist der Basiskommunikationsmechanismus im DCE. Er realisiert synchrone Fernaufrufe von Prozeduren im Rahmen des Client-Server Modells. Dabei können sich Klient und Server auch auf dem gleichen Rechnerknoten befinden. Der Aufruf der entfernten Prozedur geschieht für den Programmierer zum großen Teil transparent, d. h. unterschiedliche Formate werden automatisch konvertiert. Außerdem können beliebige Parameterstrukturen übergeben werden. Mittels Attributen kann der Program-

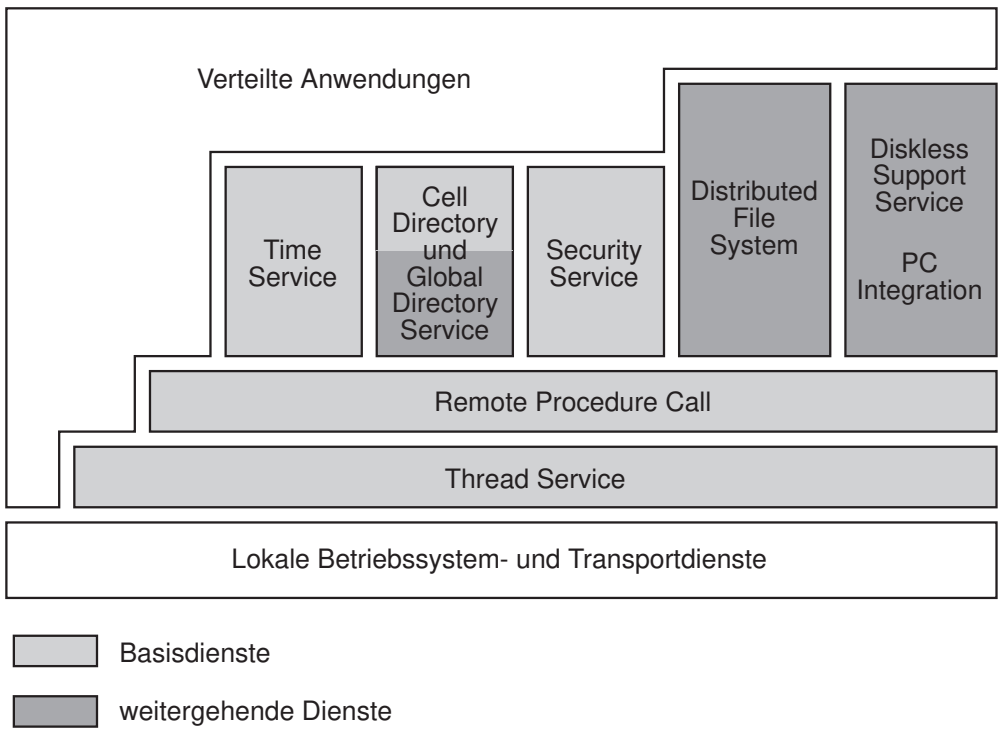

Abbildung 3.4: Die Gesamtarchitektur des OSF-DCE

mierer festlegen, ob der Aufruf einmal, mehrmals oder ohne Fehlerrückmeldung ausgeführt werden soll. Dabei können mit dem RPC ein oder mehrere Server angesprochen werden. Es ist auch möglich, im Fehlerfall mehrere Versuche durchzuführen. Es können ebenfalls alle passenden Server angesprochen werden. Dabei wird der erste Rückgabewert akzeptiert. Die Schnittstellen für den RPC werden mit einer Interface Definition Language (IDL), die an C angelehnt ist, spezifiziert.

Der Directory Service hat die zentrale Aufgabe, logische Namen auf Netzwerkadressen abzubilden. Mit diesem zentralen Directory Service sind alle Namen schnell zugänglich und können leicht konsistent gehalten werden. Der Directory Service wird in zwei Komponenten unterteilt: Cell Directory Service (CDS) und Global Directory Service (GDS). Dabei ist der CDS für die Verwaltung der Namen innerhalb einer Cell verantwortlich. Der GDS stellt das Bindeglied zwischen verschiedenen CDS dar.

Mit dem Security Service wird der Zugriff unberechtigter Benutzer oder Programme auf Daten, Dienste und Nachrichten innerhalb einer DCE-Umgebung verhindert. Dieser Dienst wird primär mit dem RPC eingesetzt. Grundsätzlich ist alle übertragene Information verschlüsselt. Jede Instanz kann Zugriffsrechte an andere Instanzen vergeben. Außerdem müssen sich die Benutzer bei der Anmeldung mit einem Paßwort authentifizieren. Die Authentifikation kann in verschiedenen Granularitäten durchgeführt werden. Der Benutzer muß sich entweder zu Beginn einer Sitzung, bei jedem Aufruf eines Dienstes oder bei jedem Übertragungspaket authentifizieren. In weiteren Stufen wird der Schutz vor Modifikation von Nachrichten garantiert oder es findet eine vollständige Verschlüsselung statt.

Der Distributed Time Service (DTS) dient zur Synchronisation der Systemuhren aller Rechner eines verteilten Systems. Dazu muß jede Cell mindestens einen Time Server enthalten. Die anderen Rechner der Cell befragen den Server in bestimmten Zeitabständen nach seiner Systemzeit. Außerdem bietet der DTS auch eine Programmierschnittstelle an. Damit kann die Systemzeit abgefragt werden und verschiedene Uhrzeiten können verglichen werden.

Das Distributed File System (DFS) verwaltet Dateien systemweit. Damit können Anwender entfernte Dateien so ansprechen, als ob sie auf dem eigenen Rechner liegen würden. Das DFS hat folgende Eigenschaften:

Lokationstransparenz: Der Ort, an dem eine Datei abgelegt ist, ist für den Anwender vollkommen transparent.

Replikation: Dateien können auf mehreren Rechnern in Kopien zum Lesen gehalten werden, um ihre Verfügbarkeit zu erhöhen.

Caching: Mit einem lokalen Cache wird der Zugriff auf Dateien so effizient wie möglich gestaltet.

Recovery: Durch Protokollierung der Zugriffe auf die Dateien wird eine Wiederherstellung nach einem Rechnerausfall ermöglicht.

Mit dem Diskless Support Service (DSS) können Rechner ohne Hintergrundspeicher in das DCE integriert werden. Dazu muß auf jedem plattenlosen Rechner ein Diskless Client zur Verfügung stehen, der mit entsprechenden Servern mit externer Dateiverwaltung kommunziert und so die Funktionalität des Hintergrundspeichers ersetzt. Auf Serverseite müssen dafür Boot Server, Swap Server und Configuration Server zur Verfügung stehen.

Die PC Integration ermöglicht es, Personal Computer mit den Betriebssystemen MS/DOS, OS/2 und Unix in der DCE-Umgebung einzusetzen. Jedoch können diese Rechner wegen ihrer begrenzten Betriebsmittel nur als Clients eingesetzt werden.

Für den Anwendungsentwickler ist primär der RPC als Kommunikationsmechanismus von Bedeutung. Er stellt sich als eine geschlossene Bibliothek von C-Funktionen dar. [Sch93]

3.2.1 Vergleich vom DCE und CORBA

In der Diplomarbeit von Martin Schoel [Sch96] werden das DCE und CORBA verglichen. Dort kommt er zu folgenden Ergebnissen: Der Hauptunterschied zwischen CORBA und DCE ist der objektoriente Ansatz von CORBA gegenüber dem herkömmlichen modularen DCE-Ansatz. CORBA bietet so eine bessere Unterstützung bestehender Systeme als das DCE. Jedoch hat das DCE einen Zeitvorsprung gegenüber CORBA, so daß robustere Implementierungen verfügbar sind. Auch bieten einige Teilsysteme wie der Thread Service oder Security Service im DCE eine bessere Basis für verteilte Systeme als sie CORBA momentan bieten kann. Für das DCE spricht auch, daß Implementierungen, die für das DCE entwickelt wurden, leichter auf neue Plattformen portiert werden können, wenn dort das

Kriterium	Tycoon	DCE
Kommunikations-mechanismen	eingebaut, abstrakt	Umgebung für Verteilung
Behandlung von Heterogenität	plattformunabhängig	zahlreiche Betriebssysteme
zusätzliche Systemdienste	Addressierungstransparenz	viele Dienste
Einbettung in Programmiersprachen	direkt in TL	Schnittstellen in C
Entwicklungs-unterstützung	wie TL-Programmierung	Editoren und Verwaltungsprogramme

Tabelle 3.2: Vergleich vom DCE und Tycoon

DCE schon verfügbar ist. Wegen einer ungenauen Spezifikation des Basic Object Adapter (BOA) bei CORBA ist die Portierung auf einen neuen ORB nur eingeschränkt möglich. Nach [Sch93] bietet CORBA insgesamt ein höheres Abstraktionsniveau zur Interaktion verteilter Objekte als das DCE.

3.2.2 Vergleich vom DCE und Tycoon

Für diesen Vergleich werden die gleichen Kriterien wie beim Vergleich von CORBA und Tycoon im Abschnitt 3.1.5 benutzt.

Die Kommunikationsmechanismen werden im DCE durch die Integration der Anwendungen in die verteilte Umgebung eingefügt. Damit sind sie für den Anwendungsentwickler leicht nutzbar.

Heterogene Systeme werden im DCE unterstützt, indem die Umgebung für zahlreiche Betriebssysteme angeboten wird. Außerdem können PCs über eine eigene Komponente integriert werden.

Das DCE bietet zum Beispiel mit dem Time Service und dem Diskless Support Service mehr zusätzliche Systemdienste als das Tycoon-System.

Die verteilte Umgebung des DCE stellt sich für den Programmierer als Sammlung von Funktionen der Programmiersprache C dar, die von ihm bei Bedarf aufgerufen werden können.

Der Entwickler wird bei seiner Arbeit im DCE durch einige Werkzeuge wie ein Editor zur Definition und Bearbeitung von Zugriffskontrollisten oder ein Kontrollprogramm zur Verwaltung der Cell Directory Services unterstützt.

Die Vergleichsergebnisse werden in Tabelle 3.2 noch einmal zusammengefaßt.

3.3 OLE: Dokumentenzentriertes Arbeiten

Object Linking and Embedding (OLE) ermöglicht es Anwendungen, Datenblöcke anderer Programme in eigene Dokumente einzubinden, ohne sich um deren Format zu kümmern. Dazu werden immer eine Client- und eine Server-Anwendungen benötigt: die Client-Anwendung empfängt die Daten, die die Server-Anwendung liefert. Die Verwaltung der Daten geschieht nach der Übergabe arbeitsteilig. Der Client behandelt den Datenblock als gesichtsloses Objekt, das er an einer bestimmten Stelle in das Dokument einfügt. Die Bearbeitung und eventuell auch die Darstellung der Daten übernimmt weiterhin der Server. Ein Dokument einer Client-Anwendung, das OLE-Objekte enthält, wird Compound Document genannt.

Wie aus dem Namen OLE hervorgeht, kann ein Dokument auf zwei verschiedene Arten zusammengesetzt werden. Beim Verknüpfen (Linking) wird das Objekt von der Server-Anwendung in eine Datei abgelegt und diese Datei als verknüpftes Objekt über die Zwischenablage des Systems in das Dokument der Client-Anwendung eingefügt. Die Client-Anwendung speichert dann alle Angaben, um das Objekt später wiederzufinden. So kann ein Objekt in verschiedene Dokumente eingefügt werden, und wenn es geändert wird, erscheint es auch im zusammengesetzten Dokument jederzeit im aktuellen Zustand. Wenn ein Objekt in ein Dokument eingebettet wird, wird die Server-Anwendung aus der Client-Anwendung heraus gestartet, das Objekt erstellt und beim Beenden der Server-Anwendung werden die Daten im Format der Server-Anwendung im Dokument der Client-Anwendung gespeichert. [Lau93]

Diese Grundlage, eine verbesserte Importmöglichkeit für Dateien verschiedener Formate, war aber nur der Anfang der OLE-Entwicklung. Der Hersteller Microsoft hat, auf diesem Konzept aufbauend, eine Umgebung für verteilte Objekte geschaffen, die als Basis für die kommenden Betriebssysteme genutzt werden soll. Innerhalb von Windows 95 ist OLE von großer Bedeutung. Ziel ist es, flexible Komponenten im System zusammenarbeiten zu lassen. Zum Beispiel wird die Textverarbeitung, die dem Benutzer unzählige Möglichkeiten bietet, in einzelne Komponenten wie Editor, Rechtschreibprüfung, Trennhilfe, Formeleditor, etc. zerlegt. Diese Komponenten werden im OLE-System als Objekte bezeichnet.

OLE basiert wie CORBA auf einem Object Request Broker, der hier Component Objekt Model (COM) genannt wird. Jedoch unterstützt COM bis jetzt nur einen einzelnen Rechner, die Objektverteilung innerhalb eines Netzwerkes ist noch nicht möglich, aber für zukünftige Versionen angekündigt.

Nach [MO94] ist *Interface* der wichtigste Begriff des Component Object Model. Interfaces sind die Bestandteile eines Objektes, über die es von außen angesprochen, verwendet und eingebunden wird. Ein Interface ist die Zusammenfassung einer Reihe von ähnlichen Methoden eines Objektes zu einer Einheit (siehe Abbildung 3.5).

Innerhalb des COM wird ein Verhandlungsprotokoll verwaltet, mit dem Klienten zur Laufzeit Referenzen auf Schnittstellen erhalten können. Für die Beschreibung der Schnittstellen exisitiert auch bei OLE eine spezielle Sprache. Außerdem können die Objekte mit einer Object Description Language (ODL) in eine Bibliothek eingetragen werden, die dem

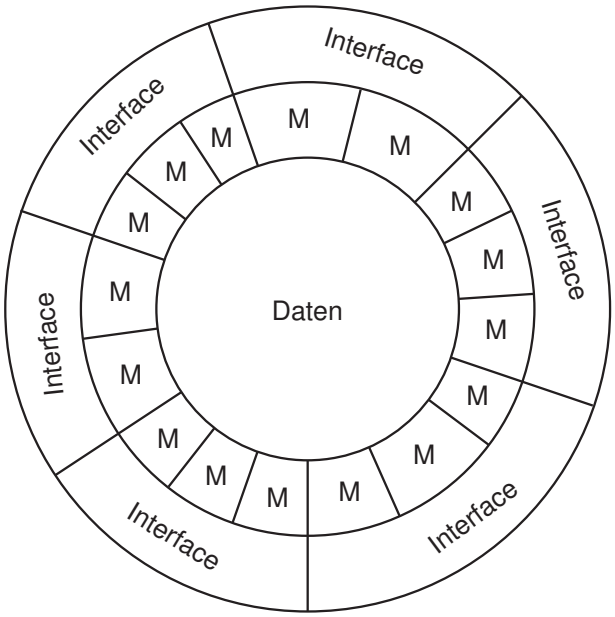

Abbildung 3.5: Ein OLE-Objekt mit Methoden und Interfaces

Interface Repository von CORBA entspricht. Die Methoden der Objekte können, wie bei CORBA, sowohl statisch wie auch dynamisch angesprochen werden.

Eine OLE-Komponente wird von einer Klasse definiert. In ihr sind eine oder mehrere Schnittstellen implementiert. Außerdem erhält die Klasse eine Class Factory, mit der eine Instanz dieser Klasse erzeugt wird. Jedoch ist eine OLE-Komponente keine alleinstehende Einheit, sie ist lediglich eine Sammlung von Schnittstellen. Für jede geforderte Funktionalität muß eine Schnittstelle bereitstehen.

Fünf wesentliche Technologien werden in OLE benutzt, um die gesamte Funktionalitiät zu bilden. OLE muß Komponenten organisieren, um sie auf dem Bildschirm darstellen zu können. Dazu werden Regeln definiert, nach denen sich beteiligte Komponenten am Bildschirmaufbau beteiligen. Andere Regeln bestimmen, wie sie Daten innerhalb einer Datei abspeichern. OLE bietet die Schnittstellen, mit denen Komponenten Daten mit anderen Komponenten austauschen können. Außerdem können Komponenten ihre Aktionen über automatische Befehlsketten (Scripte) steuern. Die dafür nötigen Technologien sind in der Abbildung 3.6 dargestellt und werden im folgenden einzeln beschrieben.

Das *Component Object Model* spezifiziert die Schnittstellen zwischen den Komponentenobjekten innerhalb einer Applikation oder zwischen verschiedenen Applikationen. Wie bei CORBA wird dabei die Implementierung von der Schnittstelle getrennt. Mittels Application Programming Interfaces (APIs) wird es den Objekten ermöglicht, Objektschnittstellen dynamisch zu finden und sie anzusprechen. Dazu müssen die Daten der Schnittstelle mit einer IDL zur Verfügung gestellt werden. Jedoch kann das COM nur Objekte auf einem Rechner suchen. Mehrfachvererbung ist im COM nicht möglich, statt dessen können mehrere Schnittstellen bereitgestellt werden. Diese Schnittstellen können in anderen Kompo-

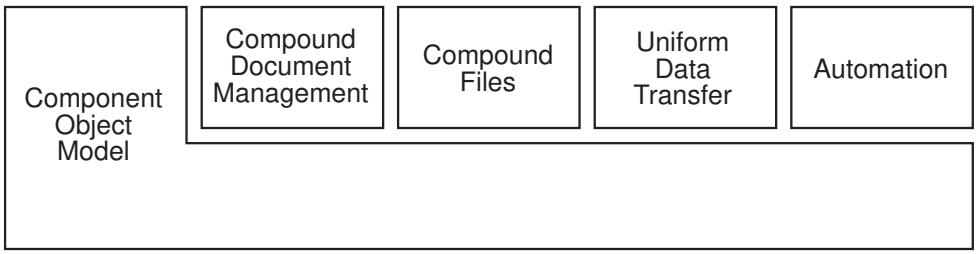

Abbildung 3.6: Die wesentlichen Technologien von OLE

nenten aggregiert werden. Damit kapseln diese Komponenten die Dienste der aggregierten Komponenten. Um ein Objekt nutzen zu können, muß der Client einen einheitlichen Klassenidentifikator absenden. Der Service Control Manager (SCM) des COM sucht in einer Registrierdatenbank nach dem Server und benachrichtigt ihn, damit er eine Instanz seiner Komponente zur Verfügung stellt.

Mit den Automatisierungs- und Scripting-Diensten (*Automation*) können Serverkomponenten von Automatisierungsklienten kontrolliert werden. Diese Klienten werden typischerweise mit einer Scripting-Sprache programmiert. Diese Technik basiert auf der Möglichkeit, Schnittstellen dynamisch anzusprechen. Mit dieser Technik können mächtige Applikationen weiter spezialisiert werden. Zum Beispiel kann man eine Tabellenkalkulation zu einer Immobilienverwaltung machen.

Für den Datenaustausch (*Uniform Data Transfer*) zwischen verschiedenen Komponenten exisitieren verschiedene Möglichkeiten, um möglichst viele Situationen und verschiedene Medien abzudecken. Daten können über eine Zwischenablage, durch Ziehen und Verschieben, über Verbindungen oder zusammengesetzte Dokumente ausgetauscht werden. Der Datenaustausch kann innerhalb eines Dokuments, mit einem anderen Dokument oder mit einer anderen Applikation stattfinden. OLE stellt verschiedene Möglichkeiten für den Datenaustausch über eine Schnittstelle zur Verfügung.

Das OLE Structured Storage System bietet ein Dateisystem innerhalb einer Datei. Die aktuelle Implementierung dieser Architektur wird *Compound Files* genannt. Diese Compound Files teilen eine physikalische Datei in mehrere Verzeichnisse und Rohdaten auf. Mit dieser Struktur können die Inhalte eines Dokumentes organisiert werden. Die beteiligten Komponenten können ihren Teil in den Compound Files selbst kontrollieren. Klienten können über mehrere Schnittstellen mit persistenten Objekten kommunizieren. Diese bilden den Übergang zu regulären Dateien. OLE bietet einen Dienst für persistente Namen, der Monikers genannt wird. Ein Moniker ist ein Name, der an persistente Daten gebunden werden kann. Sie können die assoziierten Daten über verschiedene Algorithmen suchen, die bei der Klassenerstellung angegeben werden. So können die Moniker als Zeiger für entfernte Dateien dienen oder SQL-Anfragen beinhalten, die die assoziierten Daten aus einer Datenbank ermitteln. Mit Hilfe diese Indirektion können eingebundene Daten ihren Platz im System transparent für die umgebende Datei ändern.

Der Compound Document Service aus dem *Compound Document Management* von OLE bietet Dienste für das dokumentenzentrierte Arbeiten. Dazu werden Schnittstellen

zwischen den Containeranwendungen (in älteren Versionen wurden sie Clients genannt) und den Serverkomponenten, die sie steuern, definiert. Mit diesen Schnittstellen wird ein Protokoll geliefert, das beim Zugriff auf Daten einer Anwendung den Server aktiviert und die Daten am Platz ändern läßt. Jede Serverkomponente kontrolliert die Darstellung und Speicherung seiner Daten im zusammengesetzten Dokument. Mit der Container/Server-Beziehung können Daten verschiedener Formate wie Bilder oder Ton in einem Dokument integriert werden.

3.3.1 Vergleich von OLE und CORBA

In [OHE96] liefert ein Vergleich von OLE und CORBA das folgende Ergebnis:

Der Hauptunterschied zwischen CORBA und OLE besteht darin, daß CORBA den klassischen objektorientierten Ansatz verfolgt und auf seine Komponenten ausdehnt. Jede Komponente unterstützt Mehrfachvererbung, Kapselung und Polymorphismus. Die Komponenten gehören zu Klassen, die zur Laufzeit instanziiert werden können. Die OLE-Komponenten hingegen sind keine klassischen Objekte. Sie können nicht durch Vererbung erweitert werden, sondern nur durch Kapselung andere Objekte quasi als eigene Methoden benutzen. Insgesamt ist eine OLE-Komponente eine Black Box, die, nachdem sie gekauft und installiert wurde, nicht erweiterbar ist.

Zusammenarbeit zwischen OLE-Komponenten wird nur auf der binären Ebene und nur auf einem Rechner unterstützt. OLE ist noch kein verteiltes System. Im Gegensatz dazu können CORBA-Komponenten über die Sprachen C, C++, Smalltalk und COBOL aneinander gebunden und verteilt werden. CORBA ist OLE auch auf dem Gebiet der System Services überlegen. CORBA bietet mehr Dienste als OLE an. Zusammengesetzte Dokumente werden von OLE besser unterstützt als von CORBA.

3.3.2 Vergleich von OLE und Tycoon

Bei OLE steht die Wiederverwendung von Daten im Vordergrund, um dem Benutzer dokumentenzentriertes Arbeiten zu ermöglichen. Aus dieser Motivation heraus ist ein Ansatz für verteilte Objekte gewachsen, um die Zusammenarbeit von Applikationen zu ermöglichen. Im Tycoon-System spielte die flexible Wiederverwendung von Daten bisher keine Rolle, deshalb findet man hier kein entsprechendes Gegenstück. Es ist immer die Aufgabe des Anwendungsentwicklers, bei der Wiederverwendung vorhandener Programme die Datenstrukturen entsprechend anzupassen. Dabei wird er durch die Möglichkeit der Subtypisierung in TL unterstützt. Generell ist die Wiederverwendung von Daten und Komponenten in kooperativen Informationssystemen aber ein wichtiges Mittel, um die Zusammenarbeit zu erleichtern. Deshalb wird später nochmals darauf eingegangen.

Kapitel 4

Erfassung und Analyse von Internetnutzungsstatistiken

In diesem Kapitel wird die Beispielanwendung aus WebSiteProfilern und MultiSiteAnalyzern, die die weitere Arbeit begleiten wird, vorgestellt.

Zuerst wird auf die Problematik der Nutzungsstatistiken im WorldWideWeb eingegangen. Dabei wird eine Motivation für den kommerziellen Einsatz der Statistiken zum Nachweis der Leistungsfähigkeit eines WWW-Angebots gegeben. Der WebSiteProfiler dient zur Erstellung von Nutzungsstatistiken eines WWW-Servers. Er wird anhand seiner Systemarchitektur und seines Datenbankschemas vorgestellt. Der MultiSiteAnalyzer soll die Nutzung mehrerer WWW-Server miteinander vergleichen. Dazu wird der inhaltliche Zusammenhang von WWW-Servern untersucht und eine Systemübersicht des MultiSiteAnalyzers gegeben.

4.1 Problemstellung

Jeder Informationsanbieter (Content-Provider), der Dokumente im WWW auf einem Server zum Abruf bereitstellt, erhält Protokolldateien über die Zugriffe auf sein Angebot. Sie dienten ursprünglich zur Auslastungsmessung und dokumentieren sekundengenau die Abrufe von diesem Server. Damit ist es technisch zwar möglich, die Nutzung des Angebots, nicht aber, den Zugriff individueller Nutzer zu verfolgen [Kab96].

Im HTTP-Protokoll ist aber eine Zwischenspeicherung von häufig abgerufen Dokumenten (Caching) auf Proxy-Servern vorgesehen, um die Netzbelastung zu verringern [BFF96]. Ein Ausschnitt aus dem WorldWideWeb, in dem mehrere Benutzer von einem Server Dokumente anfordern, ist in der Abbildung 4.1 dargestellt. Die Anforderungen der Dokumente gehen durch mehrere Proxy Server (P_1, P_2 oder P_3), die das Dokument in ihrem Speicher vorhalten können. Erst wenn sie das vom Benutzer angeforderte Dokument in den Caches nicht finden, wird die Anforderung vom Server S beantwortet. Wegen der Zwischenspeicher werden in den Protokolldateien der Informationsanbieter immer nur untere Nutzungsgrenzen erfaßt. Die Zugriffe auf Proxy-Server können aber auch protokolliert werden.

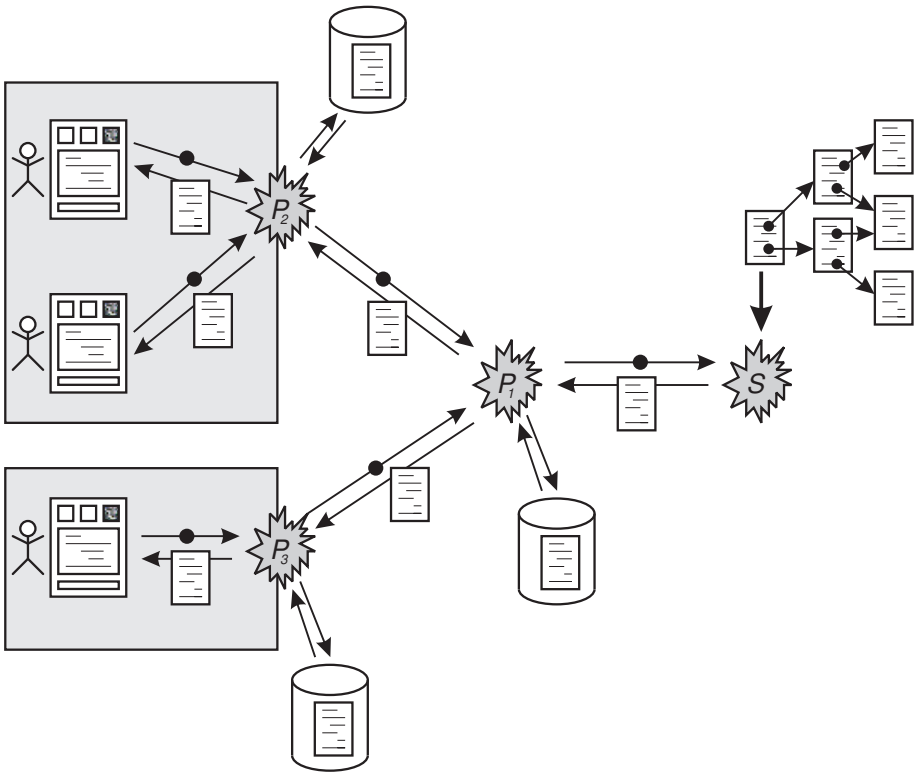

Abbildung 4.1: Caching bei der Dokumentenanforderung im WorldWideWeb

Nach [Kab96] ist jedoch verbandsübergreifend festgelegt, daß das Proxy-Server-Caching mit Hilfe kleiner CGI-Scripte umgangen wird, weil Zugriffe auf diese Art von Dateien immer bis zum Informationsanbieter durchgereicht werden. Jedoch sind Zugriffe auf Dateien, die in lokalen Cache-Speichern auf dem Rechner der Nutzer gehalten werden, für den Informationsanbieter nicht meßbar.

Generell ist der Nutzer für den Informationsabieter nicht identifizierbar. Es ist lediglich meßbar, welche Host-Rechner auf die Dateien zugegriffen haben. Jedoch können die Nutzer über inhaltliche Angaben, zum Beispiel ihre e-Mail-Adresse identifiziert werden, wenn sie es wollen.

Als Effizienzparameter werden in [Kab96] folgende Meßgrößen vorgeschlagen: *Page-Views* bezeichnen die Anzahl der Sichtkontakte von beliebigen Nutzern mit einer HTML-Seite. Wenn in der Seite Werbeanteil und Information durch das Layout getrennt sind, zählt jeder Sichtkontakt mit der Information zusätzlich als Page-View für den zugehörigen Werbeanteil. Die Summe aller Page-Views gibt Aufschluß über die Attraktivität eines Angebots. Ein *Visit* ist ein zusammenhängender Nutzungsvorgang eines WWW-Angebots. Als Nutzungsvorgang zählt ein technisch erfolgreicher Seitenzugriff (alle Daten werden an den Nutzer übertragen), wenn der Nutzer von außen auf die Seite zugreift, sich also nicht innerhalb des Navigationssystems des Angebots bewegt.

Einzelne *Hits*, die Anforderungen einzelner Dateien, stellen keine geeignte Meßgröße

dar. Eine WWW-Seite kann sich aus beliebig vielen Dateien zusammensetzen. Damit wird der Abruf einer grafikreichen Seite mit vielen Dateien einer Textseite aus einer einzelnen Datei gleichgestellt.

Daten über die Nutzung des Angebots dienen zwei grundsätzlichen Zielen. Zum einen sollen sie eine interne Bewertung des Angebots erlauben und Stärken oder Schwächen feststellen. Zum anderen sollen Effizienzkriterien für Markt- und Werbepartner veröffentlicht werden. Dazu müssen die Kriterien zuvor standardisiert und kontrolliert werden.

Online-Werbung kann auf der Grundlage von Visits und Page-Views abgerechnet werden. Analog zur klassischen Mediawerbung wird die Werbung auf der Basis der nachgewiesenen Leistungsfähigkeit des Werbeträgers vergütet. Ein anderes Modell basiert auf Werbe-Clicks (Clicks auf ein Objekt mit Weiterleitung zu Informationen des Werbetreibenden). Für eine vernünftige Lösung sollten beide Modelle kombiniert werden.

Die Messung der Reichweite von Werbung im WWW nach klassischen Methoden hat auch einen Nachteil. Die Parameter der klassischen Werbeträgeranalysen basieren auf einer Einwegkommunikation mit dem Konsumenten. Im WorldWideWeb kann der Nutzer jedoch vom Werbetreibenden eingebunden werden. Ein identifizierbarer Nutzer kann individuell begrüßt und mit abgestimmten Informationen bedient werden. Diese Qualitäten können mit den oben beschriebenen Parametern, die allein die Quantität der Nutzer betrachten, nur unzureichend erfaßt werden.

Grundsätzlich wird das Thema „Analyse von Web-Server-Nutzung“ in einer anderen Diplomarbeit [Lau97] am Arbeitsbereich DBIS behandelt.

4.2 Der WebSiteProfiler

Hier wird das erste Programm des Gesamtsystems zur Erfassung und Analyse von Internetnutzungsstatistiken vorgestellt. Zuerst wird die Systemarchitektur erläutert. Dann wird mit Hilfe des Datenbankschemas die inhaltliche Struktur der Statistiken aufgezeigt. Dadurch wird deutlich, welche Informationen über die Benutzung des WWW-Servers vorhanden sind. Der WebSiteProfiler soll, soweit wie es möglich ist, individuelle Nutzer und Sitzungen von Nutzern oder Host-Rechnern erfassen.

4.2.1 Die Systemarchitektur des WebSiteProfilers

Zur Darstellung der Systemarchitektur des WebSiteProfilers beschränke ich mich auf das Übersichtsmodell für das System und das relationale Datenbankschema.

Aus der Sicht eines Dienstnehmers gibt der WebSiteProfiler folgende Dienste nach außen weiter:

- Erstellung von Nutzungsstatistiken über bestimmte Zeiträume hinweg.
- Aktualisierung der Datengrundlage für die Statistiken.
- Archivierung bisher erstellter Statistiken.

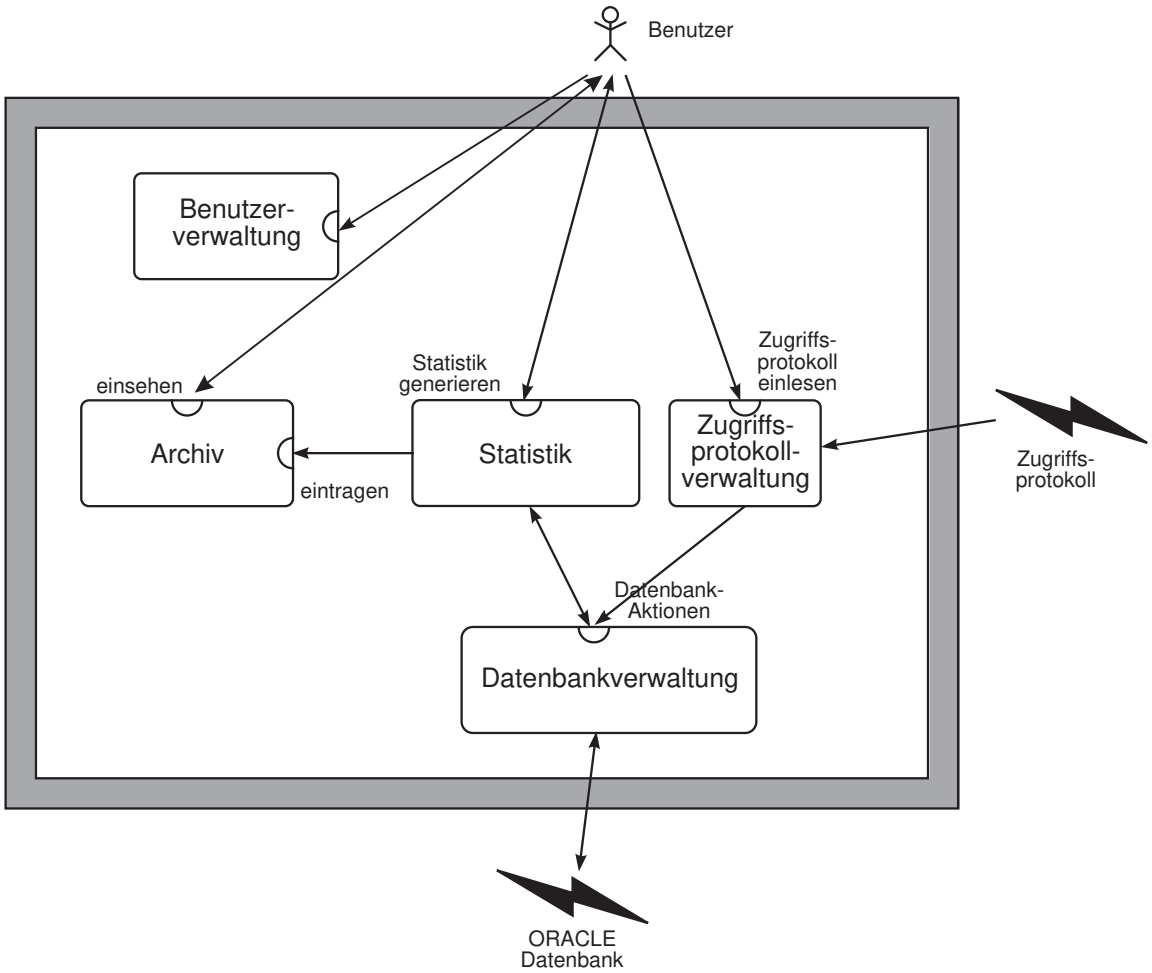

Abbildung 4.2: Übersichtsdiagramm für den WebSiteProfiler

- Verwaltung zugelassener Benutzer.

Die Datenbank für die Statistiken wird aus der Zugriffsprotokolldatei gewonnen, die von jedem WWW-Server automatisch erstellt wird. Die Zugriffsprotokolldatei wird in bestimmten Abständen auf neue Einträge hin durchsucht, die dann in die Datenbank aufgenommen werden. Der Inhalt der Datenbank wird in Abschnitt 4.2.2 vorgestellt.

Das Übersichtsmodell aus Abbildung 4.2 stellt die Komponenten des WebSiteProfilers in einer Schichtenarchitektur dar. Es hält sich an die Notation des Übersichtsmodells aus Abschitt A.2.

Außerhalb des Systems stehen die Elemente *Datenbank, Zugriffsprotokoll* und der *Benutzer*. Die *Datenbank* wird in einem eigenen System von ORACLE verwaltet. Es existieren geeignete Schnittstellen, um Anfragen an die Datenbank stellen zu können und weitere Daten einzufügen.

Das *Zugriffsprotokoll* wird vom Verwaltungsmodul ausgelesen.

Das Anfragemodul bietet den Dienst *Statistik generieren* an. Dazu wird eine Anfrage

an die Datenbank gestellt und die Antwort der Datenbank wird zu einer Grafik aufbereitet.

Das Modul Zugriffsprotokollverwaltung durchsucht die Protokolldatei nach neuen Einträgen. Sie werden in entsprechende Tupel aufgeteilt und in die Datenbank eingefügt.

Im Archivmodul werden alle erstellten Statistiken gespeichert. Man kann auf die Statistiken im Archiv entweder gezielt zugreifen oder eine Liste aller Einträge bekommen.

4.2.2 Das Datenbankschema des WebSiteProfilers

Die Miniwelt für die Datenbank des WebSiteProfilers besteht im wesentlichen aus Gegenständen des WorldWideWeb. Das zugehörige Gegenstands-Beziehungs-Modell ist in Abbildung 4.3 zu sehen. Das Gegenstands-Beziehungsmodell ist in der Notation des Objektstrukturdiagramms aus Abschnitt A.1 dargestellt. Alle Objekte sind mit ihren Attributen versehen. Um Nutzungsstatistiken über einen WWW-Server zu erstellen, benötigt man die folgenden Gegenstände und Beziehungen zwischen ihnen. [LKK93]

Benutzer Für geschützte Dokumente ist der Zugriff erst nach vorhergehender Authentifikation möglich. Dazu wird ein eindeutiger Benutzername vergeben. Bei der Anforderung von frei zugänglichen Seiten bleibt der Benutzer anonym. Daher ist der Benutzername in der Datenbank als optionales Feld gekennzeichnet. Wenn ein neuer Benutzer auf den Server zugreift, wird der Zeitpunkt seines ersten Auftretens festgehalten.

Rechner Der Rechner, von dem aus die Dokumente angefordert werden, ist durch seine eindeutige IP-Adresse identifiziert. Auch hier wird der Zeitpunkt des ersten Auftretens gespeichert.

Domäne Das Internet ist in verschiedene Domänen unterteilt. Dabei ist jeder Rechner eindeutig einer Domäne zugeordnet.

Dokument Die Dateien, die auf dem Server zum Abruf bereitliegen.

Angebote Ein WorldWideWeb-Server kann mehrere Angebote enthalten. Zu jedem Angebot gehören verschiedene Dokumente eines Anbieters.

Sitzung Wenn ein Benutzer oder ein Rechner mehrere Anfragen in einem Zusammenhang stellen, werden die Anfragen in einer Sitzung zusammengefaßt. Zu jeder Sitzung wird ihr Beginn, ihre Dauer und eine Referenz auf die nächste Sitzung des gleichen Rechners oder Benutzers gespeichert.

Anfragen sind die zentralen Gegenstände der Miniwelt. Jede erfolgreich durchgeführte Anfrage wird in der Datenbank eingetragen. Der Eintrag enthält das angeforderte Dokument, evtl. den Benutzernamen, wenn er nicht anonym bleibt, die Sitzung, aus der die Anfrage stammt und den Rechner, von dem aus die Anfrage gestellt wird. Außerdem wird der Zeitpunkt der Anfrage und die Dauer bis zur nächsten Anfrage aus der gleichen Sitzung gespeichert.

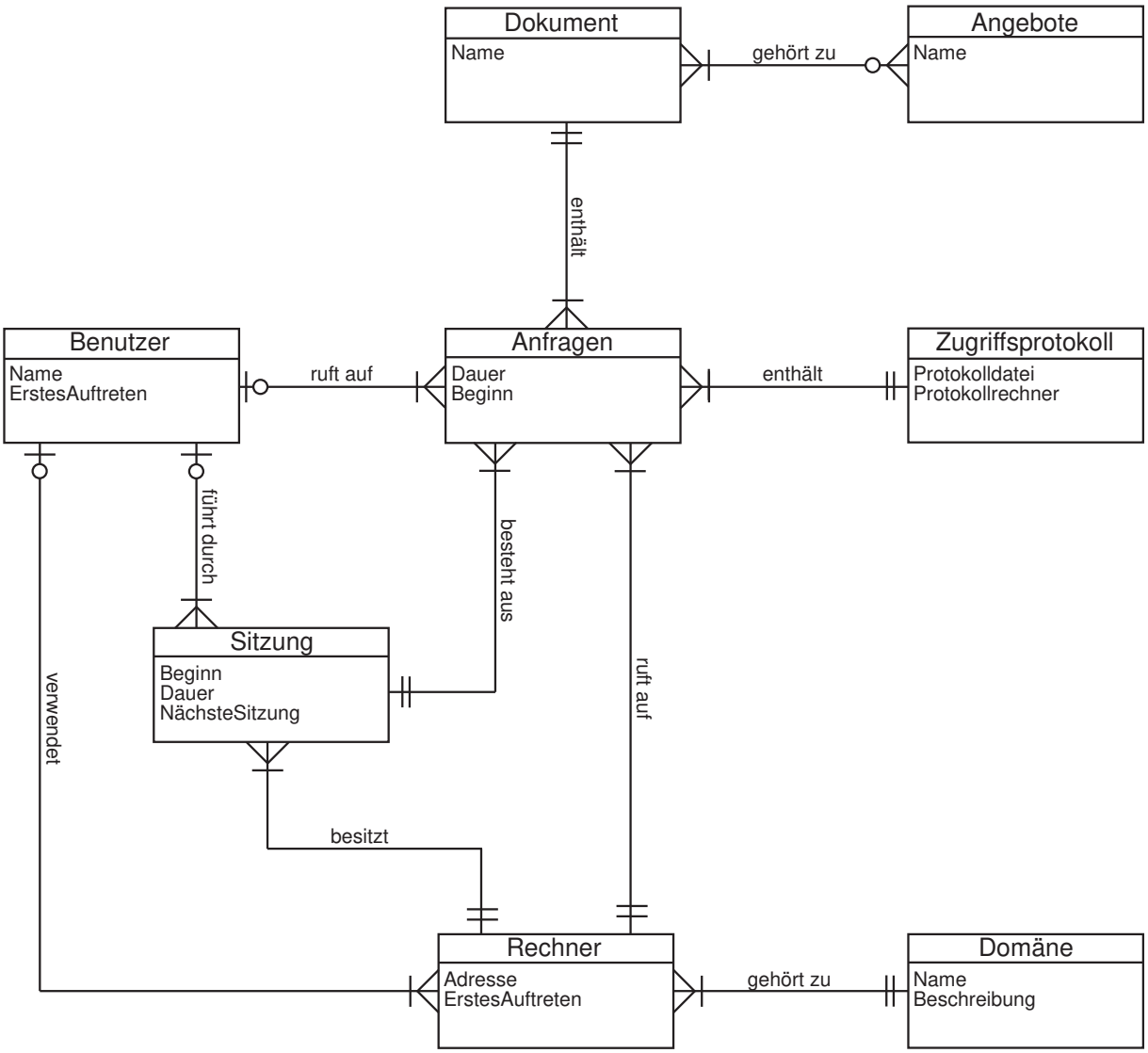

Abbildung 4.3: Objektstrukturdiagramm für die Datenbank des WebSiteProfilers

Zugriffsprotokolldatei Die Datei, in der alle Anfragen an den Server protokolliert werden.

Die Informationsquelle für die Gegenstände ist die Zugriffsprotokolldatei, die zu jedem WWW-Server gehört. In ihr stehen für jede Anforderung von Dokumenten, die an den Server gerichtet werden, folgende Daten: Die Internet-Adresse des Rechners, von dem die Anforderung kommt, eventuell ein Benutzername, wenn der Benutzer auf geschützte Seiten zugreift, die Uhrzeit der Anforderung, der Dokumentenname, ein Fehlercode und die Menge der übertragenen Daten. Diese Daten werden in der zeitlichen Reihenfolge, wie die Dokumente angefordert wurden, verzeichnet.

Auf diesem Datenbankschema sind 26 verschiedene Anfragen möglich, die in Statistiken umgesetzt werden.

Die wichtigsten davon sind:

- Anzahl der Anfrage pro Stunde an den WebServer
- Anzahl der Anfragen pro Tag an den WebServer
- Anzahl der Benutzer des WebServers pro Tag
- Anzahl der Rechner, die den WebServer benutzen, nach Tagen aggregiert.

4.3 Der MultiSiteAnalyzer

In diesem Abschnitt wird zuerst auf den inhaltlichen Zusammenhang von Servern im World-WideWeb eingegangen. Daraus ergeben sich Anforderungen an das System, das anschließend vorgestellt wird. Am Ende werden Kommunikationsmöglichkeiten für die Programme WebSiteProfiler und MultiSiteAnalyzer aufgezeigt.

4.3.1 Der inhaltliche Zusammenhang verschiedener WWW-Server

Ganz kurz zusammengefaßt bildet der MultiSiteAnalyzer den integrativen Teil einer verteilten Datenbankanwendung für Internetnutzungsstatistiken. Während die Erstellung dieser Statistiken schon in Abschnitt 4.2 behandelt wurde, wird jetzt auf den Zusammenhang dieser Statistiken, die über das ganze Internet verteilt an jeder Website anfallen, eingegangen.

Dieser Zusammenhang ist ausschnittsweise und beispielhaft in der Abbildung 4.4 dargestellt. Am linken Bildrand sind die Nutzer des Internet eingezeichnet. Am oberen Bildrand stehen Verbände, die das Mediengeschehen kontrollieren und messen wollen, zum Beispiel die GfK (Gesellschaft für Konsumforschung) oder IVW (Informationsgesellschaft über die Verbreitung von Werbeträgern). Sie nehmen ihre Aufgabe bei den „alten“ Medien wie Fernsehen und Zeitungen durch die Veröffentlichung von Einschaltquoten und der Anzahl

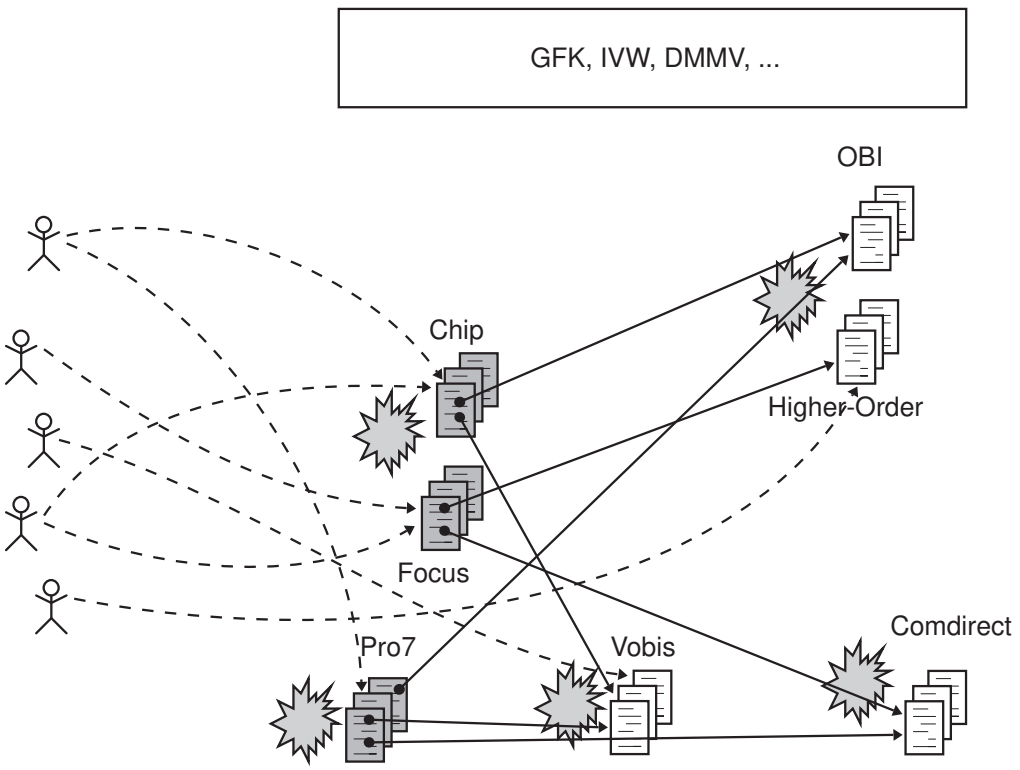

Abbildung 4.4: Der Zusammenhang von Internetnutzungsstatistiken

verkaufter Exemplare einer Zeitung wahr. Die Verbände wollen solche Zahlen auch für das Internet erheben und verbreiten.

Im Zentrum stehen die Websites verschiedener Informationsanbieter. Sie können wiederum beliebig viele Angebote bereitstellen. Die Websites sind mit Sternen dargestellt, die Angebote durch Seiten. Die Angebote werden außerdem nach Medien und Firmen unterschieden. Auch wenn diese Unterscheidung in der Realität nicht haltbar ist, soll sie hier zur Verdeutlichung des Sachverhaltes dienen. Die Medien zeichnen sich durch Anzeigen, die die Firmen in deren Angeboten schalten, aus. Die Anzeigen können als Links gestaltet werden, mit denen die Leser zu den Firmenangeboten geleitet werden.

Der MultiSiteAnalyzer soll jetzt zum einen quasi die Links rückwärts verfolgen und die Nutzungsstatistiken über die Seiten, auf denen die Anzeigen geschaltet sind, einholen, miteinander vergleichen und gegenüberstellen. Für die Medienverbände zum anderen spielen die Links auf andere Sites keine Rolle, sie untersuchen verschiedene Sites unabhängig von ihrem Inhalt und verlegen den Zusammenhang auf eine andere Ebene. In beiden Fällen sollen die Nutzungsstatistiken auf quantitativer Ebene verglichen werden. Dazu bieten sich Ranglisten an.

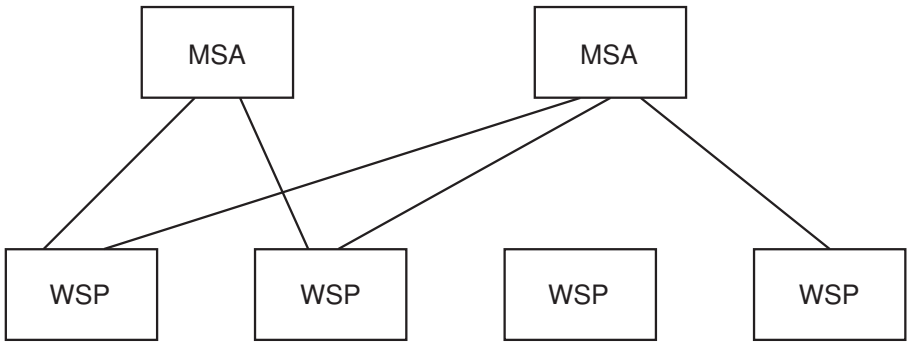

Abbildung 4.5: Die Kommunikation von WebSiteProfilern und MultiSiteAnalyzern

4.3.2 Die Systemübersicht

Der MultiSiteAnalyzer bildet einen Teil einer verteilten Datenbankanwendung. Dabei handelt es sich nach [ÖV94] um ein verteiltes, homogenes System vieler Datenbanken. Die WebSiteProfiler, die verteilt im Internet liegen, bilden die einzelnen Datenbanken. Der MultiSiteAnalyzer stellt Anfragen an diese Datenbanken und stellt die Einzelergebnisse in einem Anfrageergebnis zusammen.

Bei der Kommunikation von WebSiteProfilern und MultiSiteAnalyzern ist prinzipiell das Szenario aus Abbildung 4.5 denkbar. Im Internet arbeiten mehrere MultiSiteAnalyzer, von denen jeder mit mehreren WebSiteProfilern kommuniziert. Da die Angebote einer Website für mehrere Vergleiche interessant sind, werden wiederum verschiedene MultiSiteAnalyzer ihre Statistiken anfordern.

Die WebSiteProfiler sind autonome Systeme, die ihre Datenbestände selbst verwalten. Dem MultiSiteAnalyzer, und nur ihm, wird erlaubt, von außen Anfragen an die Datenbanken zu stellen. Sonst darf er keine Änderungen an den Daten vornehmen.

Desweiteren müssen die Daten, die von den WebSiteProfilern gewonnen werden, im MultiSiteAnalyzer aufbereitet werden. Die oben genannten Medienverbände weisen ihre Statistiken in der Regel als Rangliste aus. Daran orientiert sich der MultiSiteAnalyzer. Diese Rangliste wird dem Benutzer als Ergebnis seiner Anfrage geliefert.

Insgesamt kommt man zu dem Systemaufbau aus Abbildung 4.6. Auch in diesem Programm wird eine Benutzerverwaltung und ein Archiv eingesetzt.

4.3.3 Kommunikation des MultiSiteAnalyzers mit den WebSiteProfilern

Die offene Frage nach der Vorstellung beider Systemübersichten ist die nach der Kommunikation des MultiSiteAnalyzers mit den verschiedenen WebSiteProfilern. Hierfür sind verschiedenste Möglichkeiten denkbar. So könnte die Kommunikation unabhängig von existierenden Systemen von Grund auf selbst entwickelt werden. Dabei muß der Programmierer viele Probleme selbst lösen. Eine andere Möglichkeit wäre es, in beiden Programmen Kommunikationsschnittstellen vorzusehen, die mit einem geeigneten System zusammen-

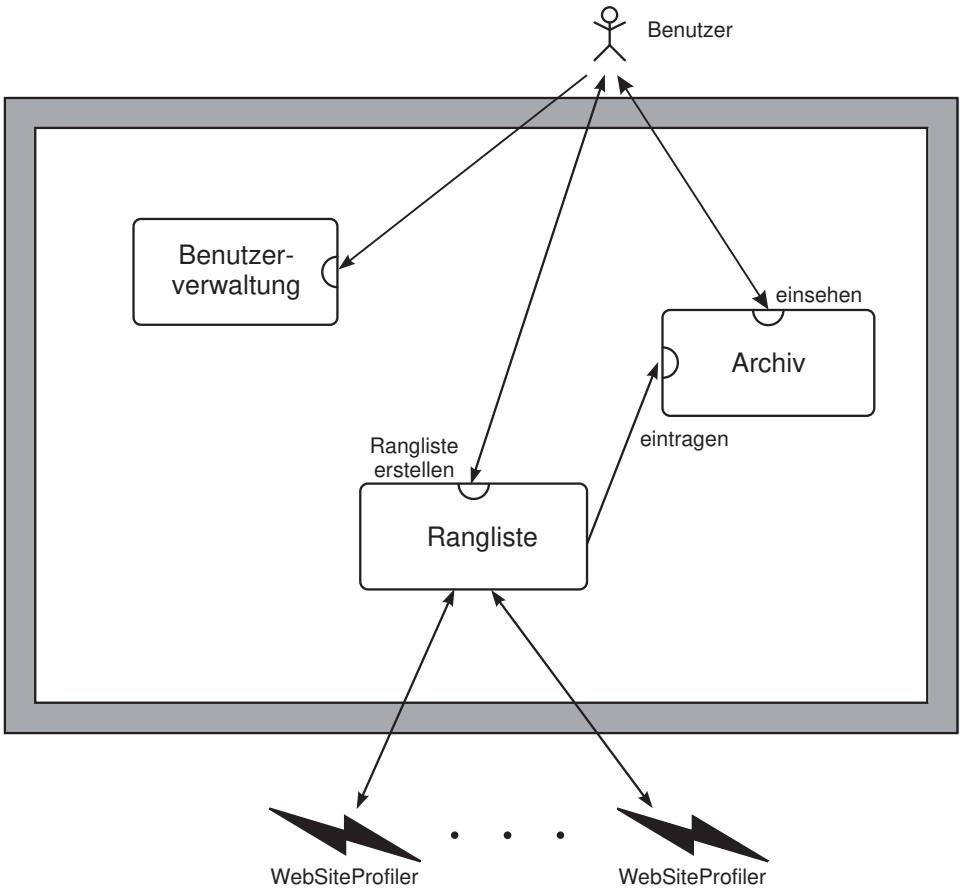

Abbildung 4.6: Die Systemübersicht für den MultiSiteAnalyzer

arbeiten. Man könnte außerdem beide Programme in ein verteiltes System wie CORBA oder das DCE einbetten, um diese mächtigen Kommunikationsmöglichkeiten zu nutzen. Alle bisherigen Vorschläge haben gemeinsam, daß neben der Kommunikationsschnittstelle in beiden Programmen eine geeignete Benutzerschnittstelle vorhanden sein muß.

Um diesen Nachteil auszugleichen, wird im nächsten Kapitel das Modell der Business Conversations vorgestellt. Mit diesem Modell ist es möglich, selbstentwickelte Anwendungen in eine Umgebung einzufügen, in der sowohl die Kommunikation mit anderen Anwendungen als auch die Benutzung durch menschliche Akteure abgewickelt wird.

Kapitel 5

Das Modell der Business Conversations

Nachdem in Kapitel 4 ein Beispiel für eine kooperative Anwendung vorgestellt wurde, soll in diesem Kapitel ein Modell für eine Umgebung kooperativer Anwendungen dargestellt werden. Dieses Modell wird im folgenden erläutert. Außerdem wird der dynamische Aspekt der Kooperation beschrieben und anhand eines Beispiels verdeutlicht.

5.1 Die Motivation für das Modell

Kooperative Informationssysteme bestehen in Anlehnung an „Cooperative Information Systems: A Manifesto“ [DDJ+96] aus zahlreichen Systemen, die über weite und komplexe Netzwerke verteilt sind. Kooperation bedeutet, daß mehrere Akteure auf ein gemeinsames Ziel hinarbeiten. Generell verhalten sich Akteure kooperativ, wenn sie mit anderen Akteuren ihrer Umgebung Ziele teilen, um sie gemeinsam zu erreichen. Dazu müssen sie Informationen austauschen.

Nachfolgend sollen auch Computersysteme als Akteure, die bestimmte Tätigkeiten verrichten, betrachtet werden. Sie können zum Beispiel einen Bericht drucken oder Informationen bereithalten. Dabei kann das erforderliche Informationssystem als Ziel betrachtet werden. Wenn das Ziel in der Umgebung des Systems mit anderen Akteuren geteilt wird, und das Informationssystem einen positiven Beitrag zum Erreichen des Ziels leistet, spricht man von einem *kooperativen* Informationssystem. Zusätzliche Voraussetzungen sind die Möglichkeiten zum Informationssaustausch und daß andere Systeme auf die Funktionalität des Informationssystems zugreifen können.

Kooperative Informationssysteme unterstützen die Änderungen der Anforderungen an das System und des Systems selbst über lange Zeiträume hinweg. Sie sollen eine Abbildung des Geschäftsmodells eines Unternehmens auf das Systemmodell ermöglichen. In [DDJ+96] wird für kooperative Informationssysteme eine Schichtenarchitektur aus vier Schichten vorgeschlagen. Die Systemschicht bildet die Basis des Systems. Darüber sorgt eine Systemintegrationsschicht für die Zusammenarbeit verschiedener verteilter Systeme.

Die Arbeitsgruppenschicht und die Unternehmensschicht bilden organisatorische Aspekte des Unternehmens auf das Informationssystem ab. Die Aspekte der Kooperation finden sich in allen vier Schichten des Systems wieder. Im folgenden wird ein Softwaremodell für die Systemintegrationsschicht vorgestellt. Es unterstützt die Zusammenarbeit autonomer menschlicher oder maschineller Akteure, indem es einen Rahmen zur Koordination ihrer langandauernden Aktivitäten anbietet. Die langandauernden Aktivitäten zur Kooperation werden über Konversationen (engl. Business Conversations) modelliert.

Dieses Modell soll höhere Modelle der weiteren Systemschichten unterstützen. Dabei handelt es sich in der Arbeitsgruppenschicht um Modelle für Geschäftsprozesse, Workflow-Management oder Unterstützung der Gruppenarbeit. In der Unternehmensschicht beziehen sich die Modelle auf virtuelle Unternehmen, elektronische Handelssysteme und dezentralisierte Informationsverarbeitung.

Folgt man der Argumentation des Manifesto, soll die Systemintegrationsschicht den Systemprogrammierer durch die Abstraktion von folgenden Punkten unterstützen:

1. Kooperation über die Zeit. Die Lebensdauer aller maschinellen Akteure richtet sich nach der Dauer der Geschäftsprozesse, in denen sie eingesetzt werden. Dabei sind sie unabhängig von unterliegenden Systemgrößen wie der Programmiersprache, Lebensdauer der Betriebssystemprozesse oder Datenbankschemata.

2. Kooperation innerhalb des Raums. Maschinelle Akteure können innerhalb einer physikalisch verteilten Umgebung migrieren. Dabei sind sie nicht von speziellen Systemplattformen oder Organisationsstrukturen abhängig.

3. Kooperation in mehreren Arbeitsformen. Softwaresysteme sollen für maschinelle Akteure einheitlich benutzbar sein. Dabei werden die folgenden Arbeitsformen gleichbehandelt: einfache Stapelverarbeitung, Transaktionverarbeitung, direkte Manipulation von menschlichen Akteueren mittels formularbasierter Eingabemasken, rechnerunterstützte Zusammenarbeit menschlicher Akteure, rechnergestütztes Workflow-Management und automatische Informationsverarbeitung mittels mobiler Softwareagenten. Ebenfalls sollen verschiedene Medien auf gleiche Art unterstützt werden: E-Mail-Nachrichten, EDI-Nachrichten, HTTP-Requests, CORBA Object-Requests oder entfernte Prozeduraufrufe.

Wenn diese Abstraktion gelingt, kann sich der Programmierer auf das *was* und *wie* des speziellen Problems konzentrieren und das *wann* (zum Beispiel Stapelverarbeitung, interaktive Sitzungen oder langandauende Konversationen mit einem Kunden), *wo* (zum Beispiel zentralisierter Hostrechner, Server innerhalbs eines Unternehmens oder ein Laptop eines Verkaufsmitarbeiters) und *wer* (zum Beispiel Kunde, Verkäufer, Workflow-Mangement-System oder Internetagent) vernachlässigen. Die Applikationen werden von überflüssigem Ballast wie Datentransfer, Synchronisation und Details der Benutzeroberfläche befreit, um Änderungen der Lebensdauer, der Systemplattform oder der Arbeitsform besser als bisher durchführen zu können.

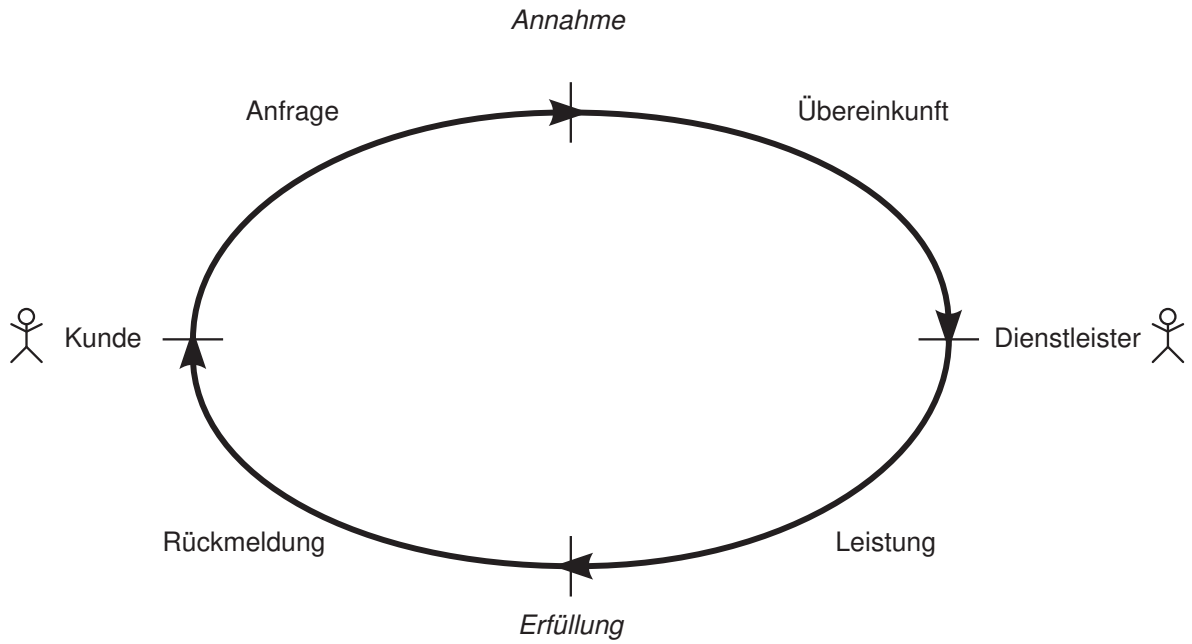

Abbildung 5.1: Kommunikation nach der Language/Action Perspektive

Die Grundlagen für die ersten beiden Punkte wurden mit dem plattformunabhängigen Tycoon-System und der Entwicklung persistenter, mobiler Threads gelegt. Den dritten Punkt sollen die folgenden Business Conversations behandeln.

5.2 Die Grundlagen des Modells

Die Basis für dieses Modell bilden Arbeiten aus dem Bereich des Computer-Support for Cooperative Work (CSCW) von Fernando Flores und Terry Winograd und deren Theorie der Sprechakte (*engl.: Speech Act Theory, Language/Action Perspective*) [Win87, FGHW88]. Sie betrachten Sprache als wichtigste Dimension für kooperative Zusammenarbeit von Menschen. Eine zentrale These ihres Modells ist, daß Menschen durch Sprache kooperieren. Daraus ergibt sich die *Language/Action Perspective* auf die Handlungen der Menschen. Linguistische Untersuchungen einer Sprache unterscheiden Syntax, Semantik und Pragmatik. Die Pragmatik behandelt die Probleme der Benutzung von Sprache. Ein klassisches Beispiel ist der Satz „In diesem Raum ist es kalt“. Wenn ein Herr ihn zu seinem Diener sagt, soll er beim Diener eine Aktion auslösen, zum Beispiel das Fenster zu schließen.

Aus der Perspektive dieses Ansatzes ist die Rolle der Sprache bei der Auslösung und Interpretation von Aktionen der wichtigste Aspekt von Untersuchungen. Eine wichtige Einheit der Kommunikation ist dabei der Sprechakt. Er kann wiederum in weitere Akte unterteilt werden. Die Sprechakte sind alle Teil einer größeren Gesprächsstruktur. Eine wichtige Klasse der Sprechakte sind die einfachen *Conversations for Action.* [Win87]

Eine Konversation mit einer Aktion als Ziel koordiniert diese Sprechakte. Durch Sprech-

akte vereinbaren die beteiligten Kommunikationspartner wechselseitig zukünftige Aktionen und bilden eine gemeinsame Vergangenheit, vor deren Hintergrund neue Sprechakte gebildet und interpretiert werden.

Das Modell der *Business Conversations* bildet einen Rahmen für die anwendungsnahe Modellierung von Informationssystemen auf der Grundlage solcher Sprechakte. Es definiert eine Terminologie und eine Semantik, mit der Informationssysteme aus der Sicht der Sprechakte modelliert und spezifiziert werden können. Dabei orientiert es sich begifflich an der Domäne der Geschäftsprozesse und erfaßt neben den statischen Aspekten eines Dienstes gleichermaßen die dynamische Interaktion von Kunde und Dienstleister.

Kernpunkt des Modells ist die Annahme, daß die Kommunikationsmuster der Sprechakte universell sind, d.h. sie sind von Art der Aktoren (Software, Menschen) und von den verwendeten Kommunikationsmedien und -protokollen (HTTP, e-Mail, CORBA, ...) unabhängig.

Sowohl die Interaktion zwischen menschlichen Akteuren, als auch die Interaktion zwischen Menschen und maschinellen Akteuren oder gar die Interaktion zwischen zwei maschinellen Akteuren, ist innerhalb des Modells transparent.

Das Modell der *Business Conversations* beschreibt die zielgerichtete Kommunikation zwischen zwei Akteuren in den Rollen Kunde und Dienstleister. Das Modell basiert auf der Beobachtung, daß eine Kommunikation mit dem Ziel einer Aktion zwischen einem Kunden und einem Dienstleister in einen Zyklus mit vier Phasen unterteilt werden kann. Diese Phasen sind in der Abbildung 5.1 zu sehen.

Die Beziehung zwischen einem Kunden und einem Dienstleister folgt immer dem skizzierten Ablauf: Erst wird der Kunde beim Dienstleister unverbindlich nach einer Dienstleistung anfragen. Am Ende dieses Schrittes steht dann die Verhandlung mit dem Dienstleister, einen gewählten Dienst tatsächlich verbindlich zu erbringen. Das ist Aufgabe der zweiten Phase, der *Übereinkunft*. Das Erbringen der vereinbarten Leistung geschieht in der dritten Phase. An deren Ende ist aus der Sicht des Dienstleisters die Leistung erbracht. In der vierten Phase, der *Rückmeldung*, wird der Kunde dem Dienstleister mitteilen, ob die erbrachte Leistung seinen Anforderungen gerecht geworden ist. Nur wenn auch diese Phase erfolgreich abgeschlossen ist, ist das gemeinsame Ziel von Kunde und Dienstleister, und damit das Ziel ihrer Kommunikation, erreicht. Diese vier Phasen werden in einer Konversation zusammengefaßt.

Der Zyklus kann durch den Verzicht auf einzelne Phasen vereinfacht werden, wenn die betrachtete Kunde-Dienstleister-Beziehung dies erlaubt. Zum Beispiel können die Phasen Anfrage und Übereinkunft zusammengefaßt werden. Dabei bleibt jedoch sowohl die Reihenfolge der Phasen als auch der Zyklus erhalten.

Das Modell der *Business Conversations* ist komplementär zu bereits bekannten Modellen (wie zum Beispiel Mainstream Objects, Entity-Relationship-Modell), die die Modellierung von Informationssystemen gestatten (siehe Abb. 5.2). Die Business Conversations versuchen, durch die Strukturierung von Dienstleistungsprozessen eine anwendungsnähere Modellierung zu unterstützen. Dadurch wird der Abstand zwischen der realen Welt und ihrer Abbildung durch Computersysteme verkleinert.

Zum Beispiel wurde der WebSiteProfiler mit mehreren Modellen entwickelt. Mit Hilfe

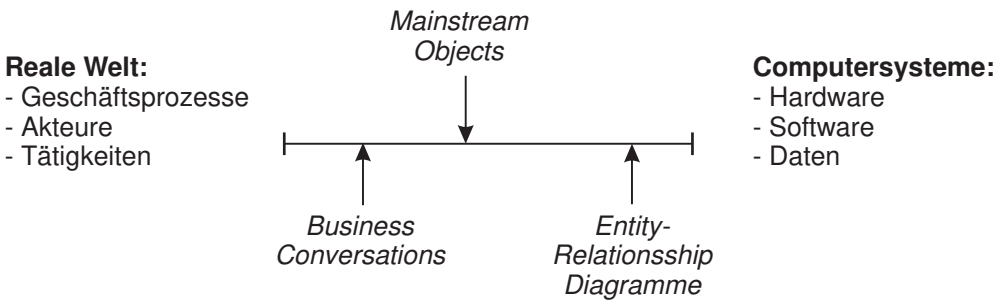

Abbildung 5.2: Modellbildung für Informationssysteme

des Entity-Relationship-Modells wurde die Darstellung der Daten in einem Datenbankschema entworfen. Das Ergebnis dieses Schrittes ist in Abschnitt 4.2.2 zu sehen. Damit ist die datenzentrierte Darstellung des WebSiteProfilers gegeben. Jedoch kann mit diesem Ergebnis noch nichts über Programmabläufe innerhalb des WebSiteProfilers ausgesagt werden. Antworten auf die Frage, ob und wie mehrere Benutzer mit dem System gleichzeitig arbeiten können, welche Aufgaben nebenläufig ausgeführt werden können oder welche Reihenfolge in einzelnen Funktionen eingehalten werden muß, kann man mit dem Modell der Mainstream Objects aus Anhang A geben. Damit hat man eine objektzentrierte Darstellung erhalten. Für die Modellierung der Kooperation mit anderen Systemen wird das Modell der Business Conversations benutzt.

5.3 Die Konzepte des Modells

Im Modell gibt es zwei Ebenen, die im folgenden getrennt beschrieben werden:

- Die Ebene der *Konversationsspezifikationen*. Auf dieser Ebene wird die Struktur möglicher Konversation beschrieben. Diese Ebene entspricht der Typebene einer Programmiersprache, mit dem Unterschied, daß Konversationsspezifikationen eine Laufzeitrepräsentation besitzen. Konversationsspezifikationen bilden Klassen von Konversationen, ihren Instanzen.

- Die Ebene der *Konversationsinstanzen*. Gegenstand dieser Ebene sind konkrete Konversationen, die zwischen einem Kunden und einem Dienstleister stattfinden. Konversationen sind immer konform zu einer zugehörigen Konversationsspezifikation. Sie werden anhand dieser Spezifikation erzeugt und sind an ein Kunde-Dienstleister-Paar gebunden.

Bevor diese beiden Ebenen erläutert werden, wird zunächst der generelle Ablauf von Konversationen erklärt.

5.3.1 Das Verarbeitungsmodell von Konversationen

In einer Konversation wird jede zielgerichtete Interaktion (also jeder Sprechakt aus der Language/Action Perspective) mit *Dialogen* realisiert. Mit Hilfe dieser Dialoge wird Information so strukturiert, daß der jeweilige Kommunikationspartner die im Sprechakt enthaltene Information seinem eigenen Kontext zuordnen kann.

Jeder Dialog beschreibt einen Zustand in der *Konversation* zwischen einem Kunden und einem Dienstleister. Im Gegensatz zur zeitlich ausgedehnten Konversation besitzt ein Dialog keine nach außen sichtbare Zwischenzustände.

Zustandsübergänge, d.h. das Beginnen eines neuen Dialogs innerhalb einer Konversation, werden durch die Auswertung und Verarbeitung (die Aktion) einer Anfrage beim Dienstleister erreicht. Dabei ist der Kunde derjenige Akteur, welcher den Zeitpunkt des Zustandsüberganges bestimmt: Indem er eine Anfrage an den Dienstleister stellt, führt dieser seine Regel, die er mit der Anfrage assoziiert hat, aus und generiert einen neuen (Antwort-)Dialog für den Kunden. Die Menge der möglichen Anfragen, die ein Kunde ausgehend von einem Zustand einer Konversation an einen Dienstleister richten kann, ist in dem Dialog enthalten, der den aktuellen Zustand der Konversation repräsentiert. Es sind genau zwei Anfragen definiert, die nicht in der Menge der Anfragen enthalten sind, die ein Dialog vorgibt:

- Die initiale Anfrage. Ein Kunde muß eine Konversation mit einem Dienstleister beginnen können, ohne sich vorher bereits in einer Konversation mit ihm zu befinden. In diesem Fall kann der Kunde genau eine Anfrage an den Dienstleister stellen, nämlich die nach der Eröffnung der Konversation.

- Eine spezielle Anfrage nach dem Abbruch der Konversation. Diese Anfrage löst ein Kunde implizit aus, wenn er über eine von Dienstleister vorgegebene Zeistspanne lang keine Anfrage an den Dienstleister stellt. Tatsächlich ist dies keine wirklich vom Kunden stammende Anfrage, sondern wird vom System des Dienstleisters im Namen des Kunden generiert. Außerdem muß dem Kunden der explizite Abbruch der Konversation erlaubt werden.

Eine Konversation zwischen einem Kunden und einem Dienstleister wird mit diesen Mechanismen auf eine Abfolge von Dialogen und Anfragen übertragen.

Der Dienstleister gibt jeweils den Dialog vor, auf den der Kunde reagieren kann. Der Kunde ist jedoch immer der *Initiator* der Konversation und der Dienstleister der *Akzeptor*. Innerhalb einer Konversation kann der Dienstleister jedoch in der Rolle eines Kunden eine andere Konversation mit einem anderen Dienstleister initiieren, wenn das zur Ausführung seiner Regel notwendig ist.

Mit der Technik der Sub-Konversation ist eine Mischform der Kunde-Dienstleister-Beziehung möglich (Abbildung 5.3). Dabei wird der Dienstleister A_2 innerhalb einer Konversation mit einem Kunden A_1 selbst zum Kunden eines weiteren Dienstleisters A_3.

Dialoge ohne einen Folgedialog, d.h. ohne eine mögliche ausgehende *Anfrage*, sind *finale* Dialoge einer Konversation.

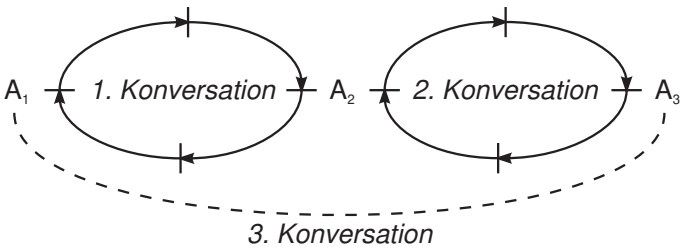

Abbildung 5.3: Verfeinerung mittels Sub-Konversationen

Jede Konversation besitzt eine definierte Menge von finalen Dialogen, die sie erreichen kann. Es gibt einen Finaldialog für das erfolgreiche Ende der Konversation (d.h. sowohl Kunde als auch Dienstleister bestätigen das erfolgreiche Erreichen des gemeinsamen Ziels) sowie einen expliziten finalen Fehlerdialog.

Die Entscheidung, unter welchen Umständen welcher Dialog zu welcher Phase zugeordnet wird, ist eine Designentscheidung bei der Anwendungsentwicklung.

Mit diesem definierten Verarbeitungsmodell bleiben beide Teilnehmer der Konversation autonome Einheiten. Wenn man diese Abläufe mit entfernten Methodenaufrufen durchführen würde, hätte der Aufrufer Bindungen an Objekte, die seine Autonomie einschränken würden. Der Dienstleister müßte sich zum Beispiel gegen alle möglichen Fehler, die beim Aufruf seiner Methoden auftreten können, absichern.

5.3.2 Konversationsspezifikationen

Die Abbildung 5.4 zeigt ein Modell für Spezifikationen von Konversationen. Das Diagramm ist in der Notation der Mainstream Objects, die im Anhang A erläutert wird, dargestellt. Alle Begriffe aus dem Modell werden im Glossar im Anhang B in ihrer deutschen und englischen Schreibweise erläutert. Die Konversationspezifikationen legen fest, welche Form die Konversationsobjekte später haben sollen. Sie umfaßt sowohl den statischen Teil der Beziehung zwischen Kunde und Dienstleister, d.h. die Datenmodellierung für den Informationsaustausch, als auch den dynamischen Teil der Beziehung, die Prozeßmodellierung für den Informationsaustausch. Beide Partner können die Spezifikation daraufhin untersuchen, ob die Konversation in ihren jeweiligen Kontext und zu ihrer Sicht auf das zu erreichende Ziel paßt.

Nach diesem Modell enthält jede Konversationsspezifikation mehrere abstrakte Dialogspezifikationen. Eine Dialogspezifikation ist als initialer Dialog ausgezeichnet, jede Konversation startet mit diesem Dialog. Die abstrakten Dialogspezifikationen bestehen entweder aus Spezifikationen von Sub-Konversationen oder von (konkreten) Dialogen. Mit den Spezifikationen von Sub-Konversationen werden Konversationsspezifikationen als Dialogspezifikationen in anderen Konversationspezifikationen wiederverwendet. Dazu können sie auf geeignete Art und Weise mit dem *Modifier* verändert werden. Dialogspezifikationen bestehen aus Inhaltsspezifikationen und Spezifikationen für Anforderungen. Jede Anforderungsspezifikation ist über die *replies-with-Beziehung* mit der Spezifikation des folgenden

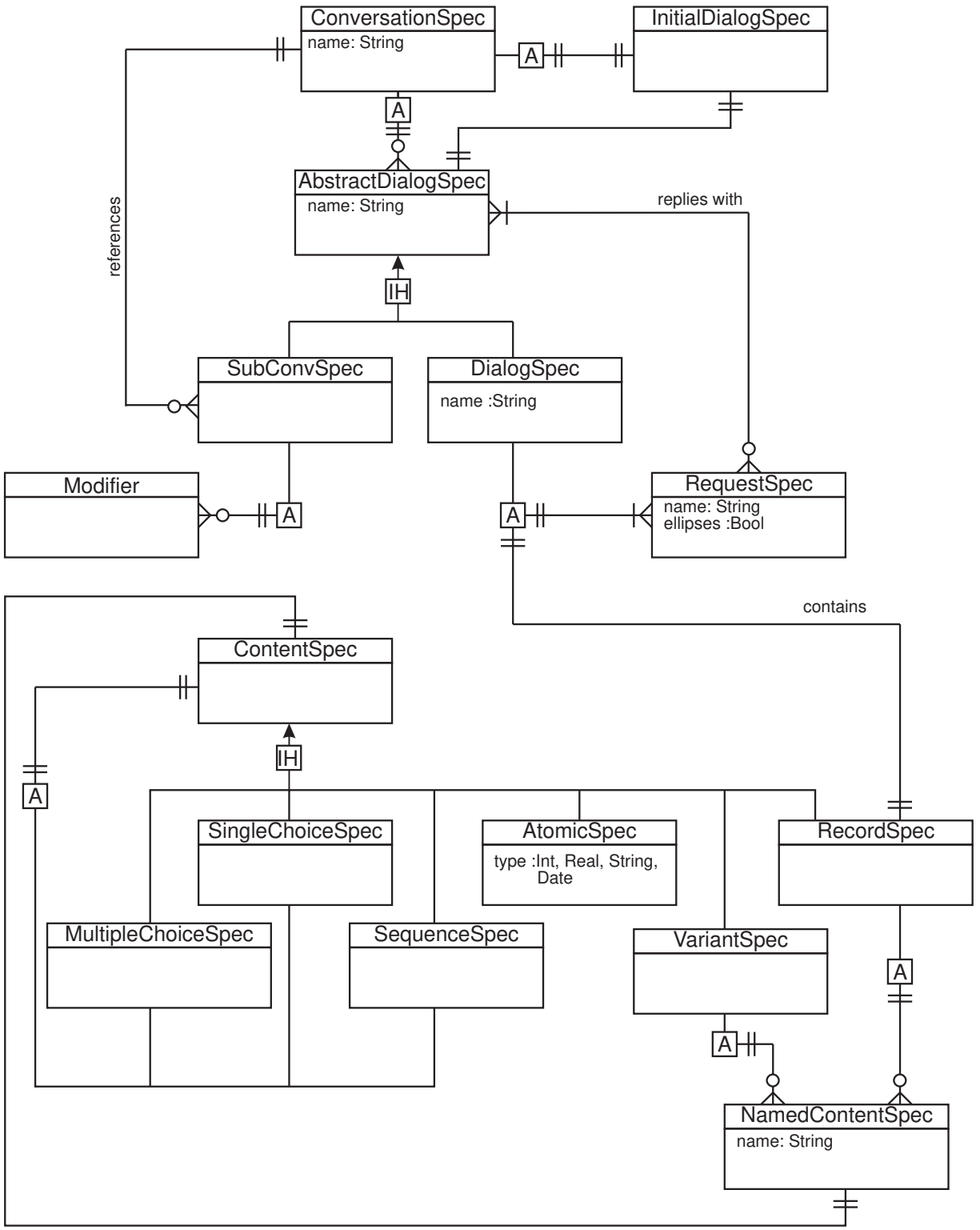

Abbildung 5.4: Spezifikationen von Konversationen, Dialogen und Inhalten

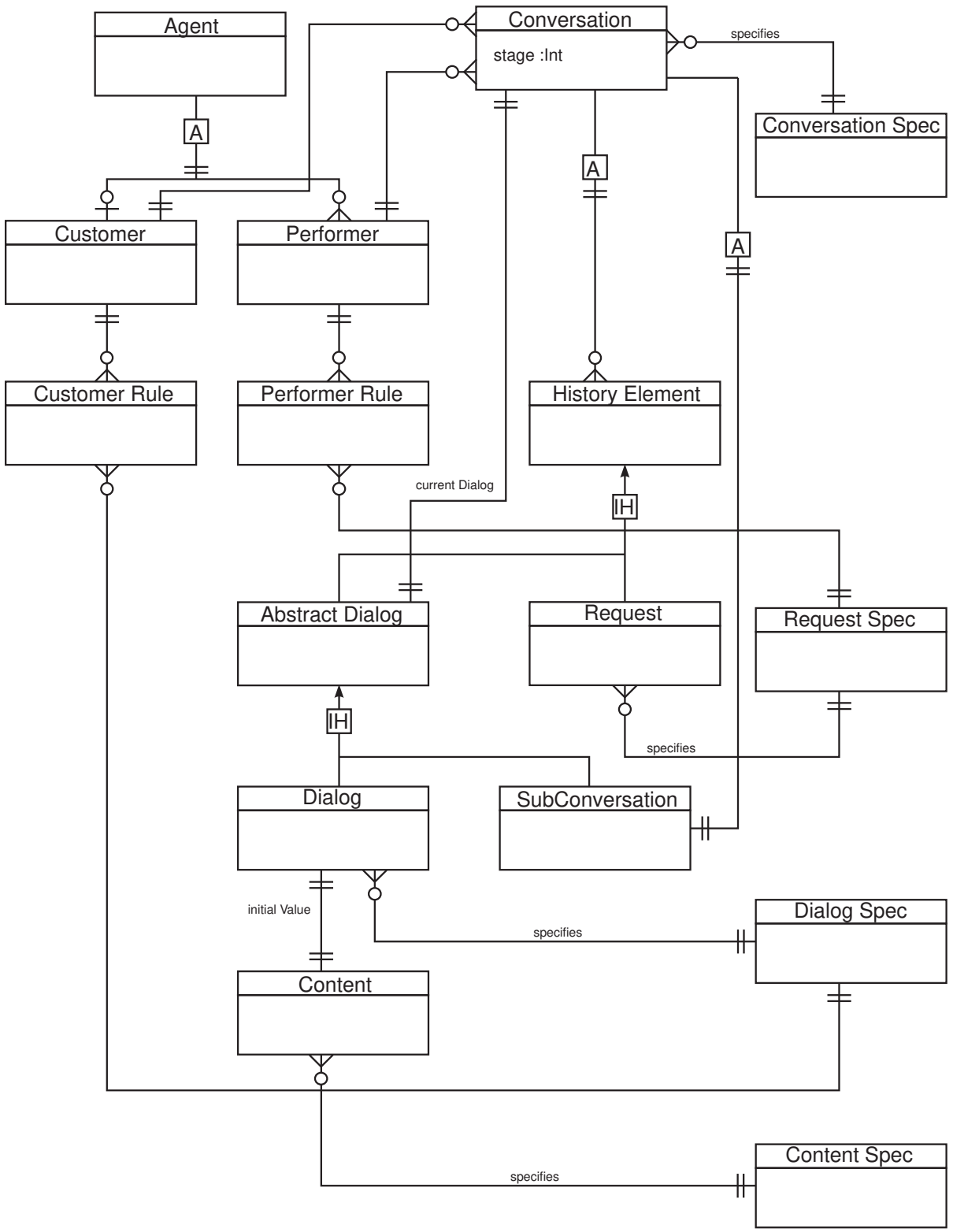

Abbildung 5.5: Instanzen von Konversationen, Dialogen und Inhalten

Dialogs verbunden. Die boolesche Variable *ellipses* zeigt an, daß sich die Aktion, die beim Dienstleister auf die Auswertung des Dialoges folgt, über mehrere Dialoge, deren Spezifikation nicht sichtbar ist, hinziehen kann.

Die Inhalte können auf verschiedene Art und Weise spezifiziert werden. Zum einen gibt es einfache Basistypen wie ganze Zahlen, rationale Zahlen, Zeichenketten oder einen Datumstyp. Komplexe Inhalte können mit Records, Varianten, Auswahllisten oder Listen modelliert werden. Dabei sind auch Schachtelungen zugelassen. Für alle Inhaltstypen existieren Operationen zum Erzeugen der Spezifikationen. Die Operationen auf den Objekten sind in der Abbildung aus Gründen der Übersichtlichkeit nicht dargestellt.

Auf der Ebene der Konversationsspezifikationen können Meta-Dienste erstellt werden, die auf diesen Spezifikationsobjekten operieren. In erster Linie sind das grafische Editoren oder Anzeiger, Modellverifikatoren, Spezifikationsmakler und ähnliches mehr. Wichtiger Nutzen der Spezifikationen ist ihre Überprüfbarkeit, durch die eine gesicherte maschinelle Abwicklung von Konversationen unter Beibehaltung der Autonomie der Partner ermöglicht wird.

Alle Spezifikationen sind Objekte erster Klasse. Sie können so benannt oder verschickt werden. Sie besitzen eine Laufzeitrepräsentation.

In der Abbildung 5.6 ist die Grammatik für die Konstruktion von Konversationsspezifikationen dargestellt. Die Notation ist aus der Darstellung der Programmiersprache TL in [Mat93] übernommen.

5.3.3 Konversationsinstanzen

Die Abbildung 5.5 zeigt den Aufbau von Instanzen von Konversationen und den Zusammenhang mit Spezifikationen.

Von einer Konversationsspezifikation können beliebig viele Konversationen instanziiert werden. Jede Konversation befindet sich zu einem Zeitpunkt in einem Quadranten der Ellipse aus Abbildung 5.1. Sie erlaubt den Zugriff auf den aktuellen Dialog (*current-dialog-Beziehung*) und enthält eine Liste von vergangen Dialogen, die durch *History-Element-Objekte* dargestellt werden. Mit den abstrakten Dialogen werden wie bei den Spezifikationen Subkonversationen und konkrete Dialoge modelliert. Jeder Dialog wird durch eine Dialogspezifikation inhaltlich vorgegeben. Aus dem Dialog heraus ist der Inhalt, der aus der Spezifikation erzeugt wurde, erreichbar. Die Objektstruktur für die Instanzen der Inhaltsspezifikationen sind in der Abbildung 5.7 zu sehen. Für jeden Inhaltstyp sind bestimmte Operationen definiert:

- Die atomaren Typen können gelesen und gesetzt werden.
- In der Liste für eine einfache Auswahl können die Alternativen gesetzt werden, man kann alle Alternativen auslesen und eine davon auswählen. Außerdem kann die gewählte Alternative bestimmt werden.
- In der Liste für mehrere Wahlmöglichkeiten können zusätzlich zu den Operationen bei einfacher Auswahl Alternativen deselektiert werden.

Conversation ::=
CONV *ConvIdentifier* ***WITH*** *InitialDialog* { *Dialog* } ***END***;

IntialDialog ::=
Dialog;

Dialog ::=
DIALOG *DialogIdentifier* ***WITH*** { *Request* } [*Content*] ***END***;

Request ::=
REQUEST *RequestIdentifier* ***TO*** *DialogIdentifier* { *DialogIdentifier* }
(***ELLIPSES*** | ***NOELLIPSES***) ***END***;

Content ::=
CONTENT *NamedContent* { *NamedContent* } ***END***;

NamedContent ::=
ContentIdentifier ***OF*** *Type;*

Type ::=
BasicType | *CompoundType;*

BasicType ::=
INT | ***REAL*** | ***STRING*** | ***DATE***;

CompoundType ::=
REC *NamedContent* { *NamedContent* } ***END*** |
VARIANT *NamedContent* { *NamedContent* } ***END*** |
MCHOICE OF *Type* |
SCHOICE OF *Type* |
SEQ OF *Type;*

ConvIdentifier ::=
identifier;
DialogIdentifier ::=
identifier;
ContentIdentifier ::=
identifier;

Abbildung 5.6: Die Grammatik für Konversationsspezifikationen

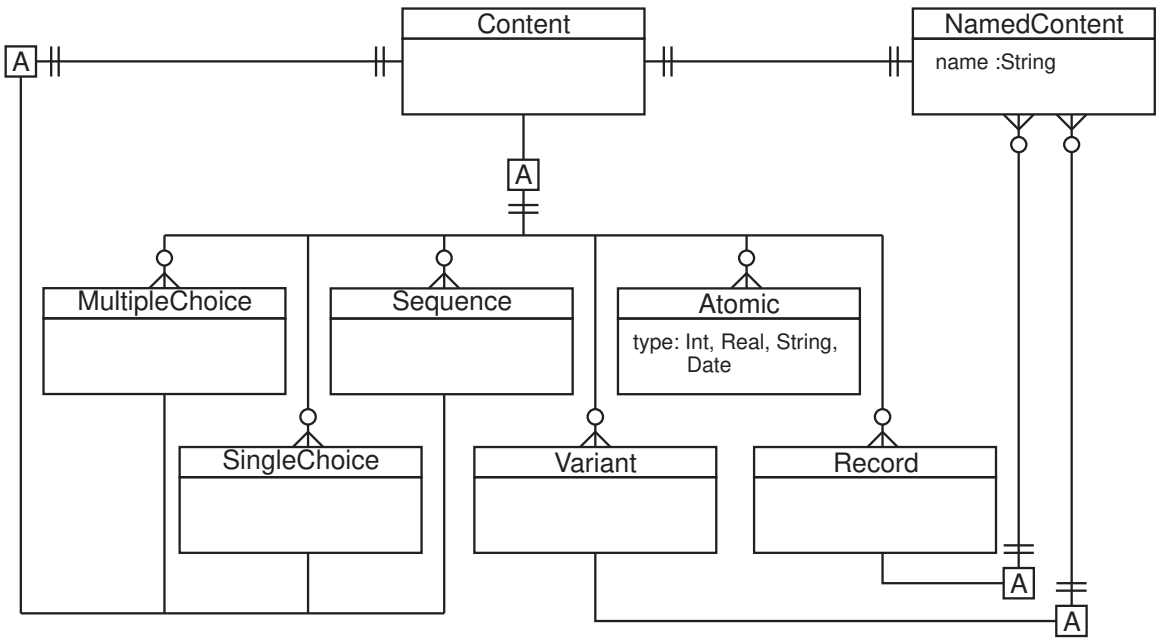

Abbildung 5.7: Instanzen der Inhaltsspezifikation

- In einem Record-Objekt kann auf die referenzierten Inhaltsobjekte zugegriffen werden.
- In einem varianten Record können außerdem Teile ausgewählt oder die Auswahl gelesen werden.
- Der Listentyp ist nicht direkt in den Dialog eingebunden und wird über Referenzen gesetzt oder gelesen. Außerdem soll er über vielfältige Auswahlfunktionen als Massendatentyp benutzt werden.

Eine Konversation wird immer zwischen einem Kunden und einem Dienstleister geführt. Um Ketten aus Ereignissen und Aktionen [DHL90] zu bilden, bekommen die beiden Partner der Konversation Regeln, mit denen sie auf Ereignisse des jeweils anderen Partners reagieren und so Aktionen ausführen. Menschliche Kunden simulieren diese Regeln, indem sie den Inhalt der Dialoge lesen und ihre Wünsche eintragen, maschinelle Kunden besitzen Programmcode, der die Dialoge entsprechend der Vorgaben behandelt. Die Regeln des Kunden enden immer mit der Übermittlung des eventuell veränderten Dialogs zusammen mit einer Anfrage. Ebenso muß der Dienstleister auf die Dialoge und Anfragen des Kunden entsprechend einer Regel reagieren. Seine Aktion endet immer damit, daß er einen neuen Dialog, den er bei Bedarf vorher mit inhaltlichen Vorgaben gefüllt hat, an den Kunden übermittelt. Durch die wechselseitige Ausführung einer Kundenregel und einer Dienstleisterregel wird die Konversation durchlaufen (siehe auch Abbildung 5.8).

Sub-Konversationen stellen sich zum Zeitpunkt ihrer Ausführung immer als Konversationen dar. Dies geht aus dem Diagramm aus Abbildung 5.5 durch die Aggregatbeziehung

$$
\begin{array}{rcll}
\text{Init:} & (r_0, d_0) & & \\
 & \downarrow & & \\
\text{Performer:} & (r_n, d_n) & \rightarrow & d_{n+1} \\
\text{Customer:} & d_n & \rightarrow & (d_n, r_{n+1})
\end{array}
$$

Abbildung 5.8: Die Regeln als Ereignis-Aktions-Paare

zwischen der Konversation und der Sub-Konversation hervor.

5.4 Invarianten des Modells

Die folgenden Invarianten sollen Interpretationsmöglichkeiten innerhalb des Modells einschränken. Ziel ist es, möglichst viele Punkte eindeutig festzulegen.

1. Alle Dialogspezifikationen einer Konversationsspezifikation haben verschiedene Namen. Alle benannten Inhaltsspezifikationen und Anfragespezifikationen einer Dialogspezifikation haben verschiedene Namen.

   Die Namen der Objekte dienen in ihren Sichtbarkeitsbereichen als Schlüssel zur Identifikation.

2. Jede Konversationsspezifikation hat eine Dialogspezifikation mit dem Namen *breakdown*.

   Damit bleibt jede Konversation immer in einem gesicherten Zustand. Es kann keine unvorhergesehenen Konfigurationen geben.

3. Jede Dialogspezifikation mit dem Namen *breakdown* hat benannte Inhaltsspezifikationen mit den Namen *lastDialog*, *reason*.

   Damit kann aus der Abbruchsituation später der Grund für den Abbruch rekonstruiert werden.

4. Alle Dialogspezifikationen mit dem Namen *breakdown* haben keine Folgedialoge und deswegen keine Anfragespezifikationen.

   Alle finalen Dialogspezifikationen beenden die Konversation.

5.5 Ein Anwendungsbeispiel

Der WebSiteProfiler aus Abschnitt 4.2 soll an dieser Stelle im Konversationsmodell dargestellt werden. Dabei liegt seine Funktionalität nach wie vor fest: Er erstellt verschiedene Statistiken über die Nutzung des WWW-Servers oder fügt das aktuelle Zugriffsprotokoll

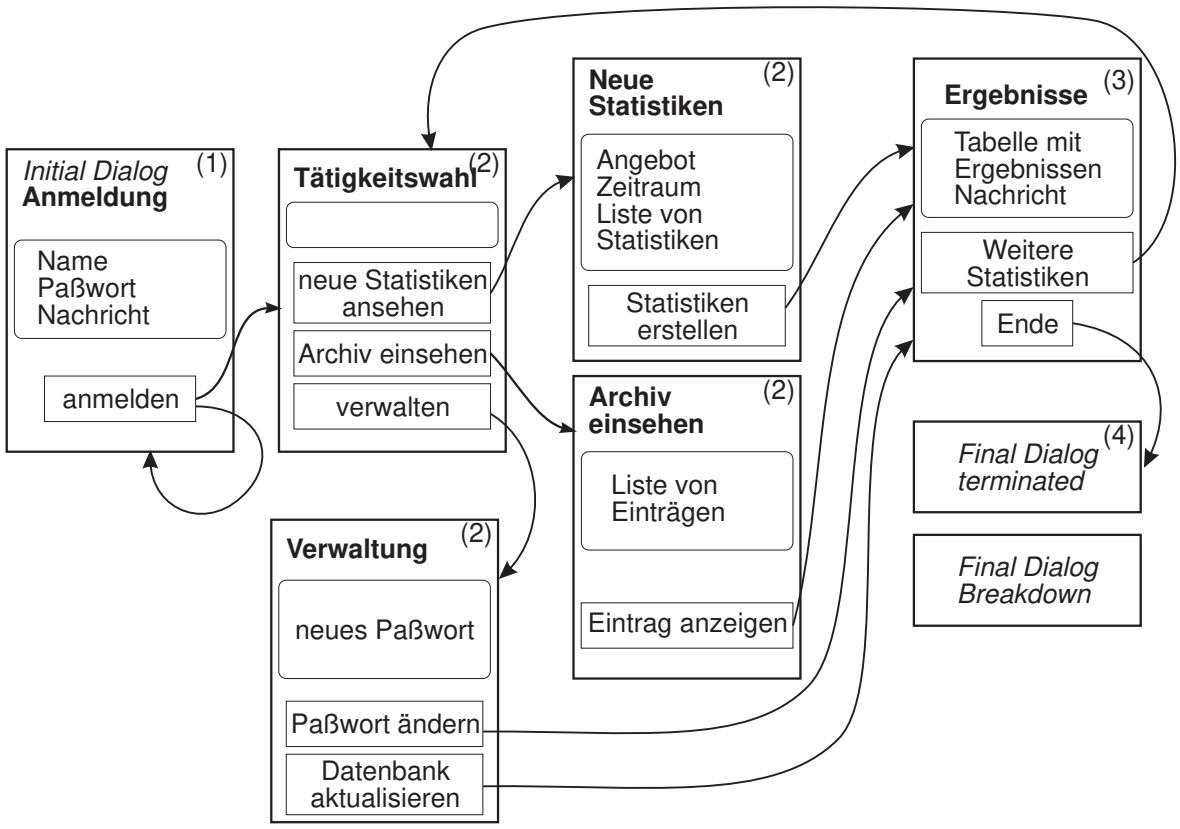

Abbildung 5.9: Konversationsspezifikation für den WSP

in die Datenbank ein. Die Visualisierung und Steuerung durch einen Benutzer wird jetzt mit den Business Conversations erstellt.

Um den WebSiteProfiler in das Modell einzufügen, muß der Programmierer eine Konversation eines Benutzers mit dem Programm entwerfen und spezifieren. Dabei tritt der WebSiteProfiler in der Rolle des Dienstleisters auf.

Eine denkbare Dialogfolge und ihre Umsetzung in eine Konversationsspezifikation ist in der Abbildung 5.9 aufgezeigt. Jeder große Kasten in der Abbildung stellt einen Dialog dar, sein Name ist fett gedruckt. In den runden Kästen steht der Inhalt des Dialogs, die unteren Kästchen bezeichnen die Anfragen des Dialoges, die zu den entsprechenden Folgedialogen führen. Die Nummern in den Dialogen zeigen die Phasen in der Konversation an, denen sie zugeordnet werden.

Im ersten Dialog soll sich der Benutzer bei dem System anmelden. Damit kann gewährleistet werden, daß keine unberechtigten Benutzer auf die Daten zugreifen. Zur Anmeldung bekommt der Benutzer einen Dialog, in dem er seinen Namen und sein Paßwort eintragen muß. Zusätzlich zu diese beiden Feldern ist das Feld für eine Nachricht an den Benutzer als Inhaltsspezifikation in der Dialogsspezifikation eingetragen. Wenn der Benutzer zum Beispiel mittels Knopfdruck eine Anfrage an das System stellt, werden die Dialoginhalte übertragen und vom System ausgewertet. Das System übermittelt dem Benutzer daraufhin

den nächsten Dialog. In diesem Beispiel kann die Auswertung auch dazu führen, daß der Benutzer wegen der fehlenden Berechtigung abgelehnt wird. Der WebSiteProfiler muß in diesem Fall entscheiden, welcher Dialog anschließend übermittelt wird. In allen anderen Fällen steht der Dialog, der auf die Anfrage folgt, fest.

Nach der Anmeldung kann der Benutzer auswählen, ob er neue Statistiken erstellen, das Archiv einsehen oder die aktuelle Zugriffsprotokolldatei in die Datenbank übernehmen möchte. In diesem Dialog braucht er keine inhaltlichen Eingaben zu machen. Durch seine Auswahl einer Anfrage wird der entsprechende Folgedialog angezeigt. Über die Anfragefolge *verwalten — Datenbank aktualisieren* wird die Funktion, in der der WebSiteProfiler die neuen Einträge des Zugriffsprotokolls gemäß dem Datenbankschema aus Abschnitt 4.2.2 in seine Datenbank übernimmt, aufgerufen. Als Ergebnis wird im Inhaltsfeld *Nachricht* des Folgedialogs die Anzahl der übernommenen Daten angezeigt. Bei den anderen beiden Dialogen *Neue Statistiken* und *Archiveinsicht* werden noch keine Funktionen des WebSiteProfilers ausgeführt. Sie kommen erst einen Dialogschritt später ins Spiel, wenn der Benutzer weitere Angaben gemacht hat. In dem Dialog, mit dem das System seine Ergebnisse präsentiert, hat der Benutzer die Wahl, weitere Statistiken erstellen zu lassen oder nicht. Durch die Schleife in der Spezifikation sind langandauernde Konversationen möglich.

In jedem Dialog erhält der Kunde die Möglichkeit, Konversationen mit der Anfrage *abbrechen* zu beenden. Ebenso kann der Dienstleister die Konversation beenden, in dem er dem Kunden den finalen Dialog *breakdown* übermittelt. Wenn die Konversation erfolgreich beendet wurde, wird der Dialog *terminated* an den Kunden zugestellt.

Auf der Basis dieser Spezifikation wird jetzt für jede durchzuführende Konversation eine Instanz erzeugt, mit der sie durchgeführt wird.

Ein Beispiel für eine Formulierung einer Dialogspezifikation aus der Abbildung 5.9 findet man in der Abbildung 5.10.

```
DIALOG Ergebnisse WITH
  REQUEST
    Weitere Statistiken TO Taetigkeitswahl NOELLIPSES
  END
  REQUEST Beenden TO Beendet NOELLIPSES END

  CONTENT Ergebnisliste OF
    SEQ OF
      SEQ OF
        REC
          unit OF INT
          count OF INT
        END
  END
END
```

Abbildung 5.10: Formulierung der Spezifikation für den Dialog mit Ergebnisstatistiken

Kapitel 6

Tycoon Business Conversations

Nachdem in Kapitel 5 das Modell der Business Conversations und der Ablauf von Konversationen erläutert wurde, wird in diesem Kapitel die Implementierung der Basissysteme für die Abwicklung von Konversationen mit dem persistenten Objektsystem Tycoon vorgestellt. Das implementierte Modell der *Business Conversations* wird so zu den *Tycoon Business Conversations*.

6.1 Ein Systemüberblick

Um das Modell der Business Conversations aus Kapitel 5 zu implementieren, wurden mehrere Module erstellt. Damit zwei Systeme über ein Netzwerk Konversationen durchführen können, brauchen sie eine Infrastruktur, mit der sie Dialoge verschicken können, um Informationen auszutauschen.

Es müssen Möglichkeiten geschaffen werden, um Spezifikationen von Konversationen zu erstellen. Desweiteren müssen diese Spezifikationen zur Laufzeit in Objekte umgesetzt werden.

Außerdem müssen die Kommunikationspartner Konversationen starten und Regeln zur Behandlung von Dialogen anlegen und ausführen können.

Die erstellten Module werden in der Abbildung 6.1 im Übersichtsdiagramm dargestellt. Die Module *ITC* und *ITCMessage* werden zur Kommunikation benutzt. *TBConv*, *TBCSpec* und *TBCContent* sind die Module, mit denen Konversationen spezifiziert und instanziiert werden können. Die Module *TBCPerformer* und *TBCCustomer* werden von den Kooperationspartnern benutzt, um Konversationen durchzuführen. Das Modul *TBCMediator* arbeitet mit übertragenen Dialogen und dient als Anbindung für weitere Systemdienste wie zum Beispiel Sicherheit oder Protokollierung.

Alle Pfeile im Diagramm spiegeln die Benutzung der Module sowohl auf Schnittstellenebene als auch auf der Ebene des Programmtextes wieder. Durch enge Schnittstellen ist aber zum Beispiel die Nutzung des ITC-Moduls durch das TBCPerformer-Modul für den Nutzer der TBCPerformer-Schnittstellen transparent.

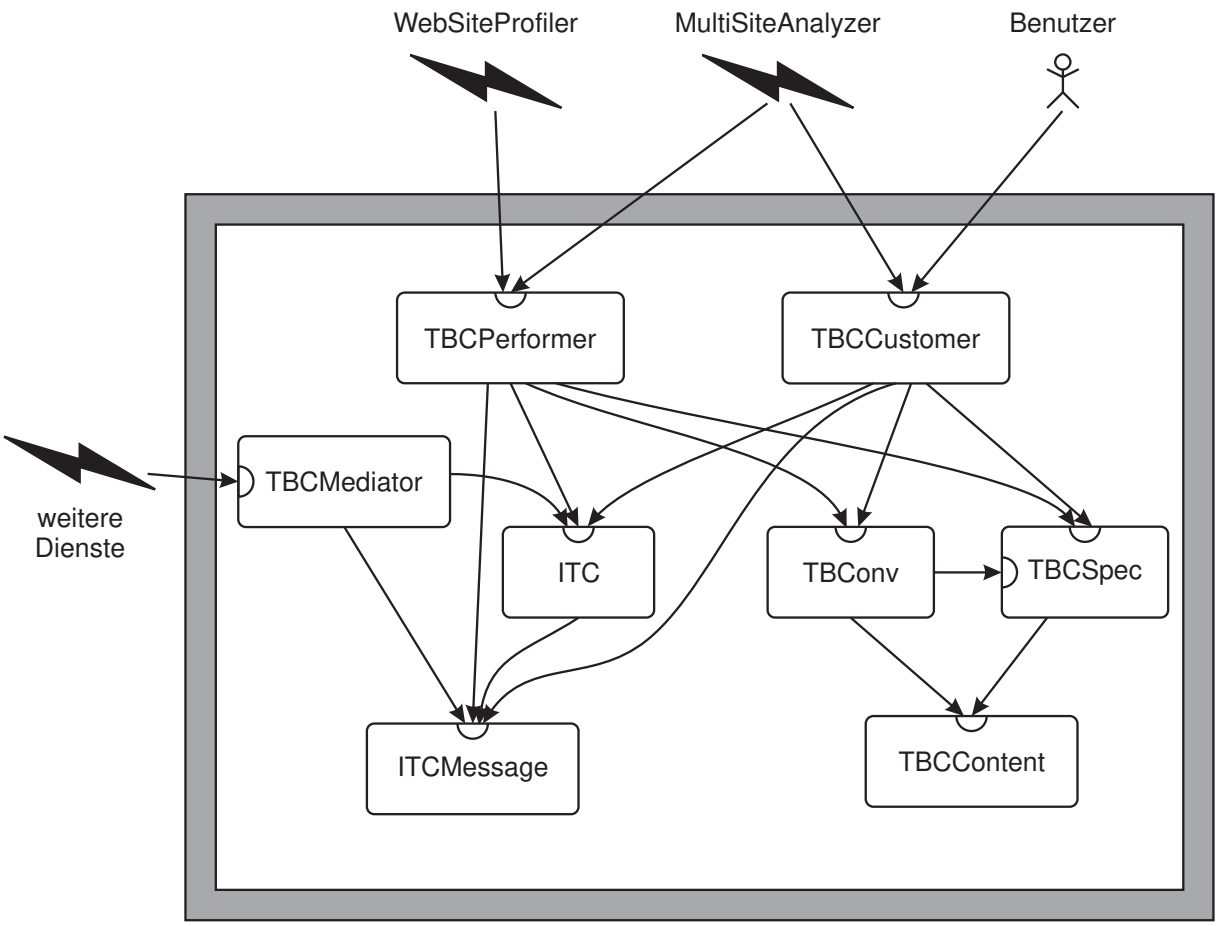

Abbildung 6.1: Die Modulstruktur der Tycoon Business Conversations

6.1.1 Die Kommunikationsmodule

Damit Anwendungen mit Hilfe der Tycoon Business Conversations kooperieren können, muß in jedem Tycoon-System ein Kommunikationsmodul vorhanden und gestartet sein.

Das System für die Tycoon Business Conversations unterstützt Nebenläufigkeit. So soll ein Dienstleister mehrere Konversationen mit verschiedenen Kunden gleichzeitig führen können. Außerdem soll die Kommunikation asynchron stattfinden. Dazu bekommt ein Kommunikationsmodul mehrere Threads, die miteinander kooperieren müssen. Um dieses Ziel zu erreichen, wurde ein Basismodul zur Kommunikation von Threads geschaffen. Die Kommunikation benutzt eine feste Menge von Nachrichten.

Das Modul *Inter Thread Communication (ITC)* spaltet in der *start*-Funktion zwei Threads ab. Der eine Thread, der an die Variable *remoteReader* gebunden ist, wartet auf eine Nachricht, die über den Unix-Socket von einem anderen Rechner zugestellt wird. Nachdem eine Nachricht empfangen wurde, wird sie mit der Funktion *deliver(...)* an den entsprechenden Thread im Objektspeicher des Kommunikationsmoduls zugestellt. Der zweite Thread, der an den Namen *localITCService* gebunden ist, wartet auf eintreffende Nach-

richten von anderen Threads des Objektspeichers. Wenn eine Nachricht eintrifft, wird sie entweder dem Mediator zugestellt, oder, falls die Nachricht schon vom Mediator behandelt wurde, an den Empfänger weitergeleitet. Dabei wird unterschieden, ob der Empfänger ein Thread im gleichen Objektspeicher oder ein Kommunikationsmodul auf einem anderen Rechner im Netzwerk ist.

Mit *Ports* werden die Kommunikationspartner identifiziert. Dabei umfaßt ein Port die Adresse des Rechners, auf dem sich der Thread befindet, einen Namen, der jeden Thread in einem Objektspeicher eindeutig kennzeichnet und einen Kanal zur Kommunikation mit Threads des gleichen Objektspeichers. Diese Kanäle wurden von der Threadsynchronisation aus einer früheren Diplomarbeit [Pie96] übernommen.

Die Übertragung von Nachrichten zwischen verschiedenen Rechnern stützt sich auf Socket-Verbindungen und entsprechende Tycoon-Bibliotheken, die die Kommunikation in das System integrieren [Mat96].

In den folgenden Abschnitten werden Module vorgestellt, die weitere Threads abspalten, um ihre Funktion nebenläufig zu erfüllen. Diese Threads benutzen das ITC-Modul, um miteinander zu kommunizieren. Dazu befinden sie sich alle Threads in einer Schleife, die mit der Funktion *rcv(...)* des ITC-Moduls auf eine eintreffende Nachricht wartet, um sie nach Erhalt zu analysieren und entsprechende Aktionen auszuführen. So können Threads mit Hilfe der Funktion *send(...)* des ITC-Moduls wiederum Informationen an ihre Partner verschicken.

Alle Nachrichten werden vor der Übermittlung an den Kommunikationspartner einem sogenannten *Mediator* zugestellt. In dieser Komponente können die Nachrichten verändert oder umgeleitet werden. Damit können zum Beispiel Sicherheits- oder Protokolldienste in das System integriert werden.

Im Mediatormodul existieren Funktionen zur An– und Abmeldung von Diensten. Alle Dienste werden in einer Liste verzeichnet. Die Dienste bekommen Prioritäten, um eine Reihenfolge bei der Abarbeitung festzulegen. Jeder Dienst muß am Ende die Nachricht an den nächsten Dienst delegieren. Wenn kein weiterer Dienst mehr registriert ist, kann die Nachricht dem Empfänger zugestellt werden. Nachrichten können somit auch vom System unterdrückt werden.

6.1.2 Module für Konversationen

Konversationen bestehen nach Abschnitt 5.3 aus zwei Teilen, den Spezifikationen und den Instanzen. Diese Unterteilung spiegelt sich auch in den Modulen wieder. Die Implementierung der Spezifikation ist nach dem Objektstrukturdiagramm aus Abbildung 5.4 vorgenommen worden.

Um eine Konversationsspezifikation zu erstellen, werden zum Beispiel Dialogspezifikationen mit Hilfe der Funktion *dialogSpec(...)* erzeugt. Die Parameterliste der Funktion enthält den Dialognamen, ein Array mit möglichen Anfragen und ein Array mit benannten Inhaltsspezifikationen. Die Spezifikationen für den Inhalt bestehen aus einem Tupel mit dem Namen und dem Konstruktor für einen Inhaltstyp. Neue Konversationsspezifikationen werden mit der *new(...)*-Funktion erstellt. Ihre Parameter umfassen den Namen der

Konversation, die Spezifikation des initialen Dialogs und ein Array mit allen Dialogspezifikationen dieser Konversation. Die Grammatik aus Abbildung 5.6 kann leicht auf diese Funktionen abgebildet werden.

Die Dialoginhalte werden in einem eigenen Modul behandelt. Dort sind sowohl Funktionen zur Spezifikation von Inhalten als auch Funktionen zur Behandlung der Instanzen vorhanden. Es existieren die Zugriffsfunktionen auf Dialoginhalte, die im Modell in Abschnitt 5.3.3 festgelegt wurden. Diese Funktionen werden innerhalb der Regeln der Kunden und Dienstleistern benötigt, um mit den Dialoginhalten arbeiten zu können.

Beim Zugriff auf den Inhaltstyp *Sequence* bekommt man lediglich eine Referenz auf die Folge. Damit sind Folgen als Massendatentyp nicht an die Dialoge, die zwischen Kunden und Dienstleistern übertragen werden, gebunden. Die Massendaten können so ausschnittsweise Inhalt von Dialogen werden.

Die Konversationsinstanzen werden im Modul *TBConv* verwaltet. Dort kann mit der Funktion *new(...)* eine neue Konversation aus einer Spezifikation erstellt werden. Dazu wird eine Tabelle mit Dialogspezifikationen angelegt. Damit sind alle Dialogspezifikation über ihren Namen erreichbar. Es wird eine leere Historie für Dialoge angelegt.

Eine Konversationsinstanz enthält außerdem Funktionen, die alte oder neue Dialogobjekte oder das aktuelle Dialogobjekt zurückgeben. Wenn ein neuer Dialog erzeugt wird, wird sein Dialoginhalt mit der Funktion *create(...)* angelegt. Er kann aus diesen Dialoginstanzen benutzt werden. Außerdem ist aus den Dialog- und Konversationsinstanzen der Zugriff auf die Spezifikationen der Dialoge oder Konversationen möglich.

Konversationen, Dialoge und Inhalte sind keine aktiven Objekte, die eigene Threads besitzen. Sie sind an den Kunden oder Dienstleister gebunden, der sie erzeugt hat.

6.1.3 Module für die Akteure

Die beiden Akteure in einer Konversation, der Kunde und der Dienstleister, wurden in den zwei Modulen *TBCCustomer* und *TBCPerformer* implementiert. In beiden Modulen sind zwei Tupel realisiert, die die beiden Akteure darstellen. Die Tupel werden jeweils mit der Funktion *new(...)* erzeugt und enthalten Funktionen zum Setzen der Konversationspezifikation und zur Behandlung von Dialogen. Der Kunde hat zusätzlich eine Funktion, um eine Konversation zu starten.

Beide Partner müssen die Dialoge einer Konversation behandeln, um Informationen auszutauschen. Der Kunde braucht für jeden Dialog eine Funktion, die den Dialoginhalt nach seinen Wünschen verändert und ihn mit einer Anfrage an den Dienstleister zurückschickt. Diese Funktionen werden als Regeln mit der Funktion *handleDialog(...)* registriert, so daß sie abgearbeitet werden können, wenn der zugehörige Dialog eintrifft. Der Dienstleister muß innerhalb einer bestimmten Konversation auf die Anfragen des Kunden reagieren und ihm einen neuen Dialog liefern. Die Aktionen, die er beim Erzeugen eines neuen Dialogs ausführt, registiert er mit der Funktion *transition(...)*. Dabei hat der Dienstleister sowohl Zugriff auf den Dialog, der vom Kunden zurückgeschickt wurde, als auch Zugriff auf den Dialog, den er an den Kunden ausgeliefert hatte. Innerhalb der Funktionen, die als Aktionen oder Regeln registiert wurden, können die Akteure andere Systeme aufrufen und

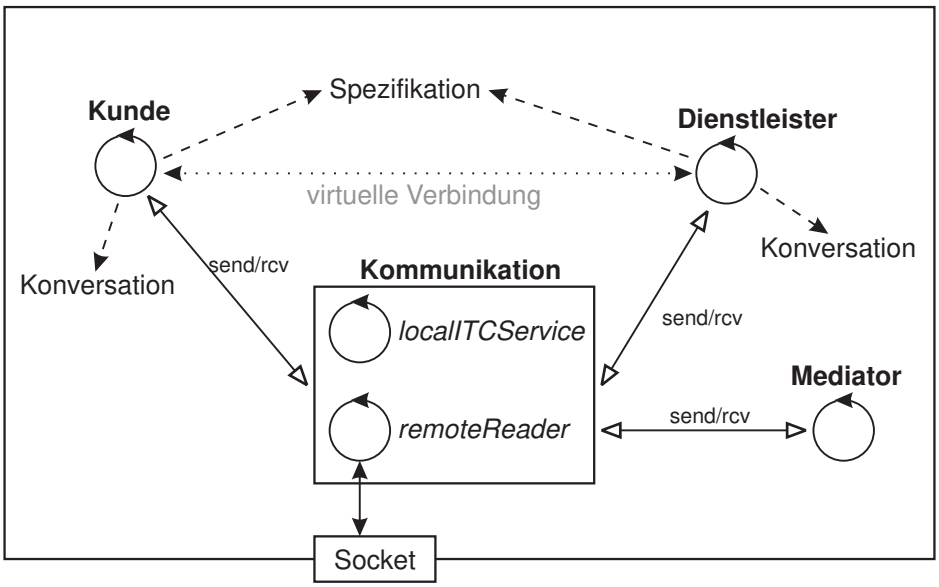

Abbildung 6.2: Die Threads im Objektspeicher

benutzen.

In den Modulen ist die Technik zum Ausliefern der Dialoge und zum Aufruf der Regeln realisiert. Beide Akteure spalten bei der Konstruktion eigene Threads ab und sind somit nebenläufig zu anderen Vorgängen. So können sich mehrere Akteure in einem Tycoon-System befinden.

Nachdem die Threads abgespalten wurden, befinden sie sich in einer Schleife und warten auf Nachrichten, die an die Akteure gerichtet sind. Die Kommunikation zwischen den Kunden- oder Dienstleisterobjekten, die mit der Funktion *new(...)* erzeugt wurden und den Threads im Objektspeicher geht über die Kommunikationsfunktionen des ITC-Moduls. Das bedeutet, daß die Funktionen wie zum Beispiel *addSpec(...)* einen Aufruf der *send(...)*-Funktion des ITC-Moduls auslöst, um dem Thread eine Nachricht mit der neuen Konversationsspezifikation zu senden.

Das gesamte Szenario der nebenläufigen Threads im Objektspeicher und ihrer Beziehungen untereinander ist in der Abbildung 6.2 noch einmal skizziert. Zwischen dem Kunden und dem Dienstleister besteht eine virtuelle Verbindung zum Austausch von Dialogen. Beide Akteure besitzen Referenzen auf die Spezifikation der Konversation und jeder Akteur referenziert eine eigene Instanz der Konversation. Das Kommunikationssystem hält über den Socket eine Verbindungsmöglichkeit zu anderen Tycoon-Systemen mit einem Kommunikationssystem. Es koordiniert die Zustellung der Nachrichten an die entsprechenden Threads unter Berücksichtigung des Mediators.

```
ON anmelden Anmeldung WSP-Conversation WITH d conv DO
  let name = d.content("Name")!string.get()
  let password = d.content("Paßwort")!string.get()
  let dia =
    if wsp.login(name password) then
      conv.newDialog("Tätigkeitswahl")
    else
      d.content("Nachricht")!string.set("Paßwort war falsch")
      d
    end
  dia
END
```

Abbildung 6.3: Beispiel für die Dienstleister-Regel zur Auswertung der Anmeldung

6.2 Die Einbettung des WebSiteProfiler

Der WebSiteProfiler aus Abschnitt 4.2 wurde in die oben beschriebene Implementierung der Tycoon Business Conversations integriert. Bisher bestand das System aus einer Sammlung von Funktionen, die entsprechende Werte zurückgegeben haben. Diese Funktionen müssen in die Dialogbehandlung integriert werden.

Der erste Schritt zur Integration des WebSiteProfilers in die Tycoon Business Conversations besteht darin, eine Spezifikation der Konversation zu erstellen. Sie wurde bereits im Abschnitt 5.5 vorgestellt. Die Spezifikation muß als Objekt im Objektspeicher vorliegen. Im zweiten Schritt wird der WebSiteProfiler als Dienstleister gestartet. Wenn der Thread gestartet ist und auf Nachrichten wartet, muß er seine Konversationsspezifikation bekommen. Dazu wird die Funktion *addSpec(...)* mit der Spezifikation als Argument aufgerufen. Danach muß er, auf seiner Konversationsspezifikation aufbauend, Regeln für die Behandlung der Anfragen und Dialoge bekommen. In den Regeln, die als TL-Funktionen an den Dienstleister gebunden werden, werden jetzt an den passenden Stellen die Funktionen des WebSiteProfilers aufgerufen.

In der Abbildung 6.3 ist eine Beispiel für die Regel zur Auswertung der Anfrage *anmelden* des ersten Dialogs dargestellt.

Nach diesen Schritten ist der WebSiteProfiler als Dienstleister im System bereit, Anfragen von Kunden nach dem Beginn der Konversation zu behandeln.

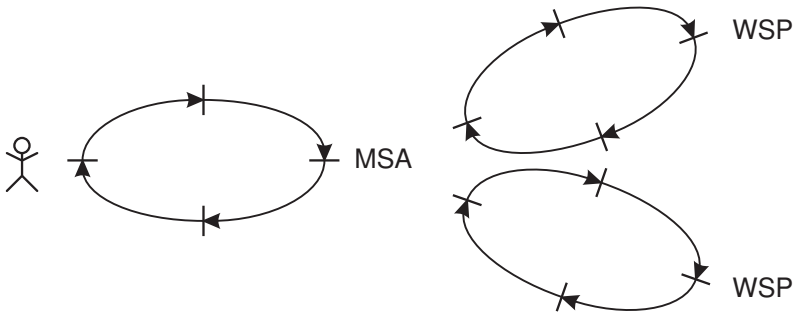

Abbildung 6.4: Integration von MultiSiteAnalyzer und WebSiteProfilern

6.3 Die Nutzung durch den MultiSiteAnalyzer

Der MultiSiteAnalyzer tritt sowohl als Dienstleister als auch als Kunde auf. Als Dienstleister erstellt er Ranglisten über die Nutzung von WWW-Servern, als Kunde erfragt er die notwendigen Daten für die Ranglisten von den WebSiteProfilern. Dieser Rollenwechsel findet innerhalb der Regeln für den Dienstleister statt. Wenn der MultiSiteAnalyzer mit der entsprechenden Anfrage aufgefordert wird, eine Rangliste zu erstellen, erzeugt er einen neuen Kunden, der mit der Konversationspezifikation für den WebSiteProfiler versehen wird und die entsprechenden Regeln zur Behandlung der Dialoge bekommt. Dabei wird der Dialoginhalt gemäß den Vorgaben des MultiSiteAnalyzer-Kunden eingetragen. Die Konversationen mit den WebSiteProfilern sind nebenläufig. Erst wenn alle als Thread abgespaltenen Konversationen mit dem finalen Dialog beendet wurden, wird die Rangliste erstellt.

Abbildung 6.4 zeigt die Integration der Konversation mit dem WebSiteProfiler in die Konversation mit dem MultiSiteAnalyzer. Wenn sich die Konversation des Benutzers mit dem MultiSiteAnalyzer in der Phase befindet, in der Ranglisten erstellt werden müssen, tritt er in Konversationen mit den WebSiteProfilern ein. Wenn diese Konversationen beendet sind, können die Ergebnisse der einzelnen Konversationen in der Rangliste des MultiSiteAnalyzers aggregiert werden. Der Benutzer des MultiSiteAnalyzers kann sich mit der Rangliste einen Eindruck von der Nutzungshäufigkeit der verglichenen WWW-Server verschaffen.

6.4 Vergleich der Tycoon Business Conversations mit anderen Systemen

In Kapitel 3 wurden drei verschiedene Ansätze für verteilte Objektsysteme vorgestellt und mit dem Tycoon-System verglichen. An dieser Stelle werden jetzt die Tycoon Business Conversations als System für kooperative Informationsysteme mit den kommerziellen Ansätzen verglichen.

Mit den Tycoon Business Conversations wurde ein System geschaffen, das einen Rah-

men für die Kooperation verschiedener Systeme in einer Kunde-Dienstleister-Beziehung vorgibt. Wenn ein Kooperationsmuster mit einer Konversation modelliert ist, kann es zur Kommunikation und zur Visualierung benutzt werden. Dies ist ein Vorteil gegenüber CORBA und dem DCE, weil dort die Benutzerführung von Applikationen nicht besonders berücksichtigt wird. Lediglich CORBA soll in Zukunft mit der Common Facility für das User-Interface (siehe Abschnitt 3.1.4) eine Schnittstelle zur Programmierung der Benutzerinteraktion bereitstellen. Ansonsten unterstützen diese Systeme lediglich die Kommunikation von Applikationen.

In den Tycoon Business Conversations wird der dynamische Aspekt von Dienstleistungen betrachtet. Während bei CORBA und dem DCE die entfernten Dienstleistungen isoliert durch einzelne Methoden- oder Prozeduraufrufe realisiert werden, wird in den Tycoon Business Conversations der Vor- und Nachlauf einer Dienstleistung ebenfalls berücksichtigt.

Die Schnittstellen, die mit der Interface Definition Language bei CORBA oder dem DCE spezifiziert wurden, sind keine Objekte erster Klasse, wie es die Spezifikationen der Tycoon Business Conversations sind. Jede Schnittstelle steht bei CORBA oder dem DCE für einzelne Dienste, die zusammenhanglos nebeneinander stehen. In den Tycoon Business Conversations bieten die Konversationen eine Einbettung der Dienste in eine geeignete Umgebung. Die Spezifikationen mit der Interface Definition Language sind systemnäher als die Spezifikationen der Tycoon Business Conversations, die sich an der Anwendung orientieren.

Im Vergleich mit OLE bieten die Tycoon Business Conversations den Vorteil der klareren Strukturierung. Mit OLE können sich Applikationen auch visualisieren. Jedoch sind die Objekte in OLE nur eine Ansammlung von Schnittstellen, die von den anderen Kommunikationspartnern aufgerufen werden. Die Konversationsspezifikation der Tycoon Business Conversations kann — wie ein Typ einer Programmiersprache — vor der Benutzung daraufhin untersucht werden, ob sie zu den Erwartungen paßt. Mit diesem Mechanismus wird die Wiederverwendung von Konversationsspezifikationen unterstützt. Zum Beispiel kann in einer Konversation zum Einkaufen in einem Warenhaus für die Abwicklung der Bezahlung der Konversationsteil von einem Kreditinstitut benutzt werden. Das Kreditinstitut wiederum kann die Konversation auch anderen Dienstleistern zur Verfügung stellen. Für diesen Mechanismus wurde in Abschnitt 5.3.2 das Dialogattribut *ellipses* eingeführt.

Kapitel 7

Vergleich verschiedener Kommunikationsformen

Das Modell der Business Conversations abstrahiert von der Art der Kommunikation. Die Grundlage ist lediglich, daß Dialoge mit ihrem Inhalt zwischen beiden Partnern ausgetauscht werden sollen. In der Implementierung der Tycoon Business Conversations werden die Dialoge über ein Unixnetzwerk mit dem entsprechenden Übertragungsprotokoll zu anderen Rechnern übertragen. Diese Übertragung kann mittels zweier verschiedener Kommunikationsformen geschehen. Bei der objektbasierten Kommunikation werden die Dialoge von zwei entfernten Kommunikationspartnern einzeln über das Netzwerk verschickt. In der agentenbasierten Kommunikationsform beauftragt der Kunde einen mobilen Agenten, die Konversation mit dem Dienstleister lokal durchzuführen. Dazu migriert der Agent auf den Rechner des Dienstleisters und führt dort eine Konversation, um die Leistung zu bekommen. Anschließend kehrt der Agent mit dem Ergebnis der Konversation zum Kunden zurück und übermittelt es ihm.

In den folgenden Abschnitten werden beide Kommunikationsformen beschrieben, damit sie anschließend verglichen werden können.

7.1 Objektbasierte Kommunikation

Bei dieser Kommunikationsform handelt es sich um die entfernte Übermittlung von Dialogen durch ein Netzwerk. Die Implementierung der Akteure, die in Kapitel 6 beschrieben wurde, zielt auf diese Form ab.

Zu einer objektbasierten Kommunikation gehört ein Server für Objekte, ein Klient, der die Objekte nutzen möchte und ein Übertragungssystem, das die Klienten für den Benutzer transparent an die Server vermittelt. In den kommerziellen Systemen aus Kapitel 3 wird dabei viel Wert auf eine komfortable Realisierung des Übertragungssystems gelegt. In der vorliegenden Implementierung der Tycoon Business Conversations ist die Übertragungsschicht nicht so komfortabel wie in den kommerziellen Systemen. Die Kommunikationspartner müssen den Namen des Rechners und des Partners auf dem Rechner

kennen, um mit ihm in Verbindung treten zu können. Jedoch können über den Mediator beliebige Metadienste wie zum Beispiel Adressauflösung integriert werden.

Der Dienstleister kann als entfernter Objektserver, der von verschiedenen Kunden Anfragen als Methodenaufrufe übermittelt bekommt, angesehen werden. Dabei werden die Dialoge, die die Anfrage nach Methoden oder die Ergebnisse der Methodenaufrufe enthalten, als eigenständige Objekte über das Netzwerk verschickt.

7.2 Kommunikation mittels mobiler Agenten

In [Whi94] wird ein weiterer Ansatz für die Zusammenarbeit von Systemen vorgestellt. Nachdem im bisherigen Client/Server-Ansatz über das Netzwerk Anfragen und die Ergebnisse von entfernten Prozeduraufrufen transportiert werden, wird mit dem Telescript-System ein Ansatz für entfernte Programmierung vorgestellt.

7.2.1 Motivation für mobile Agenten

Wenn die Netzwerktätigkeit auf entfernten Prozeduraufrufen basiert, werden Daten für die Anfrage nach einem Dienst und für die Antwort über das Netzwerk transportiert. Dabei vereinbaren die Kommunikationspartner Nachrichten, die von allen verstanden werden. Diese Vereinbarungen bilden ein Protokoll. Für entfernte Prozeduraufrufe ist es charakteristisch, daß jede Interaktion zwei Übertragungen benötigt. Bei komplexen Diensten werden entsprechend mehr Übertragungen gebraucht. Eine Folge von Interaktionen aus mehreren Prozeduraufrufen benötigt andauernde Kommunikation.

Telescript will mit der entfernten Programmierung einen neuen Ansatz für Netzwerktätigkeit verfolgen. Sie basiert auf der Übertragung von Objekten, also Daten und Programmen, die nach Empfang auf einem entfernten Rechner ausgeführt werden. Diese Objekte werden *Agenten* genannt. Die Ausführung eines Agenten besteht aus der Abarbeitung seiner Prozeduren im Kontext seiner Daten. Die Kommunikationspartner agieren bei diesem Ansatz auf der Basis vereinbarter Befehle, die Prozeduren bilden. Diese Vereinbarungen bilden eine Sprache. Charakteristisch für die entfernte Programmierung ist, daß Klient und Server ohne Hilfe des Netzwerks interagieren können. Auch eine Folge von Interaktionen benötigt keine andauernde Kommunikation.

Neben der Möglichkeit, daß der Klient zum Rechner des Servers übertragen wird, kann auch der Server für die Dienstleistung auf den Rechner des Klienten übertragen werden.

Vorteile

Der neue Ansatz bietet gegenüber der alten Form mehrere Vorteile. Ein Vorzug besteht in der besseren Leistungsfähigkeit. Die Dienste können vor Ort schneller erbracht werden, wenn nicht bei jedem Schritt der entfernte Klient das Ergebnis erhalten muß, um eine neue Anfrage abzusenden. Und je langsamer die Daten im Netzwerk übertragen werden, um so stärker wirkt sich die geringe Anzahl von Übertragungen aus. Ein weiterer Vorzug

besteht in den besseren Anpassungsmöglichkeiten der Dienste an spezielle Kundenwünsche. Dabei kann sich der Kunde aus allen Diensten, die von einem Server angeboten werden, selbst neue Dienste zusammenstellen und sie in Form eines Agenten nutzen. Zum Beispiel kann sich der Kunde selbst aus zwei Diensten, die entweder alle Dateien auf einem Server anzeigen oder eine Datei löschen, einen Dienst zusammenstellen, der alle alten Dateien löscht.

Konzepte

Die Telescript-Technologie bietet einen elektronischen Marktplatz, auf dem sich Anbieter und Kunden gegenseitig finden und interagieren können. Der Marktplatz ist weiter unterteilt und strukturiert. Er besteht aus einer Menge von *Plätzen*. Beispiele für Plätze sind der Heimatplatz eines Agenten in einem persönlichen Rechner eines privaten Nutzers oder ein Platz, der ein elektronisches Kaufhaus darstellt und von einem öffentlichen Anbieter realisiert wird. Plätze können geschachtelt sein. So kann zum Beispiel das elektronische Kaufhaus aus mehreren Abteilungen bestehen, die jeweils Plätze sind. Jeder Platz hat einen Besitzer.

Die Plätze werden von *Agenten* belegt. Während Plätze dem elektronischen Marktplatz seine statische Struktur verleihen, sind die Agenten für die dynamischen Aktivitäten verantwortlich. Die Agenten repräsentieren potentielle Anbieter und Konsumenten von Gütern und Diensten. Ein Agent bildet einen unabhängigen Prozeß, der von der Umgebung kontrolliert ausgeführt wird. Alle Agenten werden parallel ausgeführt, haben dabei aber generell keinen Einfluß auf andere Agenten.

Bisher befinden sich die Anbieter und Kunden auf dem elektronischen Marktplatz auf verschiedenen Rechnern, die durch ein Netzwerk getrennt sind. Die Kunden und Anbieter können in der Telescript-Technologie mit *Reisen* zusammenkommen. Die Agenten können sich selbst von einem Platz zu einem anderen bewegen. Die Reisen werden angetreten, um entfernte Dienstleistungen zu empfangen. Zum Beispiel kann ein Nutzer einen Agenten in das elektronische Kaufhaus schicken, damit er dort eine Eintrittskarte für eine Veranstaltung erwirbt. Für die Reisen werden die Agenten mit *Fahrkarten* versehen. Auf den Fahrkarten wird das Ziel der Agenten und die Zeit für die Reise eingetragen.

Zwei Agenten können an einem Platz interagieren. Dabei nutzt ein Agent einen Dienst, den der andere Agent anbietet. Für die Interaktion müssen sich beide Agenten *treffen*. Ein Treffen wird von einem Agenten initiiert und der zweite Agent muß dem Treffen zustimmen. Der Initiator überreicht dem Partner eine *Petition*, auf der der Name des Agenten und die Dauer des Treffens eingetragen ist. Falls das Treffen nicht erfolgreich beendet werden kann, wird der Initiator informiert und kann die Situation gemäß seinem Programm bewältigen. Wenn ein Agent den Befehl ***meet*** ausführt, bekommt er eine Referenz auf den Kommunikationspartner, die für eine direkte Interaktion genutzt werden kann.

In Abbildung 7.1 ist ein Beispiel für ein Treffen zweier Agenten gezeigt. Der Agent von Jim White ist über das Netzwerk auf den Großrechner mit dem elektronischen Kaufhaus gereist, um dort in einem Treffen mit dem Tickethändler zwei Theaterkarten für ein bestimmtes Theaterstück zu bestellen.

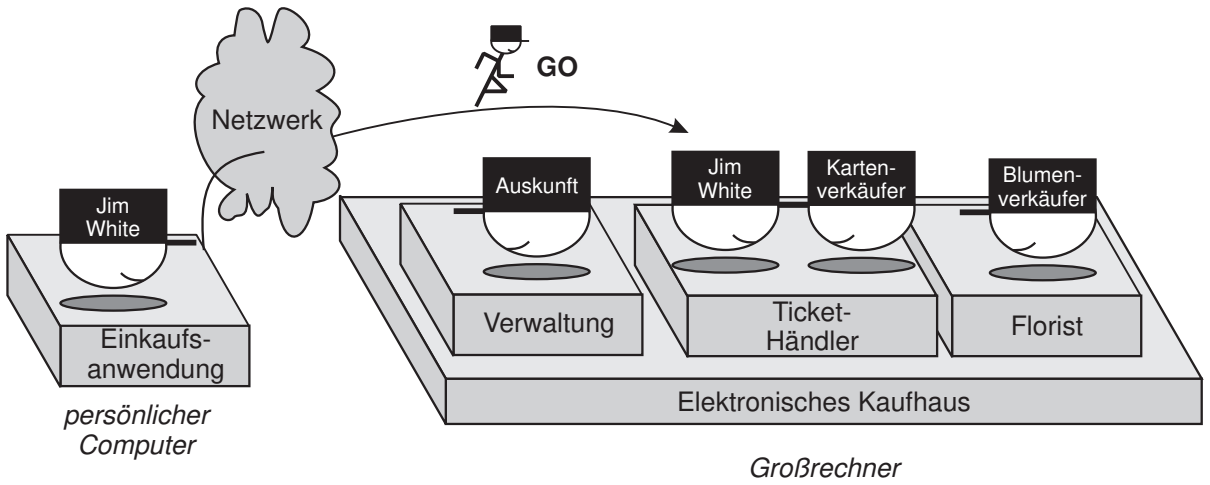

Abbildung 7.1: Beispiel für ein Treffen zweier Agenten

Agenten können auch interagieren, wenn sie sich an verschiedenen Plätzen aufhalten. Dazu benötigen sie eine *Verbindung*. Mit diesem Konzept kann zum Beispiel ein Agent am Heimatplatz über verfügbare Sitzplätze bei der Theaterveranstaltung informiert werden, der wiederum den Benutzer aussuchen läßt, welche Sitzplätze gebucht werden sollen. Der Agent am Heimatplatz informiert dann den Agenten im Kartenverkaufsbüro. Mit diesem Konzept wird der alte Kommunikationsansatz in den neuen integriert.

Die Stärke der Telescript-Technologie und des Ansatzes der entfernten Programmierung tritt erst vollständig zu Tage, wenn ein Agent mehrere Dienste kombiniert nutzt. Damit kann der Agent einen höherwertigen Dienst, der aus einzeln angebotenen Basisdiensten besteht, anbieten. Zum Beispiel kann ein Agent mehrere Plätze nacheinander besuchen. Dabei kann er die Dienstleistungen entweder alle in Anspruch nehmen oder aus allen Angeboten einige auswählen. So kann der Agent, nachdem er Theaterkarten bekommen hat, bei einem Floristen eine Lieferung von Rosen für den Tag der Theatervorstellung bestellen. Dabei kann das Ergebnis der Kartenbestellung, nämlich der Tag der Vorstellung, die Interaktion mit dem Blumenverkäufer beeinflussen. In diesem Fall dient die Telescript-Technologie dazu, aus dem Verständnis für zwei verschiedene Dienste, die Buchung von Theaterkarten und die Blumenbestellung, ein Verständnis für besondere Gelegenheiten zu bekommen. Dafür muß das elektronische Einkaufszentrum nicht erweitert werden. Es kann jedoch ein weiterer Dienst für das Arrangement von besonderen Gelegenheiten angeboten werden, in dem man den Agenten aus dem Beispiel auch anderen Nutzern zur Verfügung stellt.

Sicherheitsvorrichtungen

Die Telescript-Sprache wird interpretiert und nicht in Maschinenbefehle übersetzt. Den Plätzen und Agenten stehen damit keine Befehle für den Zugriff auf Speicher, Dateisystem oder andere Ressourcen des ausführenden Rechners zur Verfügung. Jeder Agent oder

Platz besitzt einen Namen, mit dem er von anderen identifiziert werden kann. Neben seinen Namen und Besitzern spielen die *Beglaubigungen* der Agenten eine wichtige Rolle im Sicherheitskonzept der Telescript-Technologie. *Regionen*, eine Menge von Plätzen, die dem gleichen Besitzer gehören, können Agenten bei nicht ausreichenden Beglaubigungen den Eintritt verwehren. Die Beglaubigungen müssen mit entsprechenden Sicherheitsvorkehrungen gegen Fälschung und Verlust versehen werden. Zu der Beglaubigung besitzt jeder Agent einen *Passierschein*, der seine Möglichkeiten einschränkt. Der Passierschein wird erstellt, wenn der Agent programmiert wird und bei jedem Verlassen einer Region neu festgelegt. Vor dem Eintritt in eine Region muß der Agent die dort herrschenden Vorschriften anerkennen und sich entsprechend einschränken. Wenn der Agent die Region verläßt, werden die Einschränkungen wieder aufgehoben. Mit dem Passierschein erhält der Agent die Möglichkeit, bestimmte Befehle zu benutzen. So kann es einem Verkaufsagenten verboten werden, das ***go***-Kommando zu benutzen. Außerdem darf der Agent aufgrund seines Passierscheins eine bestimmte Menge von Ressourcen wie zum Beispiel Rechenzeit benutzen. Dabei wird der Ressourcenverbrauch aller Agenten von der Maschine überwacht. Mit dem Passierschein können Aktivitäten der Agenten a priori vom Besitzer eingeschränkt werden.

Technologie

Die Telescript-Sprache soll als Programmiersprache die Entwicklung von kommunizierenden Anwendungen ermöglichen. Dabei teilt sie viele Eigenschaften mit anderen Programmiersprachen. Ihre besonderen Eigenschaften sind im folgenden aufgezählt:

- objekt-orientiert
- dynamisch
- persistent
- interpretiert
- kommunikationszentriert

Die Plätze und Agenten können ohne Telescript-Maschine nicht exisitieren. Die Maschine verwaltet die Plätze und die Agenten, die sich auf den Plätzen befinden. Dazu sorgt sie für die Persistenz der Plätze und Agenten und führt ihre Programme nebenläufig aus. Schließlich sorgt die Maschine für den Transport der Agenten zu anderen Maschinen, wenn die Agenten den Befehl dazu aufrufen. Jede Telescript-Maschine besitzt drei Programmierschnittstellen, über die sie den Zugriff auf den Rechner abwickelt. Eine Schnittstelle dient dem Transport von Agenten, in dem sie auf die Kommunikationsmedien zugreift. Die zweite Schnittstelle bietet Zugriff auf den nichtflüchtigen Speicher des Systems. Die dritte Schnittstelle dient dem Zugriff auf externe Applikationen.

Mit den Protokollen der Telescript-Maschine wird verschiedenen Maschinen ermöglicht, miteinander zu kommunizieren. Über diese Verbindungen können Agenten von einer Maschine zur anderen transportiert werden. Die Protokolle arbeiten auf zwei Ebenen. Auf der höheren Ebene wird beschrieben, wie Agenten, ihre Daten, Programme und Ausführungszustände für den Transport darzustellen sind. Während die Darstellung innerhalb der Maschine für andere Partner keine Rolle spielt, müssen sie einheitlich transportiert werden. Auf der unteren Ebene wird der Transport von einer Maschine zur anderen abgewickelt.

Verwaltung, elektronische Post und Unternehmen

Das Konzept der Agenten wird auch zur Verwaltung der Maschine eingesetzt. Intern arbeiten Agenten an Verwaltungsaufgaben, die auch netzweit zusammengefaßt werden können. Die Telescript-Technologie kann außerdem als Basistechnologie für elektronische Post benutzt werden. Dabei sind die Nachrichten als Agenten und die Postfächer als Plätze, die von einem Agenten verwaltet werden, implementiert. Damit können sowohl die Nachrichten als auch die Postfächer an die Anforderungen der Benutzer angepaßt werden. So kann zum Beispiel ein Postfach ein Verzeichnis von Telefonnummern enthalten, anhand dessen die Nachrichten weiter zugestellt werden. Mit der Telescript-Technologie kann auch ein abgeschlossenes Firmennetz bedient werden. Zum Beispiel kann die Reisekostenabrechnung mit einem Agenten von allen zuständigen Abteilungen bearbeitet werden.

7.2.2 Ablauf von agentenbasierter Kommunikation

Die Implementierung der Umgebung für mobile Agenten nach [Joh97] stützt sich auf das Modell der Business Conversations. Die Akteure *Kunde* und *Dienstleister* wurden beide zu einem Agenten generalisiert. Ein Agent kann sich frei in der Infrastruktur bewegen, er kann mit anderen Agenten Konversationen beginnen und durchführen und dabei sowohl Kunden- wie auch Dienstleister-Rollen annehmen.

Die Infrastruktur ist an die Telescript-Technologie angelehnt. Das bedeutet, daß es auch Agenten und Plätze gibt. Jedoch benutzen die Agenten zur Kooperation die Konversationen aus dem Modell aus Kapitel 5. Mit der Benutzung dieses Konzeptes fallen viele Sicherheitsprobleme, die in der Telescript-Technologie behandelt werden, weg. Weil die Agenten nur noch Dialoge austauschen, können sie in ihrer Umgebung keine unerlaubten Befehle ausführen oder für sie nicht zugelassene Schnittstellen aufrufen.

Die implementierte Infrastruktur muß in jedem Objektspeicher des Tycoon-Systems, der Agenten unterstützen soll, vorhanden sein. Sie ist das Analogon zur Telescript-Maschine. Sie stellt den Agenten Dienste zur Migration und zum Einleiten von Treffen zur Verfügung.

Jeder Agent bekommt einen Auftrag in einer Funktion, der abgearbeitet wird, sobald der Agent erzeugt wurde. Innerhalb dieses Auftrages muß der Agent Spezifikationen von Konversationen erhalten und mit wenigstens einer Rolle belegt werden, um eine sinnvolle Tätigkeit zu erbringen. In dieser Funktion kann der Befehl ***migrateTo*** aufgerufen werden, damit der Agent zu einem anderen Platz wandert. Dort kann er mit dem ***meet***-Befehl

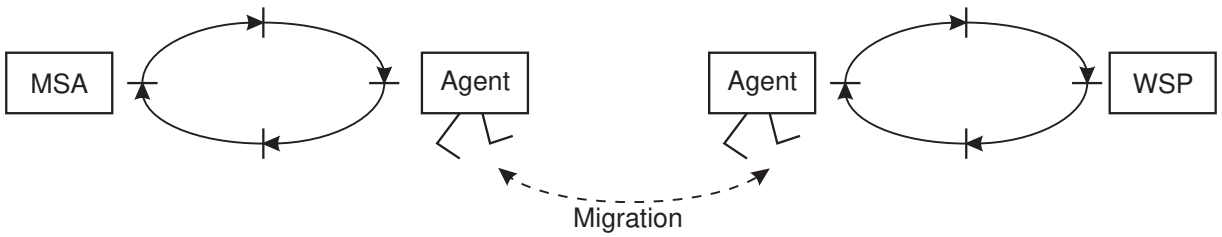

Abbildung 7.2: Der Ablauf von agentenbasierter Kommunikation

andere Agenten treffen. Sie führen miteinander Konversationen. Neben dem Auftrag bekommt ein Agent eine ***meet***-Funktion, mit der er auf Anfragen von anderen Agenten nach einem Treffen reagiert. Diese Funktion gibt als Ergebnis entweder eine Referenz auf den Agenten zurück oder verweigert ein Treffen mit dem Initiator. Mit der Referenz des anderen Agenten kann eine Konversation begonnen werden.

In der Abbildung 7.2 ist der Ablauf der Kooperation von MultiSiteAnalyzer und WebSiteProfiler dargestellt. Der MultiSiteAnalyzer bekommt vom Benutzer den Auftrag, bestimmte Statistiken zu aggregieren. Er erzeugt einen Agenten und beauftragt ihn mit einer Konversation, die gewünschten Daten vom WebSiteProfiler zu holen. Dann migiert er zum WebSiteProfiler und führt dort eine Konversation, aus der er das entsprechende Ergebnis erhält. Dann migriert er zurück zum MultiSiteAnalyzer, um die dortige Konversation mit der Übermittlung der Statistiken zu beenden.

7.3 Ein Vergleich beider Arten

Der Unterschied zwischen beiden Kommunikationsformen besteht lediglich in der Menge der über das Netz transportierten Daten und der Anzahl der Transportvorgänge. Durch die Spezifikation kann der Kunde die Konversation vorher überprüfen, ob sie seinen Anforderungen entspricht. Damit läßt sich die Konversation weitestgehend vorhersehen und die meisten Fälle, die in der Konversation auftreten können, werden im Auftrag, den der Agent bekommt, berücksichtigt. So kann oftmals die Dienstleistung mit dem fertigen Auftrag des Kunden durch einen Besuch des Agenten erbracht werden. Damit werden die Dienstleistungen mit nur zwei Übertragungen des Agenten vom Kunden zum Dienstleister und wieder zurück erbracht.

Konversationen können auf der Basis der Agenten ohne ständige Verbindung zum Netzwerk abgewickelt werden. Wenn der Agent seinen Auftrag bekommen hat, werden alle weiteren Schritte autonom durchgeführt. Der Kunde kann gerade bei Dienstleistungen, die viel Zeit benötigen, die Verbindungen zum Netzwerk abbauen und erst später, beim erwarteten Wiedereintreffen des Agenten, die Verbindung zum Netz wieder aufnehmen, um das Ergebnis zu erhalten.

Der Rahmen zur Kooperation von Agenten wird mit den Tycoon Business Conversations geliefert. Die Implementierung der Tycoon Business Conversations aus [Joh97] stellt neben der Möglichkeit der objektbasierten Kommunikation auch eine Umgebung für mobi-

le Agenten zur Verfügung. In dieser Implementierung kann der Benutzer bei Beginn einer Konversation entscheiden, welche Kommunikationsform er nutzen möchte. Bei standardisierten Dienstleistungen wie dem WebSiteProfiler bietet sich die agentenbasierte Form aufgund der geringeren Netzbelastung an. Wenn für die Dienstleistung aber zwischen mehreren, ständig wechselnden Vorgaben des Dienstleisters entschieden werden muß, sollte sie der Kunde direkt abwickeln. Ein Beispiel dafür ist der Einkauf in einem Warenhaus mit ständig wechselnden Angeboten.

Der Wechsel zwischen beiden Kommunikationsformen kann ohne Änderung der Systemkonfiguration geschehen.

In beiden Kommunikationsformen bilden Konversationen die Basis für den Informationsaustausch. Auch bei der Kooperation der autonomen Agenten werden Konversationen zur Synchronisation benutzt.

Kapitel 8

Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit noch einmal rückblickend zusammengefaßt werden. Die kommerzielle Relevanz der Arbeit und die praktischen Einsatzmöglichkeiten werden untersucht. Am Ende folgt ein Ausblick auf zukünftige Arbeiten und Erweiterungsmöglichkeiten.

8.1 Zusammenfassung

Am Anfang der Arbeit wurde das Tycoon-System vorgestellt. In Kapitel 2 sollte neuen Benutzern anhand einiger Programmbeispiele ein erster Eindruck von den Eigenschaften des Systems und der Programmierung in TL vermittelt werden. In Kapitel 3 wurden drei verschiedene kommerzielle Systeme für verteilte Objekte vorgestellt. Alle Systeme wurden mit Tycoon verglichen, um die Stellung von Tycoon auf dem Markt für verteilte System zu finden. In Kapitel 4 wurden zwei Anwendungsprogramme zur Erfassung und Analyse von Internetnutzungsstatistiken vorgestellt. Beide Systeme müssen eng miteinander kooperieren. Dabei wurde auf verschiedene Kooperationsmöglichkeiten hingewiesen, um im nächsten Kapitel einen allgemeinen Entwurf einer Umgebung für kooperative Informationssysteme vorzustellen. Das Modell basiert auf Konversationen, die ein Kunde und ein Dienstleister miteinander führen, um eine Leistung zu vereinbaren und zu erbringen. In Kapitel 6 wurde die Implementierung des Modells im Tycoon-System vorgestellt. Es wurde gezeigt, wie die Anwendungen aus Kapitel 4 mit diesem Konzept kooperieren können. In Kapitel 7 wurde die Medienunabhängigkeit der Business Conversations für einen Vergleich der Kommunikation auf der Basis von Objekten und der Kommunikation mittels mobiler Agenten genutzt. Der Vergleich zeigte, daß beide Kommunikationsformen nebeneinander stehen können und daß der Wechsel zwischen beiden Formen mit den Tycoon Business Conversations ohne Aufwand möglich ist.

8.2 Praktische Einsatzmöglichkeiten

In diesem Abschnitt wird die kommerzielle Relevanz der Arbeit untersucht. Dem bisher implementierten Prototyp kann man noch keine kommerzielle Relevanz zusprechen. Jedoch

hat das Modell der Business Conversations mit besser ausgestatteten Werkzeugen Chancen, am Markt der kommunizierenden Applikationen teilzunehmen.

8.2.1 Integration bestehender Applikationen

Mit den Business Conversations ist es leicht, bestehende Applikationen als Dienstleister für weitere Kunden zur Verfügung zu stellen. Damit braucht keine bestehende Software von Grund auf neu entwickelt werden, um sie mit anderen Akteuren kooperieren zu lassen. Der zusätzlich Aufwand zur Einbindung in die Umgebung beschränkt sich auf die Entwicklung einer Spezifikation, die die Dienste der Anwendung in Konversationen integriert und die Entwicklung von Regeln, die die Anwendung entsprechend einbinden.

8.2.2 Vorteile beim Einsatz der Tycoon Business Conversations

Alle Anwendungen, die in den Business Conversations als Dienstleister auftreten, können zu der Kommunikationsmöglichkeit und dem entfernten Zugriff auch visualisiert werden. Damit ist die separate und teilweise aufwendige Entwicklung einer Benutzerschnittstelle nicht mehr nötig.

8.2.3 Meßbarkeit der WWW-Server-Reichweiten

Während sich die Business Conversations im vorliegenden Stadium noch nicht kommerziell verwenden lassen, ist die Nachfrage nach Meßwerkzeugen für die Reichweite der Internet-Werbung größer. Nach einer AFP-Meldung vom 6. Dezember 1996 aus New York wurde auf einer Konferenz zum Thema Online-Werbung mitgeteilt, daß Werbung schon heute zu einem erheblichen Teil zu den Einkünften aus dem Internet beiträgt. Jedoch halten sich die traditionell großen Auftraggeber aus der Werbebranche in diesem Bereich noch zurück, weil die Wirkung der Werbung noch nicht ausreichend überprüfbar ist.

Der MultiSiteAnalyzer kann jedoch mit dem Vergleich der Nutzungsstatistiken mehrerer Server seinen Teil zur Messung der Werbeverbreitung beitragen.

8.3 Ausblick

Wie in Abschnitt 1.1 bereits genannt, ist diese Arbeit der Ausgangspunkt für weitere Projektarbeiten am Arbeitsbereich. Die Grundlage aus dieser Arbeit kann in zwei Richtungen erweitert werden. Zum einen kann die Basistechnologie der Business Conversations weiter ausgebaut werden. Diese Grundlagen-Arbeiten müssen sich mit speziellen Aspekten wie der Überprüfung von Konversationsspezifikationen oder der Wiederverwendung fremder Spezifikationen in eigenen Spezifikationen befassen. Zum anderen kann man auf den Business Conversations noch weitere Dienste aufbauen oder anbinden. So ist die Visualisierung von Dialogen noch zu erstellen. Desweiteren kann die Entwicklung neuer Spezifikation mit Hilfe grafischer Editoren oder anderen Werkzeugen noch benutzerfreundlicher gestaltet

werden. Man kann auch einen Sicherheitsdienst in das System integrieren, damit beide Kommunikationspartner die Identität ihres Gegenüber verläßlich kennen.

Auch die Palette der verfügbaren Übertragungsmedien kann erweitert werden. Damit kann die Anbindung bestehender Anwendungen an das Internet ermöglicht werden.

Anhang A

Überblick über die Methode der *Mainstream Objects*

In diesem Abschnitt wird ein Modellierungssystem vorgestellt, um die benutzten Modelle leichter verständlich und vergleichbar darzustellen.

Die Methode der *Mainstream Objects* [YWT+95] dient zur Modellierung von Geschäftsvorgängen auf einer objektorientierten Basis. Ziel der Methode ist es, einen Geschäftsvorgang so abzubilden, daß die Umsetzung in ein Computersystem mit Hilfe einer objektorientierten Programmiersprache leicht fällt. Um dieses Ziel zu erreichen, wird der Vorgang mit Hilfe mehrerer Modelltypen von verschiedenen Seiten betrachtet.

A.1 Das Objektstrukturmodell

Das Objektstrukturmodell spielt die zentrale Rolle in dieser Entwurfsmethode. In diesem Modell werden die Klassen, die im Projekt eingesetzt werden, gefunden. Außerdem werden die Verbindungen zwischen Klassen untersucht und abgebildet, so daß das Modell die Grundlage des Entwurfs bildet. Dieses Modell spiegelt die Welt des Benutzers wider, weil die gleichen Begriffe wie in der Realität benutzt werden.

Neben dem Objektstrukturmodell präzisieren weitere Modelle andere Aspekte des Systementwurfs. Diese Modelle sind das Transaktionsfolgenmodell, die Interaktion der Objekte, das Objektzyklusmodell und das Überblicksmodell.

Die Abbildung A.1 zeigt, wann welches Modell im Projektzyklus erstellt werden soll. Dabei nimmt das Objektstrukturmodell im Laufe des Projektes einen immer größeren Umfang ein.

Der wachsende Umfang zeigt die steigende Detaillierung. Bis zum logischen Entwurf werden die Objekte der abzubildenden Realität behandelt. Während des logischen Entwurfs wird die Struktur des zu erstellenden Anwendungsprogramms mit einbezogen. Während des physischen Entwurfs kommen die fehlenden Implementierungsdetails hinzu.

Das Objektstrukturmodell besteht aus den folgenden Komponenten:

- Gegenstandsobjekte

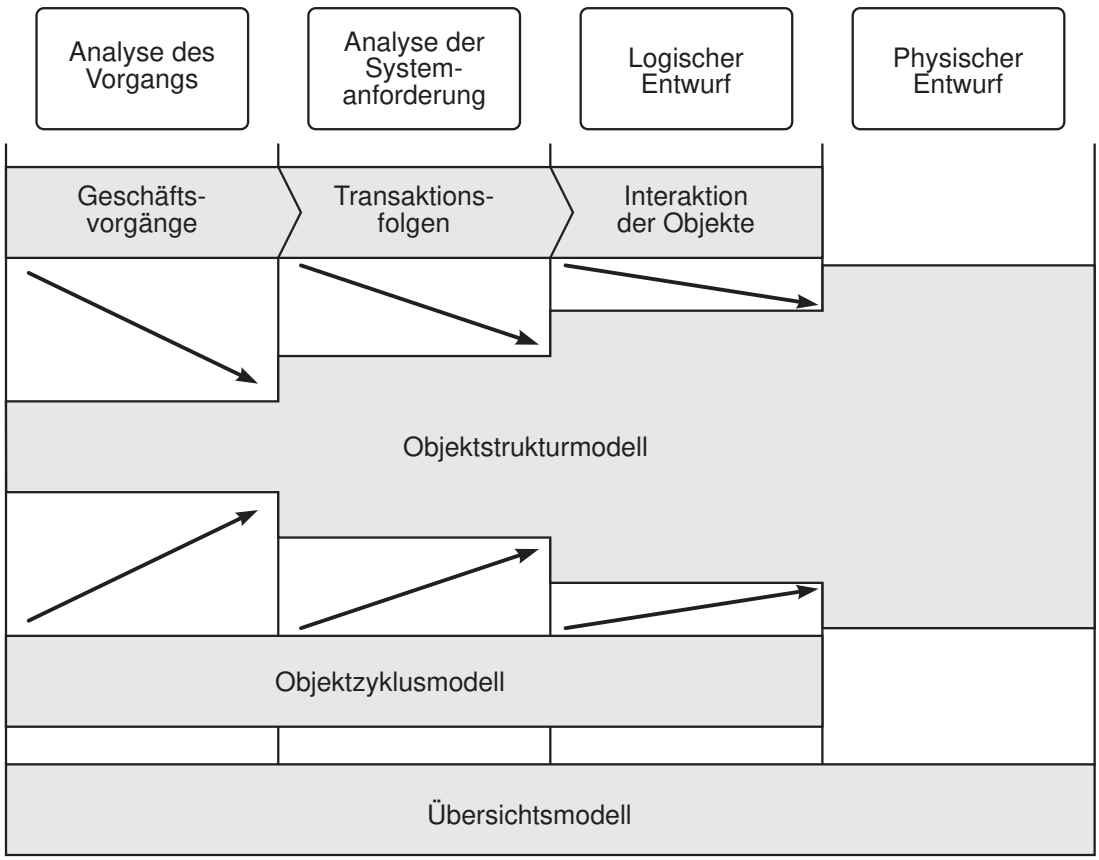

Abbildung A.1: Einsatz der Modelle im Projektzyklus

- Schnittstellenobjekte
- Kontrollobjekte
- abstrakte Objektklassen

In der Abbildung A.2 sind die zugehörigen Symbole zu sehen.

Obwohl immer von Objekten gesprochen wird, sind eigentlich Objektklassen gemeint.

Zur weiteren Beschreibung der Objekte gehören die Operationen und Attribute eines Objekts und die Bedingungen an die Attribute.

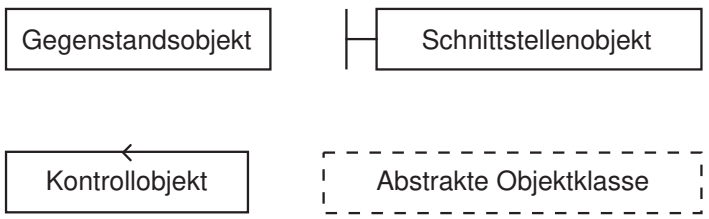

Abbildung A.2: Darstellung der Objekttypen

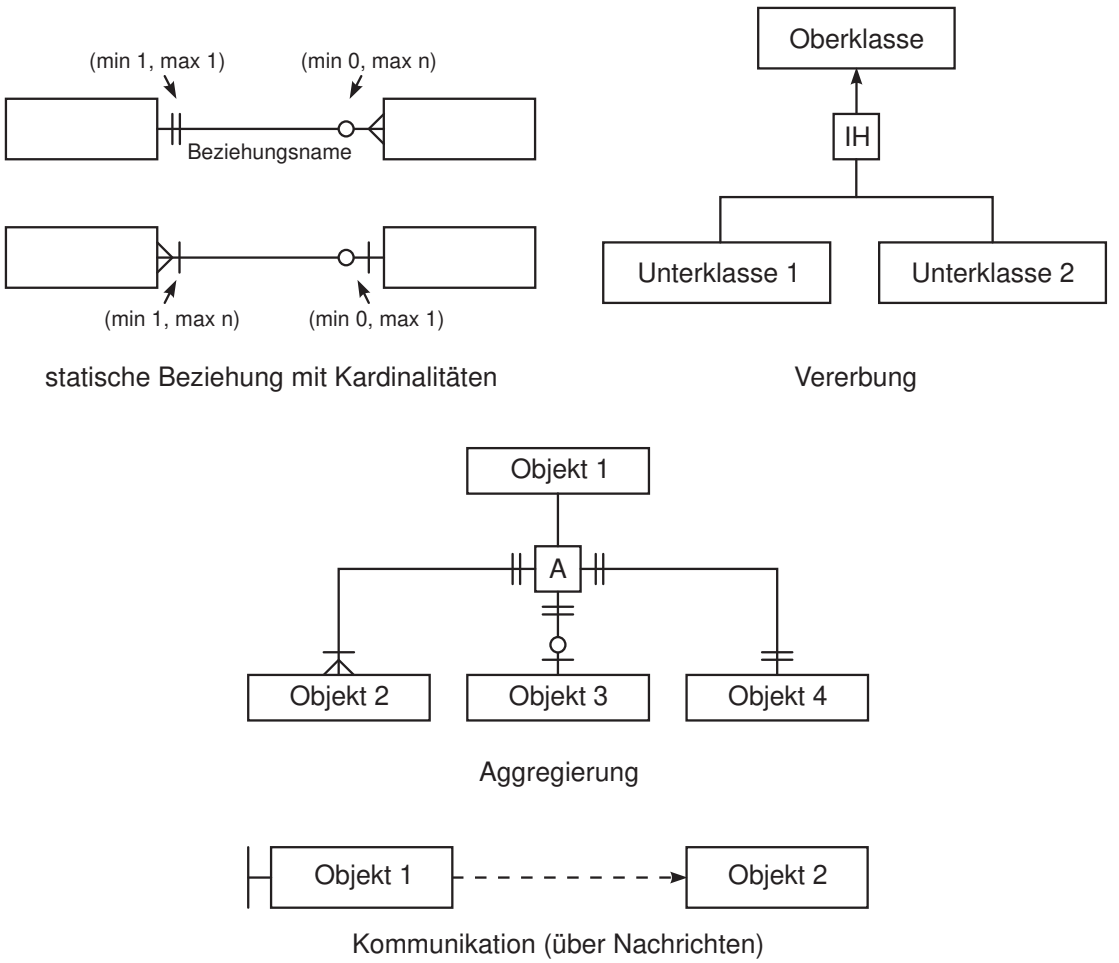

Abbildung A.3: Beziehungstypen im Objektstrukturdiagramm

Im Objektstrukturmodell werden zu den Objekten die folgenden Beziehungen zwischen den Objekten dargestellt (siehe auch Abbildung A.3):

Statische Beziehungen modellieren die Beziehung zwischen den Objekten. Sie sind an Gegenstands-Beziehungs-Modelle angelehnt. Die Beziehungen können mit Namen versehen werden. Über Kardinalitäten werden die Zahlen der verbundenen Objekte spezifiziert. Hierbei können für jedes der beiden verbunden Objekte Kardinalitäten vom Typ $(\min 1, \max 1)$, $(\min 0, \max 1)$, $(\min 1, \max n)$ oder $(\min 0, \max n)$ angegeben werden.

Vererbung wird durch Pfeile zur allgemeinen Klasse dargestellt, obwohl eigentlich die Attribute und Operationen der allgemeineren Klasse in der spezialisierten Klasse wieder auftauchen und dort erweitert werden.

Aggregierung zeigt, welche Objekte zu neuen Objekten zusammengefaßt werden. Auch hier können die oben genannten Kardinalitäten eingearbeitet werden.

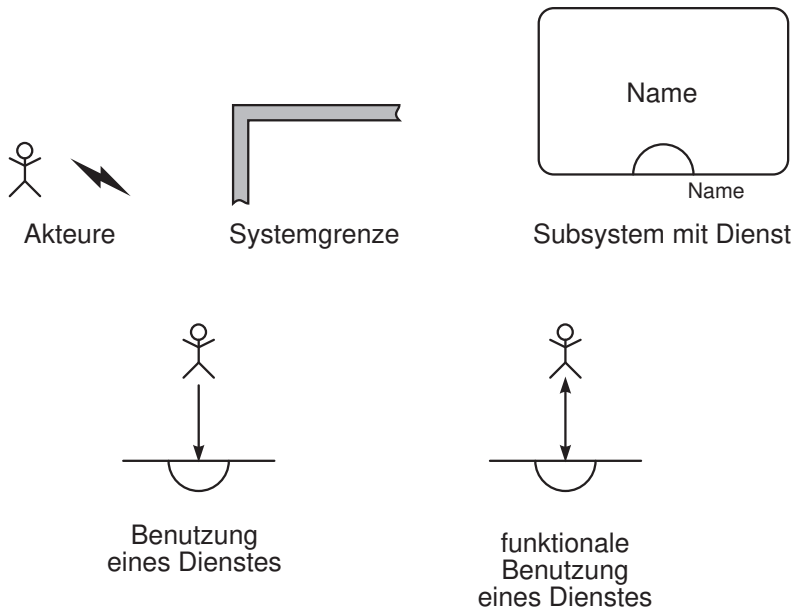

Abbildung A.4: Komponenten des Übersichtsmodells

Kommunikation über Nachrichten ist die einzige Verbindungsart, die Objekte verschiedener Typen erreichen kann.

Die Modelle für Geschäftsvorgänge, Transaktionsfolgen, Objektinteraktionen und Objektzyklen spielen in der Arbeit keine Rolle und werden daher nicht beschrieben.

A.2 Das Übersichtsmodell

Um die Entwicklung von großen Systemen in kleinere, handhabbare Aufgaben zu unterteilen, versucht man Subsysteme zu finden, die von verschieden Entwicklergruppen bearbeitet werden können. Die Unterteilung von Systemen wird mit dem Übersichtsmodell erleichtert. Außerdem können die Entwicklergruppen mit diesem Modell die Stellung ihres Subsystems im Gesamtsystem im Blick behalten.

Das Zusammenspiel der verschiedenen Subsysteme wird über Dienste geregelt, die den anderen Subsystemen zur Verfügung gestellt werden.

Im Übersichtsmodell werden die folgenden Komponenten aus Abbildung A.4 aufgenommen:

Akteure: Personen werden als Strichmännchen und maschinelle Akteure als Blitz dargestellt.

Systemgrenze: Alles, was innerhalb der Systemgrenze liegt, wird vom Modell behandelt. Die Akteure liegen außerhalb.

Subsysteme: Jedes Subsystem ist ein Kasten mit abgerundeten Ecken. Jeder Kasten wird mit seinem Namen versehen.

Dienste: Die Dienste werden durch Halbkreise in den Kästen der Subsysteme dargestellt. Auch die Dienste werden über ihre Namen identifiziert.

Benutzung von Diensten: Wenn ein Akteur oder ein Subsystem einen Dienst benutzt, zeigt ein Pfeil vom Akteur auf den Dienst.

funktionale Benutzung von Diensten: Eine eigene Erweiterung, die von den Autoren der Methode nicht vorgesehen ist, erleichtert die Darstellung in dieser Arbeit. Wenn ein Dienst benutzt wird, gibt er anschließend eine Antwort an den Aufrufer zurück.

Anhang B

Glossar für Business Conversations

Konversation (*conversation*): Der Oberbegriff für eine Folge von Dialogen zwischen einem *Kunden* und einem *Dienstleister*, mit dem Ziel einer Aktion. Eine Konversation kann einzelne (Teil-) Konversationen wiederverwenden.

Sprechakt (*speech act*): Ein Begriff aus der Language-Action Sicht auf CSCW nach Flores et al. [FGHW88]: „. . . human beeings are fundamentally linguistic beings: action happens in language in a world constituted by language.“. Genauer werden Sprechakte in Zyklen von vier Phasen unterteilt: *Anfrage*, *Übereinkunft*, *Leistung* und *Rückmeldung*. In einer technischen Umsetzung des Modells wird ein Sprechakt als *Konversation* bezeichnet.

Phase (*phase*) Granularität des Sprechaktmodells. Die Einteilung in Phasen muß nicht in einer Implementierung wiedergespiegelt werden. Sie dient allein der Veranschaulichung des Ablaufs von Sprechakten.

Anfrage (I) (*request*) Bezeichnung für die erste Phase des Modells.

Anfrage (II) (*request*) Die Bearbeitung einer Anfrage führt im Modell in einen neuen Dialog. Anfragen sind die einzige Möglichkeit, in einer Konversation einen neuen Zustand zu erreichen.

Übereinkunft (*agreement*) Bezeichnung für die zweite Phase des Modells.

Leistung (*performance*) Bezeichnung für die dritte Phase des Modells.

Rückmeldung (*feedback*) Bezeichnung für die vierte Phase des Modells.

Kunde (*customer*) Initiator einer Konversation. Innerhalb eines Sprechakts ist die Rolle des Kunden festgelegt. Aus der Sicht einer übergeordneten Konversation kann der Kunde jedoch gegenüber einem anderen Kunden wiederum in der Rolle des *Dienstleisters* auftreten.

Dienstleister (*performer*) Akzeptor einer Konversation. Innerhalb eines Sprechakts ist die Rolle des Dienstleisters festgelegt. Aus der Sicht einer übergeordneten Konversation kann der Dienstleister jedoch gegenüber einem anderen Dienstleister wiederum in der Rolle des *Kunden* auftreten.

Akteur (*Actor*) Sowohl der *Kunde* als auch der *Dienstleister* nehmen aktiv (daher Akteur) an der Konversation teil.

Regel (*Rule*) Jeder Akteur führt Aktionen aus, die Regeln genannt werden. Der Dienstleister reagiert aufgrund einer Anfrage gemäß einer zugehörigen Regel und erzeugt einen neuen Dialog. Der Kunde wiederum reagiert auf den Dialog gemäß seiner eigenen Regeln.

Annahme (*acceptance*) Die ersten beiden Phasen eines *Sprechakt*zyklus.

Erfüllung (*completion*) Die letzten beiden Phasen eines *Sprechakt*zyklus.

Dialog (*dialog*) Ein Schritt einer Phase eines Sprechakts. Das Mittel des Dialogs ist nicht bestimmt. Eine Phase kann beliebig viele Dialogschritte erfordern. Dialoge enthalten strukturierte Informationen, ihren *Inhalt*.

Inhalt (*content*) Ein Teil eines Dialogs in Form eines inspizierbaren Bedeutungsträgers.

Modifikator (*modifier*) Mit Modifikatoren kann eine bestehende Konversation in Form einer Sub-Konversation an die Erfordernisse der einbettenden Konversation angepasst werden. Dialoge der Sub-Konversation können konfiguriert, automatisiert oder ausgeblendet werden.

Historie (*history*) Die Historie bildet die gemeinsame Vergangenheit von Kunde und Dienstleister innerhalb einer Konversation. Sie wird transparent vom System fortgeschrieben und enthält alle relevanten Daten einer Konversation, insbesondere alle Anfragen und Dialoge. Sowohl Kunde als auch Dienstleister haben vollständigen Zugang zu den gesammelten Daten, die von beiden Seiten als verbindlich betrachtet werden.

Abbruch (*breakdown*) Ein Abbruch ist eine spezielle Anfrage an einen Dienstleister in einer Fehlersituation. Diese Anfragen werden in der Regel vom System im Namen des Kunden an den Dienstleister gestellt (zum Beispiel bei einer Zeitüberschreitung des Kunden in einem Dialog).

Konversations-Stadium (*conversation stage*) Eine Konversation befindet sich immer in einem definierten Stadium. Damit werden die Phasen des Spechaktzyklus widergespiegelt.

Literaturverzeichnis

[AB87] M.P. Atkinson and P. Bunemann. Types and Persistence in Database Programming Languages. *ACM Computing Surveys*, 19(2), June 1987.

[ABD+90] M. Atkinson, F. Bancilhon, D. De Witt, K. Dittrich, D. Maier, and S. Zdonik. The Object-Oriented Database System Manifesto. In *Deductive and Object-oriented Databases*. Elsevier Science Publishers, Amsterdam, Netherlands, 1990.

[AM95] Malcolm Atkinson and Ronald Morrison. Orthogonally Persistent Object Systems. *VLDB Journal*, 4(3):319–401, 1995.

[BFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol — HTTP/1.0. Internet-Draft draft-ietf-http-v10-spec-05.ps, HTTP Working Group, February 1996.

[DDJ+96] Giorgio De Michelis, Eric Dubois, Matthias Jarke, Florian Matthes, John Mylopoulos, Klaus Pohl, Joachim Schmidt, Carson Woo, and Eric Yu. Cooperative Information Systems: A Manifesto. In John Mylopoulos, Giorgio De Michelis, and Florian Matthes, editors, *Architectural Issues in Cooperative Information Systems: A Tutorial*, June 1996.

[DHL90] Umeschwar Dayal, Meichum Hsu, and Rioka Ladin. Organizing Long Running Activities with Triggers and Transactions. *ACM SIGMOD*, 1990.

[EH96] Ramez Elmasri and Abdelsalam Helal. Standards for Database Interoperability and Portability: CORBA, ODMG and SQL. Tutorial Presented at the International Conference on Cooperative Information Systems (CoopIS '96), Brussels, Belgium, June 1996.

[EW94] Oren Etzioni and Daniel Weld. A Softbot-based Interface to the Internet. *CACM*, July 1994.

[FGHW88] Fernando Flores, Michael Graves, Brad Hartfield, and Terry Winograd. Computer Systems and the Design of Organisational Interaction. *ACM Transaction on Office Information Systems*, 6(2):153–172, April 1988.

[GM96] A. Gawecki and F. Matthes. Exploiting Persistent Intermediate Code Representations in Open Database Environments. In *Proceedings of the 5th Conference on Extending Database Technology, EDBT'96*, Avignon, France, March 1996. Springer-Verlag. (to appear).

[Hoh95] Fritz Hohl. Konzeption eines einfachen Agentensystems und Implementation eines Prototyps. Diplomarbeit, Fakultät Informatik, Universität Stuttgart, 1995.

[HP94] G. Hamel and C. Parhalad. *Competing for the Future*. Harvard Business School Press, Boston, Massachusetts, 1994.

[Joh97] Nico Johannisson. Eine Umgebung für mobile Agenten: Agentenbasierte verteilte Datenbanken am Beispiel der Kopplung autonomer Internet „WebSite-Profiler". Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1997.

[Kab96] Kabel New Media. Webtracking – Die 10 wichtigsten Fragen und Antworten. http://www.kabel.de/KNMantworten.html, September 1996.

[Lau93] Thomas Lauer. Paketdienst: Grundlagen zu Object Linking and Embedding (OLE). *c't Magazin für Computertechnik*, pages 264–272, April 1993.

[Lau97] Jan T. Laub. Analyse der Web-Server-Nutzung: Krititerien, Protokolldateien und Auswertungssoftware. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1997.

[LKK93] Peter C. Lockemann, Gerhard Krüger, and Heiko Krumm. *Telekommunikation und Datenhaltung*. Hanser-Verlag, München, Wien, 1993.

[Mae94] Pattie Maes. Agents that reduce work and information overload. *CACM*, July 1994.

[Mat93] F. Matthes. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung*. Springer-Verlag, Berlin, Heidelberg, 1993.

[Mat95] F. Matthes. Higher-Order Persistent Polymorphic Programming in Tycoon. In M.P. Atkinson, editor, *Fully Integrated Data Environments*. Springer-Verlag (to appear), 1995.

[Mat96] Bernd Mathiske. *Mobilität in persistenten Objektsystemen*. Dissertation, Universität Hamburg, Fachbereich Informatik, 1996.

[MMM93] B. Mathiske, F. Matthes, and S. Müßig. The Tycoon System and Library Manual. DBIS Tycoon Report 212-93, Fachbereich Informatik, Universität Hamburg, December 1993. (Revised 19-jan-1996).

[MMS94] F. Matthes, S. Müßig, and J.W Schmidt. Persistent Polymorphic Programming in Tycoon: An Introduction. FIDE Technical Report Series FIDE/94/106, FIDE Project Coordinator, Department of Computing Sciences, University of Glasgow, Glasgow G128QQ, August 1994.

[MMS95] F. Matthes, R. Müller, and J.W. Schmidt. Towards a Unified Model of Untyped Object Stores: Experience with the Tycoon Store Protocol. In M.P. Atkinson, editor, *Fully Integrated Data Environments*. Springer-Verlag (to appear), 1995.

[MO94] Hanns-Martin Meyer and Karl Obermayr. *Objekte integrieren mit OLE2: Microsofts Basistechnologie für objektorientierte Architektur*. Springer-Verlag, Berlin, Heidelberg, 1994.

[MS93] F. Matthes and J.W. Schmidt. System Construction in the Tycoon Environment: Architectures, Interfaces and Gateways. In *Proceedings of Euro-Arch'93 Congress*. Springer-Verlag, October 1993.

[MS94] F. Matthes and J.W. Schmidt. Persistent Threads. In *Proceedings of the Twentieth International Conference on Very Large Data Bases, VLDB*, pages 403–414, Santiago, Chile, September 1994.

[MSS95] F. Matthes, G. Schröder, and J.W. Schmidt. Tycoon: A Scalable and Interoperable Persistent System Environment. In M.P. Atkinson, editor, *Fully Integrated Data Environments*. Springer-Verlag (to appear), 1995.

[MSW95] F. Matthes, J.W. Schmidt, and J. Wahlen. Using Extensible Grammars for Data Modeling. In M.P. Atkinson, editor, *Fully Integrated Data Environments*. Springer-Verlag (to appear), 1995.

[Obj95] Object Management Group. The Common Object Request Broker: Architecture and Specification. Document PTC/96-03-04, Revision 2.0, OMG, July 1995.

[Obj96] Object Management Group. What is CORBA. http://ruby.omg.org/corba.htm, June 1996.

[OHE96] Robert Orfali, Dan Harkey, and Jeri Edwards. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, New York, 1996.

[ÖV94] M.T. Özsu and P. Valduriez. Distributed Data Management: Unsolved Problems and New Issues. In T. Casavant and M. Singhal, editors, *Readings in Distributed Computing*. IEEE Computer Society Press, 1994.

[Pie96] Andreas Piellusch. Synchronisation langlebiger Aktivitäten in persistenten Objektsystemen. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, Juni 1996.

[Sch93] Alexander Schill. *DCE – Das OSF Distributed Computing Environment, Einführung und Grundlagen.* Springer-Verlag, Berlin, Heidelberg, 1993.

[Sch96] Martin Schoel. CORBA-basierte Integration von Diensten eines Umweltinformationssystems. Diplomarbeit, Fakultät für Informatik, Universität Karlsruhe, Januar 1996.

[Sie96] Jon Siegel. *CORBA Fundamentals and Programming.* John Wiley & Sons, New York, 1996.

[Whi94] James E. White. Telescript Technology: The Foundation for the Electronic Marketplace. White paper, General Magic, 1994.

[Win87] Terry A. Winograd. A Language/Action Perspective on the Design of Cooperative Work. Technical Report STAN-CS-87-1158, Department of Computer Science, Stanford University, Stanford, CA 94305, April 1987.

[YWT+95] E. Yourdon, K. Whitehead, J. Thomann, K. Oppel, and P. Nevermann. *Mainstream Objects: An Analysis and Design Approach for Business.* Yourdon Press (Prentice Hall), 1995.