Historienmanagement für kooperative Aktivitäten

Diplomarbeit

eingereicht beim
Fachbereich Informatik
der Universität Hamburg
von

Wolfgang Kruse

betreut durch

Prof. Dr. Joachim W. Schmidt

Prof. Dr. Klaus Brunnstein

8. Dezember 1997

Inhaltsverzeichnis

1. Einführung und Motivation 1

1.1 Zielsetzung 4

1.2 Gliederung 4

2. Das Modell der *Business Conversations* 7

2.1 Konzepte des Modells 7

2.1.1 Das Verarbeitungsmodell von Konversationen 7

2.1.2 Sekundäre Konversationen 9

2.1.3 Finale Dialoge 10

2.2 Die *Tycoon Business Conversations* 10

2.2.1 Konversationsspezifikationen 11

2.2.2 Konversationsinstanzen 15

2.3 Invarianten des Modells 17

3. Analyse der agentenbasierten Realisierung des *Business Conversation*-Modells ... 19

3.1 Die Architektur des Tycoon-Agentensystems 19

3.2 Domänen und Orte 21

3.3 Die Agenten 22

3.4 Das Nachrichtensubsystem 26

3.5 Der Mediator 28

3.6 Ein Beispiel für die Benutzung der Agenten 31

4. Anforderungsanalyse 37
4.1 Anforderungen an die Protokollierung 37
4.2 Alternativen zur Einbindung des Protokollierungsdienstes 38
4.3 Alternativen zum Aufbau des Protokollierungsdienstes 40
4.4 Begrenzung des Datenvolumens 42
4.5 Funktionalität des Protokollierungsdienstes 43
5. Design 45
5.1 Der Aufbau des Protokollierungsdienstes 45
5.2 Der Mediator-Dienst Archiv 47
5.3 Das Archiv 50
5.4 Der Archivar 52
6. Realisierung 55
6.1 Die Konversation 55
6.1.1 Starten einer Protokollierung 55
6.1.2 Protokolliste ansehen 57
6.1.3 Stoppen einer Protokollierung 58
6.1.4 Neustart einer Protokollierung 58
6.1.5 Löschen eines Protokolls 58
6.1.6 Dialogliste eines Protokolls ansehen 59
6.1.7 Dialog ansehen 59
6.1.8 Paßwort ändern 60
6.2 Der Archivar 60
6.3 Der Mediator-Dienst Archiv 61

6.4 Das Archiv 62
6.5 Die Integration des Protokollierungsdienstes 65
7. Nutzungsmöglichkeiten des Protokollierungsdienstes 67
7.1 Dokumentationsvorschriften 67
7.1.1 Gesetzliche Vorschriften 67
7.1.2 Vertragliche Vorschriften 68
7.2 Dokumentationseigeninteresse 69
7.2.1 Risikobegrenzung 69
7.2.2 Performanzkontrolle und -steigerung 70
8. Zusammenfassung und Ausblick 73
8.1 Zusammenfassung 73
8.2 Bewertung 73
8.3 Ausblick 75
Anhang A Glossar *Business Conversations* 77
Anhang B Agententerminologie 79
Anhang C Die Notation von OMT 81
Literaturverzeichnis 83

Abbildungsverzeichnis

Abbildung 1.1: Kommunikation gemäß den *Conversations for Action* 2

Abbildung 1.2: Modellbildung für Informationssysteme 3

Abbildung 2.1: Verfeinerung mittels sekundärer Konversationen.....9

Abbildung 2.2: Spezifikationen von Konversationen, Dialogen und Inhalten 13

Abbildung 2.3: Instanzen von Konversationen, Dialogen und Inhalten 14

Abbildung 3.1: Architektur des Tycoon-Agentensystems.....20

Abbildung 3.2: Gliederung des Agentensystems.....20

Abbildung 3.3: Orte und Agenten in den Domänen.....22

Abbildung 3.4: Die Kommunikation der Threads der Agenten 23

Abbildung 3.5: Objektstrukturmodell des Agent und seiner Rollen.....24

Abbildung 3.6: Nachrichtenaustausch während einer Konversation.....28

Abbildung 3.7: Objektstrukturmodell von Mediator und Mediator-Dienst 29

Abbildung 3.8: Nachrichtenfluß im Agentensystem 31

Abbildung 3.9: Dialoge und Anfragen der Beispielkonversation.....32

Abbildung 4.1: Systemdarstellung des Konzeptes.....42

Abbildung 5.1: Einordnung des Protokollierungsdienstes in die Schichten-Architektur 46

Abbildung 5.2: Objektstrukturmodell des Protokollierungsdienstes 47

Abbildung 5.3: Objektstrukturmodell des Mediator-Dienstes Archiv 49

Abbildung 5.4: Objektstrukturmodell des Archivs 51

Abbildung 5.5: Konversation des Archivars 53

Abbildung 6.1: Nachrichtenfluß beim Starten einer Protokollierung 56

Abbildung C.1: Darstellung von Klassen 81

Abbildung C.2: Kardinalitäten von statischen Beziehungen 82

Abbildung C.3: Vererbung 82

Abbildung C.4: Aggregation 82

1. Kapitel

Einführung und Motivation

Am Arbeitsbereich Datenbanken und Informationssysteme der Universität Hamburg entstand das Modell der *Business Conversations*, das die zielgerichtete, lang andauernde Kommunikation zweier Akteure beschreibt, die in den Rollen Kunde und Dienstleister handeln. Dieses Modell basiert auf den *Conversations for Action* von Fernando Flores und Terry Winograd ([FGHW88] und [Win87]). Es wurde als Kooperationsmodell für das Agentensystem verwendet, das Nico Johannisson [Joh97] am Arbeitsbereich entwickelt hat. Die folgenden Ausführungen zu dem Modell entsprechen im wesentlichen denen von Johannisson.

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

Basis des Modells ist die Annahme, daß die Kommunikationsmuster der Sprechakte universell sind, d.h. sie sind von der Art der Akteure (Software, Menschen) und von den verwendeten Kommunikationsmedien und -protokollen unabhängig.

Dieser Annahme folgend abstrahiert das Modell von der Art der Akteure, so daß keine Unterscheidung zwischen den Interaktionen von zwei menschlichen Akteuren, von Mensch und maschinellem Akteur oder von zwei maschinellen Akteuren stattfindet.

Die Theorie der *Conversations for Action* unterteilt eine Kommunikation mit dem Ziel einer Aktion zwischen einem Kunden und einem Dienstleister in einen Zyklus mit vier Phasen.

Die Beziehung zwischen einem Kunden und einem Dienstleister folgt dabei immer dem in der Abbildung 1.1 skizzierten Ablauf. Der erste Schritt ist die unverbindliche Anfrage des Kunden nach einer Dienstleistung. Am Ende dieses Schrittes steht dann, in der Übereinkunftsphase, die Verhandlung mit dem Dienstleister, einen gewählten Dienst tatsächlich verbindlich zu erbringen. Das Erbringen der vereinbarten Leistung geschieht in der dritten Phase, an deren Ende aus der Sicht des Dienstleisters die geforderte Leistung erbracht ist. In der vierten Phase, der Rückmeldung, teilt der Kunde dem Dienstleister mit, ob die vom Dienstleister erbrachte Leistung seinen Anforderungen entspricht.

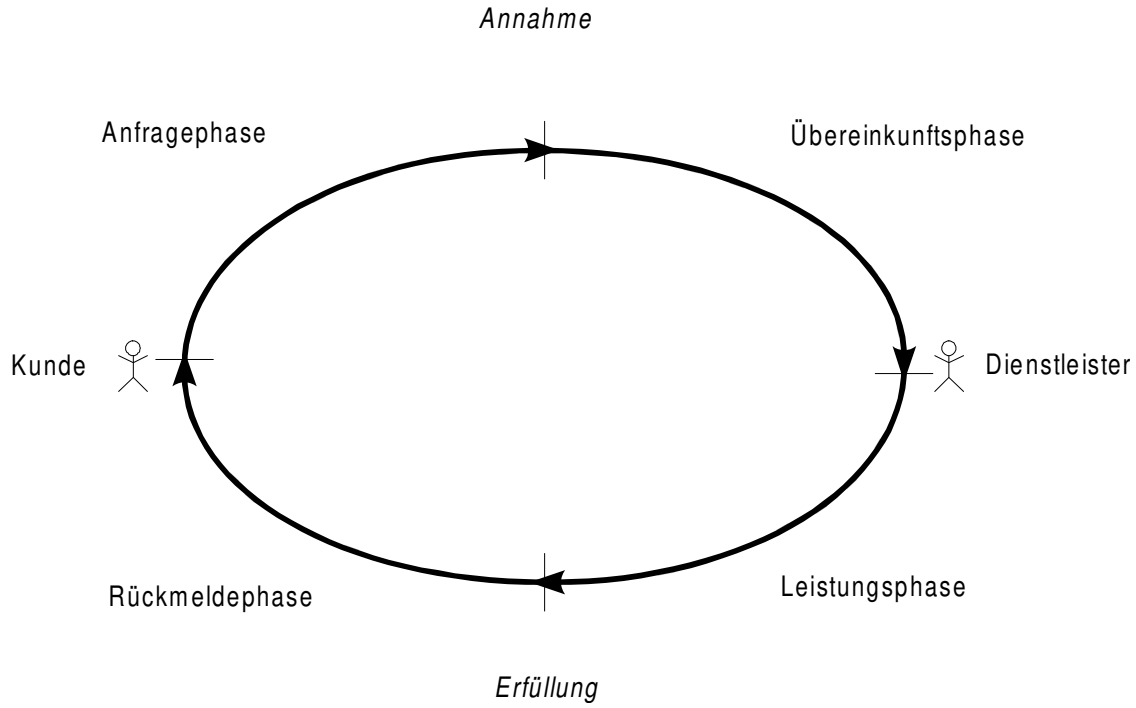

Abbildung 1.1: Kommunikation gemäß den *Conversations for Action*

Nur wenn auch diese Phase erfolgreich abgeschlossen ist, ist das gemeinsame Ziel von Kunde und Dienstleister, und damit das Ziel ihrer Kommunikation, erreicht. Der Austausch von Sprechakten innerhalb dieser vier Phasen wird unter dem Begriff der Konversation zusammengefaßt, der sich damit immer auf einen konkreten Gegenstand, die Dienstleistung oder Aktion, bezieht.

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* ergänzt bereits etablierte Modelle (wie z.B. OMT[1], Entity-Relationship-Modell), welche die Modellierung von Informationssystemen nach innen gestatten (siehe Abbildung 1.2). Diese Modelle können aus der Sicht einer Anwendung als systemnahe Modelle bezeichnet werden. Sie strukturieren Systemkomponenten *bottom-up* und objekt- bzw. datenzentriert, d.h. auf Objekt- oder Entitätsebene.

Die *Business Conversations* hingegen stellen den kooperativen Charakter eines Informationssystems in den Vordergrund. Durch die Betrachtung der von dem System nach außen erbrachten Dienstleistungsprozesse wird eine anwendungsnahe Modellierung des Systems unterstützt. Hierdurch wird der Abstand zwischen der realen Welt und ihrer Abbildung durch Informationssysteme verkleinert. Das Modell versucht nicht, Informationssysteme auf der Ebene von Objekten oder Klassen zu beschreiben. Vererbungs-, Aggregations- oder Generalisierungsbeziehungen von Objekten innerhalb des betrachteten Informationssystems müssen durch andere, für diesen Zweck geeignete Modelle beschrieben werden.

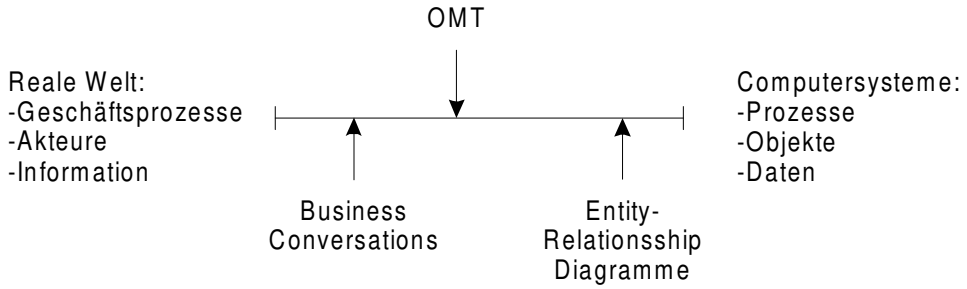

Abbildung 1.2: Modellbildung für Informationssysteme

Ein wichtiger Aspekt beim Einsatz kooperativer Informationssysteme ist die Auswertung der Abläufe im Rahmen der Geschäftsprozeßoptimierung mit dem Ziel einer ständigen Verbesserung der Geschäftsprozesse. Um solche Analysen zu ermöglichen, müssen Informationen über die Abläufe protokolliert werden. Aber auch die nachträgliche Verfolgung der Abläufe zur Kontrolle sowie die Dokumentation der Geschehnisse aus rechtlichen oder anderen Gründen wird erst durch ein Protokoll sichergestellt.

---

[1] Object Modeling Technique

1.1 Zielsetzung

Ziel dieser Arbeit ist es, für das Tycoon-Agentensystem, welches das Kooperationsmodell der *Business Conversations* verwendet, eine Protokollierungskomponente zu konzipieren und prototypisch zu realisieren. Für die Analyse des Agentensystems und den Entwurf der Protokollierungskomponente wird die objektorientierte Methode OMT verwendet [RBP+91].

Da in dieser Arbeit das Agentensystem und die *Business Conversations* zum ersten Mal als Entwicklungsgrundlage verwendet werden, stellt die Arbeit auch einen Test für deren Konzepte und Realisierungen dar. Deswegen wird auch betrachtet, ob und wieweit Änderungen notwendig werden, und ob diese lediglich die Realisierungen oder auch die Konzepte betreffen.

Das Agentensystem wurde mit der im Rahmen des *FIDE*$_2$-Projekts am Arbeitsbereich Datenbanken und Informationssysteme der Universität Hamburg entwickelten, integrierten, persistenten und polymorphen Programmierumgebung *Tycoon*[2] implementiert. Damit ist Tycoon neben dem Agentensystem die Basis für die prototypische Realisierung. Näheres zu Tycoon ist in [Mat93], [MMM93] und [MMS94] zu finden.

1.2 Gliederung

In Kapitel 2 wird das Modell der *Business Conversations* detailliert vorgestellt und dessen Realisierung analysiert. Da die Kooperation der Agenten auf der Basis dieses Modells stattfindet, muß dessen Verarbeitungsmodell auf die zu protokollierenden Daten untersucht werden.

Die Analyse des Agentensystems wird in Kapitel 3 vorgenommen. Dabei wird zunächst dessen Architektur betrachtet. Anschließend werden die Fähigkeiten der Agenten und die Kommunikationsinfrastruktur untersucht. Damit stehen die notwendigen Informationen zum Aufbau einer Protokollierungskomponente zur Verfügung.

Kapitel 4 stellt die Anforderungen an die Protokollierungskomponente und Alternativen zu deren Aufbau vor. Dabei werden die Alternativen mit Hilfe der Anforderungen verglichen und hinsichtlich ihrer Eignung bewertet.

---

[2] Typed Communicating Objects in Open Enviroments

Das Design der implementierten Protokollierungskomponente wird in Kapitel 5 detailliert beschrieben, während Kapitel 6 die Realisierung darstellt.

In Kapitel 7 werden die Nutzungsmöglichkeiten für eine Protokollierung näher betrachtet und es wird untersucht, ob die realisierte Protokollierungskomponente den verschiedenen Ansprüchen gerecht werden kann.

Kapitel 8 faßt die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf weiterführende Arbeiten.

2. Kapitel

Das Modell der *Business Conversations*

Dieses Kapitel stellt das Modell der *Business Conversations* vor. Das Modell wurde von Florian Matthes ([Mat97a] und [Mat97b]) und Nico Johannisson [Joh97] in Zusammenarbeit mit Ingo Richtsmeier [Ric97], Heiko Zade und Wolfgang Kruse entwickelt. Die Ausführungen entsprechen im wesentlichen denen von Johannisson. Nach der Betrachtung des Modells wird in Kapitel 2.2 die Analyse der Realisierung vorgenommen. Alle Begriffe der *Business Conversations* werden im Glossar im Anhang A erläutert.

Die Kooperation der Agenten im Tycoon-Agentensystem basiert auf den *Business Conversations*. Damit repräsentieren die *Business Conversations* die zu protokollierenden Daten und Zustände und werden deshalb im folgenden detailliert untersucht.

2.1 Konzepte des Modells

Ein integraler Bestandteil des in der Einführung vorgestellten Modells der *Business Conversations* ist ein vereinbartes Verarbeitungsmodell, das die Interaktion zwischen zwei Partnern definiert und ihre jeweiligen Rollen innerhalb der Interaktion, d.h. innerhalb eines Gesprächs, festlegt. Dieses Verarbeitungsmodell muß von beiden Gesprächspartnern eingehalten werden, um den korrekten Fortgang des Gesprächs zu gewährleisten.

2.1.1 Das Verarbeitungsmodell von Konversationen

In einer Konversation wird jede zielgerichtete Interaktion (d.h. jeder mögliche Sprechakt der *Conversations for Action*) durch *Dialoge* realisiert. Mit Hilfe dieser Dialoge werden Informationen so strukturiert, daß der jeweilige Kommunikationspartner die im Sprechakt enthaltene Information seinem eigenen Ausführungskontext zuordnen kann.

Dialoge entsprechen Formularen in der realen Welt; ihre Struktur ist festgelegt und durch den Verwendungszweck bedingt. In den durch die Struktur abgesteckten Grenzen können sie von allen beteiligten Partnern genutzt werden, sei es durch Ausfüllen der vorgegebenen Freifelder oder durch Untersuchung des vorhandenen Inhalts. Durch Dialoge werden von der ausgebenden Seite mögliche Folgeaktionen impliziert, die mit dem Dialog in direktem Zusammenhang stehen. Auf diese Weise repräsentiert ein Dialog einen bestimmten Punkt in einer längeren Beziehung zwischen zwei Partnern, der Schlüsse auf gültige Fortsetzungen der Beziehung in der Zukunft ermöglicht.

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

Zustandsübergänge, d.h. der Beginn eines neuen Dialogs innerhalb einer Konversation, werden durch Auswertung und Verarbeitung (die *Aktion*) einer Anfrage des Kunden beim Dienstleister durchgeführt.

Dabei ist der Kunde derjenige Akteur, welcher den Zeitpunkt des Zustandsüberganges bestimmt; indem der Kunde eine Anfrage an den Dienstleister stellt, führt dieser seine mit der Anfrage assoziierte Bearbeitungsregel 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 durch den Dialog bestimmt, welcher den aktuellen Zustand der Konversation repräsentiert. Dies entspricht in der Formularanalogie der Möglichkeit, in einem vorgegebenen Formular einen Kundenwunsch ankreuzen zu können.

Eine Konversation zwischen einem Kunden und einem Dienstleister wird mit diesen Mechanismen auf eine festgelegte Abfolge von Dialogen und Anfragen übertragen. Der Dienstleister gibt dabei jeweils einen Dialog vor, den der Kunde erhält und bearbeiten kann. Der Kunde ist jedoch immer der *Initiator* einer Konversation und der Dienstleister der *Akzeptor*.

Auf diese Weise werden Ketten aus Ereignissen und Aktionen [DHL90] gebildet. Die beiden Partner der Konversation definieren autonom Bearbeitungsregeln, mit denen sie auf die Ereignisse des jeweils anderen Partners reagieren und dadurch Aktionen ausführen.

Die gültigen Transitionen einer Konversation, wie sie durch die Regeln implementiert werden, sind durch die zugehörige Spezifikation bestimmt (siehe auch Kapitel 2.2.1).

Menschliche Akteure simulieren Kundenregeln, indem sie den Inhalt der Dialoge lesen und gemäß ihren Wünschen verändern. Maschinelle Kunden besitzen einen Programmkode für die Regeln, der die Dialoge entsprechend der kodierten Vorschrift interpretiert bzw. manipuliert. 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, dessen Felder er bei Bedarf vorher mit inhaltlichen Vorgaben gefüllt hat, an den Kunden übermittelt. Durch die abwechselnde Ausführung einer Kundenregel und einer Dienstleisterregel wird jede Konversation gemäß der zugrundeliegenden Spezifikation systematisch entwickelt.

Innerhalb einer Konversation kann ein Dienstleister, dann jedoch in der Rolle eines Kunden, eine weitere Konversation mit einem anderen Dienstleister initiieren, sofern das zur Ausführung seiner Bearbeitungsregel notwendig ist. Dieses führt zum Konzept der *sekundären Konversation*.

2.1.2 Sekundäre Konversationen

Mit der Technik der sekundären Konversation ist eine Erweiterung der Beziehung zwischen Kunde und Dienstleister möglich (siehe Abbildung 2.1). Dabei wird der Dienstleister A_2 innerhalb einer Konversation mit einem Kunden A_1 selbst zum Kunden eines weiteren Dienstleisters A_3.

Synchrone sekundäre Konversationen unterbrechen die initiierende primäre Konversation für die Dauer ihrer Ausführung. Sie bilden damit eine eingeschobene Konversation, bei der ein impliziter Rollenwechsel des Akteurs A_2 stattfindet. Wird

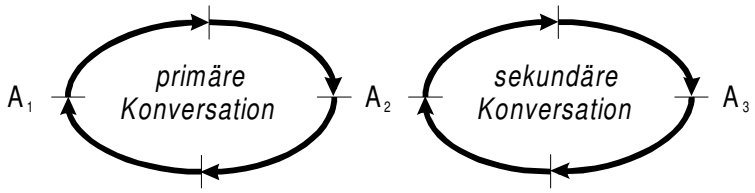

Abbildung 2.1: Verfeinerung mittels sekundärer Konversationen

eine sekundäre Konversation asynchron ausgeführt, ist keine Unterbrechung der primären Konversation notwendig. Eine asynchrone sekundäre Konversation verhält sich wie eine primäre, kann jedoch die übergeordnete Konversation *verzögern*, wobei die Atomarität der Bearbeitungsregel aufgehoben wird.

2.1.3 Finale Dialoge

Jede Konversation besitzt eine definierte Menge von *finalen* Dialogen, die sie erreichen kann. Jeder Dialog ohne einen Folgedialog, d.h. ohne eine mögliche Anfrage, ist ein finaler Dialog der Konversation. Die Konversationsbeziehung zwischen den beteiligten Partnern wird bei Erreichen eines solchen Dialogs beendet.

Das Modell definiert einen finalen Dialog für das erfolgreiche Ende jeder Konversation (d.h. sowohl Kunde als auch Dienstleister bestätigen das erfolgreiche Erreichen des gemeinsamen Ziels) sowie einen finalen Fehlerdialog. Dieser Fehlerdialog ist implizit von jedem Dialog einer Konversation aus erreichbar, ohne daß dies in der Spezifikation vermerkt werden muß. Dieser Dialog bietet die Möglichkeit, Ausnahmebehandlungen in den Bearbeitungsregeln für Konversationen vorzusehen. Eine solche Ausnahme kann von beiden Partnern jederzeit in einer Konversation ausgelöst werden. Die entsprechende zugehörige Anfrage, mit der der Kunde den Übergang in einen Ausnahmezustand anzeigt, wird durch das Modell definiert.

Die Entscheidung, unter welchen Umständen welcher Dialog welcher Phase aus Abbildung 1.1 zugeordnet wird, ist eine Designentscheidung bei der Anwendungsentwicklung. Das Phasenmodell kann von einer Anwendung genutzt werden, um Punkte innerhalb einer Konversation mit dem Partner zu vereinbaren, mit deren Erreichen beide Partner verbindliche Aussagen über den Gesprächsverlauf treffen.

2.2 Die *Tycoon Business Conversations*

Die *Tycoon Business Conversations* stellen die Realisierung des vorgestellten Modells dar, die vom Tycoon-Agentensystem verwendet wird.

Es werden wie im Modell zwei Beschreibungsebenen für Konversationen verwendet, die im folgenden getrennt vorgestellt werden:

- Die Ebene der *Konversationsspezifikationen*: Auf dieser Ebene wird die Struktur möglicher Konversationen definiert. Diese Ebene entspricht der Typebene einer Programmiersprache, mit dem Unterschied, daß Konversationsspezifikationen eine Laufzeitrepräsentation besitzen. Diese Spezifikationen bilden Fabriken (engl. *factories*), welche die Erzeugung von Konversationen, ihren Instanzen, steuern.

- 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 Konversations-

spezifikation. Sie werden anhand dieser Spezifikation erzeugt und sind an ein Kunde-Dienstleister-Paar gebunden.

2.2.1 Konversationsspezifikationen

Die Abbildung 2.2 zeigt die Struktur der Spezifikationen von Konversationen. Das Diagramm ist in der OMT-Notation, die im Anhang C erläutert wird, dargestellt.

Die Konversationsspezifikation legt fest, welche Struktur ein Konversationsobjekt im Falle einer Instantiierung haben soll. Sie umfaßt dabei sowohl den statischen Teil einer Konversation, d.h. die Datenmodellierung für den Informationsaustausch, als auch den dynamischen Teil der Beziehung zwischen Kunde und Dienstleister, die Prozeßmodellierung.

Spezifikationen sind zur Laufzeit inspizierbar, d.h. beide Partner können untersuchen, ob die Konversation in ihren jeweiligen Kontext und zu ihrer Auffassung vom Kooperationsziel paßt.

In dieser Realisierung enthält jede Konversationsspezifikation (*TBCSpec*) mehrere Dialogspezifikationen (*DialogSpec*). Eine Dialogspezifikation ist als initialer Dialog ausgezeichnet, jede Konversation beginnt mit diesem Dialog. Die Dialogspezifikation besteht aus einer Inhaltsspezifikation (TBC*Content.Spec*) und Spezifikationen für Anfragen (*RequestSpec*). Jede Anfragespezifikation ist über die *possibleReplies*-Beziehung mit den Spezifikationen der nachfolgenden Dialoge verbunden. Das Attribut *ellipses* zeigt an, ob sich die Aktion, die beim Dienstleister auf die Auswertung des Dialogs folgt, über mehrere Dialoge, deren Spezifikationen nicht sichtbar sind, hinziehen kann.

Die Inhalte können auf verschiedene Art und Weise spezifiziert werden. Zum einen gibt es einfache Basistypen wie ganze Zahlen (*Int*), rationale Zahlen (*Real*), Zeichenketten (*String*) und einen Datumstyp (*Date*). Komplexe Inhalte können mit Records (*Rcd*), Varianten (*Variant*), Auswahllisten (*MultipleChoice* und *SingleChoice*) oder Sequenzen (*Sequence*) modelliert werden, wobei auch verschachtelte Inhalte zugelassen sind. Für alle Inhaltstypen existieren Operationen zum Erzeugen der entsprechenden Spezifikationsobjekte. Die Operationen auf den Spezifikationsobjekten sind in der Abbildung aus Gründen der Übersichtlichkeit nicht dargestellt.

Eine Konversationsspezifikation wird durch *tbcSpec.new(...)* generiert. Dieser Funktion müssen der Name, die Spezifikationen der Dialoge und des Initialdialogs übergeben werden. Das erzeugte Objekt erhält den Namen und stellt zwei Opera-

tionen zur Verfügung. Durch *dialogSpecs()* wird eine Liste der Dialogspezifikationen und mit *initialDialogSpec()*[3] die Spezifikation des Initialdialogs ausgegeben.

Eine Dialogspezifikation wird durch *tbcSpec.dialogSpec(...)* erstellt. Dazu müssen der in der Konversationsspezifikation einmalige Dialogname, die Anfragespezifikationen und die Inhaltsspezifikation angegeben werden. Dieses Objekt verfügt neben seinen Namen auch über zwei Funktionen. Die Funktion *possibleRequests()* liefert eine Liste der Anfragespezifikationen, *contentSpec()* die Inhaltsspezifikation.

Mit *tbcSpec.requestSpec(...)* wird eine Anfragespezifikation erzeugt. Dazu werden ein im Dialog einmaliger Anfragename und die Namen der möglichen Folgedialoge benötigt. Das Spezifikationsobjekt enthält außer seinem Namen auch das Attribut *ellipses*, das angibt, ob es an dieser Stelle weitere, nicht aufgeführte Dialoge als Antwort geben kann. Durch die Funktion *possibleReplies()* wird die Liste der Namen der Folgedialoge ausgegeben.

Die Inhaltsspezifikation ist durch ein variantes rekursives Tupel implementiert. Die Varianten stellen die verschiedenen Inhaltstypen dar. Für jeden Typ existiert eine Funktion, die die entsprechende Typspezifikation erzeugt. Diese Typspezifikation ist schon eine Inhaltsspezifikation. Durch die Rekursion in einigen Typen ist es möglich, komplexe Inhaltsspezifikationen zu konstruieren. So gibt es z.B. den Typ *rcd*, der aus mehreren Inhaltsspezifikationen besteht.

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, sowohl in der Implementationssprache als auch im eigenen Typsystem. Sie können somit benannt, gespeichert oder verschickt werden und besitzen eine Laufzeitrepräsentation.

Die Agenten benötigen für das Führen einer Konversation die zugehörige Spezifikation. Diese wird als Generator für die Dialogobjekte benutzt, die im Laufe der Konversation ausgetauscht werden.

---

[3] Im Kode heißt die Funktion - wahrscheinlich aufgrund eines Schreibfehlers - *initalDialogSpec()*.

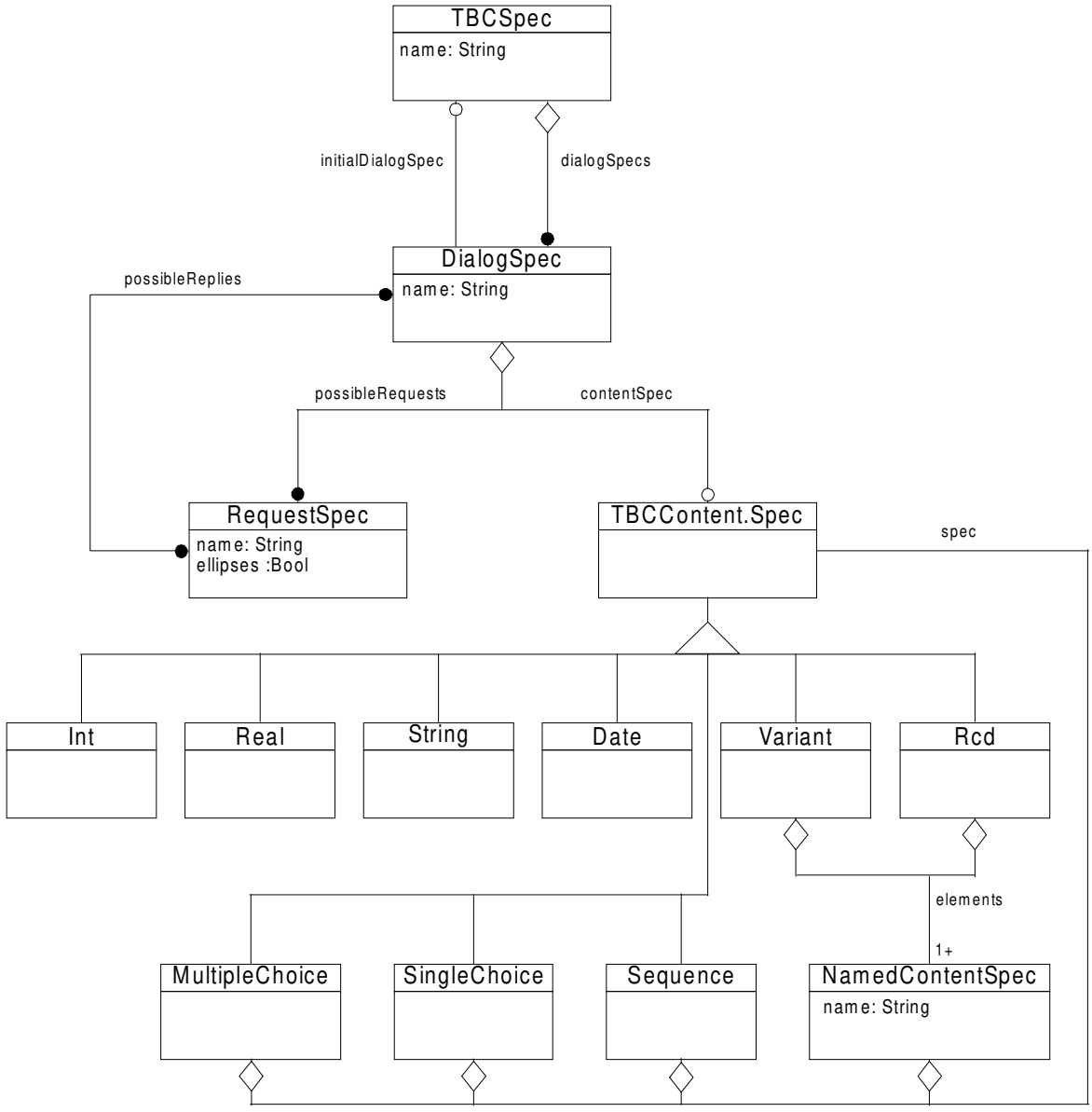

Abbildung 2.2: Spezifikationen von Konversationen, Dialogen und Inhalten

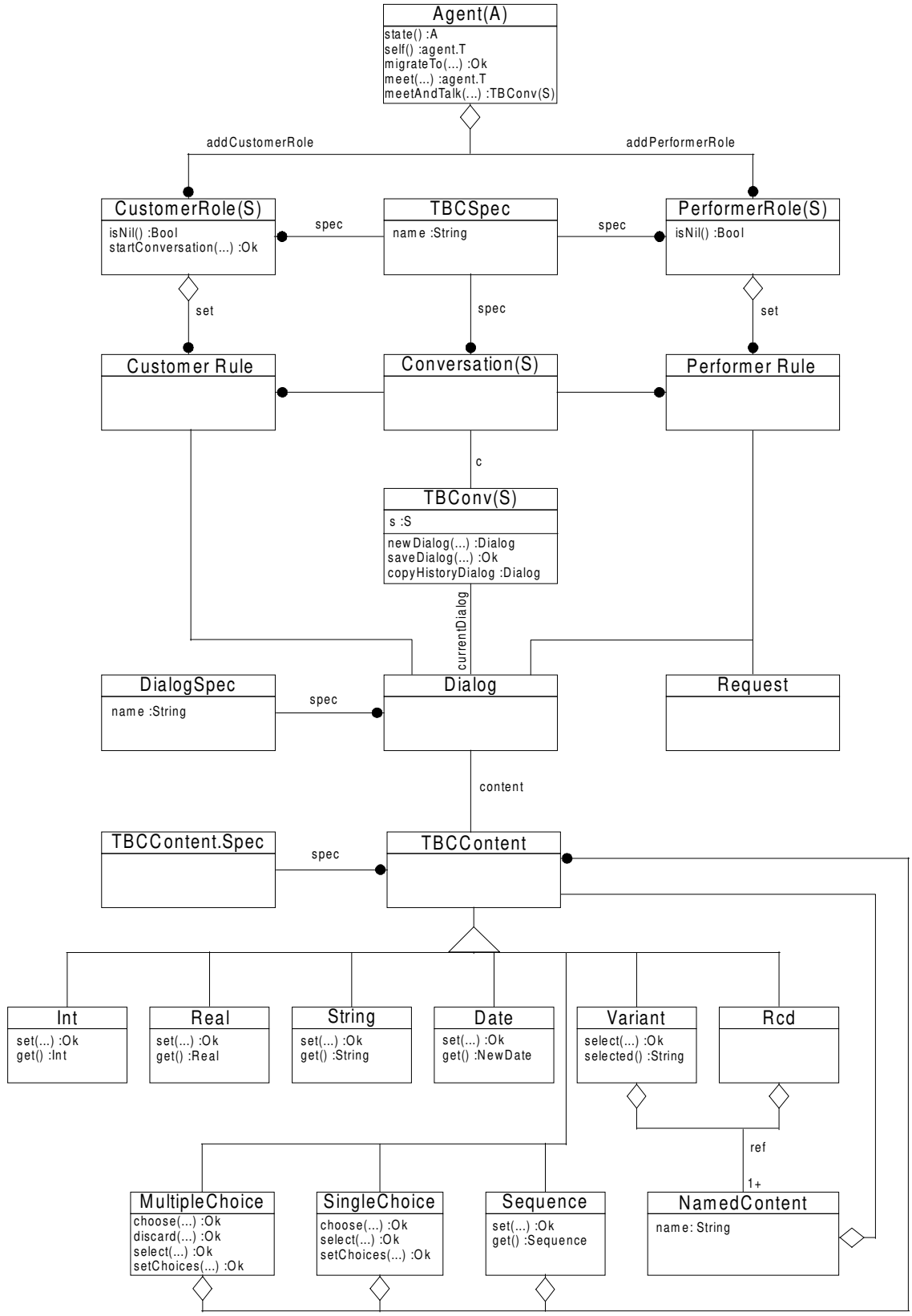

Abbildung 2.3: Instanzen von Konversationen, Dialogen und Inhalten

2.2.2 Konversationsinstanzen

Die Abbildung 2.3 zeigt den Aufbau der Instanzen von Konversationen und deren Zusammenhang mit den Spezifikationen. Da im Tycoon-Agentensystem ausschließlich Agenten die Akteure sind, wird hier gleichzeitig die Beziehung von Agent und Konversation dargestellt.

Die Agenten nehmen bezogen auf eine Konversationsspezifikation (*TBCSpec*) die Rolle des Kunden (*CustomerRole(S)*) oder des Dienstleisters (*PerformerRole(S)*) ein. Dazu kann für jeden Agenten ein beliebiger Datentyp *S* und ein Wert *s* dieses Typs angegeben werden. Diese Datenstruktur kann während der Konversation zum Ablegen von konversationsspezifischen Informationen benutzt werden. Für jede Konversationsinstanz gibt es auch eine Instanz *s*, so daß das gleichzeitige Führen von mehreren Konversationen nach der gleichen Spezifikation möglich ist.

Für die Rollen können Regeln (*Customer Rule* bzw. *Performer Rule*) erzeugt werden. Die Regel eines Kunden bezieht sich auf einen Dialog, die des Dienstleisters auf die Kombination von Dialog und Anfrage (*Request*). Durch das Eintreffen eines Dialogs beim Kunden wird die entsprechende Regel ausgewertet. Beim Dienstleister wird die Regelauswertung durch einen Dialog und die zugehörige Anfrage ausgelöst. Die Regelausführung bezieht sich immer auf die Instanz einer Konversation (*Conversation(S)*), welche aus der Konversationsspezifikation instantiiert wurde, und gilt als atomare Aktion. Beide beteiligte Agenten erhalten eine eigene Instanz der Konversation, so daß die Autonomie der Agenten gewahrt bleibt. Dadurch erhält auch jeder Agent seine eigene Datenstruktur *s*.

Das Starten der Konversation wird immer vom Kunden vorgenommen. Details dazu sind in Kapitel 3.4 zu finden. Das dadurch erzeugte Konversationsobjekt (*Conversation(S)*) ist eine für Agenten erweiterte Repräsentation des *Tycoon Business Conversations*-Objekts (*TBConv(S)*), welches die von Akteuren unabhängige Instanz der Konversation darstellt. Durch diesen Aufbau ist die Realisierung der *Tycoon Business Conversations* nicht auf das Agentenmodell beschränkt.

Der aktuelle Dialog einer Konversation wird durch die *currentDialog*-Beziehung referenziert. Ein Dialog entspricht immer der zugehörigen Dialogspezifikation (*DialogSpec*) und enthält ein Inhaltsobjekt (*TBCContent*). Dieses Objekt setzt sich aus komplexen Objekten oder aus den atomaren Objekten für ganze (*Int*) und rationale (*Real*) Zahlen, Zeichenketten (*String*) oder Datumsangaben (*Date*) zusammen. Die komplexen Objekte Variante (*Variant*) und Record (*Rcd*) bestehen aus mindestens einem benannten Inhaltsobjekt (N*amedContent*), welches wiederum ein Inhaltsobjekt enthält. Die anderen komplexen Objekte sind Sequenz (*Sequence*) und Auswahllisten (*MultipleChoice* und *SingleChoice*). Diese Objekte können mehrere Inhaltsobjekte enthalten. Auf diese Weise werden geschachtelte Inhaltsobjekte erzeugt.

Die Dialoge repräsentieren einen Zustand der Konversation. Der Wechsel von einem Dialog zum nächsten stellt einen Zustandsübergang dar. Damit enthalten die Dialoge die typisierten Zustandsinformationen der Konversation. Dementsprechend gilt es, die Dialoge zu protokollieren. Die durch Kundenregeln erzeugten Anfragen müssen ebenfalls aufgezeichnet werden, da sie für die Zustandsübergänge mitverantwortlich sind.

Die Instanz einer Konversation wird mit *tbConv.new(...)* erzeugt. Dieser Funktion muß die Konversationsspezifikation übergeben werden. Optional ist der Typ *S* einer beliebigen Datenstruktur und ein Wert *s* dieses Typs. Verwendet wird diese Funktion hauptsächlich intern im Agenten zum Starten einer Konversation. Beide an der Konversation beteiligten Agenten haben eine eigene Instanz mit eventuell verschiedenen Typen *S*.

Das erzeugte Objekt bietet außer diesem Status *s* auch die folgenden Funktionen an:

- Die Funktion *spec()* liefert die zur Konversation gehörige Spezifikation.
- Mit *saveDialog(...)* wird der angegebene Dialog in der internen Historie der Konversationsinstanz gespeichert. In dieser Historie werden eintreffende Dialoge automatisch gespeichert, nicht jedoch die Dialoge und Anfragen, die abgesandt werden.
- Über *copyHistoryDialog(...)* wird der letzte unter diesem Namen in der Historie gespeicherte Dialog wieder ausgelesen. Wird ein Dialog in einer Konversation mehrfach verwendet, ist nur der letzte erreichbar. Nach dem Ende einer Konversation löschen die Agenten die Konversationsinstanz und damit die Historie. Dadurch bleibt die Größe der Agenten begrenzt.
- Durch *currentDialog()* wird der aktuelle Dialog ausgegeben.
- Die Funktion *newDialog(...)* erzeugt eine Instanz eines Dialogs aus dessen Spezifikation. Der Dialog wird über seinen Namen ausgewählt.

Das Dialogobjekt, das mit der oben aufgeführten Funktion *newDialog(...)* erzeugt wurde, stellt drei Operationen zur Verfügung:

- Mit *spec()* ist der Zugriff auf die Dialogspezifikation möglich.
- Der Inhalt des Dialogs wird von *content()* ausgegeben.
- Bei der Verwendung der verschiedenen Generierungsfunktionen für Inhaltsobjekte besteht der Inhalt des Dialogs aus dem schon erwähnten Typ *rcd*, der

als Behälter den eigentlichen Inhalt umfaßt. Mit *content1(...)* kann über den Namen auf einen Bestandteil des *rcd*-Typs direkt zugegriffen werden.

2.3 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 einer Dialogspezifikation haben verschiedene Namen. Alle Anfragespezifikationen einer Dialogspezifikation haben verschiedene Namen.

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

2. Jede Konversationsspezifikation hat eine finale Dialogspezifikation mit dem Namen *breakdown*. Diese Dialogspezifikation ist Bestandteil des realisierten Systems und muß daher nicht für jede Konversationsspezifikation separat erstellt werden.

   Durch diesen Dialog bleibt jede Konversation auch in Ausnahmefällen in einem gesicherten Zustand. Es kann keine unvorhergesehenen Konfigurationen geben.

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

   Damit kann aus der Abbruchsituation später der Grund für den Abbruch sowie der letzte erfolgreich bearbeitete Dialog rekonstruiert werden.

4. Finale Dialogspezifikationen haben per se keine Folgedialoge und daher auch keine Anfragespezifikationen. Wird ein solcher finaler Dialog in einer Konversation erreicht, so wird diese bei beiden Partnern beendet.

3. Kapitel

Analyse der agentenbasierten Realisierung des *Business Conversation*-Modells

In diesem Kapitel wird zunächst die Architektur des Tycoon-Agentensystems beschrieben und anschließend eine Analyse der für diese Arbeit relevanten Teile vorgenommen. Die Analyse betrifft insbesondere die Agenten selber und das Nachrichtensubsystem. Dabei wird die Kommunikation während einer Konversation und das Konzept des Mediators betrachtet. Die Begriffe des Tycoon-Agentensystems werden im Anhang B erläutert.

3.1 Die Architektur des Tycoon-Agentensystems

Das Tycoon-Agentensystem wurde von Nico Johannisson [Joh97] 1996-97 im Rahmen seiner Diplomarbeit konzipiert und prototypisch implementiert. Es benutzt für die Kooperation der Agenten das Konzept der *Business Conversations*. Diese Entwicklungsumgebung wurde für die Realisierung des Protokollierungsdienstes genutzt und soll deshalb zunächst analysiert werden.

Die Architektur des Tycoon-Agentensystems beruht auf dem Schichtenmodell, wie es in Abbildung 3.1 dargestellt ist. Von einer Unterscheidung in Kunde und Dienstleister wird hierbei abgesehen, da in der Regel auf jedem Rechnersystem sowohl Dienstanbieter wie -nutzer zu finden sind.

In der Ebene der Anwendungsschicht befinden sich unabhängige Komponenten, die durch das Agentensystem mit dem Ziel einer Kooperation integriert werden und so für den Nutzer einen Dienst erbringen können. Dabei muß von einer inhomogenen Anwendungsstruktur zwischen den verschiedenen Rechnern ausgegangen werden.

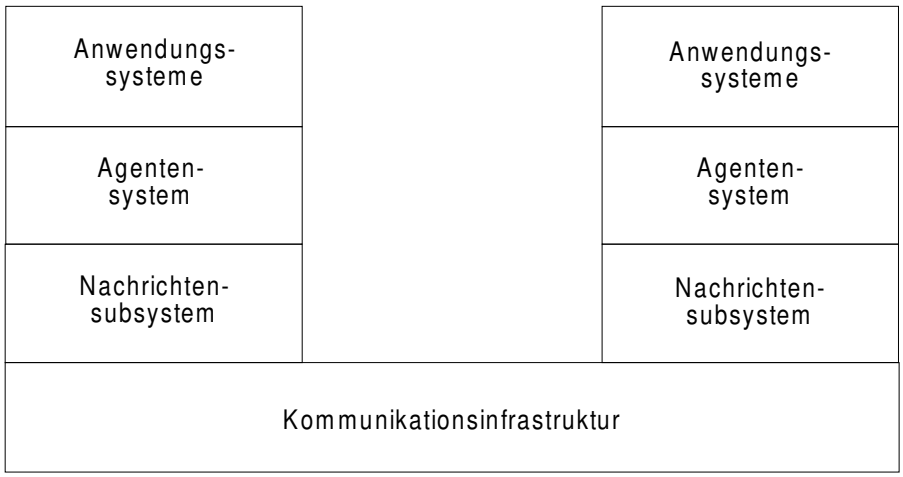

Abbildung 3.1: Architektur des Tycoon-Agentensystems

Die angesprochene Integration ist eine wichtige Leistung des Agentensystems. Eine andere Leistung besteht darin, die Aufenthaltsmöglichkeiten der Agenten gemäß Abbildung 3.2 zu gliedern. Dazu werden die Konzepte *Domäne* (*Domain*) und *Ort* (*Place*) benutzt. Eine Domäne ist einem Knoten des Rechnernetzes zugeordnet und besteht aus mindestens einem Ort, dem Wurzelort (*rootPlace*). In Kapitel 3.2 werden diese Konzepte genauer dargestellt.

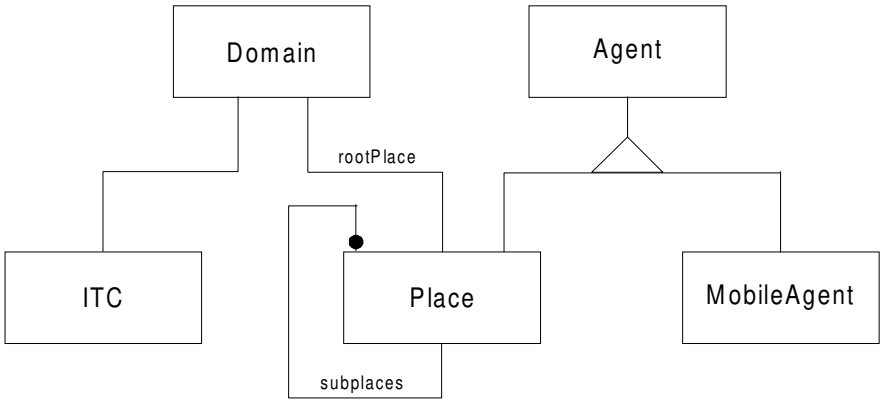

Abbildung 3.2: Gliederung des Agentensystems

Das Agentensystem unterstützt lokalen und domänenübergreifenden Ortswechsel durch *Migration*, indem es das erforderliche Protokoll abwickelt. Weiterhin stellt es

Funktionen zur Generierung von Agenten und Orten zur Verfügung. In Kapitel 3.3 wird die Erzeugung und Benutzung von Agenten genauer untersucht.

Das Nachrichtensubsystem (*ITC*[4]) bildet die Grundlage der Kommunikation innerhalb einer Domäne sowie der Kommunikation zwischen verschiedenen Domänen. Seine Aufgabe besteht darin, das Agentensystem von den Details der Netzwerkprogrammierung[5] zu befreien, eine ortsunabhängige Kommunikation zu ermöglichen und die Mobilität der Agenten zu unterstützen. Dazu wird ein asynchrones, nachrichtenorientiertes Verfahren benutzt. Weitere Ausführungen befinden sich in Kapitel 3.4. Ein Bestandteil des Nachrichtensubsystems, der Mediator, ist für diese Arbeit von besonderer Bedeutung. Auf dieses Konzept wird in Kapitel 3.5 näher eingegangen.

3.2 Domänen und Orte

Die Domäne gliedert die Aufenthaltsmöglichkeiten der Agenten durch eine Ortestruktur. Sie enthält immer einen Wurzelort. Da Orte wiederum Orte enthalten können, ergibt sich ein hierarchischer Aufbau, der zur Modellierung einer Unternehmensstruktur verwendet werden kann.

Durch das Starten des Agentensystems wird die Domäne erzeugt. Das Nachrichtensubsystem muß ebenfalls gestartet werden. Diese Ressource wird in jeder Domäne vorausgesetzt.

Ein Ort ist eine mögliche Spezialisierung eines Agenten, ein mobiler Agent eine andere. Der Unterschied zum mobilen Agenten besteht in dem stationärem Charakter der Orte. Für die mobilen Agenten stellen die Orte die Anlaufstationen dar, zu denen die mobilen Agenten migrieren können. Die Orte können aber auch den Zutritt verweigern. Da sie auch in der Lage sind, Konversationen zu führen, lassen sich auf diese Weise Systemdienste wie z.B. Broker einrichten.

Die Abbildung 3.3 zeigt den Zusammenhang von Domänen und Orten, denen wiederum die mobilen Agenten zugeordnet sind. Alle Agenten besitzen einen Kommunikationsport, der die Verbindung zum Nachrichtensubsystem herstellt. Alle Nachrichten, die ein Agent erhält, werden über diesen Port empfangen. Das Versenden einer Nachricht wird durch Aufruf der Funktion *itc.send(...)* erreicht. Näheres zum Nachrichtensubsystem ist in Kapitel 3.4 zu finden.

---

[4] Inter-Thread-Communication

[5] Im System werden *Sockets* verwendet.

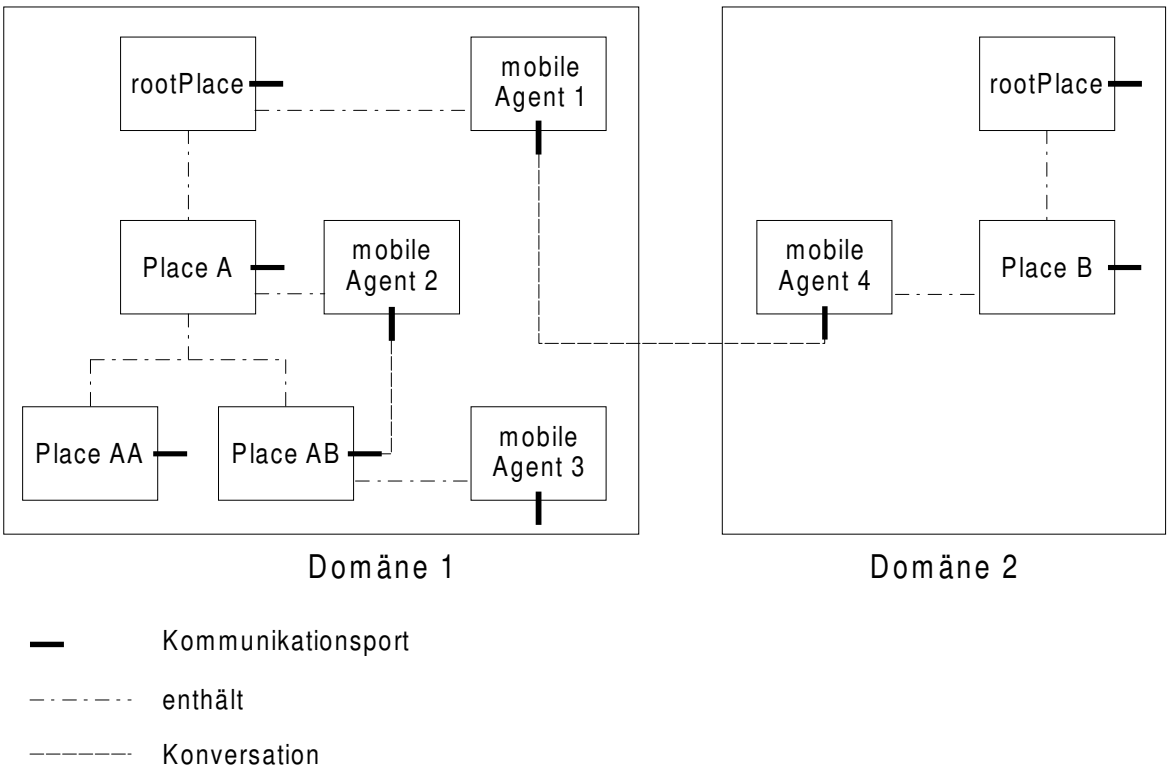

Abbildung 3.3: Orte und Agenten in den Domänen

Wenn im weiteren Verlauf der Arbeit Aussagen zu Agenten gemacht werden, betrifft dies sowohl mobile Agenten als auch Orte. Sollte eine Unterscheidung notwendig sein, wird dies entsprechend erwähnt.

3.3 Die Agenten

Die Agenten sind die aktiven Komponenten des Systems. Sie erfüllen ihre Aufgaben, indem sie Konversationen führen.

Johannisson benutzt für seine Agenten folgende Definition von Russel Beale und Andrew Wood [BW94]: „Ein Agent kann als eine relativ einfache, heterogene, autonome und kommunizierende Software-Komponente definiert werden; viele dieser Agenten existieren gleichzeitig und arbeiten zusammen, um eine Aufgabe für einen Benutzer zu erfüllen.“ Da sich die Agenten aufgrund eigener Entscheidung an andere Orte und auf andere Rechner bewegen können, zählen sie zur Klasse der mobilen Agenten. Eine Migration im Internet ist möglich, sofern auf dem Zielrechner Agenten- und Nachrichtensubsystem aktiv sind.

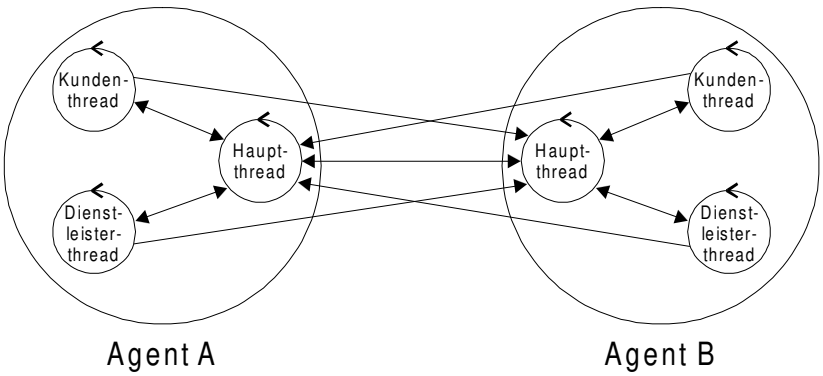

Abbildung 3.4: Die Kommunikation der Threads der Agenten

Agenten bestehen aus einem System- und einem Anwendungsteil. Der Systemteil stellt die Verbindung zwischen Anwendungsteil und Agentensystem dar. Er besteht aus drei Leichtgewichtsprozessen, den sog. Threads[6]. Der Hauptthread empfängt die Nachrichten, die an den Agenten geschickt werden, verarbeitet einige selbst und leitet die anderen an den entsprechenden Subthread weiter (siehe Abbildung 3.4). Es gibt zwei Subthreads, jeweils einen für die Rollen Dienstleister und Kunde. Während das Verhalten der Subthreads durch die Regeln der Konversation beeinflußt wird, kann auf den Hauptthread nicht eingewirkt werden. Der Systemteil ist gekapselt, so daß auf die internen Strukturen des Agenten nicht zugegriffen werden kann. Auch der Anwendungsteil hat keinen Zugriff auf die internen Strukturen. Um einen Agenten zu erzeugen wird lediglich der Anwendungsteil erstellt.

Zur Generierung eines mobilen Agenten stellt das Agentensystem die Funktion *agent.new(...)* zur Verfügung. Dieser Funktion muß außer einem Namen und dem Besitzer des neuen Agenten auch ein Wert *a* einer beliebigen Datenstruktur *A* und fünf Funktionen übergeben werden. Die Datenstruktur dient dem Agenten zur Speicherung von Informationen und steht ihm in allen Methoden und Konversationen zur Verfügung. Die zu definierenden Funktionen werden zu bestimmten Zeitpunkten aufgerufen und heißen *birth(...), meet(...), die(...), leave(...)* und *enter(...)*. Die Funktion *birth(...)* wird nach der Erzeugung des Systemteils ausgeführt, *meet(...)* wenn ein anderer Agent den Agenten treffen möchte, *die(...)* vor der Terminierung des Agenten, *leave(...)* und *enter(...)* vor dem Verlassen bzw. nach Erreichen eines Ortes. Diese Funktionen bestimmen das Verhalten des Agenten in den jeweiligen Situationen und bilden damit den Anwendungsteil des Agenten. Weitere Angaben sind zur Erstellung des Agenten nicht möglich.

---

[6] Hierbei handelt es sich nicht um Betriebssystemthreads, sondern um Tycoonthreads.

Zur Erstellung eines Ortes dient die Funktion *agent.newPlace(...)*. Der einzige Unterschied zur Generierung eines mobilen Agenten besteht darin, daß die Funktionen *leave(...)* und *enter(...)* durch *incomingAgent(...)* und *outgoingAgent(...)* ersetzt sind. Diese Funktionen werden beim Eintreffen oder der Erzeugung eines Agenten am Ort bzw. beim Verlassen des Ortes durch einen Agenten aufgerufen.

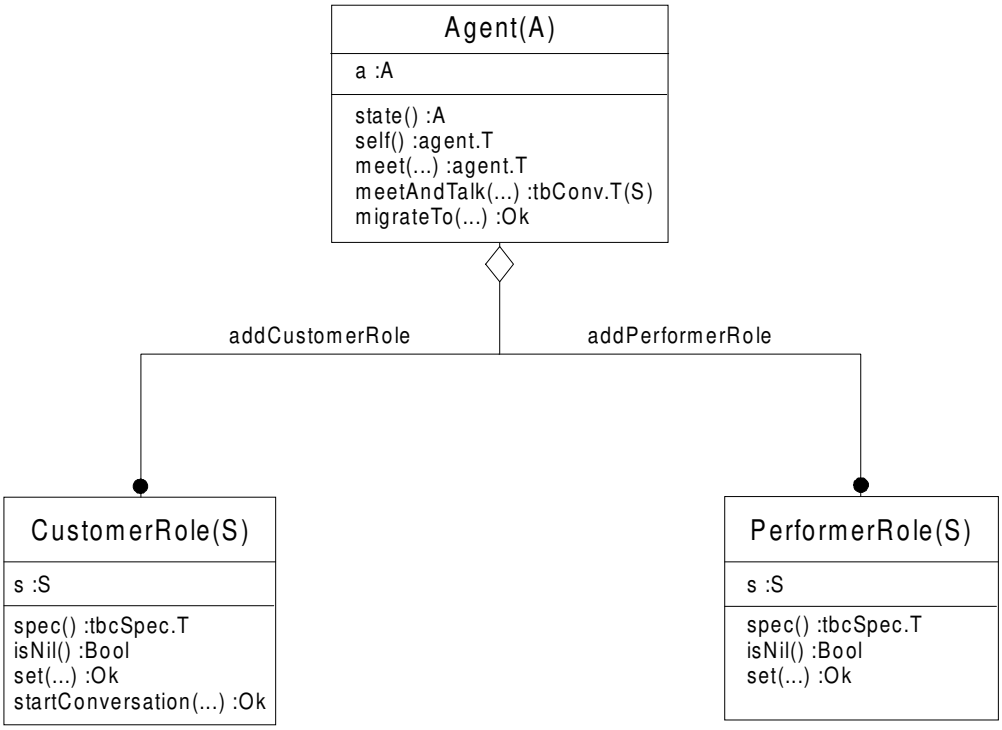

Abbildung 3.5: Objektstrukturmodell des Agent und seiner Rollen

Bei der Definition der fünf Funktionen können folgende sieben Methoden des Agenten verwendet werden:

- Mit der Methode *state()* kann der bei der Erzeugung des Agent angegebene Wert *a* abgefragt und verändert werden.

- Die Methode *self()* liefert einen Identifikator des Agenten, welcher u.a. dem Konversationspartner übergeben wird. Dieser Identifikator ist vom Typ *Agent.T* und beinhaltet lediglich den Namen des Agenten, seine eindeutige Bezeichnung, den Namen seines Besitzers und den Ort, wo er erzeugt wurde. Durch den Identifikator wird keine Bindung zwischen Agenten hergestellt, so daß die Autonomie der Agenten beim Austausch der Identifikatoren gewahrt bleibt.

- Durch die Methode *meet(...)* kann der Agent einen Identifikator eines anderen Agenten erhalten. Dazu muß dessen Name angegeben werden und die Agenten sich am gleichen Ort aufhalten[7]. Im anderen Agenten wird dabei die bei der Erzeugung angegebene *meet(...)*-Funktion ausgeführt. So kann der Agent selber entscheiden, ob er dem Treffen zustimmt.

- Die Methode *meetAndTalk(...)* ist eine Spezialisierung der *meet(...)*-Methode. Es wird nach dem Empfang des Identifikators eine synchrone Konversation begonnen und dabei die Rolle des Kunden eingenommen.

- Durch die Methode *migrateTo(...)* sind Agenten in der Lage, den Ort und die Domäne zu wechseln. Dazu ist die Zustimmung aller betroffenen Orte notwendig. Verweigert einer den Zutritt, schlägt die Migration fehl.

- Mit den Methoden *addPerformerRole(...)* und *addCustomerRole(...)* wird für eine bestimmte Konversationsspezifikation die Rolle des Dienstleisters bzw. Kunden eingenommen. Diesen Methoden kann der Typ *S* einer beliebigen Datenstruktur übergeben werden. Dieser Datentyp *S* wird beim Erzeugen einer Instanz der entsprechenden Konversation durch die schon beschriebene Funktion *tbConv.new(...)* als Parameter verwendet. Das Ergebnis dieser Methoden ist vom Typ *PerformerRole(S)* bzw. *CustomerRole(S)*.

Über die folgenden Methoden von *PerformerRole(S)* bzw. *CustomerRole(S)* wird das Verhalten des Agenten während der Konversation gesteuert:

- Die Methode *spec()* liefert die zur Konversation gehörende Spezifikation.

- Über die Methode *isNil()* läßt sich feststellen, ob für diese Rolle lediglich ein Platzhalter erzeugt wurde.

- Mit der Methode *set(...)* kann bei *PerformerRole(S)* für eine Anfrage eine Regel definiert werden. Es müssen dazu in der Methode *set(...)* die Namen des Dialogs und der Anfrage angegeben sowie eine Funktion definiert werden. Diese Funktion wird ausgeführt, wenn der Kunde in dem zugehörigen Dialog diese Anfrage ausgewählt hat. Das Ergebnis muß ein Dialog sein, da es an den Kunden gesandt wird. *CustomerRole(S)* besitzt diese Methode ebenfalls, allerdings wird hier eine Regel für einen Dialog definiert, und das Ergebnis der zu definierenden Funktion muß dementsprechend eine Anfrage sein.

---

[7] Dies ist eine Beschränkung in der Implementation, nicht im Konzept.

- Nur für *CustomerRole(S)* gibt es die Methode *startConversation(...)*, mit der eine Konversation mit dem anzugebenden Dienstleister begonnen wird. Diese Konversation findet asynchron statt, im Gegensatz zu der synchronen Konversation, die mit der Agenten-Methode *meetAndTalk(...)* gestartet wird.

Innerhalb der Regeln sind beliebige Aktionen möglich. So kann z.B. eine sekundäre Konversation geführt oder eine Migration abgewickelt werden. Letztendlich findet durch die Festlegung der Regeln die eigentliche Programmierung des Agenten statt.

3.4 Das Nachrichtensubsystem

Für die Kommunikation im Tycoon-Agentensystem wird durch das Nachrichtensubsystem ein asynchrones, nachrichtenorientiertes Verfahren realisiert. So findet die Kommunikation sowohl zwischen Agenten als auch zwischen den verschiedenen Threads eines Agenten in Form von Nachrichten statt. Die bei Funktionsaufrufen entstehende synchrone Verarbeitung und die daraus resultierende Blockierung des aufrufenden Threads wird damit verhindert und die notwendige Nebenläufigkeit erreicht. Außerdem entfällt die für Funktionsaufrufe erforderliche Bindung der Agenten, die eine Einschränkung der Autonomie darstellen würde.

Im Tycoon-Agentensystem werden ausschließlich typisierte Nachrichten verwendet. Die Nachrichten sind als variantes Tupel im Interface *ITCMessage.ti* definiert, wobei jede Variante einen Typ bestimmt. Es besteht jederzeit die Möglichkeit, neue Varianten hinzuzufügen. Das Nachrichtensubsystem ist im Modul *itc.tm* realisiert.

Nachfolgend werden die im Rahmen einer Konversation ausgetauschten Nachrichten (siehe Abbildung 3.8) daraufhin untersucht, ob sie für eine Protokollierung von Bedeutung sind.

- *customerStartAsync* beginnt eine asynchrone Konversation und wird durch die Methode *startConversation(...)* verschickt. Diese Methode kann sowohl vom Kunden- als auch vom Dienstleisterthread des Agenten aufgerufen werden. Der Empfänger der Nachricht ist der Hauptthread des gleichen Agenten. Der Agent schickt sich diese Nachricht also selbst und nimmt damit die Rolle des Kunden in der neuen Konversation ein. Die Nachricht enthält den Namen der Konversationsspezifikation, den Identifikator des Dienstleisters und einen Wert *s* der *startConversation(...)* als Parameter übergeben und dessen Typ *S* bei *addCustomerRole(S)* (siehe Kapitel 3.3) definiert wurde.

- *customerStartSync* ist die Alternative, um eine Konversation zu beginnen, und wird durch die Methode *meetAndTalk(...)* (siehe Kapitel 3.3) gesendet. Der

Agent nimmt in der neuen Konversation die Rolle des Kunden ein. Da dies eine synchrone Konversation ist, wird der Thread, der *meetAndTalk(...)* aufgerufen hat, bis zum Ende der Konversation blockiert. Deshalb darf diese Methode nur vom Dienstleisterthread verwendet werden. Der Empfänger der Nachricht ist wiederum der Hauptthread des gleichen Agenten. Die Nachricht enthält zusätzlich zu den Daten von *customerStartAsync* die Adresse eines Ports. An diesen Port muß nach dem Ende der Konversation die Nachricht *endOfConversation* geschickt werden. Der Empfänger dieser Nachricht ist der Thread, der *customerStartSync* verschickt hat.

- Der Hauptthread des Kunden sendet nach Empfangen von *customerStartAsync* bzw. *customerStartSync* die Nachricht *requestForConversationA* an den Konversationspartner, der damit zwingend die Rolle des Dienstleisters einnimmt. Die Nachricht besteht aus der Konversationsspezifikation, der internen Konversationsnummer des Kunden und dessen Identifikator.

- Der Dienstleister erklärt sein Einverständnis zur Konversation mit *conversationAcceptedA*. Diese Nachricht enthält die internen Konversationsnummern des Kunden und des Dienstleisters sowie den Namen der Konversationsspezifikation. Die interne Konversationsnummer wird vom jeweiligen Agenten nur einmal vergeben. Im Agentensystem kann sie jedoch mehrfach auftreten.

- Der Hauptthread des Kunden generiert nach dem Empfang von *conversationAcceptedA* den vom *Business Conversation*-Modell vorgegebenen Initialdialog und die zugehörige initiale Anfrage. Diese Daten schickt er zusammen mit der Konversationsnummer des Dienstleisters mittels der Nachricht *customerDialogRequestA* an den Dienstleister.

- Der Dienstleister erzeugt den ersten Dialog der Konversation gemäß der Spezifikation und verschickt ihn mit *performerDialogA*. In der Nachricht sind die Konversationsnummer des Kunden und der Dialog enthalten.

- In der anschließend stattfindenden Konversation stellt der Kunde seine Anfragen mit *customerDialogRequestA*, während der Dienstleister die Dialoge mit *performerDialogA* versendet. Diese Nachrichten werden durch das Ende der jeweiligen Regelauswertung verschickt.

- Die Konversation endet mit dem finalen Dialog. Wurde sie mit *customerStartSync* begonnen, schickt der Kundenthread jetzt an den Hauptthread die Nachricht *endOfConversation*. Diese Nachricht wird vom Hauptthread an den in *customerStartSync* angegebenen Port weitergeleitet und enthält die Instanz der Konversation.

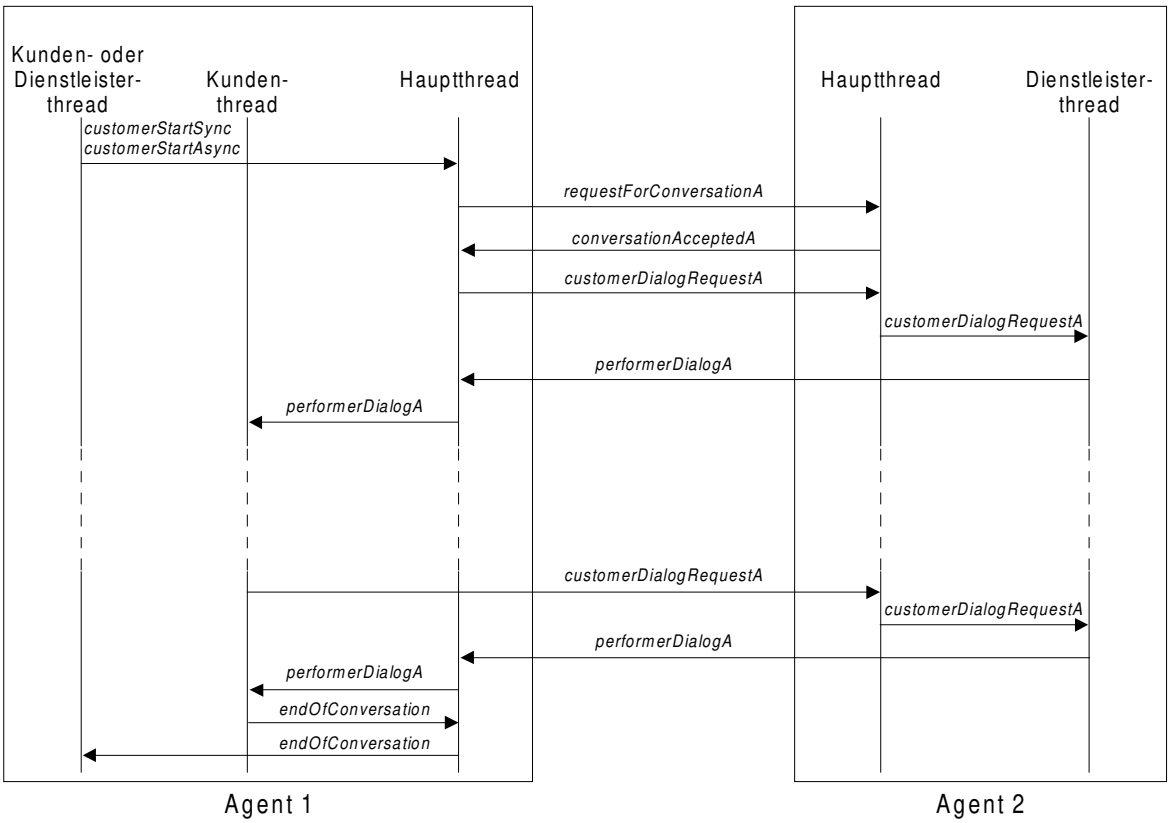

Abbildung 3.6: Nachrichtenaustausch während einer Konversation

Für die Protokollierung relevant sind die Nachrichten *performerDialogA* und *customerDialogRequestA*. Sie enthalten die Dialoge und Anfragen der Konversation. Es fällt auf, daß diese Nachrichten zusätzlich nur die Konversationsnummern enthalten, welche nicht eindeutig sind. Somit ist eine Zuordnung der Nachrichten zu den Agenten allein mit diesen Daten nicht möglich.

3.5 Der Mediator

Der Mediator ist ein optionaler Systemdienst des Nachrichtensubsystems. Es gibt pro Domäne maximal einen Mediator. Er erhält alle Nachrichten[8], bevor sie an einen lokalen Empfänger oder eine andere Domäne weitergeleitet werden. Befinden sich

[8] Ausgenommen sind nur Nachrichten, die vom Mediator oder einem Mediator-Dienst stammen, und Nachrichten an nichtöffentliche Kommunikationsports. Dieses Verfahren wird z.B. im Agenten zur Kommunikation der verschiedenen Threads benutzt.

Sender und Empfänger nicht in der gleichen Domäne, wird die Nachricht gegebenenfalls von zwei Mediatoren bearbeitet. Der Mediator hat die Möglichkeit, die Nachrichten zu kopieren, zu verändern, zu unterdrücken, umzuleiten oder unverändert zu lassen. Weder Sender noch Empfänger werden von der Bearbeitung informiert. Da mit diesem Konzept eine Protokollierung von Nachrichten ermöglicht wird, soll der realisierte Mediator ausführlich betrachtet werden.

Der implementierte Mediator beschränkt sich auf die Verwaltung von priorisierten Mediator-Diensten. Die Bearbeitung von Nachrichten findet durch diese Mediator-Dienste statt. Aufgrund dieser Realisierung besteht die Möglichkeit zur Laufzeit weitere Dienste, wie z.B. Monitoring- oder Sicherheitskomponenten, in das System zu integrieren. Jeder Mediator-Dienst erhält die zu bearbeitende Nachricht maximal einmal.

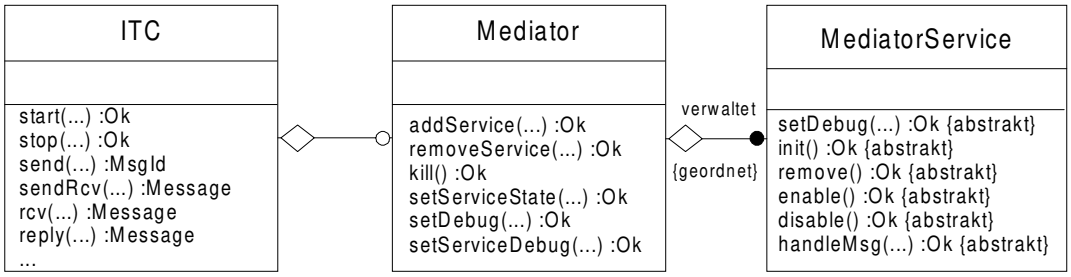

Abbildung 3.7: Objektstrukturmodell von Mediator und Mediator-Dienst

Die Abbildung 3.7 stellt den Zusammenhang zwischen dem Nachrichtensubsystem (*ITC*), dem Mediator und den Mediator-Diensten (*MediatorService*) dar.

Der Mediator wird mit *mediator.new()* gestartet. Über die folgenden Methoden kann der Mediator beeinflußt werden:

- *addService(...)* meldet einen Mediator-Dienst beim Mediator an. Dabei muß der Mediator-Dienst als Parameter übergeben werden.
- *removeService(...)* meldet einen Mediator-Dienst beim Mediator ab. Er erhält keine Nachrichten mehr vom Mediator und muß gegebenenfalls mit *addService(...)* neu angemeldet werden.
- *kill()* beendet den Mediator.

- *setServiceState(...)* deaktiviert oder reaktiviert einen Mediator-Dienst. Ist ein Mediator-Dienst deaktiviert, erhält er keine Nachrichten vom Mediator, kann aber mit *setServiceState(...)* wieder reaktiviert werden.

- *setDebug(...)* schaltet Debug-Meldungen des Mediators an oder aus.

- *setServiceDebug(...)* schaltet Debug-Meldungen für einen Mediator-Dienst an oder aus.

Ein Mediator-Dienst besteht aus Funktionen, die vom Mediator durch lokale Prozeduraufrufe in verschiedenen Situationen ausgeführt werden. Um einen Mediator-Dienst zu erstellen, ist es notwendig, die im folgenden aufgeführten Funktionen zu definieren:

- *setDebug(...)* schaltet Debug-Meldungen an oder aus. Diese Operation wird durch die Mediator-Methode *setServiceDebug(...)* aufgerufen.

- *init()* wird beim Anmelden des Mediator-Dienstes durch a*ddService(...)* ausgeführt. So kann eine Initialisierung des Mediator-Dienstes vorgenommen werden.

- *remove()* wird von *removeService(...)* und *kill()* aufgerufen. Hier können Befehle zum Beenden des Mediator-Dienstes angegeben werden.

- *enable()* wird durchgeführt, wenn der Mediator-Dienst durch *setServiceState(...)* reaktiviert wurde.

- *disable()* wird bei der Deaktivierung des Mediator-Dienstes durch *setServiceState(...)* aufgerufen.

- *handleMsg(...)* legt die eigentliche Funktionalität des Mediator-Dienstes fest. Diese Operation wird vom Mediator aufgerufen, um eine Nachricht vom Mediator-Dienst bearbeiten zu lassen. Dazu übergibt der Mediator außer der Originalnachricht auch die drei folgenden Funktionen, die bei der Definition von *handleMsg(...)* verwendet werden können. Die Funktion *setMessageBody(...)* ermöglicht dem Mediator-Dienst, die Nachricht zu verändern, indem der Rumpf der Nachricht neu festgelegt wird. Mit *send(...)* hat er die Möglichkeit, eine Nachricht zu verschicken. Im Unterschied zu dem normalen *itc.send(...)* erhält der lokale Mediator diese Nachricht nicht. Mit *delegateMessage()* bestimmt der Mediator-Dienst, daß die Nachricht an den nächsten Mediator-Dienst oder, falls es keinen weiteren Dienst gibt, an den Empfänger weitergeleitet werden soll. Ruft der Mediator-Dienst diese Funktion nicht auf, wird die Nachricht verworfen. Nur wenn alle Dienste *delegateMessage()* aufgerufen haben, erhält der Empfänger die Nachricht.

Folgende Beschreibung (siehe dazu Abbildung 3.8) verdeutlicht die einzelnen Stationen, die eine Nachricht durchläuft:

In der Domäne *A* hat sich der Mediator-Dienst *A* beim Mediator angemeldet. Der Agent *A* schickt eine Nachricht *m* an den Agenten *B* in der Domäne *B*. Das ITC *A* der Domäne *A* leitet die Nachricht zunächst an den Mediator *A* der Domäne. Der wiederum ruft den registrierten Mediator-Dienst auf. Der Mediator-Dienst gibt die Nachricht nach seiner Bearbeitung an den Mediator *A* zurück (*m'*). Da kein weiterer Mediator-Dienst angemeldet ist, geht die Nachricht an das ITC *A* weiter, welches sie an das ITC *B* der Domäne *B* schickt. Anschließend wird die Nachricht an den Mediator *B* übergeben. Da in dieser Domäne kein Mediator-Dienst angemeldet ist, wird die Nachricht an das ITC *B* zurückgegeben und vom ITC *B* dem Agenten *B* zugestellt.

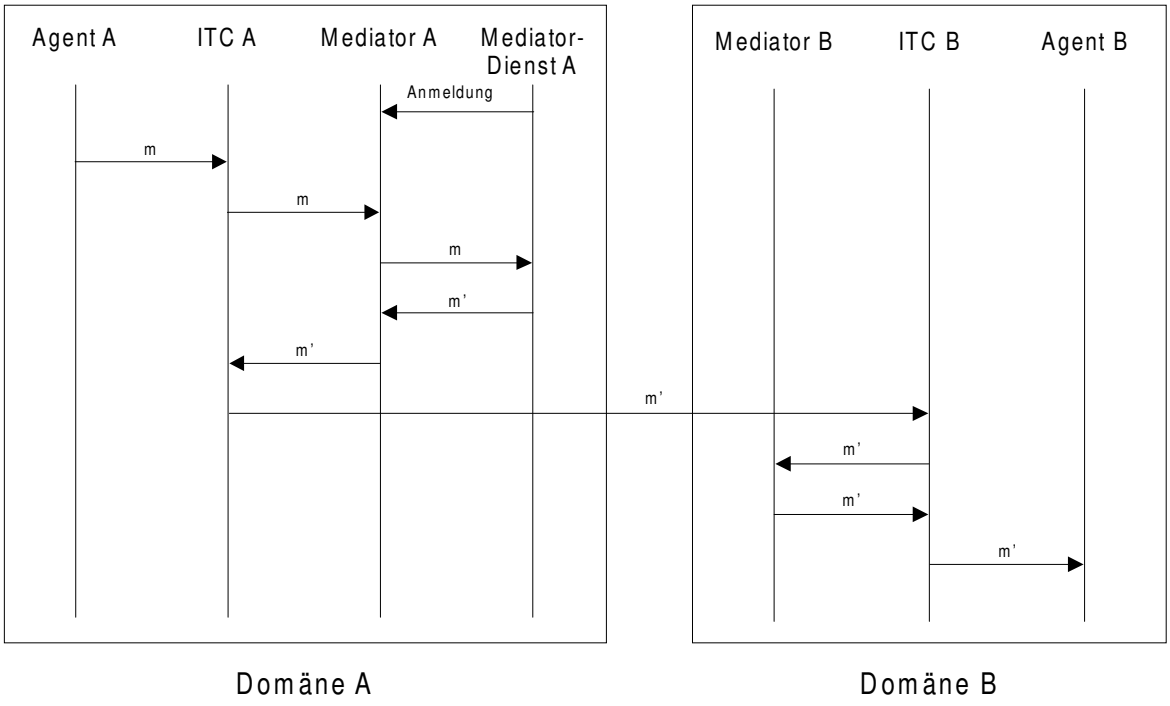

Abbildung 3.8: Nachrichtenfluß im Agentensystem

3.6 Ein Beispiel für die Benutzung der Agenten

In diesem Abschnitt wird anhand eines Beispiels die Verwendung des Agentensystems, der *Tycoon Business Conversations* und der Agenten dargestellt.

Als erstes werden die benötigten Module importiert (1a) und das Agentensystem gestartet (1b). Beim Starten des Agentensystems wird der Name der Domäne, im Beispiel „Demo“, angegeben. Der zweite Parameter dient zur Erzeugung eines eindeutigen Identifikators. Hier kann aber auch ein festgelegter Identifikator angegeben werden. Der Befehl startet gleichzeitig das Nachrichtensubsystem.

1a) Module importieren

```
import agent wwuid tbcSpec tbcContent iter;
```

1b) Agenten- und Nachrichtensubsystem starten

```
agent.startDomain(“Demo“  wwuid.new());
```

2) Konversationsspezifikation definieren

In diesem Schritt wird eine Konversationsspezifikation (siehe Kapitel 2.2.1) definiert. Diese wird anschließend von den Agenten verwendet. Die Spezifikation besteht aus drei Dialogspezifikationen. Der Dialog „Berechnung“ dient zur Addition von zwei Integerzahlen und enthält zwei Integerfelder und die Anfrage „addieren“. Der Dienstleister liefert auf dieses Anfrage hin den Dialog „Ergebnis“, der aus einem Integerfeld, welches die Summe der im Dialog „Berechnung“ eingegebenen Zahlen enthält, und den Anfragen „weiter“ und „ende“ besteht. Durch die Anfrage „weiter“ wird ein neuer Dialog „Berechnung“ angefordert, so daß mehrere Additionen möglich sind. Die Anfrage „ende“ liefert den finalen Dialog „Ende“, der weder einen Inhalt noch eine Anfrage enthält. Die Dialoge und Anfragen der Konversation werden in Abbildung 3.9 dargestellt. Die Definition der Konversationsspezifikation findet im Anschluß an die Dialogdefinitionen statt. Die Spezifikation erhält den Namen „Addierer“, der initiale Dialog der Konversation ist der Dialog „Berechnung“.

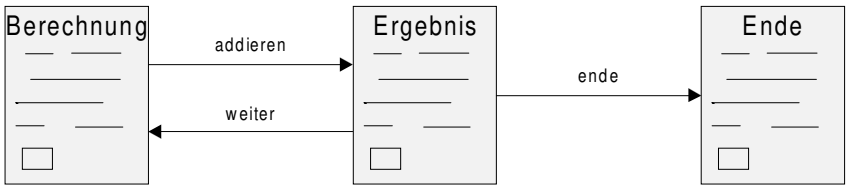

Abbildung 3.9: Dialoge und Anfragen der Beispielkonversation

```
let additionDialogSpec =
  tbcSpec.dialogSpec("Berechnung"  array
    tbcSpec.requestSpec("addieren" array  "Ergebnis" end false)
  end
  array
    tuple "1. Summand"  fun() tbcContent.intSpec() end
    tuple "2. Summand "  fun() tbcContent.intSpec() end
  end)

let ergebnisDialogSpec =
  tbcSpec.dialogSpec("Ergebnis"  array
    tbcSpec.requestSpec("weiter"  array  "Berechnung"  end false)
    tbcSpec.requestSpec("ende"  array  "Ende"  end false)
  end
  array
    tuple "Summe"  fun() tbcContent.intSpec() end
  end)

let endeDialogSpec =
  tbcSpec.dialogSpec("Ende"  array end  array  end)

let additionConvSpec =
  tbcSpec.new("Addierer" additionDialogSpec array
    additionDialogSpec
    ergebnisDialogSpec
    endeDialogSpec
  end)
```

Die Definitionen des Dienstleister- und des Kundenagenten finden im Schritt 3 statt. Nur die jeweiligen *birth(...)*-Funktionen (siehe Kapitel 3.3) sind ausführlich programmiert, da sie für das Verhalten der Agenten in der Konversation entscheidend sind.

3a) Dienstleister erstellen

Der erste Agent nimmt mit *m.addPerformerRole(...)* die Rolle des Dienstleisters für die eben definierte Konversationsspezifikation ein. Anschließend sind vier Regeln für die Konversation angegeben. Die erste reagiert auf die initiale Anfrage eines Kunden und erzeugt den Dialog „Berechnung“. Die zweite wird durch diesen Dialog und die Anfrage „addieren“ ausgelöst. Es findet zunächst die Addition der erhaltenen Zahlen statt. Das Ergebnis wird dann im Dialog „Ergebnis“ an den Kunden übertragen. Die dritte Regel reagiert auf den Dialog „Ergebnis“ und die Anfrage „weiter“. Hier wird nur ein neuer Dialog „Berechnung“ erzeugt. Wird vom Kunden im Dialog

„Ergebnis“ die Anfrage „ende“ ausgewählt, liefert die vierte Regel den finalen Dialog „Ende“. Für den Dienstleister ist die Konversation durch das Ausführen dieser Regel beendet. Die letzten beiden Angaben in diesem Teil legen den Besitzer des Agenten, in diesem Fall der Ort, an dem der Agent erzeugt wird, und seinen Namen fest.

```
let performer :agent.T = agent.new(:Ok  fun(self :agent.T) ok
   let birth(m :agent.M(Ok)) :Ok = begin
    let p = m.addPerformerRole(:Ok  additionConvSpec  ok)

    p.set(tbConv.initialDialogName()  tbConv.initialRequest()
          fun(cv :agent.Conversation(Ok)) :tbConv.Dialog
      cv.c.newDialog("Berechnung"))

    p.set("Berechnung" "addieren"
          fun(cv :agent.Conversation(Ok)) :tbConv.Dialog
     begin
      let d = cv.c.currentDialog()
      let a = d.content1("1. Summand ")!int.get()
      let b = d.content1("2. Summand ")!int.get()
      let result = a+b
      let d = cv.c.newDialog("Ergebnis")
      d.content1("Summe")!int.set(result)
      d
     end)

    p.set("Ergebnis" "weiter"  fun(cv :agent.Conversation(Ok)) :tbConv.Dialog
      cv.c.newDialog("Berechnung"))

    p.set("Ergebnis" "ende"  fun(cv :agent.Conversation(Ok)) :tbConv.Dialog
      cv.c.newDialog("Ende"))
   end)

   let meet(m :agent.M(Ok) other :agent.T spec :tbcSpec.T) :agent.T = m.self()
   let die(m :agent.M(Ok)) :Ok  = ok
   let leave(m :agent.M(Ok)  fromPlace :agent.T) :Ok = ok
   let enter(m :agent.M(Ok)  atPlace :agent.T) :Ok = ok
   agent.currentRootPlace()  "Dienstleister")
```

3b) Kunde erstellen

Bevor der zweite Agent erstellt wird, findet die Definition der Variablen *numbers* und ihres Datentyps statt. Diese Variable enthält die vom Agenten zu addierenden Zahlen. Im zweiten Agenten wird durch *m.addCustomerRole(...)* die Rolle des Kunden definiert. Hier werden drei Regeln festgelegt. Bei diesen Regeln wird auf den Datentyp *Numbers* Bezug genommen, der als Datentyp der Statusvariablen der Konversation angegeben wird (siehe Kapitel 3.3). Die erste Regel reagiert auf den Dialog „Berechnung“ und füllt die beiden Felder mit Zahlen aus der Statusvariablen der Konversation. Anschließend wird die Anfrage „addieren“ angegeben. Die zweite Regel liest aus dem Dialog „Ergebnis“ die Summe aus, und schreibt sie in die Statusvariable. Wenn noch nicht alle Zahlen addiert sind, wird die Anfrage „weiter“, ansonsten die Anfrage „ende“ ausgewählt. Die dritte Regel bestätigt lediglich den finalen Dialog. Nach der Regeldefinition wird der Befehl *c.startConversation(...)* zum Starten der Konversation gegeben. Dabei wird die Variable *numbers* angegeben, die damit Statusvariable wird.

```
Let Numbers = iter.T(Int)
let var numbers :Numbers = iter.cons(23  iter.cons(14  iter.singleton(28)));

let customer :agent.T = agent.new(:Ok fun(self :agent.T) ok
 let birth(m :agent.M(Ok)) :Ok = begin
  let c = m.addCustomerRole(:Numbers additionConvSpec)

  c.set("Berechnung"  fun(cv :agent.Conversation(Numbers)) :String
   begin
    let i = iter.head(cv.c.s)
    let j = iter.head(iter.rest(cv.c.s))
    if iter.size(cv.c.s) > 2 then
     cv.c.s := iter.rest(iter.rest(cv.c.s))
    else
     cv.c.s := iter.new(:Int)
    end
    let d = cv.c.currentDialog()
    d.content1("1. Summand ")!int.set(i)
    d.content1("2. Summand ")!int.set(j)
    "berechnen"
   end)
```

```
  c.set("Ergebnis"  fun(cv :agent.Conversation(Numbers)) :String
   begin
    let d = cv.c.currentDialog()
    let result = d.content1("Summe")!int.get()
    cv.c.s := iter.cons(result  cv.c.s)
    if iter.size(cv.c.s) > 1 then
     "weiter"
    else
     "ende"
    end
   end)

  c.set("Ende"  fun(cv :agent.Conversation(Numbers)) :String
   "")

  if iter.size(numbers) >1 then
   c.startConversation(performer  numbers)
  end
 end)

let meet(m :agent.M(Ok)  other :agent.T  spec :tbcSpec.T) :agent.T = m.self()
let die(m :agent.M(Ok)) :Ok = ok
let leave(m :agent.M(Ok)  fromPlace :agent.T) :Ok = ok
let enter(m :agent.M(Ok)  atPlace :agent.T) :Ok = ok
agent.currentRootPlace()  "Kunde")
```

4. Kapitel

Anforderungsanalyse

In diesem Kapitel werden zunächst verschiedene Anforderungen an eine Protokollierung von *Business Conversations* formuliert. Daran anschließend werden verschiedene Alternativen zum Aufbau des Protokollierungsdienst vorgestellt. Den Abschluß bilden Überlegungen zum Datenvolumen und zur benötigten Funktionalität des Dienstes.

4.1 Anforderungen an die Protokollierung

Im folgenden werden Anforderungen festgelegt, die bei der Konzeption eines Protokollierungsdienstes zu berücksichtigen sind.

- Ein Protokollierungsdienst soll die während einer Kommunikation ausgetauschten Daten aufzeichnen. Da im Tycoon-Agentensystem Konversationen als strukturiertes Kommunikationsverfahren eingesetzt werden, müssen Dialoge und Anfragen protokolliert werden. Die vor Beginn der Konversation verschickten Nachrichten dienen lediglich dem Verbindungsaufbau, während nach dem Ende der Konversation keine weitere Kommunikation stattfindet.

- Zusätzlich zu den eigentlichen Daten der Konversation müssen auch Meta-Informationen protokolliert werden. Diese dienen z.B. zur Feststellung, welche Agenten diese Konversation geführt haben. Welche Informationen gespeichert werden, ist abhängig von den Auswertungsmöglichkeiten, die der Protokollierungsdienst anbieten soll.

- Es soll möglichst wenig zusätzlicher Aufwand für die Agenten durch die Protokollierung der Konversation entstehen. Zusätzliche Aufgaben für die Agenten können die Konversation beeinträchtigen. Dies ist nicht wünschenswert.

- Die Protokollierung muß für Agenten einfach zu starten sein. Es darf kein größerer Programmieraufwand notwendig werden. Deshalb sollte eine Konversation zum Starten der Protokollierung angeboten werden.

- Damit die Agenten ohne Programmieraufwand auf ihre Protokolle zugreifen, sie bearbeiten und auswerten können, sollte es möglich sein, diese Aktionen mittels einer Konversation durchzuführen.

- Um Engpässe zu vermeiden, darf es im Agentensystem keine Beschränkung auf einen einzigen Protokollierungsdienst geben. Dadurch wird die Möglichkeit von konkurrierenden Protokollierungsdiensten, wie z.B. Notariatsdiensten, geschaffen.

- Es sollte die Möglichkeit bestehen, die Protokolle auf beliebigen, frei wählbaren Rechnern mit Protokollierungsdienst zu speichern. Auf diese Weise können die Agenten ihre Daten trotz Migration zentral verwalten.

- Um das Datenvolumen zu begrenzen, muß es möglich sein, für jede Konversation festzulegen, ob sie protokolliert werden soll. Auch eine Beschränkung auf bestimmte Teile einer Konversation ist zu berücksichtigen.

- Der Zugriff auf die Protokolle muß aus Gründen des Datenschutzes sowie sonstiger gesetzlicher Vorschriften und zum Schutz vor Industriespionage gesichert und eingeschränkt werden können.

4.2 Alternativen zur Einbindung des Protokollierungsdienstes

In diesem Abschnitt werden konzeptuelle Überlegungen zum Aufbau eines Protokollierungsdienstes vorgestellt, die sich aus der Architektur des Tycoon-Agentensystems ergeben.

Zunächst sollen die Alternativen dargestellt werden, die ein Protokollierungsdienst hat, um die zu protokollierenden Daten einer Konversation zu erhalten.

Für die Weiterleitung der Daten durch den Konversationspartner, der die Protokollierung wünscht, gibt es zwei Ansätze. Ein dritter Ansatz beruht auf dem Konzept des Mediators.

- Automatische Weiterleitung: Durch eine neu zu entwickelnde Methode des Agenten kann eine automatische Weiterleitung der Daten zum Protokollierungsdienst erfolgen. Der Entwickler des Anwendungsteils des Agenten muß diese Methode beim Start der Konversation aufrufen und angeben, wo das

Protokoll geführt werden soll. Um diese Methode zu erstellen, ist allerdings eine Änderung der Funktion *agent.new(...)*, die einen neuen Agenten erzeugt, notwendig. Es müssen alle ein- und ausgehenden Nachrichten des Agenten im Hinblick darauf untersucht werden, ob sie zu einer zu protokollierenden Konversation gehören. Weiterhin müssen neue Nachrichten hinzugefügt werden, mit denen der Protokollierungsdienst über den Beginn und das Ende einer Konversation informiert wird. Dies führt zu einer zusätzlichen Funktionalität in allen Agenten, auch wenn nur einige diese Funktionalität in Anspruch nehmen sollten.

- Regelbasierte Weiterleitung: Eine Alternative stellt ein Weiterleiten durch die Regel eines Konversationsschrittes dar. Dazu muß der Entwickler des Anwendungsteils des Agenten eine Funktion schreiben, die die Weiterleitung realisiert und die in jede Regel, aus der protokolliert werden soll, einzutragen ist. Da in einer Regel Nachrichten verschickt werden können, ist dieses Verfahren zwar möglich, bedeutet aber einen verhältnismäßig großen Aufwand, zumal dieser Aufwand für jeden Agenten mit Protokollierungswunsch wiederholt werden muß. Ein weiterer Nachteil sind die entstehenden Manipulationsmöglichkeiten, da die Daten vor der Weiterleitung verändert werden können. Ein so erzeugtes Protokoll kann nur verifiziert werden, wenn beide Konversationspartner eine Weiterleitung zu einem gemeinsamen Protokollierungsdienst vornehmen. Treten Unterschiede in den Daten auf, ist nicht zu klären, ob und welche Daten korrekt sind.

- Wie bereits in Kapitel 3.5 erläutert, können Nachrichten mittels eines Mediator-Dienstes weiterverarbeitet und umgeleitet werden. Ein Mediator-Dienst für die Protokollierung kann die Nachrichten *customerDialogRequestA* und *performerDialogA* zu einer Protokollstelle weiterleiten. Damit der Mediator-Dienst mit verschiedenen Protokollstellen zusammenarbeiten kann, muß er wissen, wohin welche Dialoge und Anfragen zu leiten sind. Diese Information kann von einem der Konversationspartner z.B. durch eine Nachricht übermittelt werden. Auf diese Weise braucht nur einer der Partner zu Beginn der Konversation aktiv zu werden; während der Konversation findet keine weitere Aktion der Konversationspartner die Protokollierung betreffend statt. Auch eine Manipulation ist nicht mehr möglich, so daß ein Notariatsdienst aufgebaut werden kann.

Da das dritte Konzept den oben geschilderten Anforderungen am besten gerecht wird, wird es bei den weiteren Überlegungen vorausgesetzt.

4.3 Alternativen zum Aufbau des Protokollierungsdienstes

In diesem Abschnitt werden drei verschiedene Konzepte für den Aufbau des Protokollierungsdienstes entworfen.

1. Eine Möglichkeit besteht darin, den Mediator-Dienst mit einer Anwendung, dem Archiv, zu koppeln. Der Mediator-Dienst übernimmt die Weiterleitung der Daten an das Archiv, welches die Speicherung und Verwaltung realisiert. Da der Mediator-Dienst lediglich eine Funktion ist, die vom Mediator aufgerufen wird, kann er keine Daten speichern. Dies bedeutet auch, daß der Mediator-Dienst die Weiterleitung immer nur an festgelegte Archive vornehmen kann. Um neue Archive zu berücksichtigen, muß der Mediator-Dienst immer erst beendet, geändert und neu gestartet werden. Verhindern läßt sich dies, indem entweder für jedes Archiv ein eigener Mediator-Dienst gestartet wird oder die Weiterleitung zunächst an eine Verteileranwendung vorgenommen wird, die eine Liste der Archive führt. Diese Liste kann mittels Nachrichten jederzeit aktualisiert werden. Die Verteileranwendung übernimmt die Weiterleitung der Daten zum Archiv.

   Ein Problem stellt die Kommunikation des Archivs mit Agenten dar. Das Archiv benötigt eine umfangreiche Konversationsunterstützung, um mit Agenten Konversationen zu führen. Anderenfalls müssen die Agenten über Nachrichten mit dem Archiv kommunizieren. Die Agenten können zwar innerhalb von Regeln Nachrichten verschicken und Antworten empfangen, aber jeder Agent, der eine Protokollierung wünscht oder eine Auswertung erreichen möchte, muß die entsprechenden Nachrichten innerhalb des Anwendungsteils verschicken. Der Programmierer kann also nicht auf dem Abstraktionsniveau der Konversationen arbeiten.

2. Eine Alternative stellt ein Agent dar, der mit dem Mediator-Dienst zusammenarbeitet und die nötige Funktionalität bietet. Dieser Agent speichert die Protokolle und kann über Konversationen anderen Agenten Zugriff auf die Daten ermöglichen. Ein Problem besteht jedoch in der Kommunikation mit dem Mediator-Dienst. Der Mediator-Dienst leitet die eintreffenden Dialoge und Anfragen an diesen Agenten weiter. Dies geschieht zu Zeitpunkten, die für den Agenten nicht vorhersehbar sind. Da der Mediator-Dienst nicht in der Lage ist, eine Konversation zu führen, findet diese Kommunikation über Nachrichten statt. Dazu muß der Agent neu zu definierende Nachrichten verstehen und verwenden. Im Systemteil der Agenten sind die Nachrichten festgelegt, die von den Agenten verstanden werden. Um weitere hinzuzufügen, muß die Funktion *agent.new(...)*, die einen Agenten erzeugt, entsprechend geändert werden. Diese

Änderung betrifft somit alle Agenten, obwohl nur Protokoll-Agenten diese Nachrichten verwenden.

3. Eine weitere Möglichkeit ergibt sich aus der Kombination der ersten beiden Alternativen. Der Mediator-Dienst leitet die Dialoge und Anfragen über eine Verteileranwendung an ein Archiv weiter. Das Archiv kann sich auf einem beliebigen Rechner befinden. Die Kommunikation zwischen Mediator-Dienst, Verteileranwendung und Archiv erfolgt über Nachrichten. Damit Agenten möglichst einfach auf das Archiv zugreifen können, wird das Archiv mit einem speziellen Agenten, dem Archivar, gekoppelt. Über diesen Archivar können Agenten mittels einer Konversation Zugriff auf das Archiv erhalten.

   Der Archivar hat zwei Möglichkeiten, das Archiv zu einer Aktion zu veranlassen; zum einen über Funktionsaufrufe, zum anderen über Nachrichten. Der Nachteil bei Funktionsaufrufen besteht in den Synchronisationsschwierigkeiten im Archiv, das von der Verteileranwendung Nachrichten erhält. Aus diesem Grund ist die Kommunikation zwischen Archivar und Archiv über Nachrichten sinnvoll. Eine Übersetzung der Konversation zwischen Agent und Archivar in Nachrichten an das Archiv kann erreicht werden, indem in den Bearbeitungsregeln der Konversation des Archivars Nachrichten an das Archiv eingebaut werden. Wichtig dabei ist, daß das Archiv nur auf Nachrichten des Archivars reagiert, es verschickt nicht unaufgefordert Nachrichten an ihn. Deshalb ist eine Erweiterung des Systemteils der Agenten um zusätzliche Nachrichten nicht nötig.

Die Abbildung 4.1 soll das dritte Konzept verdeutlichen. Die Agenten 1 und 2 führen die zu protokollierende Konversation *a*. Die Nachrichten dieser Konversation laufen über das Nachrichtensubsystem und damit auch über Mediator und Mediator-Dienst. Der Mediator-Dienst leitet die Dialoge und Anfragen an das Archiv weiter. Agent 1 führt zusätzlich die Konversation *b* mit Agent 3, dem Archivar. Diese Konversation könnte wie jede andere Konversation auch protokolliert werden. Der Archivar übersetzt die Schritte der Konversation in die Nachrichten *c* an das Archiv. Durch dieses Konzept kann jeder Agent über eine einfache Konversation auf seine Protokolle zugreifen, ohne mit Nachrichten arbeiten zu müssen.

Ein Vorteil dieses Konzeptes besteht auch darin, daß mehrere Archivare mit einem Archiv zusammenarbeiten können, und daß ein Archivar auch Zugriff auf verschiedene Archive bieten kann. Da dieses Konzept die Ansprüche an die Protokollierung am besten erfüllt, wird es bei den weiteren Überlegungen zugrunde gelegt.

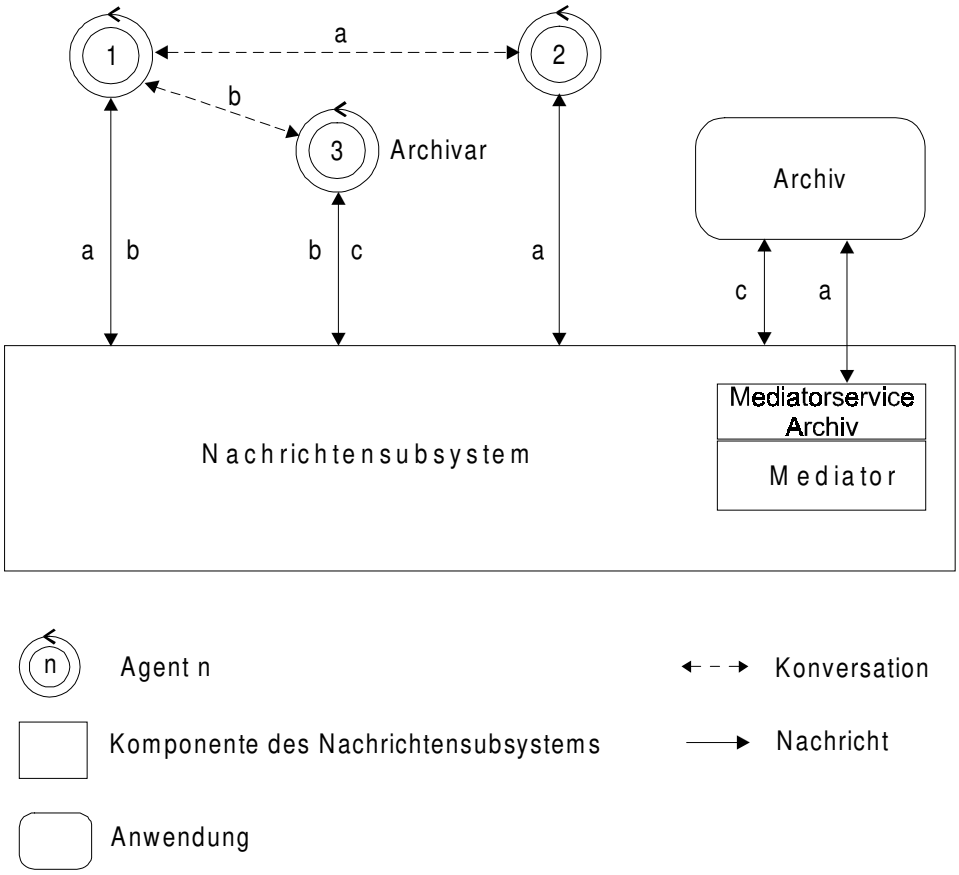

Abbildung 4.1: Systemdarstellung des Konzeptes

4.4 Begrenzung des Datenvolumens

Um das Datenvolumen zu begrenzen, kann beiden Konversationspartnern Zugriff auf das Protokoll eingeräumt werden. Damit läßt sich eine doppelte Protokollierung verhindern. Die Konversationspartner müßten sich dabei allerdings einigen, wer das Protokoll startet und wo es gespeichert werden soll. Es ist davon auszugehen, daß ein Agent die Speicherung seiner Protokolle in einem einzigen Archiv favorisiert. Eine Einigung, welches Archiv benutzt werden soll, wird deshalb wohl auf interne Netzwerke beschränkt sein. Ist eine Einigung nicht möglich, muß jeder Agent sein eigenes Protokoll erzeugen. Diese Möglichkeit muß also immer gegeben sein, ein gemeinsames Protokoll ist somit ein Spezialfall.

Als nächstes wird erörtert, welche Elemente einer Konversation zu protokollieren sind, und ob es Möglichkeiten gibt, das Datenvolumen zu begrenzen.

Wie in Kapitel 2.2.2 bereits geschildert, besteht das Inhaltsobjekt des Dialogs aus strukturierten Objekten und ihren Werten. Durch die enthaltene Strukturinformation werden zusammengesetzte Objekte wie z.B. Listen konstruiert. Diese Information muß deshalb beim Speichern erhalten bleiben.

Die Anfragen enthalten neben der eigentlichen Anfrage auch einen Dialog. Dies ist der Dialog, den der Dienstleister im letzten Schritt der Konversation geschickt hat und dessen Werte vom Kunden bearbeitet wurden. Daraus ergibt sich die Möglichkeit, statt des kompletten Dialogs nur geänderte Werte zu speichern. Dieses Vorgehen erfordert aber beim Speichern einen Vergleich der Dialoge. Außerdem muß bei späteren Zugriffen auf die Anfragen der entsprechende Dialog jedesmal wiederhergestellt werden. Da ein Dialog nur ca. 12 KB[9] groß ist, erscheint dieser Aufwand nicht gerechtfertigt.

Für eine Auswertung des Protokolls sind neben den Dialogen und Anfragen auch andere Informationen wichtig, wie z.B. über den Konversationspartner, die Konversationsspezifikation, den Beginn der Protokollierung und wann die Dialoge bzw. Anfragen aufgezeichnet wurden.

Informationen über die Verbindung, wie z.B. die Domänen und Orte, sind dagegen aufgrund der Mobilität der Agenten von untergeordneter Bedeutung, haben auf die Konversation keinen Einfluß und müssen deshalb nicht protokolliert werden. Je nachdem, welche Auswertungsmöglichkeiten angeboten werden sollen, sind diese und weitere Informationen jedoch sinnvoll.

4.5 Funktionalität des Protokollierungsdienstes

In diesem Teil werden die Basisfunktionen geschildert, die den Agenten für die Nutzung des Protokollierungsdienstes zur Verfügung stehen sollen. Grundsätzlich gilt, daß die Agenten ausschließlich mit dem Archivar im Rahmen einer Konversation in Verbindung treten. Eine direkte Kommunikation mit dem Mediator-Dienst oder dem Archiv über Nachrichten ist nicht sinnvoll.

---

[9] Diese Angabe bezieht sich auf den mit Initialwerten gefüllten Dialog „Neues Protokoll“ der Archivar-Konversation (siehe Kapitel 6.1.1). Dieser Dialog kann eine Konversationsspezifikation aufnehmen und wird dann mit ca. 30 KB für Dialoge verhältnismäßig groß.

Die wichtigste Aktion ist das Starten einer Protokollierung. Diese Aktion muß in der Konversation einfach und schnell durchzuführen sein, damit die zu protokollierende Konversation möglichst wenig beeinträchtigt wird. Der Archivar muß dafür sorgen, daß das Archiv und der Mediator-Dienst informiert werden.

Eine weitere Aktion ist das Stoppen der Protokollierung, damit unwichtige Teile einer Konversation nicht aufgezeichnet werden müssen. Daraus ergibt sich die Notwendigkeit, eine gestoppte Protokollierung fortführen zu können.

Das Löschen eines Protokolls muß ebenfalls möglich sein. Aufgrund gesetzlicher Vorschriften kann es notwendig werden, hier Einschränkungen zu machen. Ein Beispiel für eine solche Einschränkung wäre, daß das Löschen erst nach einer festgelegten Frist erfolgen darf.

Weiterhin sollte eine Übersicht über die Protokolle existieren sowie eine Liste der Dialoge und Anfragen, die zu einer Konversation gespeichert wurden. Auch die Möglichkeit zum Abrufen eines Dialogs bzw. einer Anfrage ist sinnvoll.

5. Kapitel

Design

Nachdem die konzeptuellen Überlegungen abgeschlossen sind, soll nun das resultierende Design des Protokollierungsdienstes vorgestellt werden.

5.1 Der Aufbau des Protokollierungsdienstes

Der Protokollierungsdienst benötigt die drei folgenden Komponenten:

- Einen erweiterten Mediator-Dienst: Der Mediator-Dienst Archiv (MD Archiv) besteht aus zwei Komponenten, dem eigentlichen Mediator-Dienst (*MDS Transmitter*[10]) und einer zugehörigen Anwendung (*Distributor*), die sich in der gleichen Domäne befindet. Der *MDS Transmitter* leitet alle Nachrichten, die Dialoge oder Anfragen enthalten, an den *Distributor* weiter. Der *Distributor* überprüft anhand gespeicherter Daten über zu protokollierende Konversationen, ob die Dialoge und Anfragen, die er vom *MDS Transmitter* erhält, zu einer solchen Konversation gehören und leitet sie gegebenenfalls an das zuständige Archiv weiter. Auf diese Weise wird eine zweifache Selektion der Nachrichten vorgenommen. Das Nachrichtensubsystem wird entlastet, da der *Distributor* nur relevante Daten gezielt an ein bestimmtes Archiv sendet. Ohne *Distributor* müßte der *MDS Transmitter* alle Dialoge und Anfragen an alle bekannten Archive schicken.

- Eine Anwendung: Das Archiv (*Archive*) enthält die für die Protokollierung notwendigen Datenstrukturen. Seine Funktionalität ist darauf beschränkt, auf Befehle vom *Distributor* oder vom Archivar zu reagieren.

[10] MDS steht für Mediator-Service

- Einen Agenten, den Archivar (*Archivist*): Dies ist ein Agent, der das Archiv verwaltet. Er stellt anderen Agenten eine Konversation zur Verfügung, die diesen den Zugriff auf das Archiv erleichtert.

Die drei oben beschriebenen Komponenten sind zwar für den Protokollierungsdienst notwendig, dies bedeutet aber nicht, daß die Komponenten zusammengehören. Sie sind vielmehr unabhängig und autonom.

So wie der MD Archiv Dialoge und Anfragen an verschiedene Archive verschickt, erhält auch ein Archiv nicht nur von einem MD Archiv Daten. Das Archiv wird von unterschiedlichen Archivaren abgefragt, so wie die Archivare auch verschiedene Archive verwalten können.

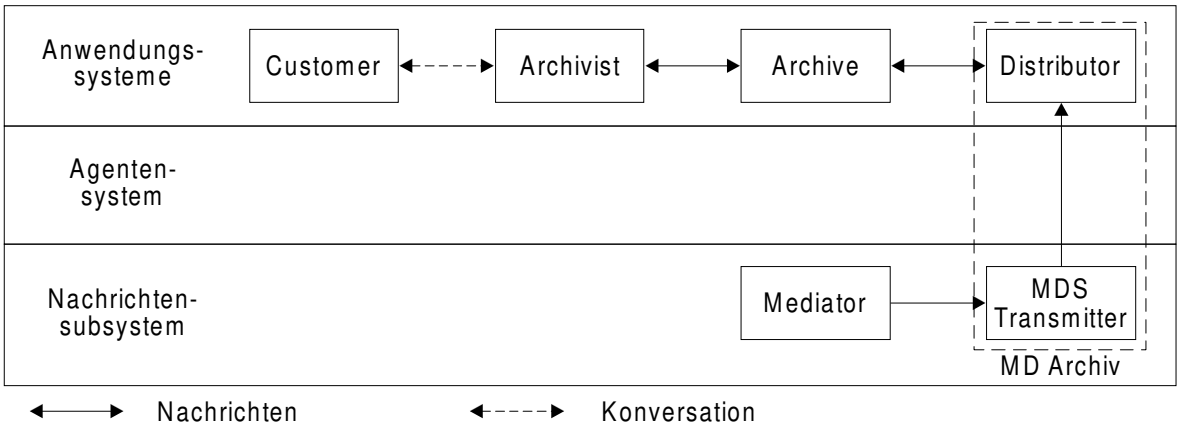

Abbildung 5.1: Einordnung des Protokollierungsdienstes in die Schichten-Architektur

Die Abbildung 5.1 ordnet die verschiedenen Komponenten den Schichten des Tycoon-Agentensystems zu. Der *Customer* ist ein Agent, der mit dem *Archivist* eine Konversation führt. Der Mediator ruft den *MDS Transmitter* auf, der dem *Distributor* Nachrichten schickt. *Archivist*, *Archive* und *Distributor* tauschen ebenfalls Nachrichten aus.

Die Abbildung 5.2 zeigt den Aufbau des Protokollierungsdienstes als Objektstrukturmodell. Der Aufbau wird in den nachfolgenden Abschnitten näher erläutert.

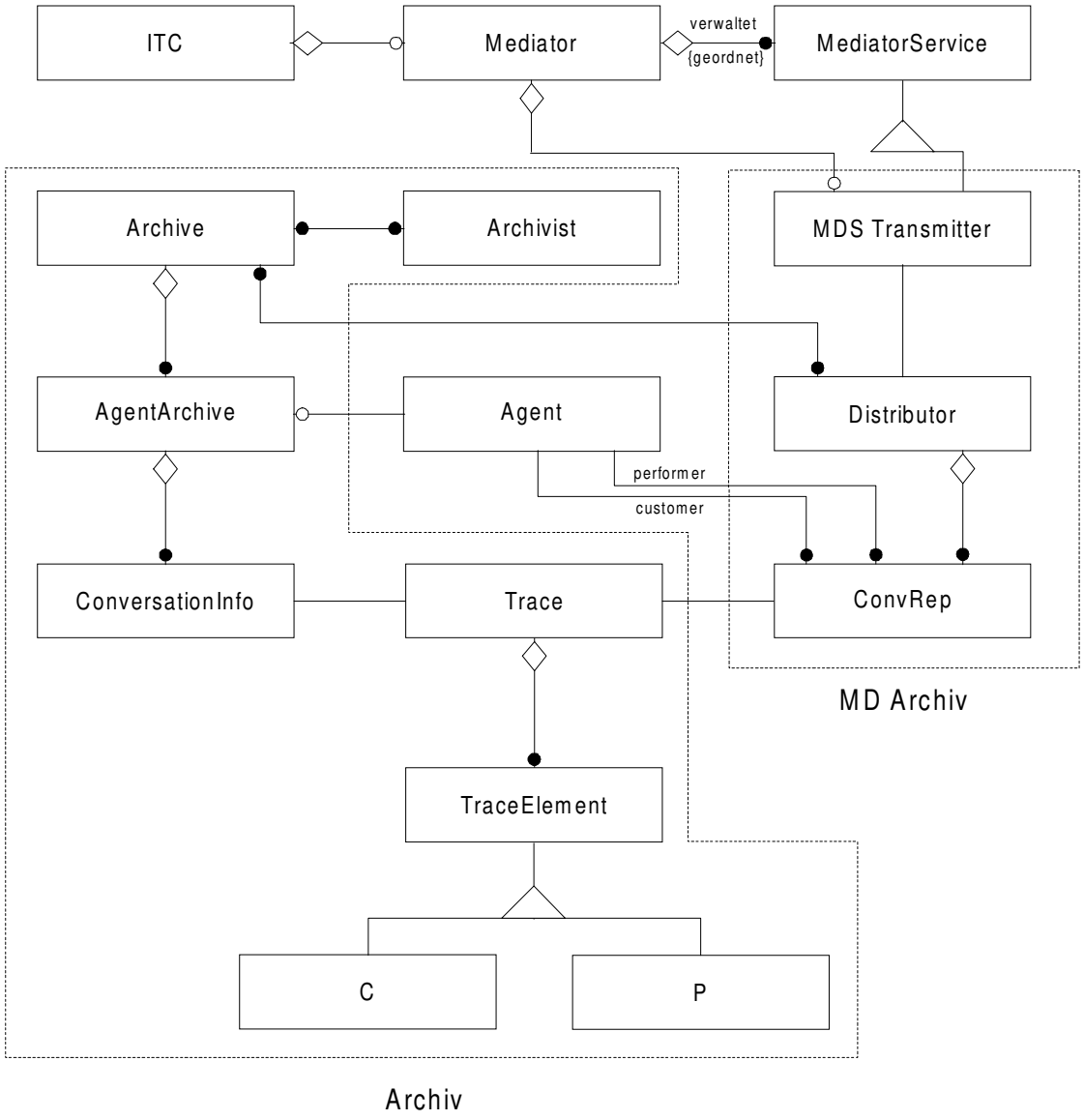

Abbildung 5.2: Objektstrukturmodell des Protokollierungsdienstes

5.2 Der Mediator-Dienst Archiv

Der MD Archiv muß zumindest in einer der Domänen gestartet sein, in denen sich die Konversationspartner befinden, deren Konversation protokolliert werden soll. Ist dies nicht der Fall, ist eine Protokollierung nicht möglich. Der MD Archiv ist so konzipiert, daß er mittels Nachrichten an beliebige Archive eine Weiterleitung vornehmen kann. Dazu ist es notwendig, daß er die Adresse des für die jeweilige

Konversation zuständigen Archivs speichert. Es werden bisher nur Dialoge und Anfragen an das Archiv verschickt. Diese Weiterleitung könnte aber gegebenenfalls auf andere Nachrichten einer Konversation, z.B. zur Initialisierung, erweitert werden. Im MD Archiv findet keine Speicherung oder Bearbeitung der Dialoge und Abfragen statt. Abbildung 5.3 zeigt das Objektstrukturmodell des MD Archiv.

In den Nachrichten *customerDialogRequestA* und *performerDialogA* sind neben Dialog und Anfrage nur die Konversationsnummern der beteiligten Agenten enthalten. Da diese Nummern im Agentensystem nicht eindeutig sind, besteht die Notwendigkeit, die beiden Nachrichten um die eindeutigen Agenten-IDs zu erweitern. Nur so kann der MD Archiv eine eindeutige Zuordnung der Nachrichten zu einer protokollierten Konversation vornehmen.

Der Mediator ruft beim Eintreffen einer Nachricht den *MDS Transmitter* auf. Handelt es sich um die Nachrichten *customerDialogRequestA* oder *performerDialogA*, sendet der *MDS Transmitter* die Dialoge und Anfragen mit den Nachrichten *mdsArchiveDialog* bzw. *mdsArchiveRequest* an den *Distributor*. Anschließend werden die Nachrichten *customerDialogRequestA* und *performerDialogA* an den Mediator zurückgegeben. Bei anderen Typen von Nachrichten geschieht dies sofort.

Der *Distributor* verwaltet eine Liste von *ConvRep*-Objekten[11]. Das *ConvRep* enthält die für den *Distributor* notwendigen Informationen einer zu protokollierenden Konversation. Dies sind die weltweit einmaligen Agenten-IDs des Kunden (*customerId*) und des Dienstleisters (*performerId*), deren Konversationsnummern (*customerConvId* und *performerConvId*), die Adresse des Archivs (*archivePlace*) und die ebenfalls einmalige ID des Protokolls (*id*).

Da sowohl die Agenten-IDs als auch die Konversationsnummern verwendet werden, können Dialoge und Anfragen von verschiedenen Konversationen der gleichen Partner auseinandergehalten werden. Damit ist auch eine Protokollierung von mehreren, gleichzeitig geführten Konversationen der Partner möglich.

Mit der Nachricht *startTrace* wird ein neues *ConvRep*-Objekt erzeugt, während es mit *terminateTrace* gelöscht wird. Erhält der *Distributor* die Nachrichten *mdsArchiveDialog* oder *mdsArchiveRequest*, wird mit Hilfe der Agenten-IDs und der Konversationsnummern festgestellt, ob der Dialog bzw. die Anfrage weitergeleitet werden muß. Ist dies der Fall, werden der Dialog oder die Anfrage unter Angabe der Protokoll-ID an die Archiv-Adresse gesandt.

---

[11] ConvRep steht für Conversation Representation

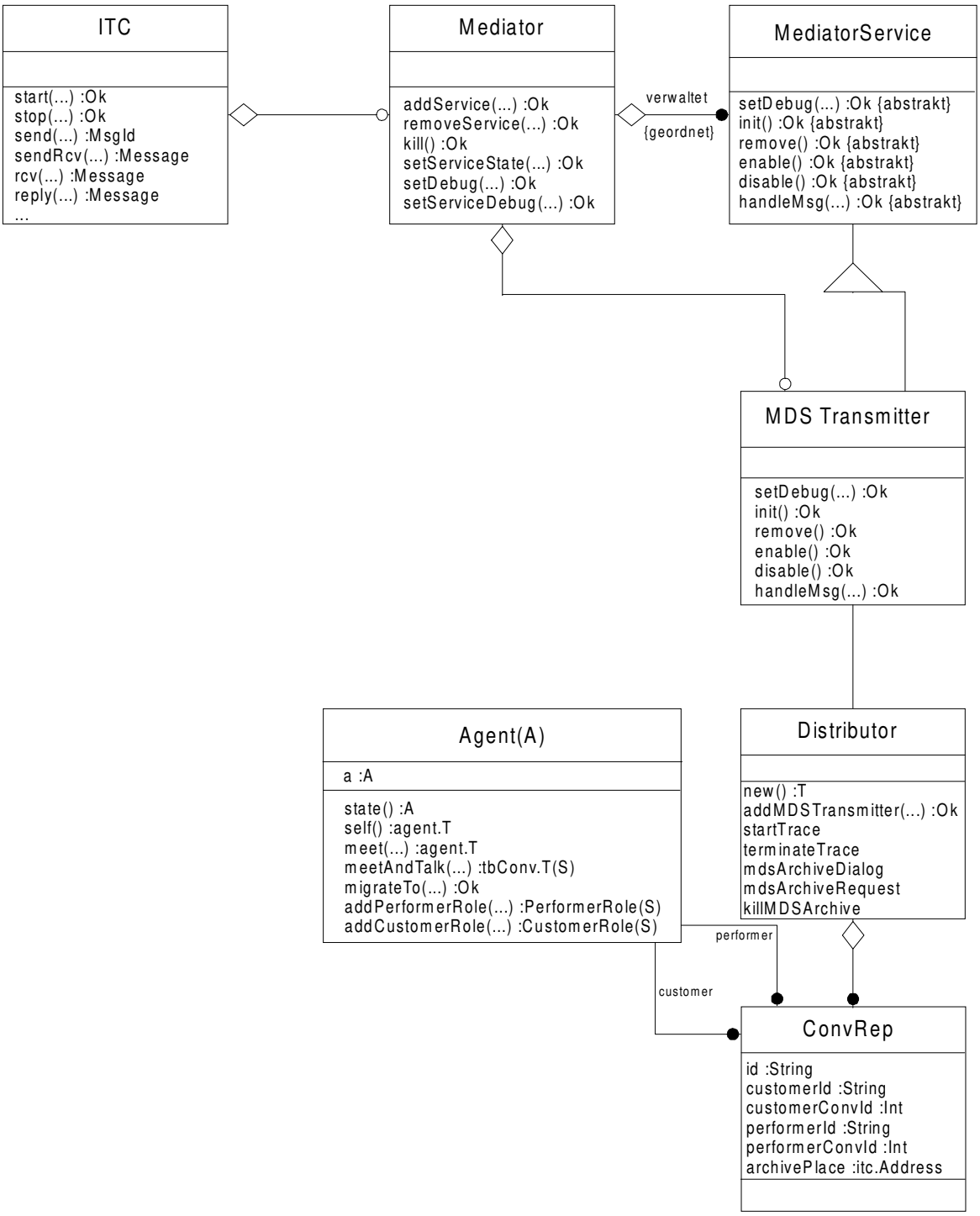

Abbildung 5.3: Objektstrukturmodell des Mediator-Dienstes Archiv

5.3 Das Archiv

Das Archiv ist als eine Anwendung konzipiert, die nur reagiert. Dies geschieht bei Nachrichten, die dem Archiv geschickt werden. Die Aufgabe des Archivs besteht in der Speicherung und Auswertung der Protokolle. Die Protokolle werden dabei dem Agenten zugeordnet, der die Protokollierung veranlaßt hat und sind durch ein Paßwort geschützt. Die Struktur des Archivs ist in Abbildung 5.4 dargestellt.

Das Archiv verwaltet Objekte vom Typ *AgentArchive*. Ein *AgentArchive* ist genau einem Agenten zugeordnet, enthält dessen Agenten-ID (*agentId*) und das Paßwort (*password*). Es verwaltet eine Liste von *ConversationInfo*-Objekten. Jeder protokollierten Konversation ist genau eine *ConversationInfo* zugeordnet. Die Attribute von *ConversationInfo* sind im folgenden erläutert:

Die Protokoll-ID (*id*) ist wie die Agenten-ID einmalig und wird vom Archiv vergeben. Die Konversationsbezeichnung (*traceName*) kann vom Agenten zur Benennung der aufgezeichneten Konversation verwendet werden. Das Attribut *convSpec* enthält die Spezifikation der Konversation. Der Name des Konversationspartners (*otherAgentName*) wurde zusätzlich zu dessen Agenten-ID (*otherAgent*) aufgenommen, um eine für menschliche Benutzer verständliche Kennzeichnung zu erreichen. Die Konversationsnummern werden in den Felder *otherAgentConvId* und *selfConvId* gespeichert. Die Rolle (*role*) des Agenten, also Kunde oder Dienstleister, wird ebenfalls festgehalten. Im Status (*status*) wird registriert, ob die Aufzeichnung noch läuft, angehalten oder beendet ist. Die Adresse des MD Archiv (*mdsArchiveAddress*) wird benötigt, um ihm Nachricht z.B. über den Abbruch einer Protokollierung zu senden. Der Beginn der Aufzeichnung (*beginOfTracing*) wird gespeichert, damit ohne Zugriff auf den *Trace* eine Unterscheidung zwischen Protokollen möglich ist, die durch Konversationen mit gleichem Partner und Konversationsspezifikation erzeugt wurden. Zusätzlich kann damit eine chronologische Sortierung erreicht werden.

Im *Trace* werden die protokollierten Daten der Konversationen gespeichert. Für jedes Protokoll wird ein *Trace* erzeugt, der wiederum die Protokoll-ID (*id*) enthält. Über die Protokoll-ID wird die zugehörige *ConversationInfo* referenziert. Das Feld *elementList* verwaltet eine Liste von *TraceElements*. Ein *TraceElement* ist entweder ein Objekt *C*[12], das sich aus Dialog, Anfrage sowie Datum und Uhrzeit der Speicherung zusammensetzt, bzw. ein Objekt *P*[13], das aus Dialog sowie Datum und Uhrzeit der Speicherung besteht.

---

[12] C steht für Customer.

[13] P steht für Performer.

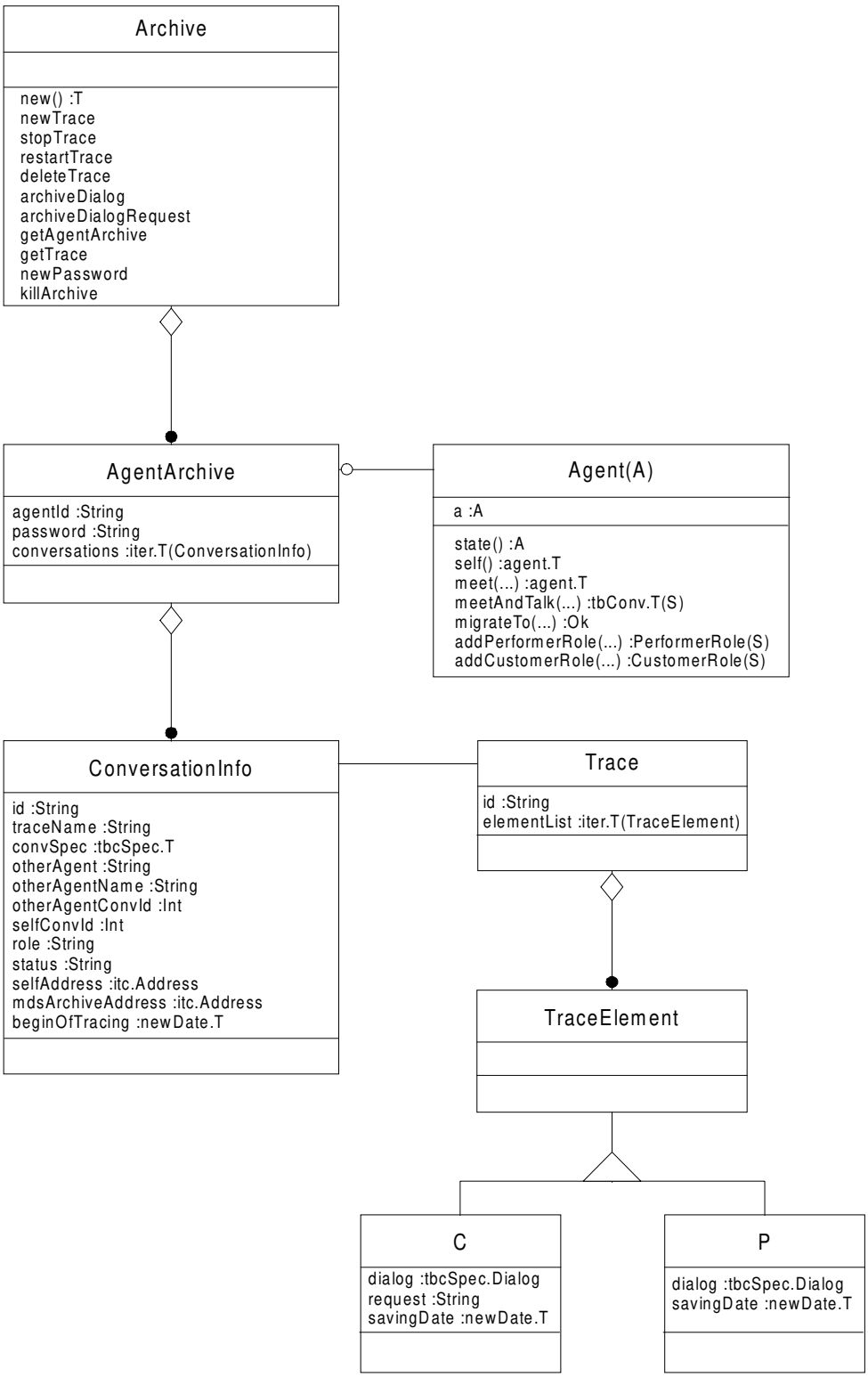

Abbildung 5.4: Objektstrukturmodell des Archivs

Dialoge, die vom Dienstleister stammen, werden als Objekt vom Typ *P* und die Dialoge und Anfragen des Kunden als Objekt vom Typ *C* gespeichert. In *elementList* wechseln sich also die beiden Objekttypen ab.

Die Teilung von *ConversationInfo* und *Trace* wurde vorgenommen, um ein Multi-Threading zu unterstützen. Die Daten, die vom MD Archiv kommen, werden ausschließlich im *Trace* gespeichert, während die meisten Aktionen des Archivars sich auf die *ConversationInfo* beschränken. Um ein Multi-Threading zu erreichen, müssen für die beiden Objekte lediglich geeignete Sperrmechanismen eingeführt werden.

5.4 Der Archivar

Der Archivar ist ein normaler Agent, der das Archiv verwaltet und für andere Agenten als Dienstleister in einer Konversation auftritt. In seinen Bearbeitungsregeln übersetzt er die Anfragen der Agenten in Nachrichten an das Archiv.

Der erste Schritt der vom Archivar angebotenen Konversation (siehe Abbildung 5.5) besteht darin, sich anzumelden. Von diesem Anmeldedialog aus kann entweder ein neues Protokoll erzeugt oder eine Übersicht über alle gespeicherten Protokolle abgerufen werden. Im Dialog „Neues Protokoll“ müssen die Daten zur Konversation eingegeben werden. Diese Aktion wird mit den Dialogen „Erfolg“ oder „Mißerfolg“ bestätigt. Über diese beiden Dialoge kann entweder das Ende der Konversation oder, falls weitere Aktionen ausgeführt werden sollen, die „Protokolliste“ erreicht werden.

In dem Dialog „Protokolliste“ kann ein im Archiv gespeichertes Protokoll ausgewählt werden. Wenn die dazugehörige Konversation noch läuft, kann die Protokollierung gestoppt werden. Wurde die Protokollierung gestoppt, kann sie weitergeführt werden. Dazu muß die beim Starten der Protokollierung angegebene Adresse des Agenten wegen einer eventuell nach dem Stoppen durchgeführten Migration im Dialog „Adresse“ bestätigt oder korrigiert werden. Die Adresse wird zur Ermittlung des MD Archiv benötigt. Weiterhin besteht die Möglichkeit, das Protokoll zu löschen. Diese Aktionen werden jeweils mit den Dialogen „Erfolg“ oder „Mißerfolg“ bestätigt.

Ergänzend kann zu einem Protokoll die Liste der gespeicherten Dialoge („Dialogliste“) aufgerufen werden. Ein ausgewählter Dialog kann angezeigt werden („Dialog“). Das Paßwort, das den Zugang zu den Protokollen ermöglicht, wird ebenfalls von der „Protokolliste“ aus geändert („Paßwort“). Auch hier findet über „Erfolg“ und „Mißerfolg“ die Bestätigung statt.

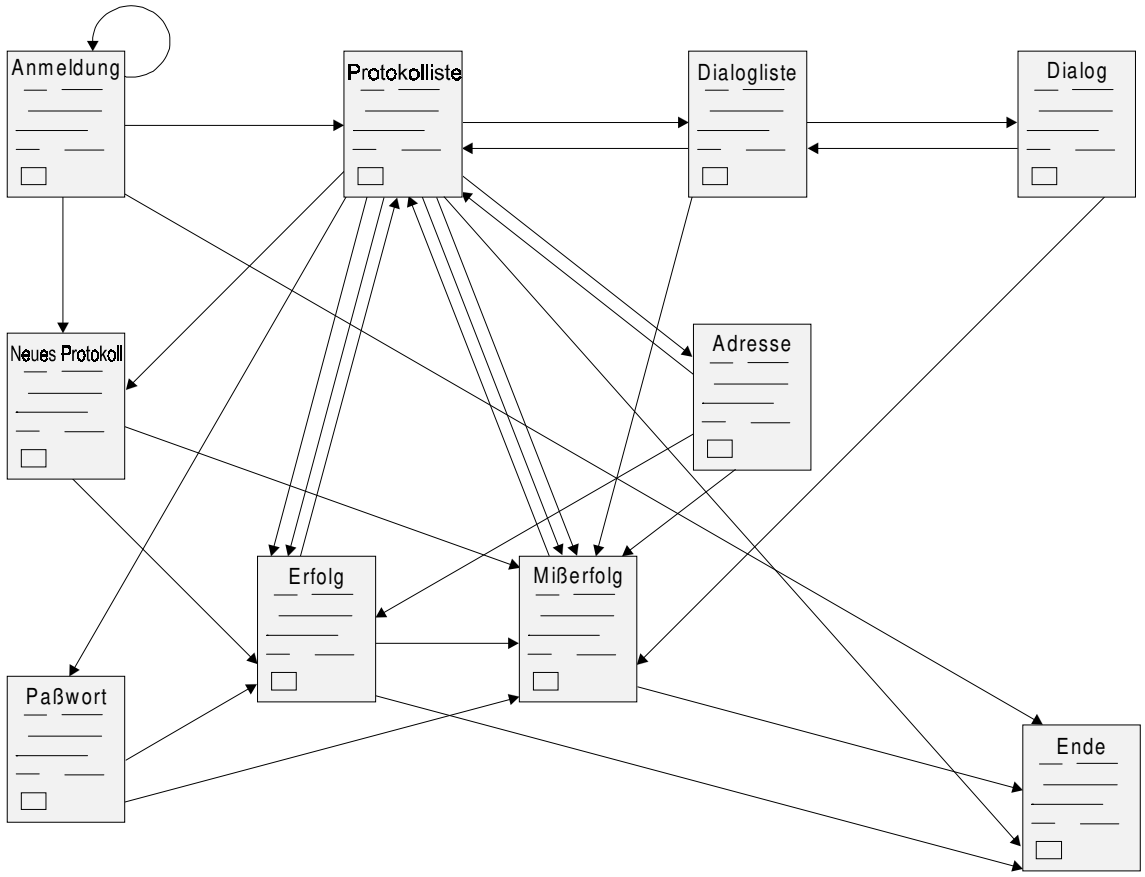

Abbildung 5.5: Konversation des Archivars

6. Kapitel

Realisierung

In diesem Kapitel wird die Realisierung des Protokollierungsdienstes beschrieben. Dazu wird zunächst anhand der Archivar-Konversation das Zusammenspiel der Komponenten und anschließend die Implementation der drei Komponenten Archivar, MD Archiv und Archiv beschrieben.

6.1 Die Konversation

In diesem Abschnitt sollen die Abläufe beschrieben werden, die bei den verschiedenen Aktionen in der Konversation eines Kunden mit dem Archivar stattfinden.

6.1.1 Starten einer Protokollierung

Um ein Protokoll zu erzeugen, beginnt der Agent als Kunde eine Konversation mit dem Archivar. Diese Konversation kann durchgeführt werden, bevor die zu protokollierende Konversation beginnt oder als sekundäre Konversation während der schon laufenden Konversation. Der Kunde füllt den Anmeldedialog mit seiner Agenten-ID und seinem Paßwort aus und wählt die Anfrage *Neues Protokoll*. Der Archivar verschickt den Dialog „Neues Protokoll“, den der Kunde folgendermaßen bearbeitet:

- Die Agenten-ID des Konversationspartners und die beiden Konversationsnummern müssen vom Kunden eingegeben werden, da der Mediator-Dienst diese Daten zur Zuordnung der Dialoge und Anfragen braucht.

- Die Eingabe der Adresse, von der aus die zu protokollierende Konversation geführt wird, ist notwendig, um die Adresse des zuständigen Mediator-Dienstes ermitteln zu können.

- Der Name des Konversationspartners, die Rolle des Agenten, der das Protokoll anlegt, eine Bezeichnung für dieses Protokoll sowie die Konversationsspezifikation können optional eingegeben werden.

Um die Konversationsspezifikation als Dialoginhalt verwenden zu können, waren Änderungen notwendig, die in Kapitel 6.2 beschrieben werden.

Wenn der Kunde den Dialog mit *OK* bestätigt, schickt der Archivar die Nachricht *newTrace* an das Archiv. Diese Nachricht enthält die Angaben aus den Dialogen „Anmeldung“ und „Neues Protokoll“.

Trifft die Nachricht *newTrace* zum Starten eines Protokolls beim Archiv ein, generiert das Archiv eine neue Protokoll-ID und schickt dem *Distributor* in der Domäne des Kunden die Nachricht *startTrace*. Der *Distributor* erzeugt daraufhin ein neues *ConvRep* und bestätigt dies dem Archiv. Das Archiv legt eine neue *ConversationInfo* mit den entsprechenden Daten an und erzeugt einen neuen *Trace*. Anschließend wird der Archivar vom Erfolg bzw. Mißerfolg dieser Aktion verständigt. Der Archivar generiert dann den entsprechenden Dialog für den Kunden.

Abbildung 6.1 zeigt den geschilderten Ablauf.

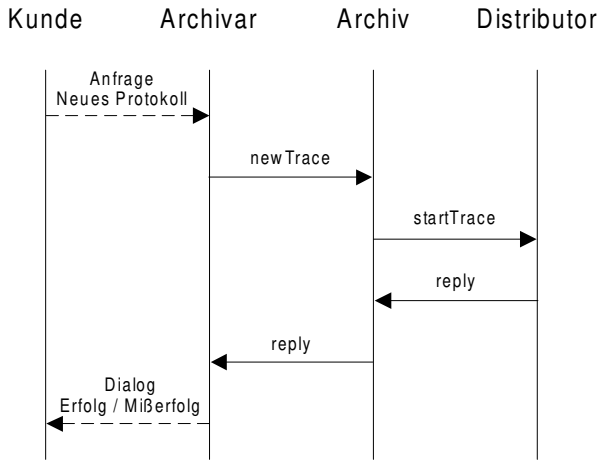

Abbildung 6.1: Nachrichtenfluß beim Starten einer Protokollierung

6.1.2 Protokolliste ansehen

Der Dialog „Protokolliste“ ist der zentrale Dialog der Konversation. Es stehen dort alle Aktionen zur Verfügung. Dies gilt auch für das Starten einer Protokollierung. Aus diesem Grund kann der Dialog von verschiedenen Dialogen aus erreicht werden.

Stellt der Kunde die Anfrage nach der Protokolliste, schickt der Archivar die Nachricht *getAgentArchive* ans Archiv. Das Archiv greift über das *AgentArchive* des Kunden auf die *ConversationInfos* zu und generiert daraus die Antwort *putAgentArchive*.

Der Archivar erzeugt aus den Daten den Dialog „Protokolliste“, der eine Liste mit folgenden Feldern enthält:

- die Bezeichnung des Protokolls, die der Kunde beim Erzeugen des Protokolls vergeben hat,
- die Protokoll-ID des Archivs,
- die Rolle des Archivar-Kunden in der protokollierten Konversation,
- den Namen des Konversationspartners,
- die ID des Partners,
- den Status der Protokollierung,
- den Namen der Konversationsspezifikation,
- die Spezifikation,
- den Beginn der Aufzeichnung und
- die Adresse, von der aus die Konversation geführt wurde oder wird.

Der Kunde kann aus der Liste einen Eintrag auswählen und folgende Aktionen durchführen:

- Stoppen der Protokollierung,
- Neustart einer gestoppten Protokollierung,
- Löschen eines Protokolls,
- Anzeigen der Dialogliste,

- Anlegen eines neuen Protokolls,
- Ändern des Paßworts und
- Beenden der Archivar-Konversation.

Für die letzten drei Aktionen muß kein Listeneintrag ausgewählt werden.

6.1.3 Stoppen einer Protokollierung

Wenn ein Agent eine laufende Protokollierung stoppen möchte, beginnt er eine Konversation mit dem Archivar. Bei seiner Anmeldung wählt er die Anfrage *Protokolliste*. Er erhält den entsprechenden Dialog und wählt das zu stoppende Protokoll und die Anfrage *Protokoll stoppen*. Der Archivar schickt die Nachricht *stopTrace* an das Archiv, das daraufhin die Nachricht *terminateTrace* an den *Distributor* schickt. Der *Distributor* löscht das entsprechende *ConvRep*. Das Archiv ändert den Status in *ConversationInfo* in *stopped* und antwortet dem Archivar. Dieser generiert daraufhin den Dialog „Erfolg“ bzw. „Mißerfolg“.

6.1.4 Neustart einer Protokollierung

Beim Neustart einer Protokollierung wird ein bestehendes, gestopptes Protokoll einer Konversation weitergeschrieben. Die neuen Dialoge und Anfragen werden an das bestehende Protokoll angefügt. Um die gestoppte Protokollierung weiterzuführen, wählt der Agent im Dialog „Protokolliste“ den entsprechenden Listeneintrag und die Anfrage *Neustart der Protokollierung*. Der Archivar generiert daraufhin den Dialog „Adresse“. In diesem Dialog muß der Kunde die Adresse bestätigen, von der aus er die protokollierte Konversation führt. Dies ist angesichts der Migrationsmöglichkeit der Agenten nötig. Wenn der Kunde den Adreßdialog mit *OK* beendet, schickt der Archivar die Nachricht *restartTrace* ans Archiv. Das Archiv wiederum schickt die Nachricht *startTrace* an den zuständigen *Distributor*. Dies ist die gleiche Nachricht wie beim Starten einer Protokollierung. Sie führt zur Erzeugung eines *ConvRep*-Objektes und der anschließenden Bestätigung an das Archiv, welches daraufhin in *ConversationInfo* den Status in *running* ändert und dem Archivar antwortet, der dann den Dialog „Erfolg“ bzw. „Mißerfolg“ generiert.

6.1.5 Löschen eines Protokolls

Wenn ein Agent ein Protokoll löschen möchte, wählt er in der Protokolliste den entsprechenden Eintrag und die Anfrage *Protokoll löschen*. Der Archivar schickt die Nachricht *deleteTrace* ans Archiv. Falls der Status der Protokollierung *running* ist,

wird dem *Distributor* die Nachricht *terminateTrace* geschickt. Das Archiv löscht sowohl die *ConversationInfo* als auch den *Trace*. War dies das einzige Protokoll des Kunden, wird auch das *AgentArchive* gelöscht. Nachdem der Archivar die Bestätigung erhalten hat, erzeugt er den Dialog „Erfolg“ bzw. „Mißerfolg“.

6.1.6 Dialogliste eines Protokolls ansehen

Um die Liste der zu einem Protokoll gespeicherten Dialoge anzuzeigen, wählt der Agent in der Protokolliste das Protokoll und die Anfrage *Dialogliste*. Der Archivar schickt die Nachricht *getTrace* ans Archiv. Mit der Antwort *putTrace* verschickt das Archiv die *TraceElements* des Protokolls. Der Archivar erzeugt daraus den Dialog „Dialogliste“, der eine Liste mit folgenden Angaben enthält:

- die Nummer des Listeneintrags,
- den Namen des Dialogs,
- die ausgewählte Anfrage, falls das Element vom Kunden stammt und somit eine Anfrage beinhaltet,
- Datum und Uhrzeit der Speicherung und
- ob der Eintrag vom Kunden oder Dienstleister stammt.

Der Agent kann einen Listeneintrag und die Anfrage *Dialog ansehen* wählen oder mit der Anfrage *Protokolliste* zum entsprechenden Dialog zurückkehren.

6.1.7 Dialog ansehen

Wählt ein Agent in der Dialogliste einen Dialog und die Anfrage *Dialog ansehen*, nimmt der Archivar keinen Kontakt zum Archiv auf, da er bereits für den Dialog „Dialogliste“ alle *TraceElements* erhalten hat. Der Archivar erzeugt den Dialog „Dialog ansehen“, indem er als Inhalt den ausgewählten Dialog einsetzt. Um einen Dialog als Dialoginhalt verwenden zu können, waren Änderungen notwendig, die in Kapitel 6.2 beschrieben werden. Der Kunde hat die Möglichkeit, sich entweder den nächsten oder den vorherigen Dialog des Protokolls anzeigen zu lassen oder die Dialogliste aufzurufen.

6.1.8 Paßwort ändern

Um sein Paßwort zu ändern, wählt der Agent in der Protokolliste die Anfrage *Paßwort ändern*. Er erhält den betreffenden Dialog und kann das neue Paßwort eingeben. Wenn er die Anfrage *OK* stellt, schickt der Archivar die Nachricht *newPassword* ans Archiv. Das ändert im *AgentArchive* das Paßwort entsprechend und bestätigt dem Archivar die Durchführung. Der erzeugt wiederum den Dialog „Erfolg“ bzw. „Mißerfolg“.

6.2 Der Archivar

Der Archivar ist im Modul *archivist.tm* realisiert und kann mit *archivist.new(...)* erzeugt werden. In der Agenten-Funktion *birth(...)* (siehe Kapitel 3.3) werden die Regeln zu der Konversation in der Regelbasis des Archivars registriert. Weitere Aktionen finden nicht statt. Der Archivar steht damit anderen Agenten als Dienstleister zur Verfügung. Die Funktion *meet(...)* (siehe Kapitel 3.3) liefert den Identifikator des Archivars, der für eine Konversation benötigt wird. Der Archivar führt keine Migration durch, damit er für die anderen Agenten jederzeit zu finden ist.

Innerhalb der Regeln kommuniziert der Archivar häufig mit dem Archiv. Die an das Archiv verschickten Nachrichten blockieren den Archivar jeweils bis zur Antwort. Er benötigt diese Antwort, um in der Konversation fortfahren zu können. Diese Einschränkung liegt im Konzept der Agenten begründet. Die Agenten sind ganz darauf ausgerichtet, Konversationen zu führen. Soll ein Agent als Front-End einer nachrichtenbasierten Anwendung eingesetzt werden, so muß er in seiner Rolle als Kunde oder Dienstleister innerhalb der Regeln Nachrichten senden und empfangen. Wenn das Empfangen von Nachrichten in einer Regel notwendig ist, blockiert der entsprechende Kunden- oder Dienstleisterthread. Das Problem läßt sich mit Einschränkungen umgehen, indem der Hauptthread des Agenten um den Typ der zu empfangenden Nachricht erweitert wird. Damit ist ein Empfangen - und Warten - innerhalb der Regel nicht mehr nötig. Allerdings kann eine Regelausführung nicht abgebrochen und später wieder aufgenommen werden, um die Regel zu Ende auszuführen. Die Regeln müssen immer von Anfang und bis zum Ende ausgeführt werden. Außerdem betrifft die Änderung des Hauptthreads alle Agenten, da sie im Agentengenerator vorgenommen werden muß. Um aufgrund der Blockade auftretende Engpässe zu vermeiden, können mehrere Archivare erzeugt werden.

Damit ein Kunde dem Archivar die Konversationsnummern der zu protokollierenden Konversation mitteilen kann, wurden Änderungen im Interface *Agent.ti* und im Modul *agent.tm* notwendig. Die Konversationsnummern waren im Systemteil der Agenten gekapselt. Um die Nummern in einer Konversation nutzen zu können,

wurden sie in *Conversation(S)* (siehe Kapitel 2.2.2) integriert. Diese Änderung betraf lediglich drei Stellen.

Um in der Archivar-Konversation eine Konversationsspezifikation als Bestandteil der Dialoge „Neues Protokoll“ und „Protokolliste“ verwenden zu können, mußte die Definition des Dialoginhalts in *TBCContent.ti* und *tbcContent.tm* geändert werden. Da das Interface *TBCContent.ti* von *TBCSpec.ti*, welches die Definition der Konversationsspezifikation enthält, importiert wird, konnte in *TBCContent.ti* nicht auf den Typ der Konversationsspezifikation Bezug genommen werden. Deshalb wurde mit einem dynamischen Typ gearbeitet.

Um zu erreichen, daß ein Dialog Bestandteil eines Dialogs sein kann, so wie es der Dialog „Dialog ansehen“ verlangt, mußte ebenfalls die Definition des Dialoginhalts geändert werden. Da auch hier wegen der Reihenfolge des Imports der Typ Dialog nicht zur Verfügung stand, wurde wiederum ein dynamischer Typ benutzt.

6.3 Der Mediator-Dienst Archiv

Der MD Archiv ist im Modul *mdsArchive.tm* implementiert. Der *Distributor* wird mit dem Befehl *mdsArchive.new()* in einer Domäne gestartet. Mit *mdsArchive.addMDSTransmitter(...)* wird der zugehörige Mediator-Dienst (*MDS Transmitter*) beim Mediator angemeldet. Vorher muß allerdings bereits der Mediator der Domäne initialisiert worden sein.

Der *MDS Transmitter* reagiert nur auf Nachrichten vom Typ *performerDialogA* und *customerDialogRequestA*. Falls eine Protokollierung des Verbindungsaufbaus zwischen den Agenten erreicht werden soll, muß der *MDS Transmitter* lediglich um die entsprechenden Nachrichten erweitert werden.

Trifft die Nachricht *performerDialogA* ein, erstellt der *MDS Transmitter* die Nachricht *mdsArchiveDialog*, welche die Agenten-IDs des Kunden und des Dienstleisters, die Konversationsnummer des Kunden sowie den Dialog enthält, und schickt sie dem *Distributor*.

Wenn die Nachricht *customerDialogRequestA* vom *MDS Transmitter* erkannt wird, erzeugt er die Nachricht *mdsArchiveRequest*, die aus den Agenten-IDs, der Konversationsnummer des Dienstleisters, dem Dialog und der Anfrage besteht, und sendet sie an den *Distributor*.

Um die Agenten-IDs verwenden zu können, wurden, wie schon in Kapitel 5.2 dargelegt, die Nachrichten *performerDialogA* und *customerDialogRequestA* um die Agenten-IDs erweitert. Dazu war lediglich eine Ergänzung der Definition der Nach-

richten im Interface *ITCMessage.ti* notwendig. Im Modul *agent.tm* mußten bei der Verwendung der Nachrichten die IDs hinzugefügt werden. Diese Änderung beschränkte sich auf acht Stellen.

Der *Distributor* verwaltet die *ConvRep*-Objekte mit einer Liste. Für jedes *ConvRep* existiert ein Eintrag. Wird der *Distributor* gestartet, befindet er sich in einer Schleife und wartet auf Nachrichten.

Wenn der *Distributor* die Nachrichten *mdsArchiveDialog* oder *mdsArchiveRequest* erhält, vergleicht er die Agenten-IDs und die Konversationsnummer aus der Nachricht mit denen seiner *ConvRep*-Liste. Stimmen die Daten mit einem *ConvRep* überein, wird entweder die Nachricht *archiveDialog* oder *archiveDialogRequest* erzeugt. Die Nachricht *archiveDialog* enthält außer dem Dialog nur die Protokoll-ID, die dem *ConvRep* entnommen wurde. Die Nachricht *archiveDialogRequest* enthält zusätzlich die Anfrage. Diese Nachrichten werden an das Archiv geschickt, dessen Adresse ebenfalls im *ConvRep* verzeichnet ist.

Trifft der finale Dialog einer Konversation ein, wird das zugehörige *ConvRep* automatisch aus der Liste gelöscht. Deshalb braucht der *Distributor* nicht vom Ende einer Konversation informiert zu werden.

Mit der Nachricht *startTrace* erhält der *Distributor* die Daten vom Archiv, die für eine Weiterleitung notwendig sind. Diese Daten sind die Protokoll-ID, die Agenten-IDs, die Konversationsnummern und die Adresse des Archivs. Gibt es für die Protokoll-ID noch kein *ConvRep* in der Liste des *Distributor*s, so wird ein solches angelegt und das Archiv darüber informiert. Ab diesem Zeitpunkt werden Dialoge und Anfrage dieser Konversation an das Archiv geschickt.

Diese Weiterleitung kann mit der Nachricht *terminateTrace* beendet werden. Die Nachricht enthält nur die Protokoll-ID. Der *Distributor* löscht daraufhin das entsprechende *ConvRep* aus seiner Liste. Soll später eine Fortsetzung der Weiterleitung erreicht werden, muß deshalb mit *startTrace* ein neuer Eintrag erzeugt werden.

Mit der Nachricht *killMDSArchive* wird der *Distributor* beendet. Damit geht auch die Liste der *ConvReps* verloren. Der *MDS Transmitter* wird über die Funktion des Mediators *removeService(...)* entfernt.

6.4 Das Archiv

Das Modul *archive.tm* realisiert das Archiv. Mit dem Befehl *archive.new()* wird das Archiv gestartet. In dieser Realisierung ist es nicht möglich, mehr als ein Archiv pro Domäne zu starten. Dies ist keine konzeptuelle Einschränkung und kann geändert

werden, indem das Starten durch Angabe einer Archivadresse parametrisiert wird. Nach dem Starten befindet sich das Archiv in einer Schleife und wartet auf Nachrichten.

Die in Kapitel 5.3 vorgestellten Objekte *AgentArchive* und *Trace* sind im Archiv als Hash-Tabellen implementiert. Jedem Agenten, der ein Protokoll erstellen läßt, wird ein Datensatz in der *AgentArchive*-Tabelle zugeordnet. Die Tabelle enthält die in Abbildung 5.4 aufgeführten Attribute. Die Agenten-ID dient als Primärschlüssel. Das Attribut *conversations* des *AgentArchive* verwaltet als Liste die *ConversationInfos* des Agenten. Die Attribute von *ConversationInfo* sind ebenfalls in Abbildung 5.4 dargestellt.

In der *Trace*-Tabelle wird für jedes Protokoll ein Datensatz angelegt. Die Tabelle besteht aus den in Abbildung 5.4 aufgeführten Attributen, mit der Protokoll-ID als Primärschlüssel. Das Feld *elementList* verwaltet die *TraceElements* mit einer Liste. Das *TraceElement* ist als variantes Tupel realisiert. Die Bestandteile der Varianten *C* und *P* sind ebenfalls in Abbildung 5.4 aufgeführt. Die Verknüpfung des *Trace* mit der *ConversationInfo* findet über die Protokoll-ID statt.

Die Nachricht *newTrace* erzeugt im Archiv ein neues Protokoll. Sie enthält die Agenten-ID und das Paßwort des Agenten, der das Protokoll anlegen möchte, die Adresse, von der aus er die Konversation führt, sowie die Angabe, ob er Kunde oder Dienstleister ist, die ID und den Namen des Konversationspartners, die Konversationsnummern, die Konversationsspezifikation und eine Bezeichnung für das Protokoll.

Das Archiv stellt zunächst fest, ob es in der *AgentArchive*-Tabelle einen Datensatz für diese Agenten-ID gibt und gegebenenfalls, ob das Paßwort korrekt ist. Dann erzeugt es eine eindeutige Protokoll-ID und schickt dem *Distributor* die Nachricht *startTrace*. Der zuständige *Distributor* wird aus der Adresse des Agenten ermittelt. Durch das Verschicken dieser Nachricht wird das Archiv bis zur Antwort blockiert. Dies ist keine Einschränkung des Konzeptes sondern der Realisierung. Ist der Agent dem Archiv bisher nicht bekannt, wird nach einer positiven Antwort des *Distributors* im Archiv ein neuer Datensatz in der *AgentArchive*-Tabelle angelegt. Dazu wird das angegebene Paßwort verwendet. Ist der Agent bekannt, wird kein neuer Datensatz angelegt. Im Feld *conversations* wird für das Protokoll ein neuer Eintrag, die *ConversationInfo*, erzeugt. Anschließend wird die *Trace*-Tabelle um einen neuen Datensatz mit der Protokoll-ID erweitert.

Mit der Nachricht *stopTrace* wird die Aufzeichnung eines Protokolls beendet. Die Nachricht enthält neben der Agenten-ID und dem Paßwort nur die Protokoll-ID. Nach der Überprüfung des Paßworts wird dem *Distributor* die Nachricht *terminateTrace* geschickt und der Status der Protokollierung in *ConversationInfo* in *stopped* geändert.

Die Fortsetzung einer Protokollierung, die mit *stopTrace* unterbrochen wurde, erfolgt durch die Nachricht *restartTrace*. Sie enthält ebenfalls Agenten-ID, Paßwort und Protokoll-ID. Zusätzlich ist noch die Adresse angegeben, von der aus der Agent die Konversation führt. Dies ist notwendig, da der Agent nach dem Stoppen der Protokollierung in eine andere Domäne migriert sein könnte. Das Archiv vergleicht diese Adresse mit der in *ConverationInfo* gespeicherten, um festzustellen, ob ein anderer *Distributor* zuständig ist, und ändert die gespeicherte Adresse gegebenenfalls. Anschließend erhält dann der *Distributor* die Nachricht *startTrace*. Bei einer positiven Antwort an das Archiv, wird dort der Status der Protokollierung in *ConversationInfo* in *running* geändert.

Die Nachricht *deleteTrace* dient zum Löschen eines Protokolls. Sie besteht aus der Agenten-ID, dem Paßwort und der Protokoll-ID. Wenn der Status der Protokollierung *running* ist, wird dem *Distributor* die Nachricht *terminateTrace* geschickt. In der *AgentArchive*-Tabelle wird die zugehörige *ConversationInfo* aus der Liste *conversations* ebenso gelöscht, wie der entsprechende Datensatz aus der *Trace*-Tabelle. War dies das letzte Protokoll des Agenten, wird darüber hinaus dessen Datensatz aus der *AgentArchive*-Tabelle gelöscht.

Die Dialoge einer Konversation erhält das Archiv vom *Distributor* durch die Nachricht *archiveDialog* (siehe Kapitel 6.3). Der Dialog wird als neues *TraceElement* in der *Trace*-Tabelle unter der angegebenen Protokoll-ID mit Datum und Uhrzeit des Eingangs gespeichert. Handelt es sich um einen finalen Dialog, wird außerdem der Status der Protokollierung in *ConversationInfo* in *finished* geändert.

Mit *archiveDialogRequest* (siehe Kapitel 6.3) empfängt das Archiv eine Anfrage mit zugehörigem Dialog. Diese Daten werden als ein weiteres *TraceElement* mit Eingangsdatum und -uhrzeit in der *Trace*-Tabelle gespeichert. Durch die Nachrichten *archiveDialog* und *archiveDialogRequest* erhält demnach das Archiv die Daten für das jeweilige Protokoll. Nur wenn es sich um einen finalen Dialog handelt, wird bei der Speicherung auf die *AgentArchive*-Tabelle zugegriffen.

Die Daten aus dem Feld *conversations* der *AgentArchive*-Tabelle können mit *getAgentArchive* abgerufen werden. Diese Nachricht besteht aus der Agenten-ID und dem Paßwort. Das Archiv antwortet mit *putAgentArchive*. Diese Nachricht beinhaltet die komplette *conversations*-Liste aus der *AgentArchive*-Tabelle.

Ein bestimmtes Protokoll wird durch *getTrace* angefordert. Diese Nachricht muß außer der Agenten-ID und dem Paßwort noch die Protokoll-ID enthalten. Das Archiv überprüft in der *AgentArchive*-Tabelle das Paßwort und verschickt mit der Nachricht *putTrace* die *elementList* des entsprechenden Protokolls aus der *Trace*-Tabelle.

Die Änderung des Paßworts ist mit *newPassword* möglich. Diese Nachricht besteht aus der Agenten-ID sowie dem alten und dem neuen Paßwort. Die Änderung findet sofort statt, so daß danach der Zugriff nur über das neue Paßwort möglich ist.

6.5 Die Integration des Protokollierungsdienstes

In diesem Abschnitt wird gezeigt, wie sich der Protokollierungsdienst in das Beispiel aus Kapitel 3.6 einfügt.

Im Schritt 1a (Module importieren) müssen zusätzlich die Module *archivist, archivistConvSpec, archive*, *mdsArchive* und *mediator* importiert werden. Als neuer Schritt 1c wird der Mediator gestartet, der Mediator-Dienst *MDSTransmitter* angemeldet, der Mediator initialisiert und schließlich der *Distributor* gestartet. Schritt 1d startet das Archiv.

Die Konversationsspezifikation des Archivars ist als Modul (*archivistConvSpec*) realisiert, und wird in Schritt 1a ebenfalls importiert. Die Agenten können dann mit *archivistConvSpec.convSpec()* auf die Spezifikation zugreifen.

In Schritt 3 wird der Archivar erzeugt. Er benötigt die Adresse des Archivs als Parameter. Der Archivar sollte gestartet werden, bevor andere Agenten erzeugt werden, damit der Archivar sofort Konversationen protokollieren kann.

1c) Mediator und Mediator-Dienst Archiv starten

```
let m = mediator.new();
mdsArchive.addMDSTransmitter(m);
itc.initMediator("THE-MEDIATOR");
mdsArchive.new();
```

1d) Archiv starten

```
archive.new();
```

3) Archivar starten

```
let archiveAddress = tuple "Archive"  itc.serverId() end;
archivist.new(archiveAddress);
```

7. Kapitel

Nutzungsmöglichkeiten des Protokollierungsdienstes

In diesem Kapitel werden Gründe für eine Protokollierung dargestellt und es wird untersucht, ob die daraus resultierenden Anforderungen an eine Protokollierung durch den erstellten Dienst erfüllt werden können.

Dazu werden zunächst Dokumentationsvorschriften und anschließend das Eigeninteresse des Anwenders thematisiert. Die Ausführungen erheben keinen Anspruch auf Vollständigkeit. Sie sollen exemplarisch die Bedeutung der Protokollierung aufzeigen.

7.1 Dokumentationsvorschriften

Dieser Abschnitt stellt gesetzliche und vertragliche Vorschriften vor, die eine Dokumentation notwendig machen.

7.1.1 Gesetzliche Vorschriften

Die zentrale gesetzliche Vorschrift zur Buchführungspflicht ist im Handelsgesetzbuch (HGB) zu finden. Nach § 238 Abs. 1 HGB ist jeder Kaufmann verpflichtet, „Bücher zu führen und in diesen seine Handelsgeschäfte und die Lage seines Vermögens nach den Grundsätzen ordnungsgemäßer Buchführung ersichtlich zu machen“.

Laut Steuergesetz besteht eine Buchführungs- und Aufzeichnungspflicht, die in der Abgabenordnung (AO) festgehalten ist. „§ 140 AO bestimmt, daß jeder, der nach anderen Gesetzen als den Steuergesetzen Bücher und Aufzeichnungen zu führen hat, die für die Besteuerung von Bedeutung sind, die(se) Verpflichtungen ... auch im Interesse der Besteuerung zu erfüllen hat.“ [Wöh92]

Damit sind die Dokumentationserfordernisse nach den handelsrechtlichen Grundsätzen ordnungsgemäßer Buchführung (GoB) mittelbar auch für die steuerlichen Dokumentationsanforderungen maßgeblich. Wöhe zufolge [Wöh92], hat der Gesetzgeber „den Begriff der Ordnungsmäßigkeit nirgends erschöpfend definiert". Diese Tatsache war und ist der laufenden, flexiblen Anpassung an neue Informationstechniken wie auch Organisationsprinzipien förderlich.

Die notwendige Harmonisierung im Rahmen der EU ist vom deutschen Gesetzgeber dadurch geleistet worden, daß er „... die geltenden Vorschriften über die Führung von Handelsbüchern... durch Regelungsinhalte der 4. EG-Richtlinie angereichert ..." hat [Wöh92].

Für Kapitalgesellschaften nehmen jeweils das Aktiengesetz in § 150, das Gesetz betreffend die Gesellschaften mit beschränkter Haftung (GmbH-Gesetz) in § 42 und das Gesetz über die Rechnungslegung von bestimmten Unternehmen und Konzernen (Publizitäts-Gesetz) in §§ 1 ff. betreffend der Aufzeichnungspflichten Bezug auf das HGB und die GoB.

Grundsätzlich gelten für alle gesetzlich vorgeschriebenen Aufzeichnungen definierte Archivierungsfristen von 10 bzw. 6 Jahren (HGB § 257 Abs. 4) und die Notwendigkeit der bildlichen Reproduzierbarkeit der empfangenen Handelsbriefe und der Buchungsbelege, bzw. der inhaltlichen Reproduzierbarkeit der übrigen Unterlagen, wenn die Aufbewahrung auf Bild- oder anderen Datenträgern erfolgt (HGB § 257 Abs. 3; § 261).

„Eine Aufzeichnung darf nicht in einer Weise verändert werden, daß der ursprüngliche Inhalt nicht mehr feststellbar ist. Auch solche Veränderungen dürfen nicht vorgenommen werden, deren Beschaffenheit es ungewiß läßt, ob sie ursprünglich oder erst später gemacht wurden." (HGB § 239 Abs. 3)

Weiterhin gibt es besondere gesetzliche Aufzeichnungspflichten zu technischen Prozessen (meist Input/Output- oder Rezeptur-Dokumentation) nach dem Umweltschutzrecht, dem Atomgesetz, dem Arzneimittelgesetz, der Drogenverordnung, der Gefahrgutverordnung, der verschiedenen Entsorgungsverordnungen etc.

7.1.2 Vertragliche Vorschriften

Auch vertragliche Vereinbarungen führen zu Dokumentationspflichten. So etwa

- im Innenverhältnis eines Unternehmens z.B. durch Gesellschaftervertrag, Satzung oder auch durch Betriebsvereinbarung (z.B. Arbeitszeitkonten),

- im Außenverhältnis durch unterschiedliche Vertragsarten (z.B. Versicherungsverträge, die eine regelmäßige Wartung oder Überprüfung von Maschinen, Feuermeldern etc. verlangen).

Im realisierten Archiv können alle Schritte einer Konversation aufgezeichnet werden. Da die Dialoge und Anfragen so gespeichert werden, wie die Agenten sie erhalten, ist mindestens die inhaltliche Reproduzierbarkeit sichergestellt. Außerdem gibt es keine Möglichkeit, das Protokoll zu verändern, so daß die entsprechenden Vorschriften des HGB erfüllt werden können.

Da mit dem Archiv eine komplette Dokumentation der gesamten Konversation erzeugt werden kann, muß bei der Konzeption der Dialoge lediglich darauf geachtet werden, daß sie alle Daten enthalten, die zur Erfüllung der gesetzlichen Vorschriften notwendig sind (z.B. Mwst.-Satz und -Betrag). Aus dieser Bedingung, die natürlich auch für die Schriftform einer Geschäftsbeziehung gilt, ergeben sich jedoch keine zusätzlichen Anforderungen an den Entwurf einer Konversationsspezifikation.

Die Sicherung der Archivierungsfristen liegt in der Hand des Anwenders. Ein Protokoll kann jederzeit gelöscht werden. Es ist allerdings kein Problem, das Löschen erst nach einer festgelegten Frist zu ermöglichen. Dazu muß lediglich ein Datumsvergleich vor dem Löschen stattfinden. Weiterhin kann das Löschen auch bestimmten Agenten vorbehalten bleiben. Die notwendigen Änderungen (z.B. die Einführung eines Löschkennworts) sind gering.

7.2 Dokumentationseigeninteresse

Es gibt verschiedene Gründe für den Anwender, zusätzlich zur gesetzlich vorgeschrieben Dokumentation weitere Daten zu archivieren.

7.2.1 Risikobegrenzung

Haftungsrechtliche Gründe sind z.B. das Interesse an

- der Sicherung von Beweismitteln für eventuelle spätere Rechtsstreitigkeiten aus Gewährleistungsansprüchen, Produkthaftung sowie anderen möglichen Haftungsgründen (z.B. Qualitätssicherungsmaßnahmen im Rahmen einer ISO-9000-Organisation),

- der Sicherung von Informationen zur erfolgreichen und schnellen Durchführung von Rückrufaktionen.

7.2.2 Performanzkontrolle und -steigerung

Es gibt vielfältige Gründe für die Protokollierung zu Analysezwecken. Scheer [Sch94] nennt z.B. die ständige Verbesserung der Daten, der organisatorischen Abläufe, des Ressourceneinsatzes und der -ausstattung. Dazu sei es erforderlich, die zur Verfügung stehenden aktuellen Soll- und Ist-Daten in Analysen auszuwerten. Nachfolgend sind einige Beispiele aufgeführt.

- Quantitative Analyse / Planung: Es gibt unterschiedlichste quantitative Analysen. Sie können sich z.B. auf den Absatz oder die Fertigung beziehen. In der Regel wird ein Zeitraum betrachtet, in dem für ein Zeitintervall eine Kenngröße ermittelt und ausgewertet wird. So ergibt sich z.B. für das erste Quartal pro Woche eine Absatzzahl. Bei diesen Analysen sind viele verschiedene Auswertungen denkbar (z.B. pro Woche, Monat, Quartal, Jahr etc. der Absatz eines Artikels, einer Artikelgruppe, etc.). Mit diesen Analysen wird die (Absatz-)Entwicklung betrachtet und Planungen für die Zukunft vorgenommen. Im wesentlichen handelt es sich dabei um Soll-/Ist-Vergleiche, die standardmäßig vorgenommen werden. Bei stark abweichenden Ist-Werten werden mit ad-hoc-Analysen die Ursachen erforscht.

  Sofern die für die gewünschte Analyse notwendigen Daten (z.B. Bestellung) im Rahmen von protokollierten Konversationen ausgetauscht werden, sind die Daten auch im Archiv vorhanden. Das Archiv bietet allerdings für die Auswertung keine Unterstützung. Dies gilt auch für die meisten noch folgenden Analysen. Das Archiv kann jedoch ohne großen Aufwand erweitert werden. Eine Möglichkeit besteht darin, eine generische Schnittstelle einzubauen, die Funktionen als Parameter akzeptiert. Diese Auswertungsfunktion kann dem Archiv per Nachricht geschickt werden. Sie wird vom Archiv ausgeführt und liefert das gewünschte Ergebnis, das als Antwort an den Absender versandt wird.

- Durchlaufzeiten: Die Durchlaufzeit gibt an, wie lange ein Geschäftsprozeß dauert. Ein Geschäftsprozeß besteht aus mehreren verschiedenen Arbeitsvorgängen. So kann ein Prozeß „Auftrag“ das Angebot, die Produktion, die Lieferung und die Rechnung umfassen. Es werden normalerweise minimale, durchschnittliche und maximale Durchlaufzeiten ermittelt. Die Zeiten dienen zur Optimierung und Überwachung des Prozesses. So kann eine zu hohe durchschnittliche Durchlaufzeit zu Änderungen im Ablauf des gesamten Prozesses führen. Außerdem kann aus diesen Zeiten das voraussichtliche Ende eines Vorgangs berechnet werden.

  Ein Geschäftsprozeß kann als eine *Business Conversation*, eventuell mit sekundären Konversationen, realisiert werden. Die Durchlaufzeit einer Konversation läßt sich im Archiv leicht feststellen, da alle Dialoge und Anfragen mit einem

Zeitindex abgespeichert sind. Die genauen Zeitpunkte des Eintreffens der Dialoge bzw. Anfragen beim Agenten stehen allerdings nicht fest, da die Archivdaten vom Mediator kommen. Es kann sowohl bei der Zustellung an den Empfänger als auch an das Archiv Verzögerungen, z.B. durch starke Auslastung des Netzwerks, gegeben haben. Dies muß bei der statistischen Auswertung entsprechend berücksichtigt werden. Diese Ungenauigkeiten können jedoch durch ein leistungsfähiges Netzwerk und kurze Entfernungen minimiert werden.

Die durchschnittliche Durchlaufzeit ergibt sich aus den Durchlaufzeiten aller Konversationen mit der entsprechenden Konversationsspezifikation. Dies läßt sich noch weiter differenzieren, z.B. indem für jeden Agenten eine mittlere Durchlaufzeit berechnet wird.

- Wartezeit / Bearbeitungszeit: Wenn sich die weitere Bearbeitung eines Vorgangs verzögert, z.B. durch eine ausgefallene Maschine, entstehen dort Wartezeiten. Dies kann weitreichende Folgen haben, da der Vorgang an den nächsten Bearbeitungsstationen eventuell schon erwartet wird, so daß auch dort Wartezeiten entstehen. Ziel von Geschäftsprozeßoptimierungen ist u.a. auch die Verringerung von Wartezeiten, allerdings ohne dabei Überkapazitäten zu schaffen. Die Bearbeitungszeit gibt an, wie lange für die Bearbeitung benötigt wurde. Diese Information wird für die Berechnung der Kosten des Vorgangs verwendet.

  Mit dem Archiv lassen sich weder Warte- noch Bearbeitungszeiten ermitteln, da nicht festgestellt werden kann, wann der Dialog beim Kunden und die Anfrage beim Dienstleister exakt eingetroffen sind und wann die Bearbeitung begonnen und beendet wurde. Um dies zu erreichen, müßten die Agenten beim Eintreffen eines Dialogs oder einer Anfrage sowie zu Beginn und am Ende der Regelauswertung einen Zeitindex an eine geeignete Protokollstelle senden. Das Archiv kann dahingehend erweitert werden, daß im *TraceElement* die entsprechenden Zeitpunkte aufgezeichnet werden. Dies führt allerdings zu erhöhtem Nachrichtenverkehr und stärkerer Beanspruchung des Archivs, da pro Konversationsschritt drei zusätzliche Nachrichten verschickt und protokolliert werden müssen.

  Die Summe von Warte- und Bearbeitungszeit kann durch das Archiv ermittelt werden, da die Zeitspannen zwischen Dialog und Anfrage bzw. Anfrage und Dialog bekannt sind. Allerdings sind auch hier Ungenauigkeiten durch eine mögliche Verzögerung im Netzwerk zu berücksichtigen.

- Auslastung / Engpässe: Es ist das Ziel, eine möglichst hohe Auslastung der Ressourcen (Mensch, Maschine, Software, etc.) zu erreichen, ohne daß dabei Engpässe und damit Wartezeiten auftreten. Die Auslastung eines Agenten kann über die Anzahl der geführten Konversationen ermittelt werden. Es ist notwen-

dig, zwischen einfachen und komplexen Konversationen zu unterscheiden, um den Aufwand entsprechend zu berücksichtigen.

Eine derartige Auswertung des Archivs ist möglich, da alle benötigten Daten gespeichert werden. Es besteht auch die Möglichkeit, festzustellen, zu welchen Zeiten eine besonders hohe Auslastung bei bestimmten Konversationen erreicht wird. Um Engpässe zu vermeiden, könnten Agenten, die zu dieser Zeit mit den eigenen Aufgaben nur gering ausgelastet sind, die überlasteten Agenten zeitweise unterstützen.

- Außergewöhnliche Situationen: Geschäftsbeziehungen, die nicht den Erwartungen entsprechend verlaufen, werden zwecks Analyse aussortiert. Beispiele dafür sind der häufige Abbruch des Kontaktes durch den Kunden in einer bestimmten Situation, der Ausfall eine Ressource, z.B. wegen Krankheit des Sachbearbeiters oder das Erreichen bestimmter Grenzwerte, z.B. des Kreditlimits des Kunden beim Dienstleister.

  Aus dem realisierten Archiv lassen sich derart auffällige Konversationen selektieren, da auf die einzelnen Schritte der Konversation gezielt zugegriffen werden kann. Wird ein Fehler im Agenten bemerkt, kann mit Hilfe des Archivs festgestellt werden, an welcher Stelle der Konversation der Fehler auftrat. Dies führt zu einer ständigen Verbesserung der Agenten.

- Kundenanalyse: Aus dem Verhalten der Kunden lassen sich vielfältige Rückschlüsse ziehen. So kann durch veränderte Angebotskonditionen, z.B. Rabatte, ein abgelehntes Angebot doch noch akzeptiert werden. Wichtig ist zudem, welche Artikel und Dienstleistungen viel bzw. wenig gefragt sind. Alle diese Informationen können aus dem Archiv gewonnen werden. So läßt sich z.B. auch feststellen, wie sich die Kunden bei Alternativen entschieden haben. Diese Analysemöglichkeit betrifft nicht nur externe Kunden, sondern kann auch in firmeninternen Geschäftsprozessen angewendet werden.

8. Kapitel

Zusammenfassung und Ausblick

Dieses Kapitel faßt die in der vorliegenden Arbeit gewonnenen Ergebnisse zusammen. Dazu wird zunächst das Vorgehen bei der Entwicklung des Protokollierungsdienstes beschrieben. Danach wird untersucht, ob die Realisierung die gestellten Anforderungen an einen Protokollierungsdienst erfüllt. Abschließend wird ein Überblick über weiterführende und zukünftige Arbeiten gegeben.

8.1 Zusammenfassung

In dieser Arbeit findet eine Betrachtung aller die Protokollierung betreffenden Aspekte des Tycoon-Agentensystems statt. Kapitel 2 stellt das medien- und aktorenunabhängige Kooperationsmodell der *Business Conversations* vor und untersucht die Realisierung als *Tycoon Business Conversations*. Dann wird in Kapitel 3 das Tycoon-Agentensystem selber analysiert. Diese Analyse umfaßt neben den Agenten auch Teile des Nachrichtensubsystems. Ein Beispiel für die Verwendung der Agenten ist ebenfalls enthalten. Anschließend werden in Kapitel 4 die Anforderungen an einen Protokollierungsdienst formuliert und die sich aus der Analyse ergebenden Alternativen bei der Konzeption dieses Dienstes dargestellt. Die Beschreibung des resultierenden Designs des Protokollierungsdienstes ist in Kapitel 5 zu finden, während in Kapitel 6 die Implementation vorgestellt wird. Kapitel 7 thematisiert sowohl eine Untersuchung über die Notwendigkeit einer Protokollierung als auch inwieweit der realisierte Dienst diesen Ansprüchen gerecht wird.

8.2 Bewertung

In diesem Abschnitt werden die in Kapitel 4.1 gestellten Anforderungen mit den Fähigkeiten des realisierten Protokollierungsdienstes verglichen.

- Es können alle Dialoge und Anfragen einer Konversation aufgezeichnet werden. Es besteht aber auch die Möglichkeit, nur Teile zu protokollieren. Die Agenten können für jede Konversation selber bestimmen, ob und in welchem Archiv das Protokoll gespeichert werden soll. Neben den Dialogen und Anfragen werden auch die relevanten Informationen über die Konversation gespeichert. Diese ermöglichen eine Vielzahl von Auswertungen der Protokolle. Einige Beispiele sind in Kapitel 7.2.2 aufgeführt. Auch wenn die meisten Auswertungen nicht implementiert wurden, so liegen doch alle relevanten Daten vor.

- Die Agenten, die den Protokollierungsdienst nutzen möchten, können alle Funktionen des Dienstes über eine Konversation aufrufen. Es müssen einmalig die Regeln für diese Konversation erstellt werden. Ein zusätzlicher Programmieraufwand entsteht nicht. Diese Regeln können dann von den Agenten registriert werden. Die Agenten sind nicht an der Weiterleitung der Daten an den Protokollierungsdienst beteiligt, so daß die protokollierte Konversation nicht beeinträchtigt wird. Auch eine Migration der Agenten während der Protokollierung einer Konversation stellt kein Problem dar. Der Agent teilt seine neue Adresse nach der Migration dem Archivar mit.

- Das Archiv, welches die Protokolle speichert, kann in jedem Ort einer Domäne angelegt werden. Es ist auch möglich, mehrere verschiedene Archive in der gleichen Domäne anzulegen. Damit können Engpässe vermieden und konkurrierende Dienste geschaffen werden.

- Die drei Komponenten des Protokollierungsdienstes (Archivar, Archiv und MD Archiv) haben klar definierte Schnittstellen. Daher ist es möglich, die einzelnen Komponenten unabhängig weiterzuentwickeln. So kann z.B. der Archivar alternativ oder auch zusätzlich eine anders aufgebaute Konversation anbieten. Auch die Zusammenarbeit der drei Protokollierungskomponenten mit anders konzipierten Komponenten ist realisierbar.

Die *Business Conversations* haben sich im Laufe der Arbeit als gutes Interaktionsmodell erwiesen. Die Konzeption einer Konversationsspezifikation ist aufgrund der Ähnlichkeit der *Business Conversations* mit Vorgängen der realen Welt einfach. Der Aufbau eines Dialogs erfolgt analog der Gestaltung eines Formulars. Die Erstellung von Dialogspezifikationen ist allerdings zur Zeit noch aufwendig. So umfaßt die Konversationsspezifikation für den Archivar ca. 200 Zeilen Programmkode. Da die Struktur der Dialogspezifikationen einfach ist, bietet es sich an, ein grafisches Werkzeug zur Erzeugung von Dialog- und Konversationsspezifikationen zu erstellen. Damit sollten auch Anwender selber in der Lage sein, eine ihren Ansprüchen entsprechende Spezifikation zu erstellen. Da auf diese Weise die Datenstruktur vorgegeben ist, wird Entwicklern die Erstellung der Anwendung erleichtert. Die in der

Vergangenheit vielfach aufgetretenen Mißverständnisse zwischen Anwendern und Entwicklern werden verringert.

Die Erweiterungen der Dialoginhalte, die in dieser Arbeit vorgenommen wurden, waren notwendig, da die Realisierung für den Dialoginhalt zuvor nur wenige komplexe Strukturen umfaßte. Durch modifizierte Ansprüche an die Inhalte werden auch in Zukunft noch Erweiterungen erforderlich werden. Diese Ausbaufähigkeit ist durchaus beabsichtigt, da ein Anspruch auf Vollständigkeit die für künftige Entwicklungen notwendige Flexibilität beeinträchtigt.

Das Konzept des Mediators hat sich ebenfalls bewährt. Es bietet eine Schnittstelle, um nachrichtenorientierte Dienste wie z.B. Monitoring und Sicherheit zu integrieren. Dabei ist zu berücksichtigen, daß in verschiedenen Domänen unterschiedliche Mediator-Dienste zur Anwendung kommen können. Für den Protokollierungsdienst ist es notwendig, daß der MD Archiv in der Domäne aktiv ist, in der sich der Agent befindet, der die Protokollierung veranlaßt hat.

Die Konzeption und Realisierung des Agentensystems, die Kooperation auf Basis der *Business Conversations* und die Integration von Anwendungen haben eine vielseitige Entwicklungsumgebung geschaffen. Es wurden nur minimale Änderungen notwendig, die zudem nicht auf konzeptuellen Problemen beruhten. Die Erstellung von Agenten ist aufgrund der Generatoren und der Trennung von System- und Anwendungsteil einfach. Die Agenten definieren sich aus den bekannten Konversationsspezifikation und den zugehörigen Regeln.

Der Archivar ist bedingt durch die Übersetzung der Anfragen der Kunden in Nachrichten an das Archiv und zurück mit über 800 Zeilen Programmkode relativ umfangreich. Das Archiv ist als Anwendung mit über 600 Zeilen nicht besonders groß, während der MD Archiv sogar nur ca. 200 Zeilen benötigt. Zusätzlich zu diesen Komponenten wurde eine Beispielanwendung zu Demonstrationszwecken realisiert. Die Anwendung besteht aus zwei Agenten, die eine Konversation führen, welche vom Protokollierungsdienst aufgezeichnet wird.

8.3 Ausblick

Der in dieser Arbeit erstellte Protokollierungsdienst kann auf verschiedene Weise erweitert werden. So sollten einige Standardauswertungen (siehe Kapitel 7.2.2) integriert werden. Dies führt zu einem entsprechenden Anpassungsbedarf beim Archivar.

Wenn ein Geschäftsprozeß aus mehreren Konversationen besteht, die alle protokolliert werden, fehlt im Archiv die Verknüpfung der zugehörigen Einträge. Dies führt

bei der Verfolgung des Prozesses zu einem zusätzlichen Aufwand, der durch eine entsprechende Erweiterung beseitigt werden sollte.

Die Protokolle werden gegenwärtig immer dem Agenten zugeordnet, der sie erzeugt hat. Insbesondere in größeren Firmen kann für kommerzielle Einsätze eine abteilungs- oder vorgangsbezogene Speicherung sinnvoller sein. Auch die Integration einer Benutzerverwaltung mit Zugriffsrechten ist denkbar.

Das Tycoon-Agentensystem wird zur Zeit unter Berücksichtigung der aus dem Prototyp gewonnenen Erfahrungen in Tycoon-2 neu implementiert. Die Visualisierung der Dialoge mit Hilfe eines Internetbrowsers ist ebenfalls in Arbeit. Damit besteht die Möglichkeit, daß Menschen Konversationen führen bzw. überwachen und eingreifen können. Andere notwendige Erweiterungen sind Agenten, die Brokerdienste anbieten, z.B. um festzustellen, welche Agenten in einem Ort vorhanden sind oder wo ein bestimmter Agent zu finden ist.

Anhang A

Glossar *Business Conversations*

Konversation (*conversation*): Der Oberbegriff für eine Folge von Dialogen zwischen einem *Kunden* und einem *Dienstleister* mit dem Ziel einer Aktion.

Sprechakt (*speech act*): Ein Begriff aus der Language-Action Sicht auf CSCW[14] nach T. Winograd [FGHW88]: „... human beings are fundamentally linguistic beings: action happens in language in a world constituted by language.". Genauer gesagt 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 Implementation widergespiegelt werden. Sie dient allein der Veranschaulichung des Ablaufs von Sprechakten.

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

Anfrage (II) (*request*): Die Reaktion eines Kunden auf einen Dialog. Die Bearbeitung einer Anfrage beim Dienstleister führt zu einem neuen Dialog und damit zum Zustandsübergang in der Konversation.

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

---

[14] Computer Supported Cooperative Work

kann der Kunde jedoch gegenüber einem anderen Kunden 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 Dienstleisters jedoch gegenüber einem anderen Dienstleisters in der Rolle des *Kunden* auftreten.

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

Regel (R*ule*): 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 seinen eigenen Regeln und erzeugt eine Anfrage.

Annahme (*acceptance*): Die ersten beiden Phasen eines Sprechaktzyklus.

Erfüllung (*completion*): Die letzten beiden Phasen eines Sprechaktzyklus.

Dialog (*dialog*): Ein Schritt einer Phase eines Sprechakts. Eine Phase kann beliebig viele Dialogschritte erfordern. Dialoge enthalten strukturierte Informationen, ihren *Inhalt*.

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

Historie (*history*): Die Historie bildet die gemeinsame Vergangenheit von Kunde und Dienstleister innerhalb einer Konversation. Sie wird transparent vom System fortgeschrieben und enthält alle Dialoge. Sowohl Kunde als auch Dienstleister haben eine eigene Instanz dieser Historie, die mit dem Ende der Konversation gelöscht wird.

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

Anhang B

Agententerminologie

Domäne (*domain*): Eine Komponente, die *Orte* und *Agenten* beherbergt und verwaltet. Domänen sind symbolisch adressierbar und besitzen immer mindestens einen Ort, welcher die Domäne selbst repräsentiert.

Aktivität (*activity*): Eine Aktivität ist eine beliebige Aufgabe, die durch einen oder mehrere *Agenten* modelliert wird. Dabei können Aktivitäten über Adreßräume verteilt sein.

Mobiler Software-Agent (*mobile agent*): Ein mobiler Software-Agent ist eine Komponente, welche aus einer oder mehreren aktiven Einheiten besteht. Ein Agent ist Teil mindestens einer *Aktivität*. Mobile Agenten sind in der Lage, sich über Adreßraumgrenzen zu bewegen. Agenten kommunizieren untereinander über typisierte *Nachrichten*.

Ort (*place*): Ein Ort ist eine spezielle Form eines *Agenten*, der eine *Domäne* aufgabenspezifisch logisch untergliedert. Jeder Ort ist durch einen lokal zur Domäne eindeutigen Namen identifiziert. Orte können wiederum andere Orte und Agenten beinhalten und bilden eine Hierarchie. Sie sind Sichtbarkeitsbereiche für die Adressierung von Agenten.

Kommunikationsport (*port*): An einem Kommunikationsport empfängt eine *aktive Einheit* Nachrichten. Es gibt benannte und anonyme Kommunikationsports. Eine aktive Einheit kann einen benannten Kommunikationsport über dessen Adresse identifizieren und Nachrichten an diesen senden. Ein anonymer Port kann nur Nachrichten von aktiven Einheiten empfangen, die eine Objektbindung an den (privaten) Portidentifikator besitzt.

Portidentifikator (*port identifier*): Ein Portidentifikator ist ein weltweit eindeutiger Bezeichner, der automatisch einem neu erzeugten Port zugewiesen wird. Alle Ports werden über Nachrichtensubsysteme mittels dieses Identifikators referenziert.

Nachricht (*message*): Nachrichten sind typisierte Botschaften und werden über *Ports* ausgetauscht. Es gibt einen definierten Satz von Nachrichten, welcher die Grundlage der Kooperationsprotokolle auf Systemebene zwischen Agenten bildet.

Aktive Einheit (*thread*): Eine aktive Einheit ist in der Lage, eine Nachricht an eine andere (nicht notwendigerweise verschiedene) aktive Einheit zu versenden.

Treffen (*meeting*): Ein Treffen findet zwischen zwei Agenten innerhalb eines Ortes statt. Treffen werden durch die Domäne vermittelt und koordinieren den Anfang einer Kooperation zwischen Agenten, indem sie diese untereinander sichtbar machen.

Anhang C

Die Notation von OMT

Die Objektmodelle in dieser Arbeit nutzen die Notation von OMT [RBP+91]. Nachfolgend wird die benutzte Notation erläutert. Dies stellt nur einen Ausschnitt der gesamten Methode dar. Für die Erläuterung der Methode wird auf die Literatur verwiesen.

Eine Klasse wird als Rechteck dargestellt. Sie kann nur mit dem Namen bezeichnet werden oder auch Angaben zu Attributen und Operationen enthalten (siehe Abbildung C.1). Ein Attribut wird dabei durch einen Bezeichner definiert. Zusätzlich kann der Datentyp des Attributs angegeben werden. Eine Operation wird ebenfalls mit dem Namen aufgeführt. Es können auch eine Liste der Argumente und der Ergebnisdatentyp hinzugefügt werden. Steht das Schlüsselwort *abstrakt* hinter einer Operation, bedeutet dies, daß die Operation nicht in dieser Klasse implementiert ist. Die Implementation muß in einer Unterklasse vorgenommen werden.

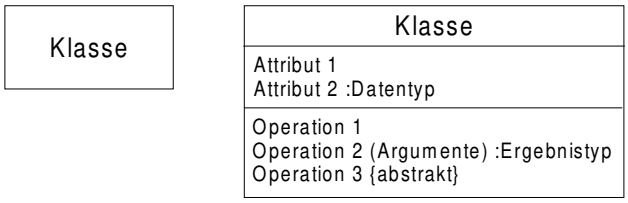

Abbildung C.1: Darstellung von Klassen

Beziehungen werden durch Linien dargestellt, die die entsprechenden Klassen verbinden (siehe Abbildung C.2). Die Beziehungen können benannt werden. Für geordnete Beziehungen existiert das Schlüsselwort *geordnet*.

Die Kardinalität von Beziehungen wird jeweils am Ende der Linie kenntlich gemacht. Die verwendeten Typen sind: (min. 1, max. 1), (min. 0, max. *n*), (min. 1, max. *n*) und (min. 0, max. 1). Zusätzlich besteht die Möglichkeit, die Kardinalität durch Angabe von Ziffern festzulegen.

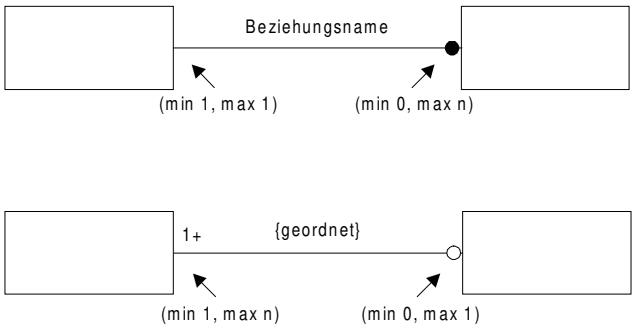

Abbildung C.2: Kardinalitäten von statischen Beziehungen

Die Vererbung wird durch ein Dreieck in der Beziehung symbolisiert (siehe Abbildung C.3). In den Unterklassen werden Attribute und Operationen der Oberklasse erweitert.

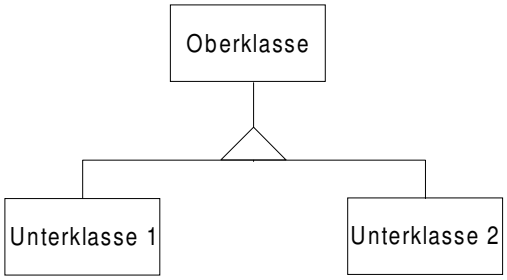

Abbildung C.3: Vererbung

Die Aggregation zeigt, welche Klassen in einer Teil-Ganzes-Beziehung stehen (siehe Abbildung C.4). Das Symbol dafür ist die Raute. Sie befindet sich auf der Ganzes-Seite der Beziehung. Kardinalitäten können ebenfalls verwendet werden.

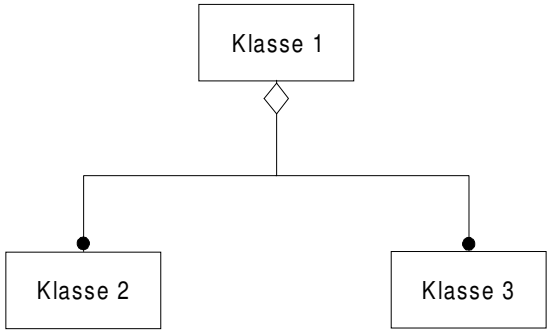

Abbildung C.4: Aggregation

Literaturverzeichnis

[BW94] Beale, Russel und Wood, Andrew. „Agent-Based Interaction“. In: *People and Computers IX*, Proceedings of HCI’94, Seite 239-245, August 1994.

[DDJ+97] De Michelis, Giorgio, Dubois, Eric, Jarke, Matthias, Matthes, Florian, Mylopoulos, John, Papazoglou, Mike, Pohl, Klaus, Schmidt, Joachim, Woo, Carson und Yu, Eric. „Cooperative Information Systems: A Manifesto“. In: Papazoglou, Mike P. and Schlageter, Gunther (Hrsg.). *Cooperative Information System: Trends and Directions*. Academic Press, 1997.

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

[Joh97] Johannisson, Nico. „Eine Umgebung für mobile Agenten: Agentenbasierte verteilte Datenbanken am Beispiel der Kooplung autonomer „Internet Web Site Profiler““. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1997.

[Mat93] Matthes, Florian. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung*. Springer Verlag, Berlin u.a. 1993.

[Mat97a] Matthes, Florian. „Business Conversations: A High-Level System Model for Agent Coordination“. (to appear in the Proceedings of DBPL-97), März 1997.

[Mat97b] Matthes, Florian. „Mobile Processes in Cooperative Information Systems“. In: *Proceedings STJA'97* (Erfurt, September 1997), Springer-Verlag, Berlin u.a.

[MMM93] Mathiske, Bernd, Matthes, Florian, und Müßig, Sven. „The Tycoon System and Library Manual“. DBIS Tycoon Report 212-93, Fachbereich Informatik, Universität Hamburg, Dezember 1993.

[MMS94] Matthes, Florian, Müßig, Sven, und Schmidt, Joachim W. „Persistent Polymorphic Programming in Tycoon: An Introduction“. FIDE Technical Report FIDE/94/106, Fachbereich Informatik, Universität Hamburg, August 1994.

[Ric97] Richtsmeier, Ingo. „Vergleich objekt- und agentenbasierter verteilter Datenbankprogrammierung am Beispiel der Kopplung autonomer Internet WebSiteProfiler“. Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 1997.

[RBP+91] Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick und Lorensen, William. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1991.

[Sch94] Scheer, August-Wilhelm. *Wirtschaftsinformatik*. 4. Aufl., Springer Verlag, 1994.

[Win87] Winograd, Terry A. „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.

[Wöh92] Wöhe, Günther. *Betriebswirtschaftliche Steuerlehre*. Bd. 1, 7. Aufl., München 1992.

[You96] Yourdon, Edward et al. *Mainstream Objects*. Prentice Hall Verlag GmbH, dt. Ausgabe 1996.

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Diplomarbeit selbständig durchgeführt und keine anderen als die benutzten Quellen und Hilfsmittel benutzt habe.

Hamburg, den 8. Dezember 1997

(Wolfgang Kruse)