university of applied sciences
gegr. 1970 fachhochschule hamburg
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIK

Diplomarbeit

Matthias Jung

Design und Realisierung einer Plattform für Informationsagenten

Studiengang Softwaretechnik

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. rer. nat. Christoph Klauck
Abgegeben am 11. Oktober 2001

BERLINER TOR 3 · D-20099 HAMBURG

Zusammenfassung

Design und Realisierung einer Plattform für Informationsagenten

Stichworte

Softwareagent, Informationsagent, Multiagentensystem, FIPA, Kooperation, Kommunikation

Zusammenfassung

In Zukunft sollen Agenten den Menschen bei komplexen Aufgaben unterstützen oder diese sogar ganz übernehmen. Eine Zukunftsvision ist, dass viele Agenten, die jeweils Experten auf speziellen Gebieten sind, in einem weltweiten Netzwerk miteinander kooperieren und Informationen und Dienste anbieten, die aber auch von Menschen untereinander genutzt werden können. Damit (fremde) Softwareagenten miteinander kooperieren können, werden gewisse Infrastrukturen vorausgesetzt. Diese werden normalerweise von einer Agentenplattform angeboten, die unter anderem die Kommunikation zwischen Agenten regelt. In dieser Arbeit wird am Beispiel eines Informationsagenten gezeigt, wie ein solcher, zukünftiger Informationsdienst aussehen könnte und es wird eine Agentenplattform entworfen, die die benötigten Infrastrukturen anbietet. Die Realisierung der Plattform und des Agenten, sowie ein kleines Anwendungsszenario bestätigen die vorausgegangenen Überlegungen, zeigen aber auch, welche Probleme bei der Verwirklichung der obigen Vision noch zu bewältigen sind.

Design and realisation of a platform hosting information-agents

Keywords

software agent, information agent, multi agent system, FIPA, cooperation, communication

Abstract

In the future, agents should support the work of humans or replace this function completely. A future vision for expert agents in a specific field is to cooperate and offer services that can be used by humans and agents in a worldwide network. For a cooperation of (foreign) software agents certain infrastructures are required. These are usually offered by an agent platform that among other things controls communication between agents. Using the example of an information agent, this document shows an approach of such an information service and the design of an agent platform supporting the mentioned infrastructures. The implementation of the platform and the agent, as well as a little scenario, confirm previous approaches and demonstrate problems with the realisation of the introduced vision.

Inhalt

Abbildungsverzeichnis

1 Einleitung

Seit einigen Jahren beeinflussen Computer unser tägliches Leben immer stärker, ob gewollt oder ungewollt. Durch Omnipräsenz und immer intelligentere Fähigkeiten, nimmt der Computer nach und nach einen völlig neuen Stellenwert ein. Während er dem Menschen noch vor wenigen Jahren als eine Art Werkzeug gedient hat, dass genutzt wurde um mehr oder weniger komplexe Tätigkeiten auszuführen, ist ein Trend zu erkennen, der einen Computer immer mehr zu einer Art Partner des Menschen werden lässt. *Personal Digital Assistants* (PDA) verwalten Adressen, Termine und andere wichtige Informationen, die sich der Benutzer nicht mehr zu merken braucht. Es soll mittlerweile Leute geben, die ohne PDA ihr Leben nicht geregelt führen können. *Personal Information Manager* (z.B. MS Outlook) verwalten nicht nur Emails, sondern beantworten diese in Abwesenheit des Benutzers oder informieren ihn über den Erhalt wichtiger neuer Emails. In gefährlichen Situationen beim Autofahren, in denen der Mensch überfordert ist, helfen Fahrerunterstützungssysteme (ABS, ESP) Unfälle zu vermeiden. Diese Entwicklung geht soweit, dass Computer als Spielzeuge sogar Haustiere imitieren und ersetzen, wie es am Beispiel *Furby* [1] oder die *Sony Legged Dogs* [2] zu erkennen ist. Ohne einen gewissen Grad an Zuwendung und Spielaufwand werden sie „traurig“ oder „sterben“ im schlimmsten Fall, wie das *Tamagotchi*.

Neue Übertragungsmedien und Kommunikationsformen verbinden zukünftig mobile Computer mit Netzwerken, wie dem Internet, und werden völlig neue Dienste ermöglichen. Auch dieser Trend ist heute schon zu erkennen, so informiert das Handy seinen Benutzer über neue Emails oder wichtige Börsen-, Politik- und Sportnachrichten. Wohin diese Entwicklung in ein paar Jahren führen könnte, soll einmal an einem kleinen Szenario dargestellt werden.

Man stelle sich vor, man frühstückt an einem sonnigen Freitagmorgen im späten September vor dem Straßencafe *Anita Wronsky* im Berliner Stadtteil Prenzlauer Berg. Auf dem Tisch befindet sich eine Tasse Milchkaffee und ein PDA. Dieser Morgen ist besonders zu genießen, da das ganze Wochenende über keine Termine anliegen und sich auf dem Konto ein Betrag von 3000,- DM angesammelt hat, der ohne Bedenken ausgeben werden kann. Schließlich wird der PDA befragt: „Was kann ich dieses Wochenende für maximal 3000,- DM unternehmen?“. Nach einigen Minuten und einem Croissant, inzwischen wurde das Frühstück gebracht, schlägt der PDA folgende Aktivitäten vor:

Freitag	15:00 Uhr	Abflug Berlin Tegel nach Barcelona, Spanien
	18:00 – 20:00Uhr.	Anreise Hotel und Checkin
	20:00 Uhr	Essen im Restaurant: Meson de Jesus
Samstag	12:00 – 14:00 Uhr	Besuch des Museu Picasso
	16:00 – 18:00 Uhr	Besuch der Sagrada Familia
	20:00 Uhr	Abendessen im Restaurant: Set Portes
	22:00 Uhr	Cocktails in Barcelonas ältester Cocktailbar: Boadas
Sonntag	10:00 – 13:00	Besuch des Marktes: Mercat Gotic D`Antiguitats
	14:00 – 16:00	Verlassen des Hotels, Rückfahrt zum Flughafen
	16:00 – 19:00 Uhr	Rückflug nach Berlin Tegel

Dieses Wochenende kostet knapp 2000,- DM, liegt damit im Budget und ist mit einem Programm nach den eigenen Vorlieben ausgefüllt. Wird dieser Vorschlag nach einiger

Bedenkzeit bestätigt werden sofort Flug und Hotel gebucht, ein Platz im Restaurant reserviert, usw.

Abbildung 1-1: Sagrada Familia in Barcelona

Dieses Szenario ist ein typisches Anwendungsgebiet für Agenten. Mit dem Einsatz des Agentenparadigmas soll es möglich werden, komplexe Aufgaben in nahezu unbekannten Umgebungen zu lösen und intelligente Anwendungen realisieren zu können. Ziel ist es, Systeme zu entwickeln, bei denen nicht mehr zu unterschieden ist, ob deren Aufgabe in Wirklichkeit von einem Menschen oder einem Computer erledigt wird. Spätestens mit der Verwirklichung dieser Ziele würde der Computer zu einem Menschen ebenbürtigen Partner werden. Ob dies jemals möglich sein wird oder nicht, wird die Zukunft zeigen, doch sind bis dorthin noch viele Hürden zu überwinden. Bezogen auf das obige Beispiel sind die größten Hürden die Spracherkennung, die Kooperation mit fremden Systemen und die Planung komplexer Abläufe.

Ziel dieser Arbeit soll sein, die heutigen Probleme bei der Verwirklichung eines solchen Dienstes aufzuzeigen und einen Ansatz zu realisieren, der als erster Schritt in eine solche Welt dienen könnte.

Dazu wird zunächst ein allgemeiner Überblick über das Themengebiet der Agenten gegeben und gezeigt, was Agenten sind und welche Arten von Agenten es gibt. Darauf aufbauend wird der Fokus von der Betrachtung eines Agenten auf die Präsenz mehrerer Agenten in einem System erweitert. Es wird auf zusätzliche Probleme und Anforderungen von Multiagentensystemen eingegangen.

Nachdem bekannt ist, was Agenten- und Mulitagentensysteme eigentlich sind, wird in Kapitel 4 die Problematik des obigen Beispiels wieder aufgegriffen. Es werden die Grenzen heutiger Systeme dargestellt und anhand der nun vorhandenen Kenntnisse über Agenten wird der Ansatz eines Informationsagenten entwickelt, der als Grundlage zur Verwirklichung des obigen Szenarios dienen könnte.

Da dieser Ansatz auf einem Multiagentensystem basiert, wird in Kapitel 5 zunächst ein Standard der *Foundation for Intelligent Physical Agents* (FIPA) vorgestellt, der eine Agentenplattform beschreibt, die als Basis des Multiagentensystems verwendet werden soll.

Kapitel 6 zeigt die Architektur einer Agentenplattform. Dabei wird zwischen den Anforderungen des Informationsagenten an die Plattform und des Ziels einer FIPA konformen Plattform abgewägt. Aufbauend auf dieser Architektur, wird ein Informationsagent beschrieben, der die Vorüberlegungen aus Kapitel 4 verwirklichen soll.

Anhand dieser Spezifikationen, wird in Kapitel 7 die Realisierung der Plattform und die des Informationsagenten beschrieben. Es werden unterschiedliche Realisierungsansätze

diskutiert und in einer abschließenden Betrachtung Detailprobleme gezeigt, die im Design nicht zu erkennen waren.

Ein beispielhaftes Szenario in Kapitel 8 soll zeigen, ob der verwirklichte Ansatz einsatzfähig ist und noch vorhandene Probleme aufdecken.

Zum Abschluss der Arbeit kommt es in Kapitel 9 noch einmal zu einer Zusammenfassung, in der auch mögliche Erweiterungen und die Reichweite des verwirklichten Ansatzes aufzeigt werden. Ein Ausblick in die Zukunft soll die Arbeit abrunden.

2 Agenten

Eine allgemein anerkannte Definition des Begriffs Agent zu geben, wie in diesem Kapitel erwartet wird, ist leider nicht möglich. Zu einem, weil der Begriff in einem weiten interdisziplinären Feld der Forschung und Entwicklung benutzt wird, in dem jede Richtung einem Agenten eine eigene Definition gibt, und zu anderem, weil der Begriff Agent in den letzten Jahren in Mode gekommen ist und jegliche Art von Software als Agent bezeichnet wird.

Zu den Forschungsgebieten, die den Begriff Agent prägen, sind unter anderem die künstliche Intelligenz (KI), die verteilte KI, Psychologie und Soziologie sowie die Entscheidungstheorie zu zählen, wie aus [Brenner et al. 98] zu entnehmen ist. Aber auch die Bereiche Netzwerke und Kommunikation bis hin zur Computerlinguistik zählen zu den beeinflussenden Forschungsrichtungen.

2.1 Begriffsklärung

Es gibt viele Erwartungen an ein System und unterschiedlichste Ansichten über zu erbringende Leistungen, um es als Agent bezeichnen zu dürfen. Dies macht es so schwierig den Bergriff greifbar zu machen. Ein Zitat von J.F. Bradshaw aus [Bradshaw 96, Seite 5] verdeutlicht dieses:

> "... why coming up with a once-and-for-all definition of agenthood is so difficult: one person's 'intelligent agent' is another persons 'smart object'; and today's 'smart object' is tomorrow's 'dumb program'."

In der Literatur sind zwei verschiedene Ansätze zu finden den Begriff Agent zu klären. Einerseits wird versucht durch eine Definition eine Eingrenzung zu finden, anderseits werden Eigenschaften aufgeführt, die ein System besitzen muss, um es zu den Agenten zu zählen. Da beide Ansätze ziemlich komplementär und recht interessant sind, werden sie in den folgenden Abschnitten vorgestellt.

2.1.1 Ansätze einer Definition

Sucht man in der Literatur nach Definitionen von Agenten, wird man recht schnell fündig. Doch sind diese recht vage gehalten und lassen sich schlecht miteinander vergleichen, da es, wie schon erwähnt, unterschiedlichste Ansichten und Auffassungen gibt, die auf unterschiedlichen Argumentationsansätzen und Definitionen weiterer Begriffe basieren. Da es bisher niemanden gelungen ist eine allgemein akzeptierte Definition oder Beschreibung zu erstellen, wird der Versuch einer weiteren Definition verzichtet und bereits vorhandene Beschreibungsversuche aufgeführt.

Dieses Problem sieht auch H. S. Nwana und würde am liebsten keine Definition geben. Er schreibt in [Nwana 96, Kapitel 4]:

> „When we really have to, we define an agent as referring to a component of software and/or hardware which is capable of acting exactly in order to accomplish tasks on behalf of its user".

Er zieht es vor den Begriff Agent als Art Sammelbegriff für speziellere Definitionen wie *knowbots*, *softbots*, *taskbots*, *personal assistants* oder *personal agents* zu nehmen.

Ein weiterer Ansatz unterscheidet zwischen Agenten und intelligenten bzw. autonomen Agenten. Dem Agenten wird die Bedeutung seines lateinischen Ursprungs zugewiesen. Das Wort Agent ist aus den dem Verb *agere* abgeleitet, was soviel bedeutet wie „handeln" oder „tun". So definieren:

Stan Franklin und Art Graesser in [Franklin 96] den Begriff Agent als:

> „… one who acts, or one who can act …“.

M. Wooldridge in [Wooldridge 99]:

> „An agent is a computer system that is situated in some environment, an that is capable of autonomous action in this environment in order to meet its design objectives“.

Mit diesen Definitionen wäre fast jedes Softwareprogramm und jeder Roboter ein Agent. Schon ein einfacher Heizungsthermostat erfüllt diese Bedingungen. Er regelt eigenständig die Temperatur nach seinen Einstellungen bzw. nach den Wünschen des Benutzers. Er hat einen Temperaturfühler über den die Umgebung wahrgenommen wird und kann entsprechend Handeln. Entweder stellt er die Heizung ein oder schaltet sie aus.

Von einfachen Computersystemen abheben tun sich die intelligenten/autonomen Agenten. So schreiben:

Franklin und Graesser in [Franklin 96]:

> ”An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.”

Wooldridge in [Wooldridge 99]:

> ”… an intelligent agent is one that is capable of flexible autonomous action in order to meet its design objectives”.

Wobei mit dem Begriff *flexible* die drei Eigenschaften Reaktivität, Pro-Aktivität und soziale Fähigkeit gemeint ist. Diese Begriffe werden später genauer erläutert.

Beide Definitionsversuche unterscheiden zwischen (dummen) Agenten und intelligenten oder autonomen Agenten, wobei mit letzteren die Art von Agenten gemeint ist, die hier als Agent bezeichnet werden. Wooldridge ergänzt seine Definition des intelligenten Agenten im weiteren Verlauf des Textes und weist ihm Eigenschaften zu, die erfüllt werden müssen. Diese weitere gängige Form Agenten zu beschreiben wird im nächsten Abschnitt genauer behandelt.

Folgende weitere Definitionsversuche sind in der Literatur zu finden:

IBM schreibt in einem Whitepaper [Gilbert et al. 95]:

> "Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires."

Pattie Maes schreibt in [Maes et al. 95, Seite 108]:

> "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed."

All diese Definitionen vermitteln zwar einen Eindruck von Agenten, lassen einem Neuling auf diesem Gebiet nicht deutlich erkennen worin sich Agenten erst von gängigen Softwareprodukten, Expertensystemen oder einfachen Robotern abheben. Weiteres Verständnis erhält man bei zusätzlicher Betrachtung einer weiteren Beschreibungsweise von Agenten..

2.1.2 Eigenschaften von Agenten

In diesem Ansatz wird der Begriff Agent durch zugewiesene Eigenschaften abgegrenzt. Agenten können eine Menge von Eigenschaften besitzen, doch ist stark umstritten,

welches die minimalen Eigenschaften sind, die ein System zu einem Agenten machen. So ist zum Beispiel in den Augen vieler Forscher der Verteilten KI Mobilität eine zwingende Voraussetzung eines Agenten, während es in andern Forschungsrichtungen auch nicht-mobile Agenten gibt.

In der folgenden Liste, entnommen aus [Görz et al. 00, Kapitel Software-Agenten], sollen einige Eigenschaften aufgeführt werden, die bei der Betrachtung von Agenten eine besondere Rolle spielen. Sie überschneiden sich zum Teil oder schließen sich gegenseitig aus, doch da jeder Agent nur eine Teilmenge dieser Eigenschaften aufweisen muss, ist dies nicht schlimm.

- **Andauernde Verfügbarkeit / Aktivität**
  Agenten sind über längere Zeit hinweg verfügbar, und nehmen Aufträge von Nutzern, oder anderen Agenten an. Beispiel: Suchmaschinen in Internet.
- **Interaktion mit der Umwelt**
  Agenten nehmen Informationen aus ihrer Umwelt auf (Erfassen der Umwelt, Annehmen von Aufträgen) und beeinflussen sie durch Aktionen.
- **Eigenständigkeit (Autonomie) im Handeln**
  Agenten unterliegen keiner (unmittelbaren) Steuerung und Kontrolle des Nutzers, sie handeln eigenständig im Sinne des Aufraggebers. Sie haben auch das Recht Aufträge nicht anzunehmen, zum Beispiel bei Überlastung.
- **Situiertheit**
  Das Verhalten des Agenten ist auch das Resultat von Umwelteinflüssen.
- **Reaktivität**
  Im engeren Sinne das unmittelbare Reagieren auf Umwelteinflüsse (Stimulus Response), im allgemeinen auch für die Interaktion mit der Umwelt.
- **Zielgerichtetheit**
  Agenten verfolgen Ziele, die angepasstes Handeln über lange Zeiträume erfordern können (*planning*).
- **Pro-Aktivität**
  Der Begriff ist verwandt mit Zielgerichtetheit und wird teilweise gleichwertig gebraucht. Eine spezielle Betonung liegt auf „Eigen-Initiative“ des Agenten.
- **Deliberatives Verhalten**
  Deliberatives Verhalten basiert darauf, dass die Welt in der Agenten agieren relativ vollständig bekannt ist. Dieses Wissen wird benutzt um Aktionen zu planen und auszuführen.
- **Intelligenz**
  Der Begriff kann im Sinne analoger Erklärungen in der KI so interpretiert werden: Agenten vollführen Handlungen, die beim Menschen als intelligent gelten würden.
- **Rationalität**
  Agenten treffen sinnvolle Entscheidungen in angemessener Zeit auch unter dem Aspekt beschränkt verfügbarer eigener Ressourcen.
- **Kommunikation**
  Agenten tauschen Informationen mit Menschen und anderen Agenten aus.
- **Kooperation**
  Agenten arbeiten zusammen mit Menschen und anderen Agenten, dazu gehört auch Kommunikation

- **Mobilität**
  Agenten können eigenständig auf andere Plattformen (Rechner) migrieren. Dabei wird das Agenten Programm mit seinen aktuellen Daten übertragen und setzt seine Arbeit dort fort mit der Möglichkeit des Zugriffs auf lokale Daten.
- **Mentale Modellierung**
  Für die Strukturierung von Agenten-Programmen werden mentale Begriffe (z.B. Fähigkeiten, Annahmen, Ziele, Absichten) verwendet. Sie werden in Anlehnung an menschliche Handlungsmodelle verarbeitet.
- **Wohlwollen**
  Agenten führen Aufträge im Rahmen ihrer Möglichkeiten im Sinne des Auftraggebers aus.
- **Glaubwürdigkeit**
  Die Erscheinungsform (z.B. Gestik animierter Agenten) soll sich in glaubhafter Übereinstimmung mit ihren Aktionen. Agenten sollen als Individuen mit eigenen Zielen, Bedürfnissen und Emotionen erscheinen.
- **Soziales Verhalten**
  Agenten arbeiten mit anderen Agenten und Menschen im Sinne ihrer Aufgabe zusammen. Vorbild sind soziale Strukturen der Menschen.
- **Rollenverhalten**
  Agenten füllen spezielle Rollen aus, z.B. als Auftraggeber oder Auftragnehmer.
- **Lernfähigkeit**
  Agenten können ihr Verhalten an die Umwelt anpassen, indem sie zum Beispiel ihre Fähigkeiten oder Entscheidungsprozesse geeignet variieren.

Diese Eigenschaften machen einen Agenten *intelligent* und heben sie von einfachen Objekten, Datenbanken oder Expertensystemen deutlich ab.

2.2 Topologie

Nachdem nun bekannt ist was Agenten im allgemeinen sind und welche Eigenschaften sie aufweisen, soll nun ein Überblick über existierende Agententypen gegeben werden.

Die gröbste Einteilung, die zu machen ist, ist abhängig von der Umgebung, in der ein Agent agiert. Dies kann die reale Welt, eine virtuelle Welt innerhalb von Computern und Netzwerken sein.

2.2.1 Intelligente Roboter

Agenten, die in unserer, der physischen Welt agieren, werden zumeist intelligente Roboter genannt. Sie erledigen Aufgaben, wie Liefertätigkeiten, Inspektionstätigkeiten, oder bauen aus einfachen Gegenständen komplexe Objekte zusammen. Sie werden in Situationen eingesetzt, die für den Menschen zu gefährlich sind, (Schlagworte: *Tiefsee*, *Mienenfelder*, *Atomkraftwerke*), oder übernehmen Aufgaben, die bisher vom Menschen ausgeführt wurden, aber vom intelligenten Roboter präziser und billiger erledigt werden können. Intelligente Roboter werden in stationäre-, mobile- und Manipulationsroboter eingeteilt, wobei die Krönung der Manipulationsroboter humanuide Roboter sind, die menschenähnlich aussehen, sich auf zwei Beinen fortbewegen und zwei Arme zur Manipulation der Umwelt haben.

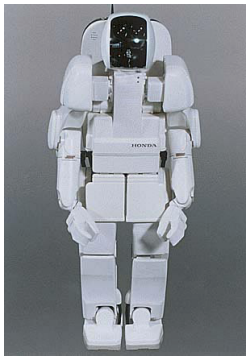


Abbildung 2-1: Honda P3 Roboter

Als Beispiele für existierende intelligente Roboter wäre der Museumsführer *RHINO* [3] zu nennen, der von der Uni Bonn in Zusammenarbeit mit der Carnegie Mellon Universität entstanden ist und Museumsbesucher durch ein Museum, von Exponat zu Exponat führt. Der Rover *Pathfinder* [4] wurde von der NASA entwickelt und zum Mars geschickt, um Fotos zu machen und Proben von der Oberfläche zu analysieren. Beim *RoboCup* [5], der Fußballweltmeisterschaft der Roboter, spielen intelligente Roboter gegeneinander Fußball.

Da diese Arbeit über Softwareagenten handelt, werden intelligente Roboter nur kurz erwähnt. Weitere Informationen können in meiner Studienarbeit unter [Jung 00] gefunden werden.

2.2.2 Softwareagenten

Agieren Agenten in den virtuellen Welten von Computern und Netzwerken, werden sie Softwareagenten genannt. Eine einheitliche Topologie von Softwareagenten gibt es leider noch nicht, ähnlich wie es auch noch keine einheitliche Definition von Agenten gibt. Deshalb sollen an dieser Stelle einige Topologien bekannter Forscher auf diesem Gebiet vorgestellt werden.

2.2.2.1 Topologie nach biologischem Modell

Franklin und Graesser halten sich in ihrer Klassifizierung in [Franklin 96] an das biologische Modell. Dieses Modell teilt Lebewesen in Art, Stamm, Klasse, Familie, Gattung und Spezies ein. Anhand dieses Modells kommen sie zu folgender Initialeinteilung:

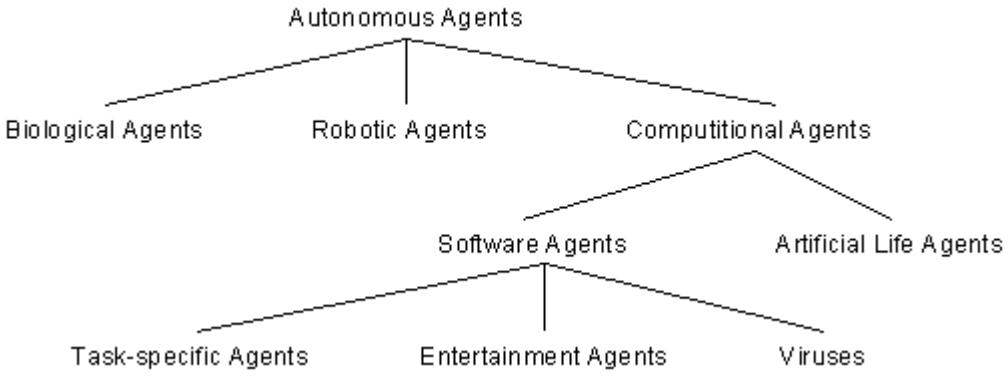


Abbildung 2-2: Topologie nach Franklin und Graesser

Unterhalb dieser Struktur schlagen sie vor Softwareagenten nach ihren Eigenschaften zu klassifizieren und stellen Eigenschaften, wie planend vs. nicht-planend, lernend vs. nicht

lernend, mobil vs. nicht-mobil usw. in einer Art Matrixorganisation gegenüber. Daraus ergeben sich dann Typen wie planende Agenten, Informations-Agenten usw.

2.2.2.2 Topologie durch Dimensionen

In dem Ansatz von H. S. Nwana aus [Nwana 96] werden Agententypen durch Zugehörigkeit in mehreren Dimensionen zu klassifiziert. Er schlägt folgende Dimensionen vor:

- **Mobilität**
  Agenten können anhand der Fähigkeit eingestuft werden, ob sie sich in Netzwerken bewegen können. Daraus lassen sich statische und mobile Agenten ableiten.
- **Reaktivität / Deliberativität**
  Abhängig, ob das Handeln auf eine interne symbolische Darstellung der Welt, oder auf aktuelle Informationen der Sensordaten beruht, sind Agenten deliberativ oder reaktiv.
- **Primäre Attribute**
  Klassifizierung anhand primärer Eigenschaften oder Ideale. Diese sind zumindest Kooperation, Lernfähigkeit und Autonomie. Aus diesen primären Eigenschafen leiten sich kooperierende -, kooperierend-lernende -, Interface- und kluge (smart) Agenten ab, wie in Abbildung 2-3 zu sehen.

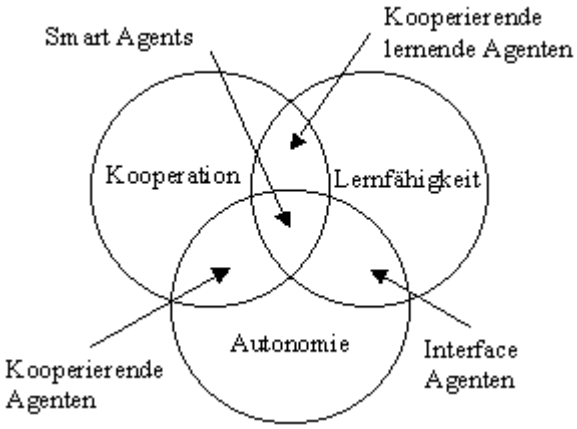


Abbildung 2-3: Topologie nach Nwana

- **Rollen**
  Agenten werden nach ihrer Rolle klassifiziert. Zum Beispiel Informations- oder Internetagenten, die das Internet nach Informationen durchsuchen.
- **Hybride Agenten**
  Diese Agenten kombinieren verschiedene Ansätze.
- **Sekundäre Attribute**
  Vielseitigkeit, Aufrichtigkeit, Glaubwürdigkeit ...

Die Kombinationen dieser 6 Dimensionen lässt eine riesige Anzahl von Agententypen zu. So kann es zum Beispiel stationäre deliberative Interfaceagenten geben, aber auch mobile deliberative Informationsagenten. Um die Übersicht zu behalten, reduziert Nwana all diese möglichen Typen zu einer Liste aus 7 Typen, die die meisten gängigen existierenden Agententypen abdecken. Diese sind:

- Kooperierende Agenten
  Die Charakteristik dieser Agenten schließt Autonomie, Kooperation (mit anderen Agenten), soziale Fähigkeit und Proaktivität ein. Weiter sollten sie rational und autonom in offenen und zeitabhängigen Multiagentenumgebungen arbeiten können. Zumeist sind sie statisch.

- Interface Agenten
  Ein Interface Agent arbeitet autonom und ist lernfähig. Er spiegelt die Metapher eines persönlichen Assistenten wieder, der zusammen mit seinem Benutzer an den selben Aufgaben arbeitet. Ein wichtiger Unterschied zu kooperierenden Agenten ist, dass er lediglich mit seinen Benutzer kooperiert, nicht aber mit anderen Agenten. Er lernt aus dessen Handlungen und ist fähig den Benutzer beim Einarbeiten in ein Arbeitsblatt oder eine Anwendung zu unterstützen. Er überwacht dessen Handlungen lernt neue Tastenkombinationen und kann Tipps zur effizientren Ausführung einer Aktion geben.
- Mobile Agenten
  Mobile Agenten sind Softwareeinheiten, die sich durch Netzwerke wie dem Internet bewegen, sich auf unterschiedlichen Computern ausführen, Informationen nach den Vorgaben den Benutzers sammeln und nach definierter Zeit oder Beendigung der Aufgabe zum Benutzer zurückkehren. Die Einsatzmöglichkeiten reichen von Flugreservierungen bis zu Netzwerkmanagement. Nach H. S. Nwana ist Mobilität keine Eigenschaft von Agenten, sie werden den Agenten zugeordnet, weil sie kooperieren und autonom agieren, wenn auch anders als kooperierende Agenten.
- Informations- und Internetagenten
  Informationsagenten existieren wegen der Notwendigkeit nach Werkzeugen, die uns Helfen die rasant wachsende Informationsflut im Internet zu verwalten. Sie verwalten, sammeln und passen Informationen nach den Bedürfnissen des Benutzers an. Man kann argumentieren, dass auch andere Agententypen Informationsagenten sind. Diese Überschneidungen sind unvermeidbar, da Informationsagenten danach definiert sind was sie tun, Interface- oder kooperierende Agenten danach eingeordnet sind was sie sind.
- Reaktive Agenten
  Eine weitere Klasse von Agenten sind reaktive Agenten, weil sie nicht über interne, symbolische Informationen über ihre Umgebung verfügen, stattdessen nach *stimulus-respone* Manier direkt auf dem Zustand der Umgebung reagieren, in der sie sich befinden. Sie sind relativ einfach und interagieren mit anderen Agenten auf primitiven Weisen. Dennoch ermöglichen sie recht komplexe Verhaltenstrukturen, wenn das Verhalten aller Agenten global betrachtet wird.
- Hybride Agenten
  Von einen hybriden Agenten wird gesprochen, wenn ein Agent in mindestens zwei der obigen fünf Klassen entspricht.
- Heterogene Agenten
  Diese Klasse der Agenten ist eine Erweiterung der kooperierenden Agenten. Heterogene Agenten können mit verschiedenartigen Agenten zusammenarbeiten und ermöglichen Wiederverwendung und Einbindung existierender Agenten.
- Smart Agents
  Smart Agents existieren zur Zeit noch nicht. Aus Abbildung 2-3 wird deutlich, dass ein Smart Agent in Vereinigung aller primären Attribute autonom, kooperativ und lernfähig ist.

2.3 Arbeitsdefinition eines intelligenten Agenten

Nachdem der Begriff Agent abgegrenzt und eine Übersicht über Agententypen vorgestellt wurde, soll nun anhand dieser Informationen eine Arbeitsdefinition gegeben werden. Diese soll im weiteren Verlauf dieser Arbeit gelten, wenn von Agenten gesprochen wird.

Ein Agent in dieser Arbeit ist am ehesten in die Gruppe der kooperierenden Agenten nach Definition von H. S. Nwana einzuordnen. Ein solcher Agent kooperiert mit anderen Agenten, um den Anforderungen seines Benutzers gerecht zu werden. Die Kooperationspartner werden autonom ausfindig gemacht und kontaktiert. Mobilität soll in dieser Arbeit ausgeschlossen werden.

3 Multiagentensysteme

Während der bisherigen Vorstellung von Agenten wurden bereits Eigenschaften, wie Kooperation und Kommunikation, und Klassen von Agenten vorgestellt, die die Existenz mehrerer Agenten in einem System voraussetzen. Existieren in einer realen oder virtuellen Umgebung mehrere Agenten, die sich gegenseitig beeinflussen, so wird von einem Multiagentensystem (MAS) gesprochen. Multiagentensysteme stammen ursprünglich aus der Forschung der *Verteilten Künstlichen Intelligenz*. Historisch gesehen war die VKI in die zwei Hauptstränge *Distributed Problem Solving* und *Multi-Agent Systems* geteilt. In den letzten Jahren hat der Begriff Multiagentensystem eine etwas allgemeinere Bedeutung bekommen und wird für alle Typen von Systemen benutzt, die mehrere autonome Komponenten miteinander verbinden, und umschließt auch das Lösen von verteilten Problemen. So schreibt E. Durfee in [Durfee et al. 89] ein MAS als:

> "… a loosely-coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities."

Die Vorteile eines MAS sind das Lösen komplexer Probleme, die ein einzelner Agent nicht lösen kann, die Erhöhung der Geschwindigkeit und Zuverlässigkeit solcher Systeme und die Toleranz von zweifelhaften Informationen. Ein weiterer Vorteil aus Sicht der Sicherheit ist, dass Agenten lediglich die Daten kennen, die zur Ausführung ihrer Teilaufgabe benötigt werden, alle anderen Daten bleiben ihnen unzugänglich.

Die Eigenschaften eines MAS nach [Jennings et al. 98] sind:

- Jeder Agent hat ungenügende Informationen und Fähigkeiten um das gesamte Problem zu lösen und so eine eingeschränkte Betrachtung.
- Keine globale Kontrollinstanz.
- Daten werden dezentral gehalten.
- Verarbeitung ist asynchron.

Damit Agenten in einem MAS kooperieren können, werden einige Infrastrukturen vorausgesetzt. Agenten kommunizieren durch den Versand von Nachrichten. Diese müssen einem Format entsprechen, das von den Empfängern verstanden wird. Ein MAS muss den physischen Austausch der Nachrichten ermöglichen und sollte über Möglichkeiten verfügen, Agenten des Systems ausfindig zu machen. Diese Infrastrukturen werden gewöhnlich durch eine Agentenplattform bereitgestellt, auf der Agenten dynamisch existieren. Sie bedienen sich der Dienste der Plattform.

3.1 Agentenkommunikation

Zumeist wird die Kommunikation von Agenten als wechselseitiges Senden und Empfangen von Nachrichten verstanden. Eine Nachrichtenfolge wird, wie in der menschlichen Kommunikation, Dialog genannt, für dessen gelingen eine gemeinsame Sprache und einige weitere Regeln bezüglich des Nachrichtenaustauschs (z.B. Protokolle) eingehalten werden müssen.

Im Gegensatz zu herkömmlichen verteilten Systemen, die zumeist nur auf syntaktischer Ebene kommunizieren (z.B. über *remote procedure call* (RPC), wie in CORBA angeboten), kommunizieren Agenten gewöhnlich auf einer hohen semantischen Ebene. Dies bedeutet nicht, dass Agenten keine RPC Funktionen nutzen, sondern diese eher als technische Übertagungsart oder Medium dienen. Der Inhalt einer Nachricht wird in einer Sprache dargestellt, die den hohen semantischen Anforderungen entsprechen kann.. Eine

Sprache die dies tut ist die menschliche Sprache, jedoch ist sie zu komplex und kontextsensitiv um in naher Zukunft von Computern verstanden zu werden. Die Linguistik beschäftigt sich mit dem theoretischen Aufbau von Sprachen und hat einige Ansätze hervorgebracht, die auch für eine Agentensprache interessant sind. Die Speech Act Theorie ist ein für Agentensprachen häufig verwendeter Ansatz, auf dem unter anderem die bekannte KQML und FIPA-ACL (werden beide später vorgestellt) basieren.

3.1.1 Speech Act Theorie

Der Oxforder Philosoph John Langshaw Austin hat mit seinen Vorlesungen 1955, die erst nach seinem Tode in [Austin 75] veröffentlicht wurden, zu einen umdenken der Philosophen und Sprachwissenschaftler über die bisherigen Vorstellungen über Sprache geführt. Er hat untersucht inwiefern mit dem Äußern von Sätzen Handlungen vollzogen werden. Denn mit Sprache tauscht der Mensch mehr als nur Informationen aus. Der eigentliche Inhalt der Nachricht kann sich all zu oft vom der Zweck der Nachricht unterscheiden. Die häufig gestellte Frage „Können sie mir sagen, wie spät es ist?“ ist in den seltensten Fällen mit einem „Ja“ zufriedenstellend beantwortet.

Austin unterscheidet in seinen Vorlesungen zunächst zwischen performativen und konstativen Äußerungen. Konstative Äußerungen stellen Aussagen oder Behauptungen auf, die wahr oder falsch sein können und dienen zur Beschreibung der Welt. Sätze wie: „Ich habe heute Geburtstag“ oder „Ich reise mit der Bahn“ sind konstativ. Performative Äußerungen sind sprachliche Akte, mit deren Hilfe bestimmte Handlungen vollzogen werden. Sie haben keinen beschreibenden Charakter und sind weder wahr noch falsch, sie stellen eine Handlung dar, die der Sprecher gerade ausführt oder ausführen wird. Demnach sind Sätze wie: „Ich gratuliere Dir, zum Geburtstag.“ oder „Ich ruf dich morgen an“ als performativ einzuordnen. Austin legt eine Reihe von performativen Verben fest, die er in die Klassen direktiv (bitten, befehlen, auffordern, anordnen und beauftragen), deklarativ (ernennen, verurteilen, kündigen, taufen, exkommunizieren) und expressiv (danken, gratulieren, protestieren, loben und begrüßen) aufteilt.

Später hat Austin die Unterscheidung von performativen und konstativen Äußerungen zu Gunsten von lokutionären, illokutionären und perlokutionären Akten überdacht, die in jeder sprachlichen Äußerung wiederzufinden sind.

- **Lokutionäre Akt**
  Der lokutionäre Akt bezeichnet den Akt des Aussprechens eines sinnvollen Satzes. Dazu gehört das Erzeugen der Geräusche und Laute, das Einhalten der Grammatik und die Wahl der Wörter, um über etwas zu reden (phonetischer, phatischer und rhetischer Akt).
- **Illokutionäre Akt**
  Der illokutionäre Akt ist Hauptgegenstand der Sprechakttheorie. Hier wird das gesagte in einer bestimmten Weise verwendet, z.B. als Warnung, Versprechen, Frage, Behauptung, Rat oder Befehl. Er kennzeichnet den Handlungscharakter der Äußerung.
- **Perlokutionäre Akt**
  Der perlokutionäre Akt bezeichnet die Handlung, die ein Sprecher durch seine Äußerung auf den Zuhörer erzielen will. Zum Beispiel wird auf Grund der Warnung (illokutionärer Akt) „Der Hund ist bissig“ beabsichtigt, dass der Zuhörer den Garten nicht betritt (perlokutionärer Akt).

Oftmals kann der Zuhörer den Sinn einer Nachricht erst richtig verstehen, wenn der illokutionäre Akt herausgefunden wurde. Dies stellt sich manchmal als schwer heraus und ist ein Grund für viele Verständnisfehler des Menschen. Wird der Satz „Dahinten

kommt eine scharfe Kurve“ nicht als Warnung, sondern lediglich als Information interpretiert, kann es zu einer gefährlichen Situation kommen.

3.1.2 Klassifizierung von Nachrichten

Auf Grund des illukotionären Aktes hat Austin Nachrichten in verschiedene Typen klassifiziert, die auf die Klassifizierung performativer Äußerungen basieren. Eine weiter verbreitete Klassifizierung stammt jedoch von seinem Schüler John R. Searle. Er hat Austins ursprüngliche Klassifizierung ergänzt und kommt in [Searle 76] zu folgendem Ergebnis:

- **representatives / asserives**
  Aussagen des Sprechers über die ihm bekannte Welt, wie: feststellen, melden, berichten oder informieren.
- **directies**
  Anforderungen oder Erwartungen des Sprechers an den Empfänger, wie fragen, befehlen, anweisen oder verbieten.
- **commissives**
  Äußerungen, die den Sprecher zu einer Tat in der Zukunft verpflichten, wie versprechen, geloben, garantieren oder schwören.
- **expressives**
  Äußerungen, die die Gefühle des Senders beschreiben, wie danken, gratulieren, beschweren oder grüßen.
- **declaratives**
  Äußerung, die einen Zustand, den sie beschreiben selbst herstellen, wie ernennen, sack oder nominieren.

3.1.3 Einfluss auf Agentenkommunikation

Diese Klassifizierung kann für eine Agentenkommunikationssprache genutzt werden. Beschreibt eine Nachricht den illukotionären Akt, der mit ihr ausgedrückt werden soll, können keine Missverständnisse über ihre Absicht auftreten. Zusätzlich kann der Empfänger die Nachricht einordnen, ohne ihren genauen Inhalt zu kennen. Dies kann zum Beispiel für Protokolle genutzt werden, die festgelegte Nachrichtentypen in einer bestimmten Reihenfolge erwarten.

3.2 Entwicklungen und Standardisierungen

Im Bereich der Multiagentensysteme und der Agentenkommunikation gibt es verschiedenste Entwicklungen. Einen kurzen Überblick geben die nächsten Abschnitte.

3.2.1 KQML / KIF

Die *Knowledge Query and Manipulation Language* (KQML) [6] ist eine Sprache und Protokoll, erschafft um Informationen auszutauschen. Sie kann als Agenten-Kommunikationssprache genutzt werden, um Interaktionen zwischen Agenten zu erzeugen. KQML basiert auf der soeben vorgestellten Speech Act Theorie und definiert eine Menge zulässiger performativer Äußerungen, mit denen einer Nachricht eine spezielle Handlung zuweisen wird. Der semantische Inhalt einer KQML Nachricht muss einem wohlgeformten Satz einer bestimmten Sprache entsprechen. Dies kann z.B. KIF sein. In welcher Sprache der Inhalt beschrieben ist, wird in einer KQML Nachricht mitabgebildet.

Das *Knowledge Interchange Format* (KIF) [7] wurde in der Stanford Universität entwickelt und liegt der ANSI als Standardisierungsvorschlag vor. Sie ist eine sogenannte *Content Language*, eine Sprache mit der Wissen zwischen Systemen mit

verschieden Wissensrepräsentationen ausgetauscht werden kann. KIF erlaubt beliebige Ausdrücke der Prädikatenlogik erster Stufe und die Definition von Objekten, Funktionen und Relationen abzubilden. KIF hat eine deklarative Semantik, das heißt es sind keine Bezüge zu einem Interpreter nötig, um Sätze der Sprache zu verstehen. Zum Verständnis der Semantik eines KIF Ausdrucks wird allerdings eine Ontologie benötigt.

Zur Agentenkommunikation werden beide Sprachen häufig gemeinsam verwendet. KQML beschreibt das Format einer Nachricht, die zwischen Agenten ausgetauscht wird, die Information dieser Nachricht wird in KIF ausgedrückt.

3.2.2 FIPA

Die *Foundation for Intelligent Physical Agents* (FIPA) definiert einen ausgiebigen Standard für heterogene und interagierende Agenten und agentenbasierte Systeme. Es wird neben Kommunikationssprachen und Interaktionsprotokollen für Agenten auch eine Agentenplattform spezifiziert, die FIPA Agenten verwaltet und ihnen Dienste anbietet. Der FIPA Standard scheint alle Anforderungen an ein Multiagentensystem abzudecken, die in Kapitel 3 erwähnt wurden. Die FIPA definiert lediglich Standards, die öffentlich zugänglich sind und von verschieden Organisationen implementiert werden. Somit existieren unterschiedlichste Implementierungen des Standards auf unterschiedlichsten Systemen, deren Agenten miteinander kooperieren können (sollen).

Auch die OMG unternimmt Aktivitäten im bereich der Multiagentensysteme. Diese liegen jedoch dem FIPA Standard zugrunde. Lediglich deren Tätigkeiten im Bereich mobiler Agenten basieren auf einem eigenen Ansatz dem *Mobile Agent System Interoperability Facility* (MASIF).

3.2.3 OAA

Die *Open Agent Architektur* (OAA) ist ein weiter vollständiger Ansatz zur Entwicklung von Multiagentensystemen. In ihr werden, ähnlich des FIPA Standards Kommunikationssprachen und eine Agentenplattform, hier *Facilitator* genannt, definiert. Die Architektur des *Facilitator* ist etwas anderes als die der FIPA Plattform, die Dienste, die von ihm angeboten werden, sind jedoch ähnlich. Ein erwähnenswerter Unterschied zur FIPA ist, dass die OAA zum Nachrichtenaustausch über Trigger verfügt, die gefeuert werden, wenn angegebene Bedingungen in der Systemumgebung erfüllt sind. Eine FIPA Plattform und deren Agenten sind lediglich Nachrichtengetrieben.

Die OAA ist ein Multiagentensystem, dass bereits implementiert ist. Der Quellcode ist frei verfügbar. Der *Facilitator* kann lediglich auf Windows und Unix Systemen eingesetzt werden, es gibt aber Schnittstellen in diversesten Programmiersprachen gegen die eigene Agenten implementiert werden können. Somit ist es möglich, dass Agenten auch auf anderen Systemen existieren, da lediglich über Nachrichten mit dem *Facilitator* kommuniziert wird.

4 Ansatz eines Informationsagenten

Nachdem nun ein Überblick über Agenten- und Multiagentensysteme gegeben wurde, komme ich auf das Szenario der Einleitung zurück. Jeder wird mir zustimmen, wenn ich behaupte, dass dieses Beispiel heutzutage noch sehr utopisch klingt. Warum dies so ist und wie ein erster Schritt zur Verwirklichung dieses Szenarios aussehen könnte, soll in diesem Kapitel gezeigt werden.

Dass dieses Beispiel so unrealistisch wirkt, mag an Technologien liegen, die noch lange nicht reif sind in einem so komplexen Feld eingesetzt zu werden. Da wäre zum einen der Personal Digital Assistant, der die genauen Vorlieben des Benutzers kennt und als Aktivität eine Reise auswählt und plant. Benutzerprofile werden auch heute schon erstellt, bemerkt oder unbemerkt. So zeigt ein gängiges Windows Betriebssystem nur noch die Einträge eines Menüs die relativ häufig verwendet werden und blendet alle anderen aus. Komplexe Entscheidungen treffen zu können, wie die Wahl einer Reise, die von vielen unterschiedlichen Informationen, wie dem Benutzerprofil, dem aktuellen Budget und der Anzahl der freien Tage abhängt, ist jedoch sehr schwer. Entsprechend problematisch ist auch die Planung der Reise, da ähnlich komplexe Entscheidungen tu treffen sind, wie das Reiseziel und die Wahl der Aktivitäten, die wiederum abhängig vom örtlichen Angebot, den Vorlieben des Benutzers und vom vorhandenen Budget sind.

Selbst wenn der Bereich *Planning*, zu dem es in der KI unterschiedlichste Ansätze gibt (siehe [Görz et al. 00]) außer Betracht gelassen wird, werden weitere, neue Technologien in diesem Beispiel verwendet. Die Informationen über Hotels und Restaurants in einer beliebigen Stadt werden dem PDA vor der Anfrage des Benutzers nicht bekannt gewesen sein. Die Wahl des Reiseziels hätte auf nahezu jede Stadt der Welt treffen können, und es hätte auch zu einen ganz anderen Vorschlag für die Gestaltung des freien Wochenendes kommen können. Für die Planungen wird der PDA die Informationen dynamisch, nach Bedarf, z.B. über das Internet, angefordert haben. Dies ist allerdings nur möglich, solange ein oder mehrere Informationsanbieter befragt werden können. Die Kommunikation zwischen ihnen muss es ermöglichen, sehr präzise Informationen anzufordern, die vom Empfänger verstanden und weiterverarbeitet werden können. Ansonsten wäre das Buchen eines Fluges und darauf basierend das Reservieren eines Hotelzimmers in der selben Stadt kaum möglich.

Abstrahiert man den letzten Ansatz, erkennt man zwei wichtige Eigenschaften von Agenten, nämlich Kooperation und Autonomie. Mit etwas anderen Worten ausgedrückt kontaktiert der PDA den oder die Informationsanbieter eigenständig und diese kooperieren miteinander, um die ihnen vorgegebenen Ziele zu erreichen. Auf der einen Seite ist das Ziel die Planung einer Reise, auf der anderen Seite sollen Informationen angeboten, eventuell sogar Waren oder Dienstleistungen verkauft werden. Demnach ist das Beispiel aus der Einleitung ein typisches Szenario für eine Multiagentenumgebung, die in diesem Fall aus Informationsagenten und einer Art Planungs- oder Freizeitagenten besteht. Das Besondere daran ist, dass die Agenten miteinander kooperieren können, obwohl sie sich nicht genau kennen, und nicht in einem, sondern in vielen verschiedenen, durch das Internet lose miteinander verbundenen Systemen existieren.

Bevor es jedoch Agenten gibt, die eigenständig Waren verkaufen oder komplexe Planungen durchführen können, müssen die dazu benötigten Informationen bereit gestellt werden. Deshalb soll der weitere Fokus zunächst auf die Anforderungen eines solchen Informationsdienstes gelegt werden. Heutige Informationsagenten können leider nicht als Basis komplexer Dienste genutzt werden. Die Gründe dafür sollen im folgenden Abschnitt beschrieben werden, bevor Überlegungen gemacht werden, wie

Informationsagenten aussehen könnten mit denen obiges Szenario verwirklicht werden könnte.

4.1 Probleme heutiger Informationsagenten

In der Literatur werden häufig Internet-Suchmaschinen, wie Lycos, als Informationsagenten bezeichnet, da sie eigenständig das Internet durchsuchen und auf dieser Basis dem Benutzer Informationen anbieten können. Meiner Meinung nach geschieht dies jedoch auf einer wenig intelligenten, sondern eher algorithmischen Ebene, weshalb ich den Begriff Agent in diesem Umfeld nicht verwenden würde. Doch auf eine Diskussion über deren Zugehörigkeit zu den Agenten möchte ich an dieser Stelle verzichten, ich denke auf die Problematik der Abgrenzung wurde in Kapitel 2 ausreichend hingewiesen.

Informationsagenten, wie Spider oder Webcrawler, basieren auf einen Ansatz, bei dem es nicht notwendig ist die Semantik, der angebotenen Informationen zu verstehen. Stark vereinfacht gesehen werden Schlagworte in HTML Seiten in riesigen Tabellen der korrespondierenden Internetadresse zugeordnet werden. Einen genaueren Einblick in die Funktionsweise von Suchmaschinen kann im Internet unter dem Schlagwort: *Information Retrieval* gefunden werden. HTML ist lediglich eine Sprache, durch die der Aufbau, jedoch nicht der Inhalt einer Seite beschrieben werden kann. Der Inhalt einer solchen HTML Seite ist zumeist in menschlicher Sprache beschrieben oder in solcher Struktur dargestellt, dass er nur durch menschliche Intelligenz und Intuition verstanden werden kann. Agenten haben leider noch nicht die Fähigkeit, solche Informationen interpretieren zu können. Dazu müssten die Informationen in einer speziellen, dem Agenten bekannten und verständlichen Sprache dargestellt werden.

Durch den Verzicht auf das Verständnis der angebotenen Informationen sind heutige Informationsagenten in der Lage, zu jedem beliebigen Thema Informationen anzubieten. Ein weiterer Nachteil dieses Ansatzes ist die daraus resultierende, starke Einschränkung der Suchanfragemöglichkeiten. Etwas überspitzt beschrieben ist es zur Zeit nur möglich eine Suche durch wichtige Schlagworte einzuschränken, die durch UND und ODER Beziehungen verknüpft sind. Dies hat zur Folge, dass eine Suchanfrage nur ungenau eingeschränkt werden kann und die Ergebnisse oftmals nicht die eigentlich gesuchte Information enthalten. Jeder hat schon einmal einige Suchbegriffe in eine Suchmaschine eingegeben und als Antwort mehrere tausend Treffer erhalten, von denen nur die wenigsten auf eine Seite mit der gewünschten Information verwiesen haben. Komplexe Anfragen zu einem speziellen Thema sind mit dieser ungenauen Art der Katalogisierung und Beschreibung der Ressourcen nicht möglich. Eine Anfrage nach den Mitgliedern der Mannschaft des Fußballweltmeisters von 1986 (Argentinien) lässt sich nicht formulieren.

Ein weiteres Problem ist, dass der gesamte Inhalt des Internets katalogisiert wird. Bei einem so rasant wachsenden Informationsangebot, wie in den letzten Jahren, ist es schon heute unmöglich das gesamte Internet vollständig und aktuell zu verwalten. Dies bedeutet für den Benutzer einen deutlichen Qualitätsverlust, da ihm nicht sofort alle Informationsquellen zugänglich gemacht werden und veraltete, nicht mehr vorhandene Informationsquellen noch immer angeboten werden. Zur Verbesserung des Suchergebnisses leiten einige dieser Informationsagenten (Meta-Suchmaschine) eine Suchanfrage an andere Informationsagenten weiter und präsentieren dem Benutzer eine Kombination aller Ergebnisse. Diese Strategie erhöht zwar die Wahrscheinlichkeit die passende Information zu finden, jedoch werden weiterhin nicht aktuelle Informationen und nicht funktionierende Links angeboten.

4.2 Ein anderer Ansatz von Informationsagenten

Da heutige Suchmaschinen darauf ausgelegt sind von Menschen genutzt zu werden, ist es ihnen nicht möglich, Informationen in einer Form anzubieten, die von Agenten für ihre Ziele (z.B. Planung, Einkauf) genutzt werden können. Es wird also ein Ansatz gesucht, in dem Informationsagenten Informationen so darstellen können, dass sie auch von nicht menschlichen Benutzern inhaltlich verstanden werden.

Informationen zu jeglichen Themenbereichen dieser Welt sind zur Zeit nur mit menschlichen Sprachen beschreibbar, deren Syntax und Semantik noch nicht von Computerprogrammen verstanden werden kann. Ein etwas anderer Ansatz könnte auf Informationsagenten basieren, die lediglich Informationen zu einem speziellen Thema anbieten, dessen Umfang mit einer maschinell verständlichen Sprache (siehe Kapitel 7.1) abzubilden ist. Sie wären Experten auf diesem Themengebiet und böten einen speziellen Informationsdienst an. Zum Beispiel könnte ein Hotelinformationsdienst Auskunft über Lage und Ausstattung von Hotels geben, oder ein Restaurantdienst kulinarische Informationen geben. Erst durch Fokussierung auf einen eng begrenzten Themenbereich und das Anbieten spezieller Dienste ist es möglich präzise Anfragen zu definieren und die angeforderte Informationen in einer Sprache zu beschreiben, die von anderen Agenten verstanden wird.

Werden Informationsagenten auf spezielle Themenbereiche beschränkt, schließt sich die Frage an, inwiefern es sinnvoll und überhaupt möglich ist einen Agenten zu erzeugen, der über alle Informationen dieses Themas verfügt. Könnte es auch genügen einen Agenten nur über eine Teilmenge aller Informationen verfügen zu lassen? Die Frage lässt sich schnell beantworten, wenn man bedenkt, dass es fast unmöglich ist alle Informationen zu einem Themenbereich zu sammeln und einem Anbieter zu Verfügung stellen zu lassen. Wesentlich einfacher ist es allerdings einen Informationsagenten über die Daten aller Hotels oder Restaurants einer Stadt verfügen zu lassen. Gäbe es eine Vielzahl von Informationsanbietern wäre es jedoch für den Benutzer sehr schwer den richtigen Anbieter ausfindig zumachen.

Wie schon erwähnt, sollen diese Informationsagenten die angebotene Information semantisch verstehen. Damit sind sie in der Lage ihren eigenen Dienst, angeboten durch andere Agenten, nutzen zu können. Wie die folgende Abbildung verdeutlicht, kann somit eigenständig ein anderer Anbieter befragt werden, falls die angeforderte Information nicht in der eigenen Wissensbasis enthalten ist.

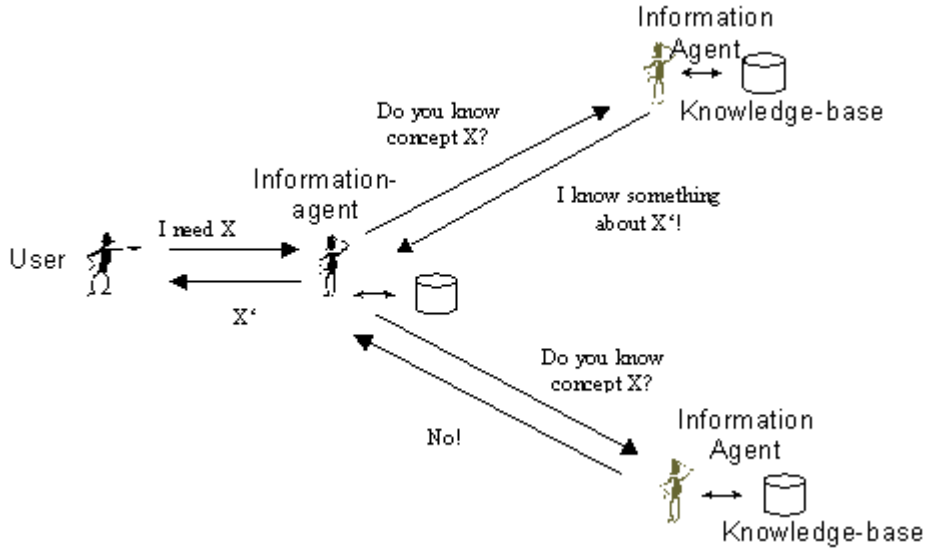


Abbildung 4-1: Kooperierende Informationsagenten

Natürlich kann ein solcher Informationsagent nicht mit der Vielzahl der angebotenen Informationen heutiger Informationsagenten mithalten, doch bringt die Spezialisierung den Vorteil, dass Dienste mit detailliertem und weiterverarbeitbarem Informationsangebot angeboten werden können. Ein weiterer Vorteil ist, dass die Agenten zur Zeit der Anfrage die Information in ihrer Wissensbasis suchen. Damit ist es möglich auch tagesaktuelle Dienste anbieten zu können, wie Hotelinformationsdienste, die Auskunft über die Anzahl der freien Betten geben können, oder TV Programminformationsdienste, die das aktuelle Fernsehprogramm kennen. Es muss jedoch bei diesem Ansatz darauf hingewiesen werden, dass spezielles Wissen über einen Dienst benötigt wird, bevor er genutzt werden kann. Die Informationsanbieter untereinander und der Benutzer eines Dienstes müssen sich auf die Darstellung der Anfrage und der Information einigen, um einen funktionierenden interoperablen Dienst anbieten zu können.

4.3 Skizze des Informationsdienstes

Verdeutlicht durch die nächsten Abbildungen, könnte ein solcher Dienst durch folgenden Ablauf aussehen realisiert werden:

Bevor ein Agent Informationen anbieten kann, muss er seinen Dienst der Pattform bekannt machen. Bekommt nun ein Agent eine Anfrage durch einen Benutzer, schaut er in seiner eigenen Wissensbasis nach, ob die Anfrage beantwortet werden kann. Falls ja, wird die Information an dem Benutzer gesendet, ansonsten werden über die Plattform andere Agenten ausfindig gemacht, die einen ähnlichen Informationsdienst anbieten und möglicherweise die gesuchte Informationen anbieten können.

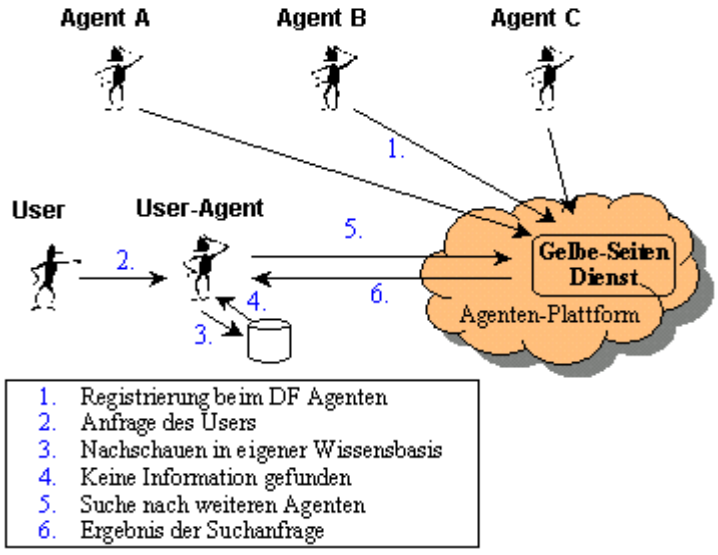


Abbildung 4-2: Auffinden potentieller Informationsanbieter

Diese werden zunächst einmal nach Vorschlägen gefragt. Der anfragende Agent wartet solange, bis Vorschläge von allen Agenten eingetroffen sind und wählt aus diesen den Besten aus.

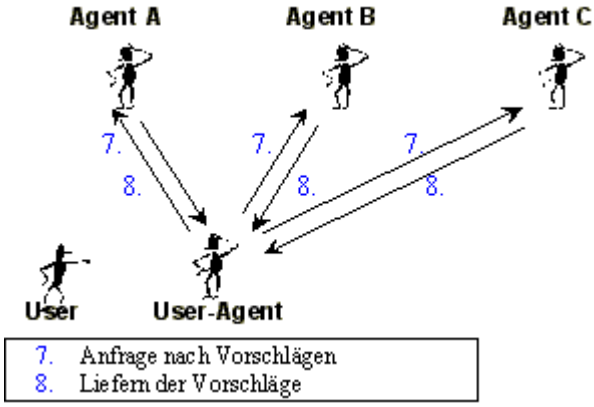


Abbildung 4-3: Anfrage nach Vorschlägen

Dem Agenten, dessen Vorschlag ausgewählt wurde, wird daraufhin mitgeteilt, dass sein Vorschlag akzeptiert wurde, alle anderen werden informiert, dass ihre Vorschläge abgelehnt wurden. Für sie ist die Konversation nun beendet. Der Agent, der den akzeptierten Vorschlag gemacht hat, schickt daraufhin die vollständige Information an den anfragenden Agenten.

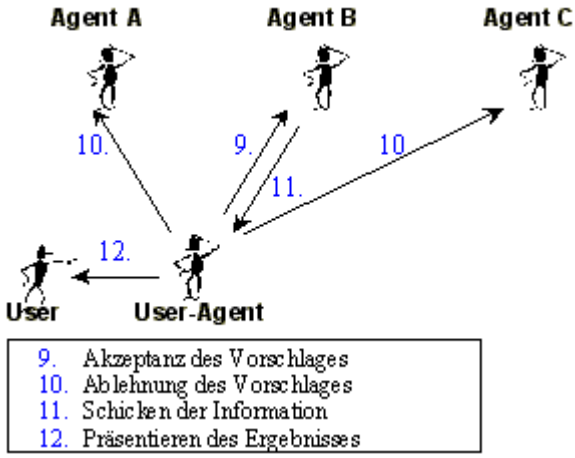


Abbildung 4-4: Senden der Information

Dieser Ablauf ist eine Möglichkeit von vielen, um Informationen zu beschaffen. Es könnte andere, einfachere aber auch komplexere Ansätze geben, der Phantasie sind keine Grenzen gesetzt.

In diesem Ansatz eines Informationsdienstes basierend auf einem Multiagentensystem existieren noch viele offene Punkte. Zum Beispiel muss geklärt werden, wie die Beschreibung eines speziellen Informationsdienstes auszusehen hat, damit der Dienst von Agenten ausfindig gemacht werden kann, die mit diesen Dienst kommunizieren. Andere noch zu klärende Punkte wären die Darstellung und Auswahl von Vorschlägen, die Darstellung der Information und das Kooperieren mit Informationsagenten auf anderen Plattformen. Es ist nur schwer denkbar, dass weltweit alle Agenten auf nur einer Plattform existieren werden. Wünschenswert wäre es, wenn mehrere Plattformen miteinander verbunden wären und die Agenten über die Plattformengrenzen hinaus miteinander kooperieren könnten, wie folgende Abbildung zeigt.

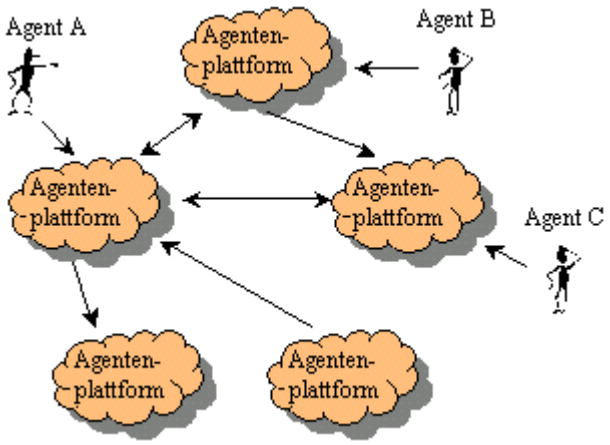


Abbildung 4-5: Mögliches Vernetzung von Agentenplattformen

Inwiefern und wie diese offenen Punkte gelöst werden können, oder andere Wege gefunden werden müssen, ist unter anderem von Funktionen der Agentenplattform abhängig. Die Anforderungen dieses Ansatzes an ein Multiagentensystem sollen nun diskutiert werden.

4.4 Anforderungen an eine Agentenplattform

Dass dieser Ansatz ein typisches Multiagentensystem beschreibt, wurde schon festgestellt. Damit er verwirklicht werden kann, müssen auch alle typischen Probleme eines solchen Systems gelöst werden, wie:

- Kommunikation (Darstellung und Austausch der Nachrichten)
- Verwaltung der Agenten
- Auffinden von Agenten

Diese werden im allgemeinen von einer Agentenplattform gelöst, die die Umgebung der Agenten darstellt. Agenten existieren auf einer Agentenplattform, bedienen sich ihrer Dienste und richten sich nach ihren Vorgaben. Für die Anforderungen des obigen Ansatzes muss eine Agentenplattform folgende Voraussetzungen erfüllen:

- Austausch von Nachrichten (physisch)
  Die Agenten müssen in der Lage sein, Nachrichten untereinander austauschen zu können.
- Definition der Nachrichten
  Das Format der Nachrichten muss spezifiziert sein, damit es den Agenten möglich ist Nachrichten anderer Agenten verstehen zu können.
- Unterstützung beim Führen von Dialogen
  Zum Austausch einer Informationen müssen möglicherweise mehrere Nachrichten zwischen den Agenten ausgetauscht werden.
- Auffinden der Dienste
  Es muss möglich sein, Informationsdienste ausfindig zu machen bzw. zu veröffentlichen.

Um diese Anforderungen bereitzustellen, sind Vor- und Nachteile eines eigenen Plattformentwurfs gegen die Verwendung eines standardisierten Ansatzes abzuwägen.

Ein eigener Entwurf bietet den Vorteil, genau die Funktionen anbieten zu können, die für die zu realisierende Aufgabe benötigt werden. So kann zum Beispiel das Format der

Nachrichten, die unter den Agenten ausgetauscht werden, auf die wirklich benötigten Daten und eine einfache Darstellung beschränkt werden. Die Implementierung einer Plattform nach eigener Definition würde wahrscheinlich deutlich weniger Aufwand verursachen, als bei der Verwendung eines Standards anfallen würde. Nachteil ist aber, dass die Umgebung der Agenten stark beschränkt ist. Es würden lediglich Agenten miteinander kooperieren können, die selbst implementiert wurden. Die Kooperation mit fremden Informationsanbietern, wie im Beispiel in der Einleitung dargestellt, wäre ausgeschlossen. Ein weiterer Grund, der für die Verwendung von Standards spricht, ist die geringere Gefahr von konzeptionellen Fehlern, da sie zumeist gut durchdacht sind. Eine standardisierte Architektur berücksichtigt Probleme, die in einem eigenen Ansatz erst sehr spät auffallen könnten, wie die Fehlerbehandlung, und zu aufwendigen Redesign führen könnte.

Auch wenn ein gewisser Mehraufwand und eventuelle Unklarheiten bei der Verwendung eines Standards auftreten könnten, sprechen die Vorteile für die Verwendung und gegen eine Eigenentwicklung, gerade in Anbetracht des ursprünglichen Ziels, völlig fremde Informationsagenten in einem offenen System wie dem Internet, miteinander kooperieren zu lassen.

Nach dieser grundlegenden Entscheidung ist nun festzulegen, welcher Standard benutzt werden soll. In Kapitel 3.2 wurden einige Entwicklungen im Bereich Multiagentensysteme vorgestellt. Von denen ist die Kombination KQML - KIF auszuschließen, da sie die oben definierten Anforderungen nicht vollständig erfüllen können. Anders sieht dies bei dem FIPA Standard und der OAA aus. Beide erfüllen alle Kriterien und kommen als zu verwendendes Multiagentensystem infrage. Der einzige Vorteil des FIPA Standards ist, dass es unterschiedlichste Implementierungen und somit auch unterschiedlichste Hilfsmittel zur Erzeugung und Verwaltung von Agenten gibt. Verwendet werden soll der FIPA Standard, welches sich nicht technisch begründen lässt. Entscheidend ist, dass die Firma, die diese Arbeit begleitet, Mitglied der FIPA ist und einige Forschungsarbeit für die FIPA geleistet hat und Erfahrungen mit dem Standard vorhanden sind.

Bevor zum Design des obigen Ansatzes übergegangen wird und die noch offen Fragen des Informationsdienstes geklärt werden, wird zunächst der FIPA Standard vorgestellt.

5 FIPA Standard

Die *Foundation for Intelligent Physical Agents* (FIPA) [8] wurde 1996 gegründet und ist eine Non-Profit Organisation, die sich aus über 50 kommerziellen und akademischen Mitgliedern zusammensetzt. Dazu gehören bekannte Firmen, wie zum Beispiel Nortel, BT, Fujitsu oder Siemens. Ziel ist es, einen weltweit anerkannten Standard zu definieren, durch den heterogene Agentensysteme verschiedenster Anbieter miteinander kooperieren können. Über die Jahre wurden verschiedene Standards verabschiedet und weiterentwickelt. Der FIPA-97 Standard ist der erste, feste Standard gewesen, gefolgt von den experimentellen Weiterentwicklungen FIPA-98 und FIPA-99. Erst der FIPA-2000 Standard soll als weiterer fester Standard verabschiedet werden, ist aber auch noch in einem experimentellen Zustand. Da alle FIPA Mitglieder die Absicht erklären die aktuellste Version unterstützen zu wollen und diese Version die Vorgängerversionen zu ersetzen scheint, wird an dieser Stelle der experimentelle FIPA 2000 Standard vorgestellt. Zur Zeit sind noch nicht alle FIPA Dokumente überarbeitet, weshalb zum Teil ältere Versionen immer noch gültig sind und verwendet werden.

5.1 Überblick

Der FIPA Standard besteht aus zahlreichen Spezifikationen, die ein vollständiges Multiagentensystem beschreiben. Diese Spezifikationen sind grob in fünf Teile einzuteilen. Auf oberster Ebene werden Beispielapplikationen vorgestellt in denen FIPA Agenten eingesetzt werden können. Auf der mittleren Ebene werden abstrakte Architekturen vorgestellt, die benötigt werden, um eine Agentenumgebung und Agentendienste zu verwirklichen. Die unterste Ebene ist in drei Teile gegliedert und beschreibt Funktionen und Grundlagen, die ein Multiagentensystem bereitstellen muss. Diese Struktur wird noch einmal von folgender Abbildung verdeutlicht, entnommen aus [9].

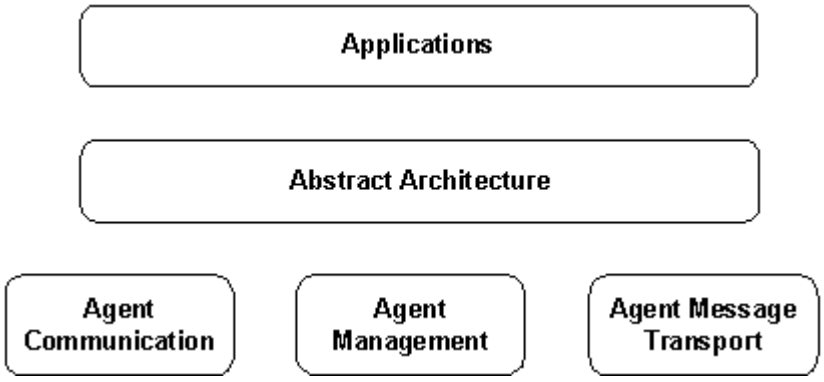


Abbildung 5-1: Struktur des FIPA Standards

Die Spezifikationen des ersten Teils der untersten Ebene beschreiben die Kommunikation der Agenten und definieren die Typen der Nachrichten (performative Äußerung, siehe 3.1.2), die Contentsprachen und Interaktionsprotokolle. Der Agent Management Teil beschreibt die Struktur der Agentenplattform und der letzte Teil die Darstellungsformen der Nachrichten und die Transportmechanismen, über die Nachrichten ausgetauscht werden.

Im folgenden werden lediglich Ausschnitte aus den letzten drei Teilen genauer betrachtet, da sie die Kernkomponenten des Standards darstellen.

5.2 Management

Aus dem FIPA Agenten Management sollen zwei Spezifikationen vorgestellt werden. Leider liegt hierzu noch kein Dokument des FIPA 2000 Standards vor, so dass die Architektur der 97er Version vorgestellt wird.

5.2.1 Agenten Plattform

Die Agentenplattform (AP) ist ein zentraler Bestandteil des FIPA Standards und stellt eine Umgebung dar, in der Agenten existieren können. Sie bietet Mechanismen zum Austausch von Nachrichten, zum Auffinden anderer Agenten und verwaltet die Agenten der Plattform.

Einen Überblick über die Komponenten einer FIPA Plattform und deren struktureller Aufbau zeigt Abbildung 5-2. Detaillierte Informationen zum Aufbau und der Funktionen einer FIPA Plattform werden nachfolgend erklärt und sind in [FIPA 97a] zu finden.

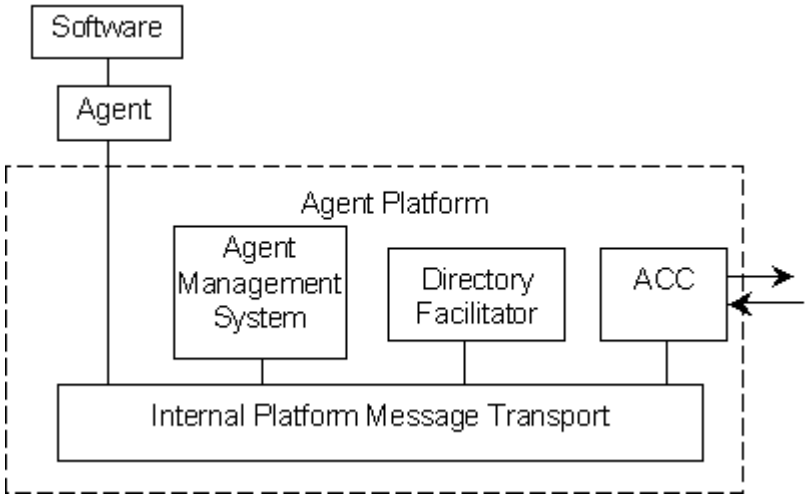


Abbildung 5-2: FIPA Agentenplattform Architektur

Die Plattform setzt sich aus folgenden 4 Komponenten zusammen, die zum Teil selbst als Agenten realisiert werden.

- **Internal Platform Message Transport**
  Die technische Grundlage um Nachrichten innerhalb der Plattform auszutauschen, bildet die *Internal Platform Message Transport* (IPMT) Komponente. Ihre Aufgabe ist, Nachrichten innerhalb der Plattform physisch von einen Sender zum Empfänger zu übermitteln. Der Agent kann sicher über eine erfolgreiche Zustellung der Nachricht sein, wenn er keine Fehlermeldung erhält. Die FIPA legt nicht fest welches Transportprotokoll zur Verwirklichung dieser Komponente zu verwenden ist, so dass CORBA, Java RMI, aber auch HTTP oder SMTP benutzt werden können. Eine Agentenplattform darf mehrere Transportprotokolle unterstützen. Die Agenten der Plattform müssen zumindest eines dieser Protokolle nutzen können.
- **Agent Communication Channel**
  Nachrichten von einen Agenten zu einem anderen Agenten werden über den *Agent Communication Channel* (ACC) geschickt. Eine Nachricht, die an einen anderen Agenten verschickt werden soll, wird zunächst an den ACC geschickt, dieser ist für die weitere Zustellung zuständig. Je nach Adressant wird die Nachricht plattformintern zu einem existierenden Agenten weitergeleitet, oder plattformextern an den ACC einer anderen Plattform geschickt. Dieser ist für die weitere Zustellung zuständig. Der Transport einer Nachricht zwischen

verschiedenen Plattformen geschieht über IIOP, welches später noch genauer vorgestellt wird. Auf einer Plattform kann es nur einen ACC geben.

- **Agent Management System**
  Die Managementkomponente der AP ist das *Agent Management System* (AMS). Sie wird als Agent realisiert und existiert genau einmal auf einer Plattform. Sie hat Kontrolle über den ACC und dessen Benutzung. Die Aufgabe des AMS Agenten ist die Verwaltung der Aktivitäten auf einer Plattform, wozu das Erschaffen und Zerstören von Agenten gehört sowie die Überwachung der Migration von Agenten und die Entscheidung, ob sich ein Agent an der Plattform anmelden darf. Das AMS verwaltet eine Liste der auf der Plattform existierenden Agenten und kennt deren physischen Transportadressen, die abhängig von der IPMT sind.
- **Directory Facilitator**
  Der *Directory Facilitator* (DF) ist ein Agent, der eine Art *Gelbe Seiten* Verzeichnis anbietet und anderen Agenten Funktionen zum Registrieren, Deregistrieren und Ändern der eigenen Beschreibung zur Verfügung stellt. Zudem ermöglicht der DF Agent anderen Agenten diese Registrierungen nach bestimmten Kriterien ausfindig zu machen. Es existiert mindestens ein DF auf einer Agenten Plattform. DF Agenten können eine Suchanfrage an andere DF Agenten weiterleiten, um ein größeres Sucherergebnis zu erzielen.

5.2.2 Lebenszyklus eines Agenten

FIPA Agenten existieren auf einer Agentenplattform und machen sich ihre Fähigkeiten zu nutze, um die eigenen Ziele zu erreichen. Im Prinzip sind Agenten Programme, die in Einklang mit anderen Programmen (Agentenplattform) laufen müssen. Dazu muss es Mechanismen geben, die Aktivitäten, wie das Starten und Beenden eines Agenten, regeln und verfolgen. Aus diesem Grund definiert die FIPA ein detailliertes Lebenszyklusmodell eines Agenten, das von der Plattform verwaltet wird.

Mögliche Zustände (Kreise) und Zustandsübergänge (Linien) dieses Lebenszyklus werden in folgender Abbildung dargestellt.

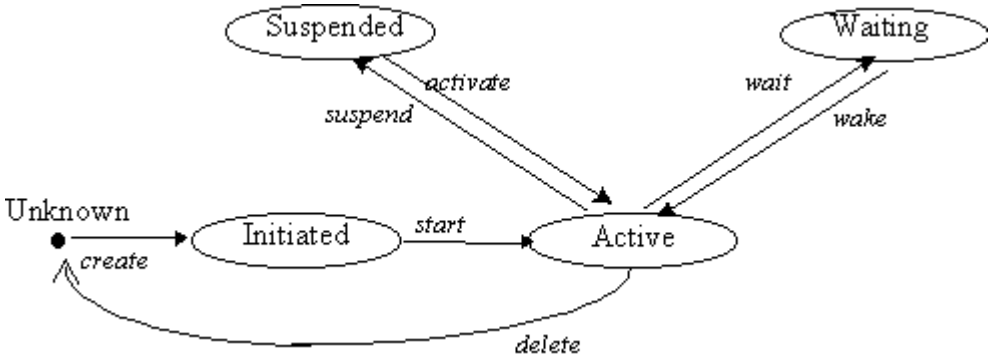


Abbildung 5-3: Lebenszyklus eines FIPA Agenten

Die genaue Bedeutung der Zustände und ihrer Übergänge kann aus [FIPA 97a] entnommen werden.

5.3 Nachrichten

Neben einer Managementumgebung ist ein weiterer wichtiger Zweig des FIPA Standards die Regulierung der Kommunikation. FIPA Agenten kommunizieren untereinander, indem sie Nachrichten austauschen. Damit sie die Nachrichten verstehen, müssen sie einem von der FIPA definierten Format entsprechen.

5.3.1 Agent Communication Language

Nachrichten, die FIPA Agenten austauschen, müssen der *Agent Communication Language* (ACL) entsprechen. In ihr werden grundlegende Informationen über eine Nachricht abgebildet, wie der Sender oder die Empfänger. Der semantische Inhalt der Nachricht (Content) wird durch die ACL nicht beschrieben, sondern lediglich in ihr als String dargestellt. Zusätzliche Parameter einer ACL Nachricht geben Auskunft über das Format des Contents. Durch diese Aufteilung ist die ACL flexibel verwendbar, da jegliche Art von Informationen dargestellt werden kann.

Die ACL basiert auf der Speech Act Theorie, die in Kapitel 3.1.1 vorgestellt wurde. Jede ACL Nachricht besitzt einen Typ, der einer performativen Äußerung entspricht, und dem Content der Nachricht eine bestimmte Handlungsbedeutung gibt. So entspricht eine Nachricht des Typs *request* einer Anforderung, die der Empfänger ausführen soll. Eine Nachricht, der eine performative Äußerung zugewiesen, ist wird von der FIPA auch *Communicative Act* (CA) genannt. Die zulässigen CA, die eine ACL Nachricht annehmen kann, und ihre semantische Bedeutung sind von der FIPA exakt spezifiziert und werden im übernächsten Abschnitt vorgestellt.

5.3.1.1 ACL Grammatik

Die FIPA erlaubt die Darstellung der ACL in mehreren Formaten. Neben einer XML basierten Darstellung ist auch eine Stringdarstellung erlaubt, welche hier vorgestellt werden soll. Die Stringdarstellung einer ACL Nachricht ist einfach und übersichtlich. Sie beginnt mit einer öffnenden und endet mit einer schließenden Klammer. Der erste Term entspricht einem gültigen CA, und beschreibt die performative Aussage der Nachricht. Darauf folgen einige, größtenteils optionale *key/value* Paare, deren Reihenfolge unbestimmt ist.

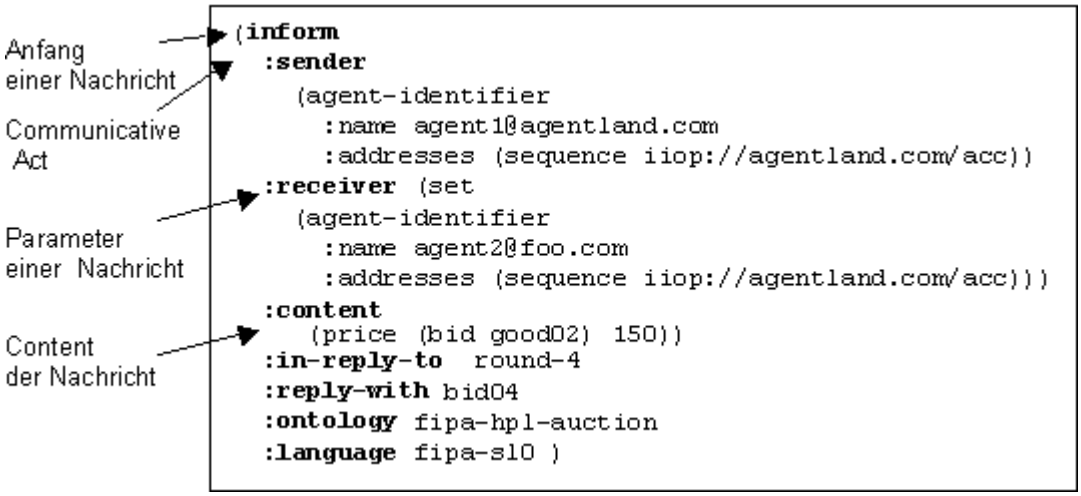


Abbildung 5-4: Aufbau einer ACL Nachricht

Folgende Liste beschreibt alle gültigen Parameter einer ACL Nachricht und deren Bedeutung. Die gesamte Definition der ACL in EBNF ist in Anhang B zu finden, oder kann in [FIPA 2000a] nachgelesen werden.

Teilnehmer der Kommunikation

- `:sender`
  Sender der Nachricht.
- `:receiver`
  Empfänger der Nachricht (mehrere möglich).
- `:reply-to`
  Die Antwort dieser Nachricht wird an diesen Empfänger geschickt.

Inhalt der Nachricht

- **`:content`**
  Content der Nachricht.

Beschreibung des Contents

- **`:language`**
  Darstellung des Contents.
- **`:encoding`**
  Codierung des Contents.
- **`:ontology`**
  Ontologie des Contents.

Kontrolle der Konversation

- **`:reply-with`**
  Dieser Wert wird bei einer Antwort in das Feld `:in-reply-to` übernommen.
- **`:reply-by`**
  Zeit oder Datum, wann eine Nachricht spätestens beantwortet sein muss.
- **`:in-reply-to`**
  Wert des `:reply-with` Parameters der Vorgängernachricht.
- **`:protocol`**
  Protokoll zu dem diese Nachricht angehört.
- **`:conversation-id`**
  ID der Konversation zu der diese Nachricht gehört.

Zusätzlich ist es dem Anwender erlaubt, eigene Parameter zu definieren. Diese müssen mit dem Präfix ":x-" beginnen.

5.3.1.2 Communicative Acts

Durch die Zuweisung eines *Communicative Acts* (wie Anfrage, Information oder Fehler) kann auf die Absicht einer ACL Nachricht geschlossen werden ohne den genauen Inhalt zu kennen. Missverständnisse bei der Interpretation des Contents vermieden werden. Der CA kann beispielweise dem Content „`x == y`" die Bedeutung einer Anfrage (`request`) geben, die mit wahr oder falsch beantwortet werden soll, aber auch die Änderung der gemeinsamen Umgebung bekannt geben (`inform`).

Mit dem FIPA 2000 Standard werden 22 verschiedene CA definiert. Da eine Beschreibung aller etwas ausufernd wäre, sollen an dieser Stelle nur die zukünftig verwendeten und somit die für diese Arbeit wichtigsten Typen aufgelistet werden. Eine vollständige Liste mit einer Beschreibung der CAs kann in [FIPA 2000b] gefunden werden.

- **Accept Proposal**
  Eine solche Nachricht akzeptiert einen zuvor gemachten Vorschlag. Der Sender gibt dem Empfänger zu verstehen, dass er den von ihn kommende Vorschlag annimmt.
- **Agree**
  `Agree` gibt dem Empfänger zu verstehen, dass eine zuvor erhaltene Anweisung (z.B. `request`), möglicherweise in der Zukunft, ausgeführt wird. Implizit bedeutet diese Nachricht, dass der Sender die vorige Nachricht verstanden hat. Zusätzlich kann einem `agree` zum Beispiel die Zeit beigefügt werden, die beschreibt wann die Aktion voraussichtlich ausgeführt wird.

- **Call for Proposal**
  Der Sender dieser Nachricht bittet dem Empfänger um Vorschläge zu einer bestimmten Aktion. Zum Beispiel um ein Preisangebot für ein Objekt während einer Auktion.
- **Failure**
  Dem Empfänger einer `failure` Nachricht wird mitgeteilt, dass eine Aktion der vorhergehenden Nachricht nicht ausgeführt werden konnte, obwohl die Aktion zuvor als ausführbar eingestuft wurde. Zum Beispiel wird mit einem `failure` CA geantwortet, wenn ein DF Agent eine Anforderung zum Registrieren eines Agenten bekommt, dieser aber schon registriert ist.
- **Inform**
  Der Sender informiert dem Empfänger über einen Fakt seiner Wissensbasis, oder Umgebung. Er hält diese Aussage für wahr und beabsichtigt, dass der Empfänger sie auch für wahr nimmt und noch nicht kennt.
- **Not Understood**
  Dem `not-understood` CA geht immer eine Aktion/Nachricht des Empfängers an den Sender voraus. Die Bedeutung ist, dass der Sender die ursprüngliche Nachricht des Empfängers nicht verstanden hat. Dem Content einer solchen Nachricht sollte die nicht verstandene Nachricht und ein Grund beigefügt sein.
- **Propose**
  Mit diesem CA wird eine Aktion vorgeschlagen, die vom Sender ausgeführt werden kann. Zum Beispiel kann ein Angebot zum Kauf eines Objektes gemacht werden.
- **Refuse**
  Ein `refuse` teilt dem Empfänger mit, dass eine Aktion, die beim Sender angefordert wurde, nicht ausgeführt werden konnte.
- **Reject Proposal**
  Ein zuvor gemachter Vorschlag wurde vom Sender dieser Nachricht abgelehnt.
- **Request**
  Der Sender fordert den Empfänger auf eine Aktion auszuführen. Der Content dieser Nachricht beschreibt die auszuführende Aktion. Zum Beispiel wird der DF Agent angewiesen den Sender zu registrieren oder nach bestimmten anderen Agenten zu suchen.

Der CA beschreibt lediglich die Bedeutung des Contents. Darstellung und Inhalt des Contents sind völlig unabhängig davon. Ob ein Agent eine Nachricht verstehen kann, ist abhängig von dessen Darstellung und Semantik.

5.3.2 Content einer Nachricht

Wie schon zuvor erwähnt ist die ACL unabhängig von der Darstellung des Contents einer Nachricht. Dieser wird zumeist durch eine Contentsprache dargestellt, die Feststellungen (propositions), Handlungen (actions) und Ausdrücke (terms) darstellen kann. Eine Contentsprache ist die FIPA Semantic Language (SL), die als Standardsprache verwendet wird. Sie dient unter anderem zur Kommunikation mit AMS und DF Agenten. Die Flexibilität der ACL erlaubt es auch andere Contentsprachen zu verwenden, wie das Knowledge Interchange Format (KIF). Die FIPA SL ist eine sehr umfassende Sprache, deren Komplexität durch Definition von Untermengen etwas vermindert wird. Die SL0 ist die kleinste Untermenge der SL, gefolgt von SL1 und SL2. Für die meisten Aufgaben genügt es die SL0 Untermenge zu verwenden. Ähnlich wie die ACL kann auch die SL in mehreren Darstellungen abgebildet werden. Entnommen aus [FIPA 2000e] ist in Anhang C die Stringdarstellung der SL0 in EBNF dargestellt.

Entsprechende XML und RDF Darstellungen sind unter der FIPA Homepage [8] zu finden.

5.3.3 Semantik einer Nachricht

Mit einer Contentsprache kann Wissen in XML oder einem einfachen String dargestellt werden. Die Bedeutung dieses Wissens, sprich die Semantik, kann mit einer solchen Sprache nicht beschrieben werden. Hierzu werden Ontologien verwendet, die von mehreren Agenten oder Diensten geteilt werden. Sie bieten einen Wortschatz zur Beschreibung und Übermittlung von Wissen über dasselbe Thema und eine Menge von Eigenschaften und Beziehungen der Entitäten dieses Wortschatzes an. Die FIPA definiert eine Ontologie in [FIPA 99a] in Kapitel 5.2.1 folgendermaßen:

„An ontology explicitly specifies the concepts and associations within a domain in a way that is formal, objective, and unambiguous. This includes the objects, quantitative and qualitative information, distinctions, and relationships. Common (or shared) ontologies allow the sharing and reuse of knowledge (about the domain of discourse) among software entities (i.e. programs or agents)".

5.3.3.1 Agent Management Ontologie

Der Content, der in den Nachrichten der AMS und DF Agenten erwartet und versendet wird, wird in FIPA SL0 dargestellt und durch die Agent Management (AM) Ontologie beschrieben. Diese Ontologie, ist in [FIPA 2000f] definiert und besteht im Prinzip aus drei Teilen.

Zunächst beschreibt sie die Funktionen, die von den DF und AMS Agenten angeboten werden. Dieses sind:

DF Funktionen
register
deregister
modify
Search

AMS Funktionen
register
deregister
modify
search
get-description
quit

Darauf folgend werden die Datenstrukturen beschrieben, die zur Verwendung dieser Funktionen benötigt werden.

df-agent-description

Parameter	Typ
name	agent-identifier
services	Set of service-description
protocol	Set of String
ontology	Set of String
language	Set of String

ap-description

Parameter	Typ
name	String
transport-profile	ap-transport-description
mobility	Boolean
dynamic	Boolean

service-description

Parameter	Typ
name	String
type	String
protocol	Set of String
ontology	Set of String
language	Set of String
ownership	String
properties	Set of property

ams-agent-description

Parameter	Typ
name	agent-identifier
ownership	String
state	String

agent-identifier

Parameter	Typ
name	Word
addresses	Sequence of URL
resolvers	Sequence of agent-identifier

search-constraints

Parameter	Typ
max-depth	Int
max-results	Int

Property

Parameter	Typ
name	String
value	Term

Zuletzt werden Ausnahmesituationen, die bei der Ausführung einer Handlung auftreten können, durch Exceptions beschrieben. Diese können aus [FIPA 2000f] entnommen werden.

5.4 Protokolle

Nur die wenigsten Absichten oder Ziele eines Agenten lassen sich mit dem Austausch einer Nachricht erreichen. Selbst einfache Konversationen benötigen zumeist den Austausch mehrerer Nachrichten zwischen Initiator und Teilnehmer (zukünftig als Responder bezeichnet), wie zum Beispiel eine einfache Anfrage und deren Beantwortung. Werden mehrere Nachrichten ausgetauscht, um eine Aufgabe zu erfüllen, wird von einem Dialog geredet. Damit Initiator und Responder erfolgreiche Dialoge führen können und die ausgetauschten Nachrichten verstehen, bzw. wissen, welchem Format eine Antwort entsprechen muss, werden die meisten Dialoge durch Interaktionsprotokolle geregelt. Ein Interaktionsprotokoll ist eine fest definierte Abfolge von Nachrichten und regelt den Kommunikationsablauf.

Protokolle werden bei jeder Art von Kommunikation zwischen Computersystemen eingesetzt. So definiert das ISO/OSI Referenzmodell einen Protokollstapel mit sieben aufeinander aufbauenden Schichten, zum Austausch von Nachrichten zwischen Computersystemen, wie in Abbildung 5-5 zu sehen ist. Jede Schicht / Protokoll hat eine bestimmte Aufgabe und bietet der darüber liegenden und der darunter liegenden Schicht einen Dienst an. So legt z.B. die Bitübertragungsschicht eine von der Sicherungsschicht erhaltene Nachricht in bestimmter Codierung auf das Medium (Kabel, Licht, Funk), die von der Bitübertragungsschicht des Empfängers entgegengenommen, verarbeitet und an die Sicherungsschicht weitergereicht wird. Eine Nachricht wird physisch von der

obersten, der Anwendungsschicht, an die jeweils untere Schicht weitergereicht, bis sie über das Medium übertragen wird. Auf Seite des Empfängers wird sie von der Bitübertragungsschicht entgegengenommen und an die jeweils nächste Schicht weitergereicht, bis sie bei der Anwendung des Empfängers angekommen ist. Jede Schicht hat dabei spezielle Aufgaben zu erfüllen, damit der Nachrichtenaustausch funktionieren kann. Die Aktionen einer Schicht, die auf Seite des Senders und des Empfängers ausgeführt werden, werden Protokoll genannt.

Eine Implementierung dieses Modells gibt es nicht, jedoch lassen sich gängige, bekannte Protokolle auf einige dieser Schichten abbilden. So beinhaltet Ethernet die Funktionen der Bitübertragungs- und der Sicherungsschicht, TCP/IP die Funktionen der Netzwerk- und Transportschicht und HTTP, SMTP, CORBA oder IIOP die der Anwendungsschicht. Weitere Details zu diesem Modell und dessen reale Umsetzung sind in [Tannenbaum 96] zu finden.

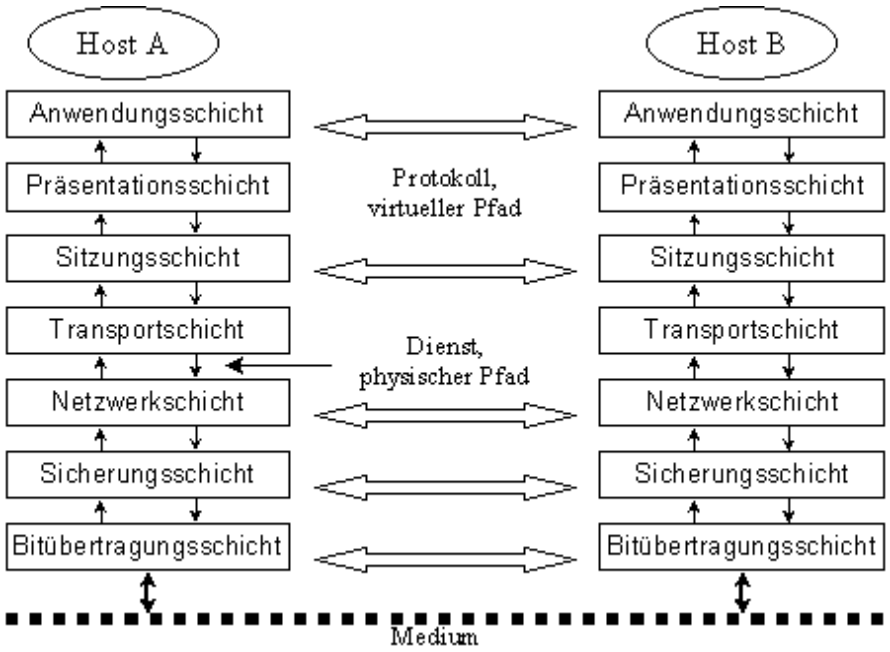


Abbildung 5-5: ISO/OSI Referenzmodell

Die Anwendungsschicht dieses Protokollstapels würde der IPMT Komponente einer FIPA Plattform entsprechen. Darauf aufbauend definiert die FIPA in einer weiteren Protokollschicht eine Reihe von Protokollen, nach denen Dialoge geregelt ablaufen können. Ein solches Protokoll wird durch eine bestimmte Reihenfolge von ACL Nachrichten mit festgelegten *Communicative Acts* definiert. Trifft eine ACL Nachricht mit einem, vom Protokoll abweichenden Typ ein, schlägt die Konversation fehl. Dass eine ACL Nachricht einem bestimmten Protokoll angehört, ist am Wert des `:protocol` Parameters zu erkennen.

Die FIPA definiert folgende 11 Protokolle zu denen es jeweils eigene Spezifikationen gibt:

- Request Interaction Protocol
- Query Interaction Protocol
- Request When Interaction Protocol
- Contract Net Interaction Protocol
- Iterated Contract Net Interaction Protocol
- English Auction Interaction Protocol

- Dutch Auction Interaction Protocol
- Brokering Interaction Protocol
- Recruiting Interaction Protocol
- Subscribe Interaction Protocol
- Propose Interaction Protocol

Zusätzlich können eigene Protokolle definiert werden.

Zur Verdeutlichung sollen zwei Protokolle vorgestellt werden, die in späteren Teilen dieser Arbeit verwendet werden.

5.4.1 Request Interaction Protocol

Das *Request Interaction Protocol* [FIPA 2000c] dient dem Initiator dazu eine Anforderung an einen Responder zu stellen, der den Initiator über den Ausgang der Anforderung informiert. Die beteiligten CA und ihre Reihenfolge werden in Abbildung 5-6 grafisch dargestellt. Der kursive Text bezieht sich auf den Content der Nachricht.

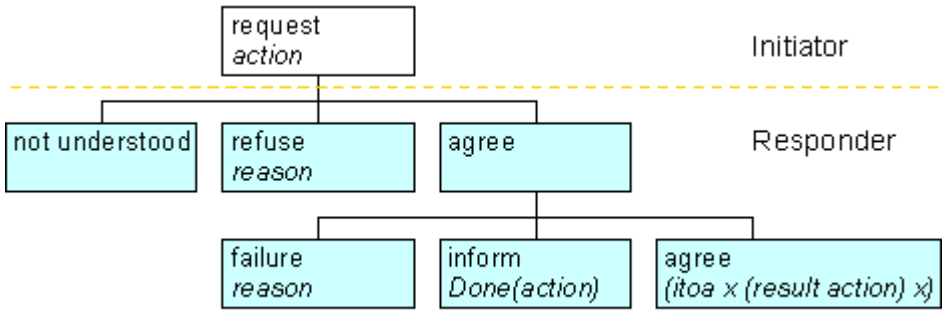


Abbildung 5-6: FIPA-Request Protokoll

Dieses Protokoll wird unter anderem benutzt, um mit DF- und AMS Agenten zu kommunizieren. Ein beispielhafter Dialog einer Registrierung beim DF Agenten ist in Anhang D zufinden.

5.4.2 Iterated Contract Net Interaction Protocol

Das *Iterated Contract Net* (ICN) Protokoll [FIPA 2000d] ist gedacht, um Verhandlungen mit einem oder mehreren Agenten zu führen. Der Initiator fragt nach Vorschlägen und wählt aus den eingegangenen Vorschlägen den oder die besten heraus. Daraufhin wird jedem Teilnehmer mitgeteilt, ob sein Vorschlag akzeptiert oder abgelehnt wurde. Es ist möglich nach der Ablehnung eines Vorschlags weitere Vorschläge vom Anbieter anzufordern. Abbildung 5-7 stellt diesen Ablauf noch einmal grafisch dar.

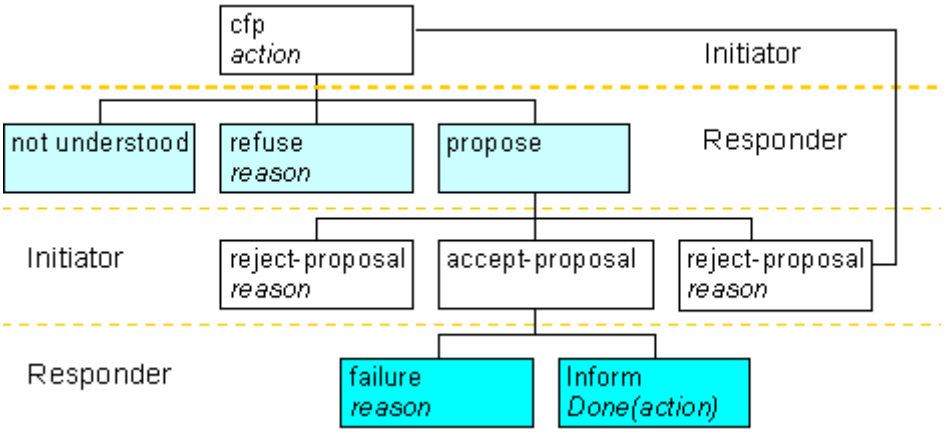


Abbildung 5-7: FIPA - Iterated Contract Net Protokoll

Ein Beispiel für das ICN Protokoll wäre eine Verkaufsverhandlung. Der Initiator möchte ein Auto kaufen und holt Angebote bei verschiedenen Händlern ein. Sind alle Angebote zu teuer, lehnt er alle Vorschläge ab und fragt nach Angeboten für ein ähnliches Auto (z.B. ohne Klimaanlage). Bekommt er nun ein Auto vorgeschlagen, dass seinen Vorstellungen entspricht, wird es akzeptiert. Alle anderen werden abgelehnt. Diesen Ablauf verdeutlicht das Flussdiagramm in Abbildung 5-8 noch einmal.

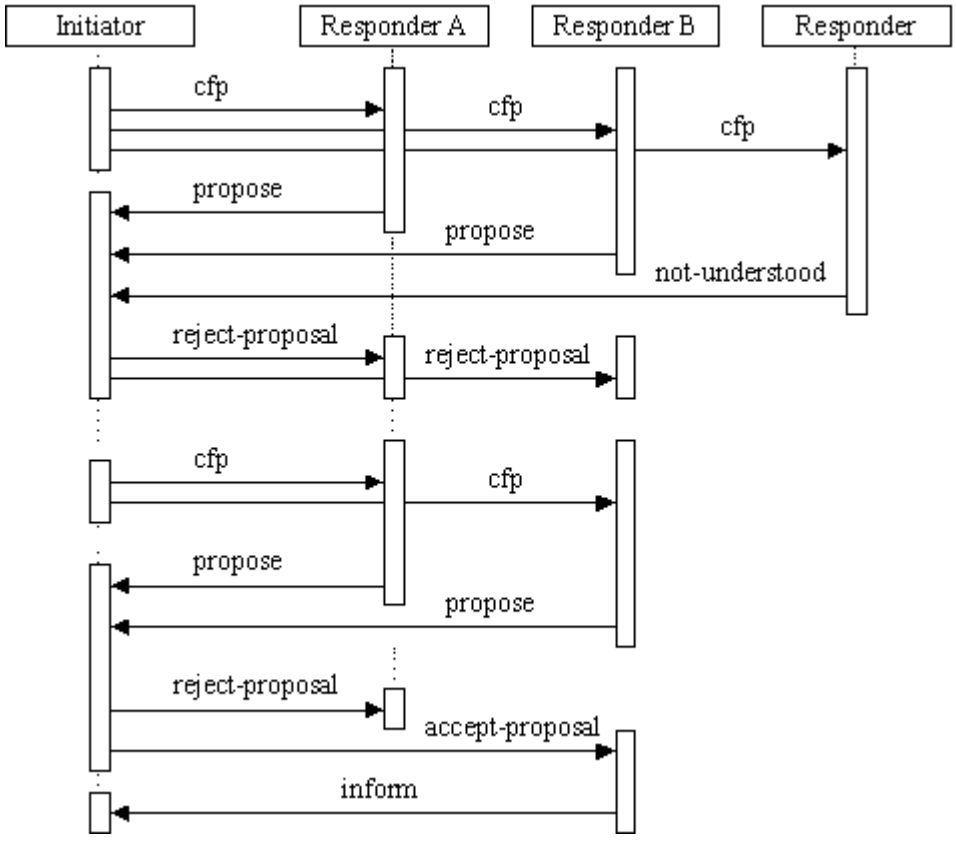


Abbildung 5-8: Ablauf des ICN Protokolls

5.5 Kritik

Die Kritik, die an dem FIPA Standard an dieser Stelle zu richten ist, fällt recht positiv aus. Es scheinen alle Anforderungen an ein Multiagentensystem berücksichtigt zu werden. Ein Neuling in dem Bereich Multiagentensysteme wird einige Schwierigkeiten haben die vielen verschiedenen Spezifikationen zu ordnen und wird mit einer Vielzahl vom neuen Begriffen konfrontiert, die zum Teil etwas deutlicher erklärt werden könnten.

Der FIPA 2000 Standard ist um einiges schlüssiger als seine Vorgängerversionen, jedoch vermisse ich an einigen Stellen ein ausführliches Fehlermanagement. Auch das Auffinden anderer FIPA Plattformen könnte problematisch werden.

5.6 Existierende FIPA Plattformen

Der FIPA Standard wurde in den letzten Jahren weiterentwickelt und verbessert. Viele Erfahrungen wurden von Mitgliedern gesammelt, die den Standard in einer eigenen Plattform implementiert und Agenten entwickelt haben. Neben zahlreichen Eigenimplementierungen existieren einige bekannte Plattformen und Frameworks, die zum Teil samt Quellcode im Internet frei verfügbar sind.

Die bekanntesten Plattformen sollen hier kurz vorgestellt werden:

- **FIPA-OS *(FIPA-Open Source)***
  FIPA-OS ist eine *open source* Implementierung der Firma Emorphia. Sie ist plattformunabhängig in Java implementiert und komponentenbasiert. Die Mehrzahl aller aktuellen FIPA Spezifikationen wird von ihr unterstützt und fortlaufend aktualisiert. Durch den Einsatz von FIPA-OS wird die einfache und schnelle Entwicklung von Agenten ermöglicht. Weitere Informationen zu FIPA-OS können im Internet unter [10] gefunden werden.
- **JADE (*Java Agent DEvelopment Framework*)**
  JADE [11] ist ein Framework entwickelt von der Firma CSelt und wie FIPA-OS vollständig in Java entwickelt. Es erlaubt eine Agentenplattform auf verschiedenen Rechnern mit unterschiedlichen Betriebssystemen zu verteilen. Zusätzlich werden Debugwerkzeuge und GUIs zur Verwaltung der Agenten angeboten. JADE-Agenten sind innerhalb einer Plattform mobil.
- **AAP (*APRIL Agent Platform*)**
  Die AAP wurde von den Fujitsu Laboratories in Kalifornien entwickelt. Zur Implementierung benutzen sie die *Agent Process Interaction Language* (APRIL), die von Francis G. McCabe und Keith L. Clark am Imperial College entwickelt wurde. APRIL, zu der Details unter [McCabe 95] und [12] gefunden werden können, ist eine symbolische Sprache und wurde konzipiert, um verteilte Anwendungen zu bauen. Sie ist objekt-basiert und bietet aktive Objekte als Prozesse an, deren Methoden durch das Senden von Nachrichten aktiviert werden. Die Eigenschaft APRIL's, verteilte Anwendungen entwickeln zu können, wurde in der AAP benutzt. Die AAP bietet die typischen FIPA Funktionalitäten, ist aber nur für verschiedene Unix Derivate und Linux Systeme verfügbar. Die AAP und APRIL sind *open source* Projekte und können unter [13] heruntergeladen werden.

Alle Plattformen geben an, zu 100 Prozent FIPA konform zu sein. Dies würde bedeuten, dass Agenten auf unterschiedlichsten Plattformen miteinander kooperieren können. Bisher gesammelte Erfahrungen zeigen, dass dies zumeist nur zutrifft, wenn Agenten auf den gleichen Plattformimplementierungen existieren. Abweichungen von den FIPA Spezifizierungen fallen so nicht auf, da Sender und Empfänger dieselbe Basisimplementierung verwenden. Sobald Agenten unterschiedlicher FIPA Implementierungen miteinander kooperieren sollen, treten häufig unerwartete Probleme auf. Minimalste Abweichungen vom Standard führen zu Inkompatibilitäten. Um diese Probleme zu erkennen und zu beheben wurde ein groß angelegtes Projekt gestartet, in dem Agenten unterschiedlichster Implementierungen miteinander kooperieren sollen.

5.7 AgentCities

In dem Projekt AgentCities [14] wird das Potential von FIPA Agentenanwendungen getestet. Ziel ist es ein weltweites, öffentlich zugängliches und ständig verfügbares Netzwerk von FIPA Plattformen aufzubauen. Auf jeder Plattform sollen Agenten Dienste anbieten, die Informationen über eine wirkliche Stadt (London, Berlin, Paris, San Francisco ...) bereitstellen. Jeder Teilnehmer dieses Projekts baut in seiner Plattform eine eigene AgentCity auf, die Informationsdienste über deren Einrichtungen, wie Bars, Restaurants, Hotels oder Theater anbieten. Anwendungen sollen in der Lage sein, über Agenten diese Dienste zu nutzen und irgendwann auf Basis dieser Dienste neue, komplexere Dienste anzubieten.

Das AgentCities Projekt ist noch recht jung und besteht momentan aus neun Städten, die jeweils unterschiedlichste Testdienste anbieten. Ziel dieses Projektes ist es, die Einsatzfähigkeit des FIPA Standards zu zeigen und Probleme, die lediglich in komplexen und weitverteilten Umgebungen auftreten, zu entdecken und zu beheben.

Die existierenden Städte basieren zumeist auf eine der oben vorgestellten Plattformimplementierungen oder auf Eigenentwicklungen.

6 Design der Agentenplattform und des Informationsagenten

Der vorgestellte FIPA Standard scheint als Grundlage einer Agentenplattform geeignet zu sein, da alle Anforderungen an eine Agentenplattform aus Kapitel 4.4 des zu verwirklichenden Informationsagenten erfüllt werden. Der

- Austausch von Nachrichten
  wird durch den ACC und dem IPMT geregelt,
- die Definition der Nachrichten
  wird durch die Sprachen ACL und SL beschrieben,
- Unterstützung beim Führen von Dialogen
  gibt es durch fest definierte Protokolle und
- Auffinden der Dienste
  geschieht durch den DF Agent.

Durch dieses Ergebnis wird die getroffene Entscheidung den FIPA Standard als Basis der benötigten Agentenplattform zu verwenden bekräftigt. Durch die nun vorhanden Kenntnisse über den FIPA Standard sollen in diesem Kapitel die noch offenen Punkte aus 4.3 zur Verwirklichung eines Informationsagenten geklärt werden und ein Ansatz gezeigt werden, wie ein solcher Informationsagent aussehen könnte. Zunächst soll die Aufmerksamkeit wieder auf die Plattform gerichtet bleiben, die hier angesprochenen Punkte werden in Kapitel 6.3 wieder aufgegriffen.

Bei der Vorstellung des FIPA Standards wurden bereits in Kapitel 5.5 existierende Implementierungen vorgestellt. Eine wichtige Frage, die an dieser Stelle geklärt werden muss, ist, inwiefern es sinnvoll ist eine weitere, eigene Implementierung vorzunehmen oder eine existierende Plattform als Grundlage zu nutzen.

Die wichtigste Voraussetzung für kooperierende Agenten ist, dass sie miteinander kommunizieren können. Der FIPA Standard dient dazu als eine wichtige Grundlage, jedoch müssen unterschiedliche Plattformen exakt den Vorgaben des Standards entsprechen, um miteinander kompatibel zu sein. Eine Evaluierung einiger in Kapitel 5.5 vorgestellten, frei verfügbaren Plattformen hat ergeben, dass Nachrichten zwischen Agenten problemlos ausgetauscht werden können, solange sie sich auf einer Plattform oder unterschiedlichen Instanzen einer Plattform befinden. Agenten auf unterschiedlichen Implementierungen können nur ungenügend, oder gar nicht miteinander kommunizieren. Dies scheint daran zu liegen, dass in den Versionen 97–99 des FIPA Standards, wichtige Details zu kompliziert, oder nicht eindeutig definiert wurden und jede Plattformimplementierung eine eigene Lösung verwirklicht hat. Ein weiteres Problem ist, dass die Entwickler selbst im Standardisierungsgremium sitzen und eigene Ideen in den Standard bringen wollen, die vorsorglich schon einmal implementiert werden. Abhilfe soll die Version 2000 des Standards bringen, die jedoch nur von einer Plattform vollständig unterstützt wurde. Aufgrund dieser Testergebnisse scheint es sinnvoll zu sein eine eigene Plattform zu entwickeln, die eventuell auf die Abweichungen anderer Plattformen vom Standard individuell angepasst werden kann.

Ein weiterer Grund für die Implementierung einer eigenen Plattform ist, dass nicht auszuschließen ist, dass Plattform zur Verwirklichung des Informationsagenten zusätzliche Funktionen anbieten muss, die über den FIPA Standard hinausgehen und dementsprechend auch nicht von anderen Plattformen angeboten werden. Eine Änderung des Quellcodes dieser Plattformen ist ausgeschlossen, da diese in jeder neuen Version nahgezogen werden müssten.

Diese Gründe sind ausschlaggebend für eine eigene Implementierung des FIPA Standards. Neben der Spezifikation des Informationsagenten ist ein weiteres Ziel dieses Kapitels das Design einer FIPA konformen Multiagentenplattform, die zunächst lediglich den obigen Anforderungen des Informationsagenten entspricht.

Der FIPA Standard ist mit seinen 36 aktuellen Spezifikationen ziemlich komplex und es ist sehr aufwendig ihn in allen Details zu implementieren. Darum soll zunächst herausgefunden werden, welches die interessanten Teile des Standards sind, die dringend als Basis für eine funktionsfähige FIPA Plattform benötigt und implementiert werden müssen und welche zu vernachlässigen sind.

Da ein Schwerpunkt der benötigten Plattform auf Kommunikation beruht, spielen Aspekte wie Mobilität der Agenten bei der zu entwickelnden Plattform zunächst keine Rolle und können unberücksichtigt bleiben. Vielmehr ist es wichtig, dass Agenten Nachrichten austauschen können und diese auch verstehen. Es kann eine weitere Einschränkung gemacht werden, die den Aufwand der Implementierung deutlich senkt. Eine FIPA Plattform kann Nachrichten an Agenten weiterleiten, die sich auf anderen Plattformen (-implementierungen) befinden. Somit ist es möglich mit allen FIPA Agenten zu kommunizieren, ohne dass jemals fremde Agenten auf der eigenen Plattform existieren und angemeldet zu sein brauchen. Mit dieser Einschränkung ist die Plattform weiterhin (zumindest nach außen) FIPA konform und es könnten möglicherweise weitere Teile des FIPA Standards, wie das Management der Agenten oder ihre Lebenszyklen, zunächst stark vereinfacht oder gar nicht implementiert werden.

Auf Basis dieser Einschränkungen gehen die folgenden beiden Unterkapitel etwas detaillierter auf die Komponenten des FIPA Standards ein und sollen in groben Zügen die Architektur der eigenen FIPA Plattform aufzeigen. Dabei wird top-down vorgegangen und mit den einzelnen Komponenten der Plattform begonnen, bevor auf die Details der Kommunikation eingegangen wird. Nachdem das Design der eigenen FIPA Plattform vollständig ist, wird überlegt, wie der zu verwirklichende Informationsagent auf Basis FIPA Plattform und der Überlegungen aus Kapitel 4.3 aussehen könnte.

6.1 Plattform

Eine FIPA Plattform besteht aus verschiedenen Komponenten, die bereits in Kapitel 5.2 und durch Abbildung 5-2 vorgestellt wurden. Wie eben erwähnt, ist es nicht unbedingt notwendig den gesamten FIPA Standard zu implementieren, um den Anforderungen des Informationsagenten zu genügen und um (zumindest anderen Plattformen gegenüber) FIPA konform zu bleiben. An dieser Stelle wird im Detail diskutiert, welche Plattformfunktionalität dazu minimal benötigt wird. Vorweg zeigt Abbildung 6-1 schon einmal die mögliche Struktur der Plattform.

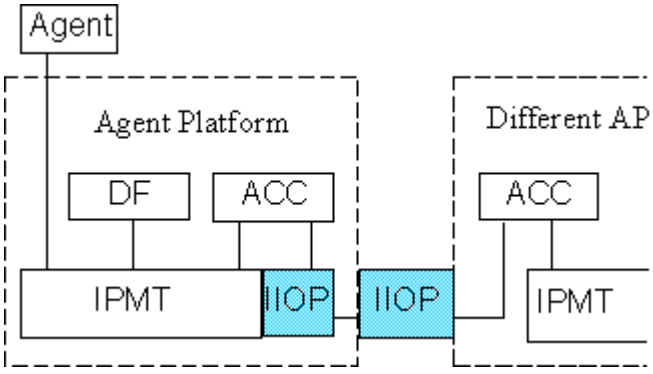


Abbildung 6-1: Mögliche Struktur der eigenen FIPA Plattform

6.1.1 Internal Platform Message Transport

Diese Komponente, auch IPMT abgekürzt, transportiert eine Nachricht von einem Agenten zum ACC bzw. vom ACC zu einem Agenten auf der Plattform. Dabei hat sie zwei Voraussetzungen zu erfüllen, um FIPA konform zu sein.

Die Agenten einer Plattform (einschließlich ACC) müssen physisch nicht zwingend auf einem Computer existieren, sie können im Netzwerk verteilt sein. Selbst, wenn sie sich auf einem Computer befinden, können sie in verschiedenen Prozessen existieren, die keinen gemeinsamen Speicher teilen. Die IPMT muss also Interprozess-Kommunikation ermöglichen, welche prinzipiell über unterschiedlichste Ansätze realisiert werden kann, wie HTTP, SMTP, CORBA oder andere dem Middleware Bereich angehörige Standards und Produkte. Das Protokoll, über das die plattform-interne Kommunikation geschieht, wird von der FIPA nicht vorgeschrieben.

Des Weiteren soll ein Agent mit mehreren Anderen gleichzeitig kommunizieren und geregelte Dialoge über mehrere Nachrichten hinweg führen können. Dies ist mit einer synchronen Kommunikationsform wie dem *remote procedure call* (RPC), nur schlecht möglich, da der sendende Agent seinen Programmablauf unterbricht, bis eine Antwort erhalten wurde. Eine Anfrage bei einem Agenten könnte sehr viel Zeit in Anspruch nehmen, und es wäre unperformant, wenn alle anderen laufenden Dialoge des Agenten auf die Abarbeitung des aktuellen warten müssten. Ein weiterer Nachteil der synchronen Kommunikationsform besteht bei Nachrichten, die keine Antwort benötigen. Der eigene Ablauf würde trotzdem unterbrochen werden, bis das Protokoll den Aufruf, mit einer leeren Antwort beendet. Eine andere Kommunikationsform, die diese Nachteile vermeidet, ist asynchrones Messaging. Hier wird der eigene Programmablauf fortgesetzt, sobald die Nachricht den Empfänger erreicht hat. Nachteil von dieser Kommunikationsform ist, dass eingehende Nachrichten bestehenden Dialogen explizit zugeordnet werden müssen, was den Verwaltungsaufwand erhöht. Dieser Nachteil steht jedoch in keinem Verhältnis zu den Nachteilen der synchronen Kommunikation, weshalb die IPMT asynchron Nachrichten austauschen soll, wie von der FIPA auch empfohlen wird.

Zusammenfassend muss die IPMT Komponente es ermöglichen, Nachrichten zwischen unterschiedlichen Prozessen oder Computern asynchron auszutauschen. Auf welche Weise dies geschieht, wird von der FIPA nicht vorgeschrieben und kann in unterschiedlichen Plattformimplementierungen variieren. FIPA Agenten können deshalb nur auf Plattformen existieren, deren Art der IPMT Realisierung sie unterstützen. Wie diese Komponente in der eigenen Plattform realisiert werden soll, wird erst im nächsten Kapitel entschieden.

Damit Agenten mit unterschiedlichen IPMT Realisierungen trotzdem miteinander kommunizieren können, stellt der ACC die Funktionalität bereit Nachrichten an andere Plattformen weiterzuleiten. Für diese Art der Nachrichtenübertragung wird von der FIPA IIOP als einheitliches Protokoll vorgeschrieben, welches der ACC implementieren soll.

6.1.2 Agent Communication Channel

Die Aufgabe des Agent Communication Channels (ACC) ist das Ausliefern von ACL Nachrichten. Diese Komponente wird seit dem FIPA 2000 Standard nicht mehr als eigener Agent implementiert, ist aber ein sehr zentraler Bestandteil der Plattform und existiert nur einmal auf einer Plattform.

Der ACC unterscheidet zwei unterschiedliche Arten Nachrichten weiterzuleiten. Einerseits werden Nachrichten plattformintern an Empfänger weitergeleitet, die auf der selben Plattform existieren. Dazu sind die Voraussetzungen nötig, dass aus der

Empfängeradresse erkannt werden muss, ob sich der Empfänger auf der selben Plattform befindet und auf die Protokolladresse der IPMT Komponente geschlossen werden kann.

Befindet sich anderseits der Empfänger der Nachricht auf einer anderen Plattform, so wird die Nachricht an den ACC der fremden Plattform weitergeleitet. Dieser ist für die weitere Auslieferung der Nachricht zuständig. Für plattformexterne Kommunikation wird von der FIPA das, auf CORBA basierende, Protokoll IIOP vorgeschrieben. IIOP [Ruh et al. 99] ist ein Protokoll, dass unterschiedlichste CORBA Implementierungen interoperabel macht. Ein Objekt, das über IIOP ansprechbar sein soll, wird einem IIOP unterstützenden ORB bekannt gemacht. Dabei wird eine weltweit eindeutige, standardisierte Adresse (IOR genannt) erzeugt, über die jemand anders ein Proxyobjekt erzeugen und über dies Methoden auf dem Originalobjekt aufrufen kann. Da laut FIPA der ACC die Nachrichten fremder Plattformen entgegennehmen soll muss der ACC an einem IIOP ORB exportiert werden. Es mag gute Gründe der FIPA geben, die Funktionalität des IIOP Protokolls direkt im ACC zu implementieren. Meiner Meinung nach ist die Plattformarchitektur übersichtlicher, wenn nur eine Komponente den physischen Nachrichtentransport regelt. Aus diesem Grund soll die IMPT Komponente den plattforminternen und -externen Nachrichtentransport zumindest physisch regeln, was nicht bedeuten soll, dass der ACC nicht mehr zwischen internem und externem Nachrichtenaustausch zu unterscheiden braucht.

Damit Nachrichten vom ACC an Agenten anderer Plattformen weitergeleitet werden können, muss dem ACC die IORs der ACC dieser Plattformen bekannt sein. Dies könnte auf zwei unterschiedliche Weisen geschehen. Entweder beinhaltet die Empfängeradresse der zu versendenden ACL Nachricht die IOR und hat folgendes Format: „`agent@IOR:3453004352003453400...`“, oder die Empfängerplattform ist dem ACC bekannt und die IOR kann aus dem Namen der Plattform bestimmt werden. Dann könnte die Empfängeradresse das Format „`agent@agentland`“ haben. Genaueres hierzu soll in der Realisierung diskutiert werden.

Zum Empfangen von Nachrichten fremder Plattformen, muss der ACC einem IIOP ORB bekannt gemacht und die dabei erzeugte IOR an andere Agenten oder Plattformen verbreitet werden. Die Art der Verbreitung der IOR ist von der FIPA leider nicht streng vorgeschrieben, so dass es mehrere Ansätze in der Praxis gibt. Eine mögliche Lösung dieses Problems, die Interoperabilität zu anderen Implementierungen sicher stellt, wird auch in der Realisierung im Kapitel 7.2.3 diskutiert.

6.1.3 Agent Management System

Das Agent Management System (AMS), bekannt aus Kapitel 5.2, ist ein Agent, der alle Aktivitäten auf einer Plattform überwacht. Es kann lediglich ein AMS Agent l auf einer Plattform existieren. Zu seinen Aufgaben gehören unter anderem die Kontrolle über die Nutzung des ACC, sowie die Erzeugung und Zerstörung von Agenten.

Am Anfang dieses Kapitels wurde beschrieben, dass die Plattform in einer ersten Implementierung zunächst weder Migration von Agenten, noch die Existenz fremder Agenten zu unterstützen braucht. Kompatibilität zu anderen FIPA Plattformen wird durch den FIPA konformen Versand von ACL Nachrichten an andere Plattformen sicher gestellt. Mit diesen Einschränkungen wird die Funktionalität des AMS Agenten nicht mehr zwingend benötigt, denn die eigenen Agenten können sich auch selbst verwalten, an der Plattform an- und abmelden und ihren Lebenszyklus selbst bestimmen. Dies bringt den Vorteil, dass weder der AMS Agent implementiert werden muss, noch alle Agenten die Kommunikation mit dem AMS Agenten, nach den Regeln des *FIPA Request* Protokolls und der Agent Management Ontologie, zu unterstützen brauchen. Die einzig wichtige Information, die die Plattform über existierende Agenten benötigt, ist die

Zuordnung des Namens des Agenten zu seiner IPMT Adresse, damit Nachrichten vom ACC an den Agenten weitergeleitet werden können. Da diese Information jedoch nur vom ACC benötigt wird, kann dieser diese auch direkt verwalten. Beim Start müsste jeder Agent seinen Namen und seine IPMT Adresse dem ACC bekannt machen, anstatt eine vollständige FIPA konforme Beschreibung beim AMS zu registrieren. Diese einfache Bekanntmachung bräuchte nach keinen speziellen FIPA Kommunikationsregeln geschehen und könnte durch einfachen RPC Aufruf beim ACC realisiert werden.

Die zu realisierende FIPA Plattform würde somit nur noch aus der Kommunikationsschicht (IPMT) und der zentralen Instanz ACC bestehen, was die Kompatibilität zu anderen FIPA-Plattformen, wie beschrieben, nicht einschränkt. Der DF Agent, als weiterer Teil einer FIPA Plattform, ist von diesem Ansatz nur wenig betroffen. Einzige Änderung ist, dass er nicht vom AMS gestartet wird und sich dort auch nicht anmelden muss, sondern explizit gestartet wird und seinen Namen und Adresse beim ACC registriert.

6.1.4 Directory Facilitator

Der vom *Directory Facilitator* (DF) Agenten angebotene Dienst stellt eine Art *Gelbe Seiten* Dienst dar. Andere Agenten melden die von ihnen angebotenen Dienste beim DF Agent an und schlagen bereits registrierte Dienste nach. Der DF Dienst darf nicht mit der dynamischen Bindung von Objekten an einen CORBA ORB gleichgesetzt werden, da der DF Agent nach Registrierungen befragt werden kann, die bestimmten Eigenschaften entsprechen und lediglich diese als Ergebnis liefert. Der DF Dienst ist weder neu noch einzigartig und vergleichbar mit dem JINI *Lookup Service* [Oaks et al. 00], [15] oder des *UDDI* [16] Standards für *Web Services* [17], wenn auch beide deutlich jüngerem Datums sind als der DF Dienst.

Die Datenstrukturen mit denen der Dienst genutzt wird sowie die angebotenen Operationen (*register*, *deregister*, *modify* und *search*) werden von der Agent Management Ontologie definiert. Diese Ontologie wurde schon kurz in Kapitel 5.3.3 vorgestellt und auf sie wird im folgenden Kapitel 6.2.1 noch genauer eingegangen. Der Laut FIPA sollten die DF Registrierungen persistent gespeichert werden, damit sie nach einem Ausfall des DF Agenten weiterhin vorhanden sind. Ein registrierter Agent braucht somit nicht periodisch zu prüfen, ob seine Registrierung noch existiert. Die Speicherung der Daten könnte in einer Datenbank oder einem Directory Service stattfinden.

Meiner Meinung nach, ist dieser Ansatz für einen Einsatz außerhalb einer begrenzten Testumgebung nicht ausreichend, da anzunehmen ist, dass nicht alle Agenten ihre Registrierungen konsequent löschen werden und es sehr schnell zu ungültigen Einträgen kommen wird. Eine mögliche Lösung, wie sie in JINI durch eine Art Leasing stattfindet, bei der nach einem bestimmten Zeitintervall die Registrierung erneuert werden muss, ist von der FIPA noch nicht angedacht. In der zu verwirklichenden Plattform wird soll dieses Problem auch ausgeklammert und optimistisch davon ausgegangen werden, dass jeder Agent seine Registrierung löscht, bevor er beendet wird.

Ein weitere Fähigkeit, die der DF Agent neben der Registrierung und Suche nach anderen Agenten unterstützen muss, ist die *Federated Search*. Dabei wird eine Suchanfrage an alle weiteren, dem DF bekannten DF Agenten weitergeleitet. Das Suchergebnis aller Agenten wird, bereinigt von doppelten Einträgen, an den anfragenden Agenten zurückgegeben. Somit wird die Wahrscheinlichkeit, einen der Suchanfrage entsprechenden Dienst zu finden, stark erhöht, und der Agent braucht nicht zwingend mehrere DF Agenten zu kennen und zu befragen. Hieraus ist zu folgern, dass der DF Agent seinen eigenen Dienst, angeboten von anderen Agenten, nutzen können muss.

Die Kommunikation mit dem DF Agenten findet über das *FIPA Request* Protokoll statt. Dieses einfache Protokoll muss vom Nutzer des DF Dienstes und vom DF selbst unterstützt werden. Eine mögliche Abbildung und Nutzung der Protokolle allgemein und speziell des *FIPA Request* Protokolls wird im nächsten Abschnitt beschrieben.

Somit sollte das Design einer FIPA konformen Agentenplattform, die den Anforderungen aus Kapitel 4 entspricht, vollständig sein. Sie sollte alle Anforderungen erfüllen, so dass Informationsagenten untereinander Informationen anbieten und austauschen können. Zusammenfassend verweise ich noch einmal auf Abbildung 6-1, die das Design der bisher beschriebenen Plattform zeigt.

Bisher wurde lediglich Aufbau und Funktionalität der Agentenplattform besprochen, doch der FIPA Standard regelt auch die Grundlagen zur Kommunikation und definiert Sprachen und Protokolle, die es Agenten ermöglichen, die untereinander ausgetauschten Nachrichten zu verstehen. Mit diesen Teilen des Standards beschäftigt sich der folgende Abschnitt.

6.2 FIPA Kommunikation

Mit Kommunikation ist in diesem Abschnitt nicht mehr der physische Austausch von Nachrichten gemeint, sondern die Reihenfolge, der Aufbau und die inhaltliche Struktur der Nachrichten, die untereinander ausgetauscht werden. Dabei wird zunächst der Aufbau von ACL Nachrichten beschrieben, gefolgt von der Darstellung ihres Inhaltes. Abschließend wird auf die Abbildung der FIPA Protokolle eingegangen.

6.2.1 Nachrichten

Die Kommunikation zwischen FIPA Agenten basiert auf dem Austausch von ACL Nachrichten. Diese werden zumeist im Stringformat untereinander ausgetauscht. In der aktuellsten Version des FIPA Standards ist eine XML Darstellung der ACL vorgesehen, die jedoch bislang von keiner weiteren Plattform unterstützt wird und aus diesem Grund zunächst vernachlässigt werden kann. Ein FIPA Agent muss in der Lage sein, ACL Strings zu verstehen und zu erzeugen. Generell ist es recht kompliziert auf die Informationen einer Zeichenkette zuzugreifen, weshalb Nachrichten bekannten Formats (hier die Agent Communication Language) geparst und in eine interne Darstellung umgewandelt werden. In einer objektorientierten Sprache würde eine ACL Nachricht durch ein Objekt oder Objektbaum, bestehend aus unterschiedlichen Objekten, repräsentiert werden, die die Informationen der Stringnachricht enthalten. Für das Umwandeln einer ACL Nachricht aus einem String in eine interne Darstellung und zurück werden demnach Parser und Composer benötigt.

Eine ACL Nachricht ist so definiert, dass sie Informationen über die Bedeutung ihres Inhalts (performativer Akt), sowie zu dessen Darstellung besitzt. Der Inhalt selbst (Content), wird durch sie jedoch nicht beschrieben, sondern als einfacher String dargestellt. Dieser Ansatz macht eine ACL Nachricht unabgängig von der Information, die mit ihr ausgedrückt werden soll. Das Format des Contentstrings wird durch die speziellen ACL Parameter `language` und `ontology` beschrieben. Inwiefern ein Agent in der Lage ist den Content einer ACL Nachricht zu verstehen, hängt von den Werten dieser Parameter ab.

6.2.2 Content der Nachrichten

Der Content einer ACL Nachricht wird durch eine Ontologie definiert, die den semantischen Inhalt unabhängig von der syntaktischen Darstellung beschreibt. Je nach Dienst, den der Agent anbietet oder nutzt, muss der Content der ACL Nachrichten der

jeweiligen Ontologie entsprechen. Mit ihr werden zum Beispiel gültige Aktionen und Daten eines Dienstes definiert. Ähnlich der ACL muss auch der Content auf eine ontologieabhängige, interne Darstellung abzubilden sein, die dem Agenten einfachen Zugriff auf die Daten der Nachricht ermöglicht. Zusätzlich zu dieser Abbildung werden Interpreter und Composer zu Umwandlung benötigt.

Agent Management Ontologie

Die Agent Management Ontologie beschreibt den Content, der von AMS und DF Agent verstanden wird. Die Struktur dieser Ontologie ist bereits aus 5.3.3 bekannt und soll hier nicht wiederholt aufgeführt werden. Im letzten Abschnitt wurde festgelegt, dass zunächst nur der DF Agent realisiert werden muss. Dies wirkt sich in sofern auf die Abbildung der AM Ontologie aus, als dass alle Plattform und AMS spezifischen Strukturen und Aktionen nicht verstanden und implementiert werden brauchen. Es wird nur jener Teil der Ontologie benötigt, der von DF Agenten verwendet wird.

Semantic Language

Eine Ontologie beschreibt lediglich die Semantik eines Ausdrucks, unabhängig von der syntaktischen Darstellung. Eine Sprache zur Darstellung einer Ontologie ist die Semantic Language (SL), die bereits in 5.3.2 vorgestellt wurde, und ihre Untermengen SL0 - SL2. Die SL beschreibt die syntaktische Struktur einfacher logischer Ausdrücke und allgemeiner Strukturen, wie Aktionen, Sets oder Terme. Basierend darauf können mit der SL Ontologien auf einen String abgebildet werden.

Die Verarbeitung des Contents einer ACL Nachricht sieht demnach folgendermaßen aus. Der SL String wird von einem SL Parser geparst und in eine interne SL Darstellung umgewandelt. Abhängig von der Ontologie des Contents wandelt ein spezieller Interpreter diese SL Darstellung in eine ontologieabhängige Darstellung um, die vom Agenten verarbeitet werden kann. Die Erzeugung eines SL Strings geschieht invers, wie der folgenden Abbildung zu entnehmen ist.

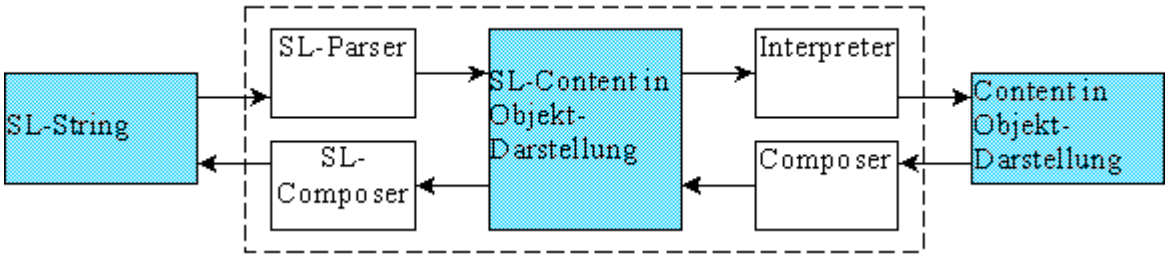


Abbildung 6-2: Verarbeitung des Contents einer ACL Nachricht

Vorteil dieser Trennung ist, dass die selbe Aussage (Semantik), in unterschiedlichen Darstellungen, durch verschiedene Contentsprachen, ausgedrückt werden kann. Theoretisch könnte der Content einer ACL Nachricht auch in KIF (siehe 3.2.1) dargestellt werden. Wäre der Empfänger in der Lage einen KIF String parsen zu können und würde er über einen entsprechenden KIF-Ontologieinterpreter verfügen, kann der Inhalt der Nachricht trotz unterschiedlicher Darstellung zu FIPA SL verstanden werden.

Nachrichten, die an den DF Agent geschickt werden, müssen auf obige Weise verarbeitet werden. Die AM Ontologie ist so einfach definiert, dass sie problemlos in SL0, der kleinsten SL Untermenge, abgebildet werden kann. Es wird also ein SL0 Parser und Composer benötigt, die den Content in eine interne SL0 Darstellung und aus dieser Darstellung in einen String umwandeln, und ein AM Interpreter, der aus der SL0 Darstellung die AM Semantik interpretieren kann.

An dieser Stelle sind ist es einem FIPA Agenten möglich einzelne ACL Nachrichten zu empfangen, zu verstehen und zu versenden. Doch viele Handlungen können nur schwer durch den Versand einzelner Nachrichten erledigt werden. Oftmals möchte der Sender über den Ausgang einer Anweisung oder dessen Ergebnis informiert werden. Damit diese Dialoge geregelt ablaufen, werden sie durch FIPA Protokolle beschrieben.

6.2.3 FIPA Protokolle

FIPA Protokolle (siehe Kapitel 5.4) werden benötigt, um Dialoge nach festgelegten Regeln führen zu können. Sie ermöglichen Agenten beliebig komplexe Aufgaben untereinander zu erledigen. Ein FIPA Protokoll ist eine Definition mehrerer, aufeinanderfolgender, zusammengehöriger ACL Nachrichten, deren performativer Akt genau festgelegt ist. Ein Dialog zwischen verschiedenen Agenten, der nach einem Protokoll geführt wird, soll in Zukunft Konversation genannt werden. Da Agenten miteinander asynchron kommunizieren, brauchen sie auf keine direkt folgende Antwort zu warten, nachdem sie eine Nachricht versendet haben. Sie können in der Zwischenzeit andere, offene Aufgaben erledigen, wie weitere Konversationen bearbeiten. Diese Flexibilität bedeutet jedoch, dass eingehende ACL Nachrichten einer speziellen Konversation zugeordnet werden müssen, um sie richtig verarbeiten zu können. Diese Zuordnung geschieht durch spezielle Parameter der ACL Nachricht, die das Protokoll und die ID der Konversation beschreiben.

Betrachtet man die unbestimmt eingehenden Nachrichten als Teil ihrer Umgebung, so könnte man sagen, dass der Agent auf seine Umwelt reagiert, indem er Nachrichten entgegennimmt und verarbeitet. Die Fähigkeit, Nachrichten entgegenzunehmen, könnte man mit einem Sensor vergleichen. Dieser etwas abstrakte Ansatz soll einen FIPA Agenten nicht als reaktiv darstellen, das wäre etwas zu gewagt, da nur Nachrichten verarbeitet werden, die an ihm adressiert sind. Vielmehr soll an den, in der Agentenwelt weit verbreiteten, verhaltensbasierten Ansatz von Rodney Brooks und der Subsumption Architektur erinnert werden, siehe [Brooks 90]. In diesem Ansatz wird davon ausgegangen, dass ein Agent seine Umwelt durch Sensoren wahrnimmt. Verschiedenste Behaviours beeinflussen die Aktionen des Agenten, abhängig von den Daten, die von den Sensoren geliefert werden. Sie werden parallel ausgeführt und beeinflussen das Gesamtverhalten des Agenten. Unterschiedliche Prioritäten der einzelnen Behaviours regeln die Auswirkung auf das Gesamtverhalten. Als Beispiel wäre ein Roboter zu nennen, dessen Aufgabe es ist, einen Korridor entlag zufahren, ohne an Wände anzustoßen. Er wird durch zwei Behaviours (durch Prozesse verwirklicht) gesteuert; das eine lässt ihn immer geradeaus fahren; das andere ändert die Fahrtrichtung, wenn eine Wand zu nahe kommt. Ändert der Richtungswechsel nicht schnell genug den Abstand zur Wand, wird die Vorwärtsbewegung verlangsamt, die Drehung jedoch fortgeführt. Damit das Gesamtverhalten funktioniert und Konflikte unter den Handlungen der einzelnen Behaviours vermieden werden, haben beide unterschiedliche Prioritäten. In diesem Fall besteht ein Konflikt, wenn das eine Behaviour den Roboter verlangsamen will, da eine Wand im Weg steht, das andere jedoch den Roboter mit konstanter Geschwindigkeit weiter geradeaus fahren lassen will.

Diese Architektur könnte vereinfacht ein interessanter Ansatz sein, die FIPA Protokolle abzubilden. Werden die Prioritäten der einzelnen Behaviours außer Acht gelassen, könnte ein Dienst, dessen Nutzung auf einem bestimmten Protokoll basiert, durch eine Art Behaviour abgebildet werden. Das Behaviour verwaltet alle laufenden Konversationen und führt bei jeder ankommenden Nachricht eine, vom Zustand der Konversation abhängige, durch den Dienst festgelegte Aktion aus.

Eventuell ist es möglich, die Verwendung eines Protokolls durch ein Behaviour soweit kapseln zu können, dass die Implementierung unterschiedlicher Dienste, die dasselbe Protokoll nutzen, sehr einfach und frei von der Protokolllogik geschehen kann. Getestet werden kann dieser Ansatz zum einen beim DF Agenten, dessen Kommunikation über das *FIPA Request* Protokoll abgewickelt wird, und zum anderen natürlich beim Informationsagenten, bei dem noch nicht feststeht welches Protokoll verwendet wird.

Somit sind alle wichtigen Aspekte der Kommunikation und der Plattformarchitektur des FIPA Standards besprochen. Basierend auf diesem Design kann nun der Informationsagent detailliert spezifiziert werden.

6.3 Informationsagent

Das Ziel ist einen kooperierenden Informationsagenten zu verwirklichen, der ein Experte in einen begrenzten Themengebiet ist. Kooperation soll ermöglichen, sehr viele Informationen anbieten zu können, die über das eigene Wissen hinausgehen. Er soll die von ihm angebotene Information verstehen, damit sehr präzise Anfragen gestellt werden können.

In Kapitel 4 wurde bereits ein Ansatz skizziert, der zeigt, auf welche Weise Informationen unter Agenten ausgetauscht werden können. Zusammengefasst sieht dieser Ansatz so aus, dass ein Agent von seinem Benutzer nach Informationen gefragt wird. Kann er diese nicht selbst liefern, werden weitere Agenten befragt, die eventuell über die gewünschte Informationen verfügen. Ein möglicher Ablauf dieses Prozesses wurde in den Abbildungen auf den Seiten 19f beschrieben und soll nun konkretisiert werden. In den folgenden Abschnitten werden dazu die Unklarheiten, die dieser Ansatz noch beinhaltet und die bisher ausgeklammert wurden, diskutiert. Lösungswege auf Basis des FIPA Standards sollen gefunden werden.

Zusätzlich soll angemerkt werden, dass von einer friedlichen Umgebung ausgegangen wird, in der alle Informationsagenten existieren. Es wird erwartet, dass alle Agenten gewillt sind, nach ihren Möglichkeiten eine sinnvolle Aktion auszuführen. Die Möglichkeit, dass Informationsagenten existieren könnten, die störende und ungewollte Aktivitäten ausführen, wird ausgeschlossen.

Um einen Überblick zu erhalten, werden die wichtigsten zu klärenden Probleme vorweg schon einmal aufgeführt, bevor sie im Detail diskutiert werden:

- Wie werden Informationsagenten gefunden, die möglicherweise die gesuchte Information anbieten können?
- Wie kann der Ablauf einer Konversation mit den Mitteln des FIPA Standards verwirklicht werden?
- Wie sieht der Content der auszutauschenden ACL Nachrichten aus?
- Und zuletzt, wie werden die Informationen dargestellt?

Der erste Schritt eines Agenten ist das Auffinden von Informationsagenten, die in frage kommen, die gesuchte Informationen anzubieten. Da hierzu einige Kenntnisse über den Informationsdienst benötigt werden, soll dieser Schritt erst an späterer Stelle beschrieben werden. Zunächst soll überlegt werden, wie der zu verwirklichende Informationsdienst im Detail aussehen könnte.

6.3.1 Definition des Informationsdienstes

Bisher wurde lediglich von Informationsagenten und von auszutauschenden Informationen gesprochen, jedoch wurde außer Betracht gelassen, welche Art von

Informationen ausgetauscht werden sollen und in welchem Format diese dargestellt werden könnten. Im Prinzip gibt es zwei unterschiedliche Ansätze der Informationsbeschreibung.

Einer ist schon vom DF Agenten bekannt, wobei alle Daten und Aktionen durch eine spezielle Ontologie beschrieben wurden. Mit diesem Ansatz können spezielle Dienste, wie Hotelinformations-, oder TV-Programmdienste angeboten werden, indem für jeden Dienst eine eigene Ontologie definiert wird, die alle benötigten Daten beschreibt. Nachteil ist jedoch, dass Agenten somit spezielles Wissen über einen Dienst haben müssen, um ihn nutzen zu können. Die Ontologie, die einen Dienst beschreibt, muss standardisiert und allen Agenten bekannt sein, ansonsten ist eine erfolgreiche Konversation nicht möglich. Vorteil ist, dass bei diesem Ansatz sehr präzise Anfragen gestellt und entsprechend präzise Informationen geantwortet werden können.

Ein anderer Ansatz wäre eine allgemeine Ontologie zu definieren, die Anfragen und Vorschläge unabhängig von der Art der Informationen beschreibt, nicht aber die auszutauschende Information. Diese könnte dann unterschiedlichste Formate und Inhalte haben, müsste lediglich in einem String abzubilden sein. Sie könnte aus einer HTML Seite, deren Inhalt nur von Menschen zu verstehen ist, einem XML Dokument, einem Prolog-Ausdruck oder einem Satz einer speziellen Ontologiedefinition, dargestellt durch die SL, bestehen. Somit würde das Problem des Verstehens der Information auf den Benutzer des Informationsdienstes verlagert werden. Vorteil wäre, dass jegliche Art von Informationsagenten miteinander kommunizieren könnten. Nachteil ist, dass die angeforderte Information eventuell vom Nutzer nicht verstanden werden könnte. Dies könnte verhindert werden, wenn die beim DF registrierte Beschreibung des Anbieters um das Format der angebotenen Information erweitert wird. Der suchende Agent würde somit nur mit Anbietern verhandeln, deren Informationsdarstellung er auch versteht. Werden unterschiedliche Formate vom Benutzer verstanden, könnte beispielweise mit Agenten verhandelt werden, die Bilder liefern und mit anderen, die HTML Seiten anbieten. Eine solche beispielhafte Registrierung ist in der folgenden Abbildung 6-3 zu finden.

```
(df-agent-description
  :name
    (agent-identifier
      :name  hugo@agentland.com
      :addresses (sequence  iiop://agentland.com:9000/acc))
  :language (set fipa-sl0)
  :ontology (set fipa-agent-management)
  :protocol (set fipa-request)
  :services (set
    (service-description
      :name information-service
      :type information
      :ontology knowledge
      :properties (set
        (property
          :name context
          :value plant)
        (property
          :name result-type
          :value html)))))
```

Abbildung 6-3: Beispiel DF Registrierung

Der Ansatz, jegliche Art von Informationen durch allgemeine Anfragen und Vorschläge anfordern und auswählen zu können, schränkt die Präzision der Beschreibung der Informationen stark ein, da Anfragemöglichkeiten nicht mehr themenspezifisch dargestellt werden können. Eine mögliche Anfrage, die unabhängig von

Themenkenntnissen ist, könnte zum Beispiel ein Begriff sein. Eine Anfrage hätte somit die Bedeutung: „Welche Information hast du zum Thema Segeln?“. Ein Vorschlag könnten ein oder mehrere weitere Begriffe sein, die die verfügbare Information beschreiben.

Es ist also eine Entscheidung zu treffen, ob der Informationsagent sehr spezielle Informationen anbieten oder ein allgemeiner Informationsdienst realisiert werden soll. Beide Ansätze sind so unterschiedlich, dass Vor- und Nachteile nur schwer gegeneinander abzuwägen sind. In Kapitel 4.1 wurde an den heutigen Informationsagenten kritisiert, dass Suchanfragen nicht präzise genug eingeschränkt werden können. Dieses Problem besteht mit dem letzten Ansatz leider auch. Der erste Ansatz wird im Prinzip schon durch den DF Agenten und der Agent Management Ontologie realisiert. Der Informationsagent müsste lediglich eine weitere Ontologie definieren, die einen weiteren Dienst, zum Beispiel über Hotelinformationen, definiert und geeignete Suchanfragen und Vorschläge abbildet. Da es nicht sehr sinnvoll ist, den selben Ansatz zweimal zu implementieren, soll der zu verwirklichende Informationsdienst nach dem letzten Ansatz, trotz der erwähnten Einschränkungen realisiert werden.

6.3.2 Auffinden der Informationsanbieter

Aus dieser Entscheidung ergibt sich eine Konsequenz, die sich auf die Auswahl möglicher Informationsanbieter auswirkt. Sollen bestimmte Informationen zu einem Begriff ausfindig gemacht werden, muss sicher gestellt werden, dass beide Kooperationspartner über den selben Begriff verhandeln. So hat zum Beispiel der Begriff *Blatt* für jemanden, der an Bücher denkt, eine völlig andere Bedeutung, als für jemanden, der an Pflanzen denkt. In Kapitel 4.2 wurde schon bemerkt, dass sich diese Art der Informationsagenten von den bisherigen darin unterscheidet, dass jeder Agent über Spezialkenntnisse zu einem oder mehreren Themenbereichen verfügen soll, anstatt Informationen zu allen Bereichen liefern zu können. Da es jedoch zu Doppeldeutigkeiten kommen kann, wenn Anbieter lediglich nach Informationen zu einem Begriff gefragt werden, muss jeder Informationsanbieter einem Themenbereich zugeordnet werden, anhand dessen ein nach Informationen suchender Agent eine Vorauswahl der Anbieter treffen kann. Auf diese Weise würden nur Agenten zum Begriff *Blatt* befragt, die Informationen im Bereich Biologie oder Pflanzen anbieten.

Dieser Ansatz wirft jedoch einige weitere Probleme auf. Es muss geklärt werden, welche Themenbereiche überhaupt existieren und unter welchem Bereich Agenten zu suchen sind, wenn Informationen gesucht werden. Ein möglicher Ansatz wäre, alle Bereiche durch einen einfachen Ontologiebaum zu beschreiben. In diesem Fall wird der Begriff Ontologie in seiner ursprünglichen Bedeutung verwendet, die eine Begriffswelt beschreibt. Die FIPA nutzt den Begriff eher, um Datenstrukturen und Aktionen unabhängig von ihrer syntaktischen Darstellung zu definieren. Ein beispielhafter Ontologiebaum wird in Abbildung 6-4 gezeigt. Er verbindet Begriffe, die miteinander in Beziehung stehen. In diesem Fall deuten die Beziehungen eine Spezialisierung des Bereiches an. In der Theorie der Ontologien gibt es die verschiedensten Arten von Beziehungen. Dass dieses Thema schnell sehr komplex werden kann, zeigt eine beispielhafte Beziehung zwischen den Begriffen *Getreide* und *Nagetier*, deren Bedeutung eher durch *frisst*, *benötig* oder allgemeiner *hat etwas zu tun mit* ausgedrückt werden kann. Hier soll nur von einer Art der Beziehungen zwischen den Begriffen einer Spezialisierung ausgegangen werden. Weitere Informationen über die Theorie der Ontologien und ihrer Einsatzmöglichkeiten können unter [18] gefunden werden.

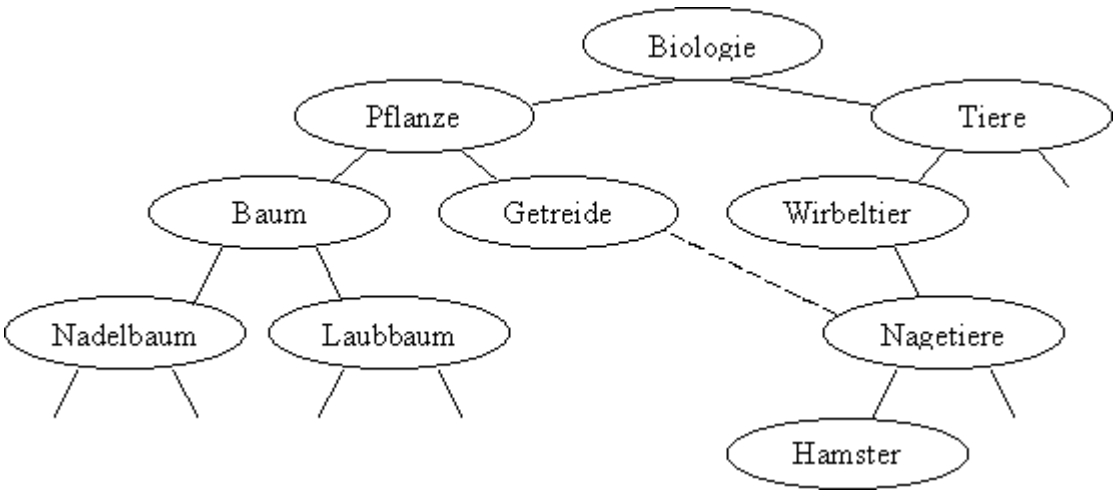


Abbildung 6-4: Darstellung eines einfachen Ontologiebaums

Ein sogenannter Themenbereich, der einen Knoten im Ontologiebaum darstellt, wird auch als Kontext bezeichnet. Da nicht auszuschließen ist, dass ein Begriff in einem Ontologiebaum doppelt vorkommen könnte, soll die Darstellung eines Kontextes die hierarchische Struktur des Baums widerspiegeln und in Anlehnung an die Domänenstruktur des Internets den vollen Pfad beinhalten. Ein gültiger Kontext würde zum Beispiel durch *Biologie.Pflanze.Baum* dargestellt.

Die Verwendung eines Kontextes hat zunächst den Vorteil Agenten aufzufinden, die Experten auf einem Gebiet sind. Er bietet den weiteren Vorteil, dass zusätzlich Agenten befragt werden können, die eigentlich Experten in spezielleren oder allgemeineren Kontexten sind, wahrscheinlich die Information aber auch liefern könnten. Neben allen Agenten, die dem Kontext *Biologie.Pflanze* zugeordnet sind, könnte es auch sinnvoll sein, alle Agenten nach dem Begriff *Blatt* zu befragen, die dem Kontext *Biologie* oder dem spezielleren Bereich *Baum* zugeordnet sind.

Ein Problem dieses Ansatzes ist jedoch, dass die Struktur eines solchen Baums allen Agenten bekannt sein muss, damit er sinnvoll genutzt werden kann. Da unsere Welt zu komplex ist, um sie in einem Kontextbaum abzubilden, verschärft sich dieses Problem zusätzlich, da unterschiedlichste Bäume zur Verfügung stehen müssten. Hier wurde ein Auszug aus der Biologie als Beispiel verwendet, denkbar wären Ontologien zu jedem erdenklichen Bereich, wie Musik oder Bücher, um auf den anfangs gewählten Begriff *Blatt* zurückzukommen. Dieses Problem besteht nicht nur in diesem Ansatz, sondern allgemein, wenn Informationen zu beliebigen Themen verwaltet werden sollen und kann nur durch Standardisierung gelöst werden.

Dass dies Problem auch in der realen Welt ernsthaft ist, ist bei gängigen Suchmaschinen, wie Altavista [19] oder Yahoo [20], zu erkennen. Sie bieten neben der allgemeinen Suchfunktion auch ein Portal an, dass aus einem Baum besteht, der mit allgemeinen Themen, wie Unterhaltung und Computer, beginnt und mit speziellen Themen, wie Popmusik oder Agententechnologie abschließt. Es wird versucht eine Vielzahl der denkbaren Themenbereiche in diesem Baum darzustellen, der natürlich lückenhaft und grob gegliedert ist. Die in einem Knoten angebotenen Informationen sind Links zu Internetseiten, die manuell, nicht von der Suchmaschine, eingefügt wurden. Betrachtet man die Indizes unterschiedlicher Suchmaschinen, stellt man fest, dass sie unterschiedliche Strukturen haben und nicht standardisiert sind. Sollen Agenten oder andere Programme eigenständig die Informationen in einem solchen Baum pflegen, müsste ihnen die Struktur bekannt sein, sprich der Baum standardisiert sein.

Da das Problem der Mehrdeutigkeit eines Begriffs anders kaum zu lösen ist, soll dieser Ansatz, trotz seiner Probleme verwendet werden. In einem Beispielszenario wird ein eigens definierter Kontextbaum ausreichend sein.

Zur Umsetzung dieses Ansatzes bietet sich der DF Dienst der FIPA Plattform an. Informationsanbieter registrieren sich beim DF unter einem zuvor definierten Kontext. Nach Informationen suchende Agenten durchsuchen den DF Dienst und schränken die Suche nach Anbietern eines bestimmten Kontextes ein. Diese zusätzliche Information wird durch die AM Ontologie leider nicht direkt unterstützt. Für solche Fälle kann jedoch ein `property` Konstrukt genutzt werden, in dem zusätzliche Daten durch ein *name-value* Paar angegeben werden können. Eine beispielhafte DF Registrierung zeigt dies in Abbildung 6-3.

6.3.3 Ablauf einer Konversation

Gehen wir davon aus, dass eine Menge möglicher Informationsanbieter gefunden wurde. Die nun folgende Kommunikation mit den potentiellen Informationsanbietern muss, um erfolgreich geführt werden zu können, in Regeln gefasst werden. Aus Sicht eines Benutzers dieses Dienstes ist der erste Schritt, jeden Informationsanbieter nach Vorschlägen zu der gesuchten Information zu befragen. Diese erhaltenen Vorschläge werden bewertet, und nach Auswahl des besten wird die dem Vorschlag entsprechende Information angefordert.

Zur Regelung eines solchen Ablaufs eignet sich das *FIPA Request* Protokoll, das die Kommunikation mit dem DF Agenten regelt, nicht. Alternativ könnte das *FIPA Iterated Contract Net* Protokoll aus Kapitel 5.4.2 genutzt werden. Es wurde für den Anwendungsfall konzipiert, dass zunächst Vorschläge eingeholt werden, die daraufhin angenommen oder abgelehnt werden. Solange beide Kommunikationspartner sich an das ICN Protokoll halten, kann eine gültige Konversation geführt werden.

6.3.4 Semantik der beteiligten Nachrichten

Für eine erfolgreiche Konversation ist nun zu überlegen, wie der Content der ACL Nachrichten, die an einer Konversation beteiligt sind, aussehen könnte. Dazu muss überlegt werden, was die einzelnen Nachrichten semantisch ausdrücken sollen.

Die erste Nachricht, die an den oder die Informationsanbieter geschickt wird, fragt nach Vorschlägen zu einem oder mehreren Begriffen. Durch das verwendete Protokoll wird dies durch den Typ der ACL Nachricht, einem *call for proposal* (`cfp`), ausgedrückt. Es reicht aus, wenn der Content dieser Nachricht lediglich den oder die gesuchten Begriffe abbildet.

Die erwartete Antwort muss dem Typ `propose` entsprechen und bietet Vorschläge über vorhandene Informationen an. Ein solcher Vorschlag könnte wiederum durch einen einfachen Begriff dargestellt werden. Auf den ersten Blick scheint es nutzlos, wenn nach dem Begriff *Segeln* gefragt wird und als Vorschlag mit dem Begriff *Segeln* geantwortet wird (abgesehen von der Tatsache, dass der Agent genau zu diesem Thema etwas beitragen kann). Interessant wird diese einfache Darstellung aber, wenn der Agent zum Thema Segeln nichts anzubieten hat, doch Informationen zu einem ähnliches Thema, wie *Wasserski* oder *America's Cup*, anbieten kann. Jemand, der an Informationen zum Thema Segeln interessiert ist, scheint an Wassersport interessiert zu sein. Es ist anzunehmen, falls zum Thema Segeln keine Informationen gefunden werden konnten, dass er auch am Thema *Wasserski* interessiert sein könnte. Der *America's Cup* ist eine große Segelregatta und kann entsprechend zum Thema Segeln interessant sein.

Nachdem alle Vorschläge eingegangen sind, wählt der Agent den seiner Meinung treffendsten aus und schickt eine Ablehnung oder Annahme der Vorschläge an alle Agenten. Dies wird durch die vom Protokoll definierten ACL Nachrichtentypen `accept-proposal` und `reject-proposal` ausgedrückt. Der Content der ablehnenden Nachrichten könnte derselbe wie der der ursprünglichen `propose` Nachricht sein, der Content der annehmenden Nachricht könnte durch den ausgewählten Vorschlag dargestellt werden.

Wie oben festgelegt, braucht der Content, der abschließenden `inform` Nachricht hier nicht berücksichtigt zu werden, da dieses Format abhängig von der auszutauschenden Information ist.

Aus den bisher erfolgten Überlegungen kann gefolgert werden, dass der Content der an einer Konversation beteiligten ACL Nachrichten recht einfach gehalten werden können und lediglich einzelne Begriffe ausdrücken muss. Die Sprache SL0 scheint mächtig genug, um solchen Content syntaktisch darstellen zu können. Eine Ontologie, die die obigen Überlegungen abbildet, soll erst bei der Realisierung des Agenten spezifiziert werden.

6.4 Folgerungen

Alles in allem konnte ein Design gefunden werden, dass die meisten Überlegungen aus Kapitel 4 widerspiegelt. Es wurden einige Kompromisse eingegangen und Einschränkungen gemacht, die notwendig waren, um die Realisierung nicht zu aufwendig werden zu lassen. Gemeint sind Entscheidungen, wie die Plattform nur nach äußerlich FIPA konform zu halten und den AMS Agenten nicht zu verwirklichen. Zu einigen unvorhergesehen auftretenden Problemen sind Lösungsansätze diskutiert und gefunden worden, wie die Verwendung eines Kontextbaums, um einen Informationsagenten seinem Themengebiet zuordnen zu können. Der hier gewählte Ansatz einen Informationsdienst zu verwirklichen, der unabhängig von der Art der angebotenen Informationen ist, nimmt zwar bewusst eine schlechte Beschreibungsfähigkeit der gesuchten Information in Kauf, scheint es doch interessanter zu sein, als einen Dienst zu realisieren, der abhängig von der anzubietenden Information ist, wie einen Hotelauskunftsdienst. Im Hinblick auf zukünftige, komplexere Anwendungen enthalten die hier gewählten Ansätze noch einigen Stoff für Diskussionen und müssen eventuell weiter durchdacht werden.

Für die Realisierung dieses Ansatzes sehe ich zur Zeit keine großen Probleme, da viele mögliche Ungereimtheiten, gerade beim Informationsagenten, detailliert durchdacht sind. Vorhandene Implementierungen des FIPA Standards zeigen, dass er prinzipiell verwirklichbar ist. Die Entscheidung eine eigene Plattform zu implementieren ist durch den Ansatz der Registrierung der Informationsagenten unter einem bestimmten Kontext bekräftigt worden. Eine solche Funktionalität könnte nur sehr schwer mit einer fremden Plattformimplementierung angeboten werden.

Recht interessant kann der Ansatz der Abbildung der FIPA Protokolle durch Behaviours sein. Er verspricht, sehr viel Funktionalität zu kapseln, von dessen Implementierung ein Agent verschont wird. Möglicherweise braucht ein Agent durch diesen Ansatz nur dienstrelevante Teile zu implementieren.

7 Realisierung

In diesem Kapitel soll gezeigt werden, ob und wie die Spezifikationen und Ansätze des letzten Kapitels zu realisieren sind. Es wird auf die soeben beschriebenen Aspekte noch einmal, jedoch auf einer anderen, detaillierteren Ebene eingegangen und gezeigt, aus welchem Themenbereich gewisse Probleme eigentlich kommen und welche Lösungsansätze zur Verfügung stehen.

Im Gegensatz zum letzten Kapitel wird in diesem Kapitel zuerst mit der Realisierung der FIPA Sprachen und Protokolle begonnen, gefolgt von den Komponenten der FIPA Plattform. Die hier gewählte Gliederung unterscheidet sich vom Design, weil zur Realisierung einzelner Plattformkomponenten die Implementierung von Ontologien und Protokollen vorausgesetzt wird. Zudem ist die grobe Struktur des zu realisierenden Teils des FIPA Standards bekannt und genügend Überblick vorhanden, um eine detailliertere Beschreibung verstehen zu können.

Bevor auf Details eingegangen wird, noch ein paar allgemeine Anmerkungen zur Realisierung. Diese Arbeit ist in der Programmiersprache Java implementiert worden, um die bekannten Vorteile, wie Plattformunabhängigkeit, Speichermanagement und die große Klassenbibliothek zu nutzen. Zum Verständnis werden lediglich Grundkenntnisse vorausgesetzt, ansonsten sind sie z.B. in [Flanagan 99] zu erwerben. Andere Produkte, Standards und Protokolle, die benutzt werden, werden in ihrer Funktionalität ausreichend erklärt.

In einigen Fällen wird angesprochen in welchen Java Paketen vorgestellte Klassen zu finden sind. Dieses Prinzip wird nicht durchgängig beibehalten, da es an vielen Stellen unpassend ist zu jeder Klasse auf das entsprechende Paket hinzuweisen. In Anhang A wird deshalb die Paketstruktur der hier implementierten Klassen vollständig vorgestellt.

Das Fehlermanagement im einzelnen soll hier ausgeklammert werden, obwohl es ein interessanter und wichtiger Punkt ist. Jedoch würde es an wichtigen Stellen die Beschreibung des allgemeinen Ablaufs stark verkomplizieren und unübersichtlich machen. Für genauere Informationen in diesem Bereich verweise ich auf den Quellcode.

7.1 FIPA Kommunikation

Zunächst einmal wird die Abbildung der Sprachen ACL und SL0 sowie der Agent Management Ontologie beschrieben. Eine grundlegende Entscheidung über das Parsen dieser Sprachen, bzw. interpretieren dieser Ontologie ist vorweg anzumerken. Parser für die formalen Sprachen, wie ACL und SL0 könnten durch die Parsergeneratoren *lex* und *jacc* [Levine et al. 92] oder die entsprechende Java Variante *javacc* [21] generiert werden. Da jedoch bisher noch keine Erfahrungen mit diesen Tools gesammelt wurden und zumindest die ACL Grammatik sehr einfach ist, wird auf die Verwendung verzichtet und die Funktionalität manuell implementiert.

Obwohl das *FIPA Iterated Contract Net* Protokoll von keiner Komponente der Plattform, sondern nur vom Informationsagenten verwendet wird, soll es in diesem Kapitel zusammen mit dem *FIPA Request* Protokoll beschrieben werden. Das Abbilden und Verwalten der einzelnen Konversation beider Protokolle ist nach den Vorüberlegungen aus Abschnitt 6.2.3 recht komplex und leichter zu verstehen, wenn es zusammenhängend an einer Stelle mit einem einfachen und einem etwas komplizierteren Protokoll dargestellt wird.

7.1.1 Agent Communication Language

Eine ACL Nachricht wird durch die Klasse `ACLMessage` aus dem Paket *fipa.lang.acl*, abgebildet. In diesem Paket sind alle weiteren Klassen zu finden, die im Zusammenhang mit ACL Nachrichten benötigt werden. Die Inhalte der Parameter einer Nachricht, wie `:sender` oder `:receiver`, werden in gleichnamigen Instanzvariablen gespeichert. Der Typ (performativer Akt) einer Nachricht wird durch die zusätzliche Variable mit dem Namen `type` abgebildet. Der komplexe Typ `agent-identifier` beschreibt den Sender oder Empfänger einer ACL Nachricht und wird durch eine gleichnamige Klasse repräsentiert.

Das Parsen von Sätzen einer Sprache ist ein schon lange und stark durchdachtes Thema aus dem Bereich des Compilerbaus. Im Folgenden werden einige Begriffe und Ansätze aus diesem Bereich genutzt, die bei Unkenntnis in [Aho et al. 88] nachgeschlagen werden können. ACL Nachrichten werden zunächst nur im String Format übertragen. Um aus einem ACL String ein `ACLMessage` Objekt zu erzeugen, muss der String geparst werden. Die Grammatik der ACL ist kontextfrei und durch einen LL(1) Parser lesbar. Einen einfachen top-down Parser stellt die Klasse `ACLParser` dar, die aus einem ACL String ein `ACLMessage` Objekt erzeugt, das die Informationen der Nachricht enthält.

Zur lexikalischen Analyse des ACL Strings wird der String durch die Klasse `FIPATokenizer` in einzelne Token zerlegt. Überflüssige Leerzeichen, Tabulatoren und Zeilenumbrüche (siehe Abbildung 5-4) werden entfernt, und der String kann unabhängig von der optischen Darstellung geparst werden. Im Unterschied zum herkömmlichen Java Tokenizer werden auch öffnende und schließende Klammern und Text in Gänsefüßchen als einzelne Token interpretiert. Die Nachricht aus Abbildung 5-4 wird zum Beispiel in die Token „`(`“, „`inform`“, „`:sender`“, „`agent1`“... unabhängig von Leerzeichen und Zeilenumbrüchen zerlegt. Bei der lexikalischen Analyse wird häufig schon eine Zuordnung einzelner Token zu Symbolen (wie „`open-bracket-sym`“, oder „`if-sym`“) gemacht, die an den Parser gegeben werden. Davon wird wegen der Einfachheit der ACL abgesehen. Eine Interpretation der einzelnen Token findet im `ACLParser` durch Stringvergleich statt.

Es kann nicht davon ausgegangen werden, dass nur syntaktisch korrekte Nachrichten geparst werden, da auch mit fremden Agenten kommuniziert werden könnte. Das Verhalten eines Parsers bei fehlerhaften Sätzen ist ein sehr komplexes Thema, wenn versucht wird, nach einem Fehler die nächste Einsprungstelle zu finden und den Parsevorgang fortzusetzen. Diese Funktionalität ist vorteilhaft, wenn der String in die richtige Syntax gebracht werden soll, da mehrere Fehler pro Parsevorgang gefunden werden können. Ein Programmierer z.B. bräuchte den gesamten Quellcode nicht nach jedem behobenen Fehler neu zu kompilieren, um auf den nächsten aufmerksam gemacht zu werden, sondern könnte in einem Schritt mehrere Fehler beheben. Im Fall des ACL Parsers spielt das Fortfahren nach einem Fehler nur eine geringe Rolle, da der Standpunkt vertreten werden kann, dass bei einer fehlerhaften Nachricht der Parsevorgang abgebrochen und, wenn möglich, eine `not-understood` Nachricht an den Sender geschickt wird. Da dieses Verhalten viel einfacher zu implementieren ist, wirft die Klasse `ACLParser`, beim Auftreten des ersten Fehlers eine Exception, mit einer kurzen Information über den Fehler.

Zum Versand einer Nachricht wird ein `ACLMessage` Objekt mit Hilfe der Klasse `ACLComposer` in einen String umgewandelt. Parameter, deren Werte nicht gesetzt sind, werden dabei, um FIPA konform zu bleiben, nicht im String abgebildet. Damit die Nachricht auch für Menschen komfortabel lesbar ist, werden zusätzliche Leerzeichen und Zeilenumbrüche in einen ACL String eingefügt.

Die Umwandlung einer ACL Nachricht aus der String-Darstellung in ein Java Objekt und aus einem Objekt in die String-Darstellung kommt sehr häufig vor. Bei hohem Nachrichtenaufkommen kann dies schnell zu Performanzproblemen führen, wenn für jeden Parsevorgang ein Parser- oder Composerobjekt erzeugt und nachher wieder zerstört wird. Eine mögliche Lösung basierend auf dem Singleton Pattern aus [Gamma et al. 96] wäre, nur eine Instanz der `ACLParser` und `ACLComposer` Klasse zu nutzen. Dies kann aber zu einem *Bottleneck* Problem führen, da die Nutzung jeweils synchronisiert werden müsste, damit parallele Parsevorgänge sich nicht gegenseitig behindern. Ein Ansatz, der dieses Problem nicht hat, basiert auf der Verwendung statischer Methoden. Parser- und Composerobjekte werden nicht angelegt. Die Klassen werden nur durch statische Methoden genutzt. Da es auch bei diesem Ansatz zu Problemen mit parallelen Parsevorgängen kommen könnte, wird auf die Verwendung von Klassen- und Instanzvariablen verzichtet und es werden nur lokale Variablen benutzt. Dieser Ansatz scheint der vernünftigste zu sein, da weder unnötige Objekte angelegt werden, noch der Zugriff auf die Klassen synchronisiert werden muss. Dieser Ansatz wird zusätzlich zum ACL Parser und Composer auch bei der Implementierung der SL0 und Ontologie Parser/Interpreter und Composer verwendet, ohne dass noch einmal direkt darauf eingegangen wird.

7.1.2 Semantic Language

Der Inhalt einer ACL-Nachricht wird gewöhnlich als wohlgeformter Satz der FIPA SL Grammatik oder einer ihrer Untermengen dargestellt. Aus dem Design geht hervor, dass die beiden benötigten Ontologien mit der Untermenge SL0 dargestellt werden können. Im Vergleich mit der ACL Grammatik ist die SL0 Grammatik um ein Vielfaches komplexer. Genau genommen ist die ACL durch die SL0 darstellbar. Die formale Beschreibung der SL0 Grammatik in EBNF und deren Abbildung auf Java Objekte in einem UML Diagramm sind im Anhang C zu finden. Im Prinzip wird fast jedes nicht-terminale Symbol der Grammatik durch eine eigene Klasse abgebildet. Die Klasse `SL0Content` ist das Zugriffsobjekt auf den SL Objektbaum und wird vom Parser zurückgegeben und an den Composer übergeben. Da alle weiteren Klassen, die diesen Baum bilden, in den folgenden Abschnitten nicht benötigt werden, sollen sie hier nicht weiter vorgestellt werden. Für weitere Details der Abbildung verweise ich noch einmal auf Anhang C und den Quellcode. Alle an der Abbildung und Umwandlung der SL0 beteiligten Klassen sind im Paket *fipa.lang.sl0* zu finden.

Die SL0 ist eine kontextfreie Grammatik, die leider nicht LL(1), sondern nur LL(2) parsebar ist. Da eine Umwandlung in eine LL(1) Grammatik gar nicht oder nur sehr schwer möglich ist, muss der Parser an einigen Stellen ein weiteres Zeichen im voraus lesen, bevor das folgende Symbol identifiziert werden kann, und somit bekannt ist, welches Objekt erzeugt werden muss. Die Klasse `SL0Parser` nutzt zum Parsen den schon bekannten `FIPATokenizer` und erzeugt aus einem SL0-String einen SL0 Objektbaum. Verhalten und Struktur dieser Klasse ist ähnlich dem `ACLParser`. Beim Auftreten eines Fehlers wird auch hier der Parsevorgang abgebrochen und der Agent mit einer Exception auf den ungültigen SL0 String hingewiesen.

Die entsprechende `SL0Composer` Klasse generiert aus einem SL0 Objektbaum einen entsprechenden SL0 String. Dies geschieht, indem über alle Objekte des Baums rekursiv, der Tiefe voran, iteriert wird. Auch ein SL0 String wird mit Leerzeichen und Zeilenumbrüchen versehen und optisch übersichtlich aufgebaut.

7.1.3 Agent Management Ontologie

Die Semantic Language ist so spezifiziert, dass verschiedenste Arten von Informationen durch sie dargestellt werden können. Diese müssen zusätzlich durch eine Ontologie beschrieben sein, die die Semantik der Information definiert. Hier soll die Untermenge SL0 genutzt werden, um die Begriffswelt, die durch die AM Ontologie definiert wird, syntaktisch darzustellen.

Im Design wurde gezeigt, dass nicht die gesamte AM Ontologie benötigt wird, sondern nur der Teil, den der DF Agent nutzt. Dies sind im wesentlichen die Aktionen *register*, *deregister*, *modify* und *search*, die Datenstruktur `df-agent-description` und einige mögliche Exceptions. Die soeben aufgezählten Aktionen des DF Agenten werden durch die Klasse `AMAction` repräsentiert, die als Element eine Instanz der `DFAgentDescription` Klasse enthält, die das Ontologie-Konstrukt `df-agent-description` repräsentiert. Ist eine Aktion eine Suchaktion, kann zusätzlich ein `search-constraint` angegeben werden, dass zusätzliche Optionen, wie die maximale Anzahl der Ergebnisse und die Suchtiefe beschreibt. Da es recht lang und umständlich ist die genaue Abbildung der AM Ontologie in Java Klassen zu beschreiben, zeigt folgendes UML Diagramm weitere Details:

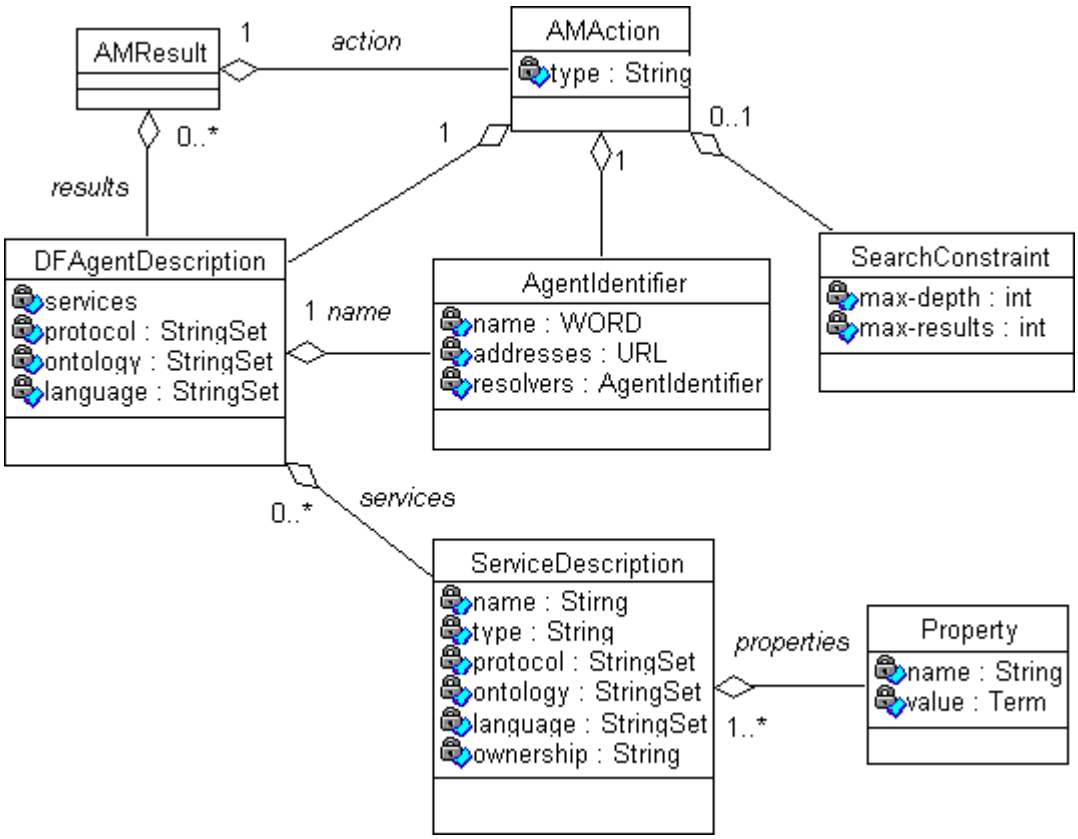


Abbildung 7-1: Abbildung der Agent Management Ontologie

Soll ein Agent den DF Dienst nutzen, so ist eine ACL Nachricht vom Typ `request` an den DF Agenten zu schicken. Der Content dieser Nachricht, der in ein `AMAction` Objekt abgebildet wird, besteht aus dem SL0 Konstrukt `action-expression`. Dieses beginnt mit dem Schlüsselwort *action* und beschreibt den Typ der Aktion, den ausführenden Agenten und die zur Aktion gehörigen Daten, die in diesem Fall eine `df-agent-description` darstellen. Zu sehen ist ein solcher Content in Abbildung 6-3. Entsprechende Interpreter und Composer sind in der Lage solch einen Ausdruck in ein `AMAction` Objekt und zurück zu konvertieren.

Das Ergebnis einer Anfrage kann entweder eine Bestätigung oder Ablehnung der Anfrage, oder eine Liste von Suchergebnissen sein. Im ersten Fall wird eine ACL `inform` Nachricht verschickt, deren Content aus einem SL0 Ausdruck besteht, der mit dem Schlüsselwort `done` beginnend den vorherigen Content wiederholt, und die Anfrage bestätigt. Bei Ablehnung wird eine `refuse` Nachricht verschickt, deren Content gleich dem der Anfrage ist. Im letzten Fall besteht der Content der `inform` Nachricht aus einem SL0 Ausdruck, der mit dem Schlüsselwort *result* eine Liste aus keinen oder mehreren `df-agent-description` Konstrukten beschreibt. Dieser Ausdruck wird im ein `AMResult` Objekt interpretiert, über das der Agent auf die Suchergebnisse zugreifen kann.

Durch die Beschreibung der AM Ontologie und ihre Abbildung in SL0 wird deutlich, dass nicht nur der Content der Nachrichten, sondern auch der Ablauf einer Konversation genau definiert sein muss, um erfolgreich kommunizieren zu können. Die soeben beschriebe Konversation mit dem DF Agenten basierte auf dem *FIPA Request* Protokoll, dessen Realisierung im folgenden Abschnitt vorgestellt wird.

7.1.4 Abbildung der Protokolle

Im Design wurde ein Ansatz vorgestellt, in dem Protokolle in Behaviours abgebildet werden, die den Ablauf des Protokolls und die Verwaltung mehrerer, parallel geführter Dialoge implementieren. Wie ein solcher Ansatz verwirklicht werden kann, soll in diesem Abschnitt gezeigt werden. Dabei werden zunächst allgemeine Überlegungen angestellt, die anschließend durch Betrachtung des *FIPA Request* Protokolls und des *FIPA Iterated Contract Net* Protokolls konkreter werden.

Konversationen

Ein Dialog, dessen Abhandlung einem Protokoll zugrunde liegt wird, wie schon beschrieben, Konversation genannt. Es gibt keinen Grund dies in der Implementierung anders zu handhaben, weshalb es die abstrakte Klasse `Conversation` gibt, von der jede spezielle Konversation erbt. Sie stellt sicher, dass jede Konversation über eine ID verfügt und darüber Auskunft geben kann, ob sie beendet ist oder nicht. Zudem wird über abstrakte Methoden sicher gestellt, dass jede `Conversation` Klasse die Methode `execute(ACLMessage acl)` und zwei weitere, an dieser Stelle unwichtige, Methoden implementiert. Die Methode `execute` wird aufgerufen, wenn eine ACL Nachricht angekommen ist, die dieser Konversation angehört und bearbeitet werden soll.

Grundsätzlich sind an einer Konversation zwei unterschiedliche Akteure beteiligt, hier Initiator und Responder genannt. Der Initiator beginnt eine neue Konversation und schickt die erste Nachricht an den oder die Responder. Für den Responder beginnt die neue Konversation mit Erhalt dieser Initialnachricht. Weitere Nachrichten werden der entsprechenden, bestehenden Konversation über eine ID zugeordnet, die im `:conversation-id` Feld der ACL Nachricht übertragen wird. Zuständig für die Vergabe dieser eindeutigen ID ist der Initiator, die aus Namen des Agenten und einer Nummer besteht.

Da ein Agent in der Rolle des Initiators nicht unbedingt auch die Rolle des Responders annehmen muss, man denke an einen Agenten, der einen Dienst in Anspruch nimmt jedoch nicht selbst anbietet, ist es sinnvoll die Implementierung des Protokolls nach Initiator und Responder zu trennen. Jedes Protokoll bietet eine `InitiatorConversation` und eine `ResponderConversation` Klasse an, die beide von der `Conversation` Klasse erben. Beide bilden den Stand der Konversation ihrerseits in einem Zustand ab. Dieser regelt, welches die nächste erwartete Nachricht ist, und was bei Erhalt dieser zu tun ist. Die einzelnen Zustände auf Initiator- und Responderseite einer Konversation können sehr

gut nach dem State Pattern aus [Gamma et al. 96] implementiert werden. Gerade bei komplexen Protokollen, kann der Verlauf der Konversation übersichtlich und einfach gehalten werden. Im Prinzip definiert eine abstrakte State Klasse eine `perform` Methode, die jeder davon abgeleitete Zustand, nach den eigenen Anforderungen überschreibt. Vor dem Versand der Nachricht wird der Folgezustand als aktueller Zustand der Konversation gesetzt oder die Konversation für beendet erklärt.

Die genaue Implementierung eines Protokolls wird in den nächsten beiden Unterabschnitten anhand zweier Beispiele gezeigt.

Behaviours

Die Abbildung eines Protokolls in eine `Conversation` Klasse ist nur der erste Schritt, denn, wie im Design beschrieben, muss ein Agent verschiedenste Konversationen parallel führen können. Damit nicht jeder Agent die Verwaltung der Konversationen erneut implementieren muss, wurde im Design angedacht ein Protokoll durch ein Behaviour darzustellen, dass diese Aufgaben kapselt.

Bevor jedoch eine Lösung gezeigt wird, ist noch ein wichtiger Punkt im Umgang mit Behaviours zu diskutieren. Im eigentlichen Sinne ist ein Behaviour ein Prozess, der ständig irgendwelche Sensordaten liest und abhängig von ihnen Reaktionen ausführt. Dies könnte in diesem Fall so ausgelegt werden, dass ein Agent Nachrichten erhalten würde und sie in einer Art Message Queue ablegen würde. Das oder die Behaviour würden durch einen Polling-Mechanismus, aktiv die an sie gerichteten Nachrichten der Queue entnehmen und verarbeiten. Diese Variante hat jedoch einige Nachteile.

Zum einen ist sie recht kompliziert, da jedes Behaviour in einem eigenen Thread oder Prozess laufen und der Zugriff auf die Message Queue synchronisiert werden müsste. Zum anderen kann vom Agenten nur schwer sichergestellt werden, das es ein Behaviour gibt, dass mit dieser Nachricht etwas anfangen kann. Sie würde ewig in der Queue verweilen, nach und nach würde die Queue mit unverstandenen Nachrichten riesig groß werden. Zwei weitere Gründe gegen eine solche Implementierung eines Behaviours sind, dass der Sender der Nachricht auf eine Antwort wartet, die er unter obigen Umständen nicht bekommen würde, und dass diese Variante einige Performanzkriterien zur Diskussion stellt. Um so höher die Pollingrate der Behaviours, desto mehr behindern sich diese gegenseitig und desto schwerer können neue Nachrichten in die Queue eingefügt und entnommen werden. Wird die Pollingrate zu niedrig gehalten, so kann einige Zeit vergehen, in der das Behaviour nichts tut, neue Nachrichten aber vorhanden sind.

Aus diesen Gründen wird eine Nachricht, die ein Agent erhält, direkt an das Behaviour weitergegeben und verarbeitet. Auch wenn dieses Konzept nicht exakt dem eines Behaviour entspricht, scheint diese Lösung zur Zeit mehr Vorteile zu bringen.

Die Implementierung sieht folgendermaßen aus. Es existiert eine Basisklasse mit dem Namen `Behaviour`. Sie besitzt ein Dictionary, in dem eine conversation-id einem `Conversation` Objekt zuordnet ist. Zusätzlich definiert sie, ähnlich wie die `Conversation` Klasse, die abstrakte Methode `execute(ACLMessage)`, die aufgerufen wird, wenn eine Nachricht für dieses Behaviour eintrifft.

Von dieser Klasse abgeleitet sind die Klassen `InitiatorBehaviour` und `ResponderBehaviour`, die die geerbte `execute(ACLMessage)` Methode jeweils unterschiedlich überschreiben. Auf Initiatorseite wird beim Aufruf dieser Methode die übergebene ACL Nachricht, anhand des `:conversation-id` Parameters an die `execute` Methode des entsprechenden `Conversation` Objektes weitergeleitet. Beim Start einer neuen Konversation muss das `InitiatorBehaviour` das neue `Conversation` Objekt zu diesem Zeitpunkt schon erzeugt haben.

Die `execute` Methode des `ResponderBehaviour` prüft, ob die ACL Nachricht einer bereits bestehenden Konversation zuzuordnen ist und leitet diese ggf. an die entsprechende Konversation weiter. Ansonsten wird geprüft, ob der Typ der eingehenden Nachricht dem Initial-Typ der Konversation entspricht und ein neues `Conversation` Objekt erzeugt.

Von den Klassen `InitiatorBehaviour` und `ResponderBehaviour` wird jeweils eine weitere Klasse abgeleitet, die die Funktionalität eines bestimmten Protokolls implementieren. Dies wird nun anhand des *FIPA Request* Protokolls gezeigt.

7.1.4.1 FIPA-Request Protokoll

Das *FIPA Request* Protokoll, beschrieben in 5.4.1, ist ein recht einfaches Protokoll. Die Abbildung einer Konversation wird durch die zwei Klassen `RequestInitiatorConversation` und `RequestResponderConversation` implementiert, die jeweils die Seite des Initiators und die des Responders des Protokolls repräsentieren. Die Verwaltung dieser Konversationen und die Schnittstelle zum Agenten implementieren die Klassen `RequestInitiatorBehaviour` und `RequestResponderBehaviour`.

Request – Initiator

Die Klasse `RequestInitiatorConversation` besitzt folgende drei Zustände, die nach dem State Pattern jeweils als Klasse implementiert werden: `RequestInitial`, `RequestAgree` und `RequestInform`. Alle sind von einer gemeinsamen Basisklasse abgeleitet und implementieren die Methode `perform(ACLMessage)` unterschiedlich. Wird eine neue Instanz der `RequestInitiatorConversation` Klasse angelegt, so befindet sie sich im `RequestInitial` Zustand. Gestartet wird die Konversation, indem die Methode `start(ACLMessage)` aufgerufen wird. Die ACL Nachricht, die ihr übergeben wird, muss zumindest den Empfänger und den Content der Nachricht beinhalten. Sie ruft die `perform` Methode des `RequestInitial` Zustandes auf. Dieser setzt den Folgezustand `RequestAgree` und verschickt die Nachricht.

Eine Antwortnachricht wird an die `execute` Methode der `RequestInitiatorConversation` Klasse weitergeleitet. Diese leitet die Antwort an die `perform` Methode des nun aktuellen `RequestAgree` Zustands weiter. Dieser erwartet entweder eine Nachricht des Typs `agree`, `not-understood` oder `refuse`. Bei Erhalt einer `agree` Nachricht, wird als einziges der Folgezustand `RequestInform` gesetzt. Der Erhalt einer Nachricht vom Typ `not-understood` oder `refuse` bedeutet, dass irgendwo ein Fehler aufgetreten ist. Die erhaltene Nachricht wird als *final-message* gespeichert und die Konversation beendet. Der Initiator der Konversation wird durch die *final-message* über den Ausgang der Konversation informiert.

Wurde eine Nachricht vom Typ `agree` erhalten, so wird im nun aktuellen `RequestInform` Zustand eine weitere Nachricht erwartet. Diese wird wiederum an die `perform` Methode des Zustandes weitergeleitet. Dieser erwartet eine Nachricht vom Typ `inform` oder `failure`. In beiden Fällen wird die Konversation beendet und die erhaltene Nachricht als *final-message* in der `RequestInitiatorConversation` Klasse gespeichert.

Die Verwaltung mehrerer Request-Konversationen übernimmt die Klasse `RequestInitiatorBehaviour`. Sie erbt vom `InitiatorBehaviour` und bietet nach außen zwei Methoden an, mit denen neue Request-Konversation gestartet werden können. Die Methode `perform(ACLMessage)` erwartet als Parameter eine ACL Nachricht, die zumindest Empfänger und Content enthält. Alle anderen wichtigen Parameter der ACL Nachricht werden, wenn auch nicht ausführlich beschrieben, in der entsprechenden InitiatorConversation oder InitiatorBehaviour Klasse gesetzt. Die

`perform` Methode startet eine neue Konversation, wartet bis sie beendet ist und gibt die *Final-Message* der Konversation als Ergebnis an den aufrufenden Agenten zurück. Dieser ist für die Interpretation des Ergebnisses zuständig. Da die Nachrichten eines Dialoges asynchron verschickt werden, ist es an dieser Stelle unumgänglich, zu warten und durch Polling das Ende der Konversation abzufragen. In diesem Fall wird ein mal pro Sekunde geprüft, ob die Konversation beendet ist. Dieses Zeitintervall könnte in folgenden Versionen auch variabel einstellbar sein.

Die zweite öffentliche Methode `performNoBlock` startet ebenso eine neue Request Konversation, stoppt jedoch den weiteren Programmablauf bis zur Beendigung nicht. Statt dessen wird das `Conversation` Objekt zurückgegeben. Der Agent ist zuständig zu prüfen, ob die Konversation beendet ist.

Soll ein spezieller Dienst über das *FIPA Request* Protokoll genutzt werden, so sollte es genügen, lediglich eine weitere Klasse vom `RequestInitiatorBehaviour` abzuleiten. In ihr brauchen nur rein dienstspezifische Funktionen implementiert werden, während die Basisklasse die vollständige Kommunikation über das *FIPA Request* Protokoll erledigt.

Request – Responder

Nachdem die Seite des Initiators des *FIPA Request* Protokolls ausführlich beschrieben wurde, ist es Zeit die Seite des Responders genauer zu betrachten. Sie ist etwas einfacher, da nur eine Nachricht des Typs `request` erwartet wird. Nach Abarbeitung der Anfrage, ist die Konversation auch schon beendet. Die Klasse `RequestResponderConversation` bildet diesen einzigen Zustand nicht über das State Pattern ab. Die Funktionalität des Protokolls wird direkt in der `execute` Methode implementiert. An dieser Stelle ist jedoch eine Besonderheit zu bemerken, die in der `RequestInitiatorConversation` nicht auftritt. Es ist zu prüfen, ob die Anfrage der ankommenden Nachricht ausgeführt werden kann und ggf. mit einer `agree` oder `refuse` Nachricht zu antworten. Daraufhin ist die Anforderung auszuführen und ein Ergebnis zurückzuliefern. Diese Funktionen können jedoch nicht von der `RequestResponderConversation` Klasse implementiert werden, da sie nur für die Funktionalität des Protokolls zuständig ist und den Content der Nachrichten nicht interpretieren kann. Diese Funktionen können nur von der Behaviourklasse implementiert werden, weshalb an dieser Stelle die abstrakten Methoden `checkRequest(ACLMessage)` und `handleRequest(ACLMessage)` des `RequestResponderBehaviour` aufgerufen werden. Diese müssen von der Behaviourklasse eines Dienstes, der das *FIPA Request* Protokoll nutzen soll, implementiert werden. Nachdem die Anfrage ausgeführt und die beiden Antwortnachrichten verschickt wurden, ist die Konversation auch schon beendet.

Die Klasse `RequestResponderBehaviour` unterscheidet sich von der entsprechenden Initiator Klasse insofern, dass sie die Methoden `perform` und `performNoBlock` nicht anbietet. Beide sind nicht notwendig, da der Responder in seiner Rolle keine neue Konversation starten kann und lediglich auf einkommende Nachrichten reagiert. Jedoch definiert diese Klasse die beiden schon erwähnten abstrakten Methoden `checkRequest` und `handleRequest`, die, von der Klasse `RequestInitiatorConversation` aufgerufen werden. Der Fall, dass während einer Konversation Funktionalität benötigt wird, die abhängig vom zu verwirklichenden Dienst ist, kann auch auf der Initiatorseite des Protokolls auftreten. Dies ist der Fall beim FIPA Iterated-Contract-Net Protokoll und wird auf dieselbe Weise, durch den Aufruf abstrakter Methoden, gelöst.

Die funktionale Trennung einer Konversation von ihrer Verwaltung sowie die Trennung der Verwaltung von der Aufgabe, die mit Hilfe eines Protokolls verwirklicht werden soll, macht es nun sehr leicht möglich verschiedenste Dienste zu implementieren, die dieses Protokoll nutzen. Zusätzlich kann ohne großem Aufwand Änderungen an der

Implementierung einer Konversation vorgenommen werden, ohne jeden verwirklichten Dienst anpassen zu müssen.

Ein Dienst, der das *FIPA Request* Protokoll nutzt, wird vom DF Agenten angeboten. Die Implementierung dieses Dienstes auf Basis der `RequestResponderBehaviour` und `RequestInitiatorBehaviour` Klassen, wird in Kapitel 7.2.4 erläutert.

7.1.4.2 FIPA-Iterated-Contract-Net Protokoll

Ein weiteres FIPA Protokoll, das benötigt wird, ist das *FIPA Iterated Contract Net* Protokoll, vorgestellt in Kapitel 5.4.2. Dieses ist etwas komplizierter als das *FIPA-Request* Protokoll, da ein zweiter Kommunikationsschritt zwischen Initiator und Responder notwendig ist. Zusätzlich unterscheidet es sich in einem weiteren Punkt vom *FIPA Request* Protokoll, da mehrere Responder an einer Konversation teilnehmen können. Da es jedoch strukturell ähnlich wie das *FIPA Request* Protokoll realisiert ist, soll es etwas knapper beschrieben werden und Punkte hervorheben, die auf der oben beschriebenen Basis individuell für jedes neue Protokoll implementiert werden müssen.

Dieses Protokoll wird auf Seite des Initiators durch die Klassen `ICNInitiatorConversation` und `ICNInitiatorBehaviour` realisiert. Gestartet wird die Konversation durch Aufruf der Methode `start`, die als Parameter eine Liste der teilnehmenden Agenten und einen String benötigt. An dieser Stelle ist die Realisierung nicht ganz frei von der Aufgabe, die der Informationsagent mit der ICN Konversation erledigt. Das zweite Argument dieser Methode ist der Begriff der an alle Informationsagenten geschickt wird, um Informationen ausfindig zu machen. Der Start einer Konversation ist noch etwas verbesserungswürdig, denn in Zukunft sollte diese Information an anderer Stelle übergeben werden.

Die Zustände der `ICNInitiatorBehaviour` Klasse werden wieder nach dem State Pattern abgebildet und heißen `ICNInitial`, `ICNPropose` und `ICNInform`. Auch hier werden alle eingehenden Nachrichten von der `execute` Methode an die `perform` Methode des jeweils aktuellen Zustandes weitergeleitet. Diese erledigen folgende Aufgaben:

- `ICNInitial`
  Zuerst wird geprüft, ob es Teilnehmer der Konversation gibt. Falls ja, wird der nächste Zustand (`ICNPropose`) gesetzt, falls nicht, ist die Konversation schon beendet. Das Ändern des Zustandes geschieht schon an dieser Stelle, da es möglich ist, dass die erste Antwort schon eintrifft bevor alle Nachrichten an alle Teilnehmer versendet sind. Daraufhin wird eine neue ACL Nachricht erzeugt, die an jeden Teilnehmer einzeln geschickt wird. Die FIPA Spezifikation sieht zwar vor, dass ACL Nachrichten mehrere Empfänger haben können, jedoch darf eine SL-Action nur an einen speziellen Agenten gerichtet werden. Aufgrund dieser Einschränkung ist für jeden Teilnehmer ein individueller Content zu erzeugen, und eine spezielle ACL Nachricht, an jedem einzeln Teilnehmer zu schicken. Das Erzeugen des Contents geschieht durch die Methode `createCFPContent`, die von der Klasse `ICNInitiatorBehaviour` abstrakt definiert wird.
- `ICNPropose`
  In diesem Zustand werden Nachrichten von Typ `propose` erwartet und alle eingehenden Vorschläge der Teilnehmer gesammelt. Die Teilnehmer, die mit Fehlernachrichten antworten, werden aus der Liste der Teilnehmer gelöscht. Haben alle Agenten geantwortet (interessant ist ein Zeitlimit, dass jedoch nicht implementiert ist), werden alle eingegangen `propose` Antworten an die `handleProposals` Methode des Behaviours geschickt. Somit ist der Agent in der

Lage über Akzeptanz und Ablehnung alle Vorschläge zu entscheiden. Als Rückgabewert dieser Methode wird eine Liste von `accept-proposal` und `reject-proposal` ACL Nachrichten erwartet, die an die entsprechenden Teilnehmer versendet werden. Da von den Teilnehmern, an die eine `reject-proposal` Nachricht geschickt wird, keine weitere Antwort erwartet wird, werden sie aus der Liste der weiterhin an der Konversation teilnehmenden Agenten, entfernt. Der nächste Zustand ist `ICNInform`.

- `ICNInform`
  In diesem Zustand werden Nachrichten vom Typ `inform` oder `failure` erwartet. Alle Antworten werden als *Final-Messages* gespeichert. Nach Antwort des letzten Teilnehmers wird die Konversation beendet.

Somit ist die Funktionalität der ICN Konversation des Initiators beschrieben. Die dazugehörige `ICNInitiatorBehaviour` Klasse muss, nach Anforderung der soeben beschriebenen Zustände, die abstrakten Methoden `handleProposals` und `createCFPContent` definieren. Ansonsten bietet sie, wie die `RequestInitiatorBehaviour` Klasse, die beiden Methoden `perform` und `performNoBlock`, zum Starten einer neuen ICN Konversation an.

Die Seite des Responder besteht aus den Klassen `ICNResponderConversation` und `ICNResponderBehaviour`. Sie haben dieselbe Struktur, wie die entsprechenden Klassen des *FIPA Request* Protokolls. Eine Konversation besteht aus den beiden Zuständen *initial* und *accept-reject*. Sie werden durch folgende Zustandsklassen, mit entsprechend unterschiedlicher `perform` Methode, realisiert.

- `ICNInitial`
  In diesem Zustand wird erwartet, dass ein ACL Nachricht des Typs `cfp` eingetroffen ist. Eine Antwortnachricht wird, durch die Methode `handleCFP` des entsprechenden Behaviours, erzeugt und versendet.

- `ICNAcceptReject`
  Nachdem ein Vorschlag an den Initiator geschickt wurde, wird eine Antwort des Typs `accept-proposal` oder `reject-proposal` erwartet. Je nach Antwort, wird die Nachricht an die Methode `handleAcceptProposal`, die eine neue Nachricht mit der Information passend zum akzeptierten Vorschlag erzeugt oder die Methode `handleRejectProposal` des Bahaviors weitergeleitet, die für mögliche Aufräumarbeiten beim Agenten genutzt werden kann.

Das `ICNResponderBehaviour` hat dementsprechend die abstrakten Methoden `handleCFP`, `handleAcceptProposal` und `handleRejectProposal` zu definieren. Ansonsten ähnelt es der Klasse `RequestResponderBehaviour`.

Auch bei der Realisierung dieses Protokolls ist eine Trennung von Protokoll und des eigentlichen Dienstes, der damit angeboten werden soll, zumeist (bis auf die oben beschriebene Ausnahme) geglückt. Soll ein Agent, einen Dienst über das ICN Protokoll anbieten oder nutzen, braucht lediglich eine entsprechend erweiterte ICN Behaviourklasse implementiert werden. Wie dies im Falle des Informationsagenten funktioniert, ist in Kapitel 7.3 zu sehen.

Die Beschreibung der Darstellung der ACL Nachrichten und ihres Contents sowie die Realisierung des FIPA request und des *FIPA Iterated Contract Net* Protokolls ist soweit beendet. Aufbauend auf die in diesem Kapitel beschriebenen Grundlagen, wird nun

gezeigt, wie eine FIPA konforme Agentenplattform aussehen kann. Es wird erwartet, dass die Abbildung des DF Dienstes auf Basis der soeben beschriebenen Behaviourklassen des *FIPA Request* Protokolls einfach möglich ist.

7.2 Plattform

Die Architektur der zu implementierenden FIPA Plattform wurde im Design spezifiziert und besteht aus den Komponenten IPMT, ACC und DF Agent. In den folgenden Abschnitten wird gezeigt, in welcher Form diese Komponenten, auf Basis der im letzten Abschnitt beschriebenen FIPA Sprachen und Protokolle, realisiert werden können.

Zusätzlich zu den einzelnen Komponenten wird ein Basisagent vorgestellt, der gemeinsame Funktionalität aller Agenten kapselt, sowie ein einfacher Testagent zum Testen der Plattformfunktionen.

7.2.1 Internal Platform Message Transport

Die *Internal Platform Message Transport* (IPMT) Schicht hat die Aufgabe, Nachrichten physisch von einem Agenten zum anderen zu transportieren. Dabei dürfen die einzelnen Agenten auf verschiedenen Computern bzw. in verschiedenen Prozessen existieren. Der Nachrichtentransport untereinander soll asynchron geschehen. Es existieren viele verschiedene Ansätze, Standards und Produkte mit denen asynchrone Interprozesskommunikation realisiert werden kann, eine eigene Implementierung ist daher nicht nötig. Jedoch ist zu diskutieren, welche Variante für diese Aufgabe die geeignetste ist.

Die wahrscheinlich bekannteste Form ist das Email Protokoll SMTP (Simple Mail Transport Protocol), bei dem der Nutzer eine Email Nachricht an einen SMTP-Server schickt, der diese solange zwischenspeichert, bis sie vom Empfänger abgeholt wird. Der Inhalt einer Email könnte eine ACL Nachricht sein. Jedoch wurde schon auf ein wichtiges Problem hingewiesen, dass zumindest im Fall der Agentenkommunikation auftritt. Normalerweise wird davon ausgegangen, dass der Empfängeragent vorhanden ist und die Nachricht sofort entgegennimmt und bearbeitet (solange er nicht überlastet ist). Ist der Agent nicht erreichbar, so wird auf der Seite des Senders eine Fehlernachricht erwartet, auf die mit entsprechenden Maßnahmen reagiert werden kann. Eine Zwischenspeicherung von ACL Nachrichten auf einem Emailserver hätte zur Folge, dass der Sender nicht weiß, wann die Nachricht vom Empfänger abgeholt wird und wann eine Antwort zu erwarten ist (dies können Minuten, Stunden oder Tage sein). In manchen Szenarien mag die Nutzung von SMTP auch vorteilhaft gegenüber Protokollen sein, deren Auslieferung einer Nachricht auf direktem Wege geschieht. Zum Beispiel bei Aufgaben, die keine Antwortnachricht erfordern oder bei denen eine Antwort auch nach langer Zeit eintreffen kann.

Ein weiter Ansatz Nachrichten asynchron zu übertragen könnte auf der, von Sun verabschiedeten, Spezifikation *Java Messaging Service* (JMS), oder anderen Messaging Systemen basieren. JMS ist ein allgemeines Interface, dass von den meisten Messaging Produkten, wie *MQSeries* [22] oder *SonicMQ* [23], unterstützt wird und ist Teil der *Java 2 Enterprise Edition* (J2EE) Spezifikation. Durch JMS ist die Anwendung bzw. der Agent unabhängig von dem speziellen Messaging Service, der hinter der JMS Schnittstelle benutzt wird.

SMTP, JMS oder ein anderes Messaging System könnten theoretisch als IPMT Komponente genutzt werden, doch erfüllen sie einer weitere Anforderung des Designs nicht. Die IPMT Komponente soll ACL Nachrichten neben den plattforminternen Transport auch zu anderen Plattformen über IIOP transportieren können. Da IIOP auf

CORBA basiert, ist es sinnvoll eine CORBA Implementierung als Grundlage, die auch IIOP unterstützt.

CORBA [24] ein Standard ist, der vom der OMG verabschiedet wurde. Es gibt verschiedenste Implementierungen und Produkte, die zukünftig Object Request Broker (ORB) genannt werden, da ein ORB die zentrale Instanz eines CORBA Produkt ist. An dieser Stelle sollen weder Vor- und Nachteile der gängigsten CORBA Implementierungen, noch die Gründe, die zur Auswahl von Voyager als CORBA ORB geführt haben, diskutiert werden. Die einzig entscheidenden Gründe für Voyager sind, dass dieses kommerzielle Produkt bereits erworben wurde und somit rechtlich eingesetzt werden darf, IIOP unterstützt und in der *Flirtmaschine* verwendet wird, die ein Agent ist, der im Szenario verwendet wird.

Im Folgenden wird nun beschrieben, wie Voyager für das Versenden und Empfangen von Nachrichten innerhalb der Plattform genutzt werden kann, und wie Nachrichten über IIOP an andere Plattformen geschickt werden können.

Voyager dient als ORB, der Nachrichten von einem Agenten zum ACC oder vom ACC an einen Agenten delegiert. Dies geschieht über den asynchronen Aufruf einer Methode auf dem (Voyager bekannten) Empfängerobjekt. Dazu müssen zwei Voraussetzungen erfüllt sein. Zunächst muss jeder Agent und der ACC, um Nachrichten erhalten zu können, folgendes Interface implementieren:

```
public interface IFipaAgent
{
  void message( java.lang.String message );
}
```

Abbildung 7-2: Öffentliches Interface jedes Agenten

Die Methode `message(String)` wird bei erhalt einer Nachricht aufgerufen. Was in dieser Methode mit den String geschieht, ist abhängig von dem Agenten und wird später beschrieben. Die zweite Voraussetzung ist, dass jeder Agent beim Voyager ORB bekannt ist, damit eine Nachricht an ihm ausgeliefert werden kann. Dazu ist es nötig den Agenten an Voyager zu exportieren. Das gemeinsame öffentliche Interface aller Agenten, über das Nachrichten versendet und empfangen werden, ist das `IFipaAgent` Interface. Jeder Agent wird als `IFipaAgent` an Voyager exportiert. Folgende Abbildung soll dieses verdeutlichen:

```
01 public void startUp()
02 {
03   // some initialization
04   String bindAddress = "DF@agentcities.agentscape.de";
05   this.print("DF is starting at: " + bindAddress + " ...");
06   try {
07     if (! Voyager.isStarted())
08       Voyager.startup(this, port_);
09     IFipaAgent agent = (IFipaAgent)Proxy.export(this, port_);
10     Namespace.bind( bindAddress, agent);
11     this.println("...done.");
12   } catch (Exception e) {
13     this.println("... failed! " + e);
14   }
15 }
```

Abbildung 7-3: Export eines Agenten an den Voyager ORB

In Zeile 09 wird das Agentenobjekt (this) an den Voyager ORB unter einem bestimmten Port exportiert. Über diesen Port ist der Agent in Zukunft physisch ansprechbar. Dabei wird ein Proxyobjekt erzeugt, das in Zeile 10 dem Voyager ORB, als `IFipaAgent`, unter einem bestimmten Namen bekannt gemacht wird. Ab sofort kann dieses Proxyobjekt

unter diesem Namen bei Voyager angefordert werden, und auf ihm die Methode `message(String)` des `IFipaAgent` Interfaces aufgerufen werden. Diese Schritte werden beim Start eines jeden Agenten ausgeführt, der Agent kann nach erfolgreichem Start Nachrichten empfangen.

Das asynchrone Versenden einer Nachricht wird in folgender Abbildung gezeigt, in der beispielhaft eine Nachricht an den ACC geschickt wird.

```
01 public void sendMessage(String msg)
02 {
03   try {
04     String bindAddress = "//"+accHost+":"+accPort+"/acc";
05     IFipaAgent acc = (IFipaAgent)Namespace.lookup(bindAddress);
06     OneWay.invoke(acc, "message", new Object[]{msg});
07   } catch (Exception e) { }
08 }
```

Abbildung 7-4: Versand von ACL Nachrichten

In Zeile 4 wird die Voyageradresse des ACC aus Rechnernamen, Port und dem Namen des exportierten Objektes (siehe auch Abbildung 7-3, Zeile 4) konstruiert. Daraufhin wird das Proxyobjekt vom Typ `IFipaAgent` anhand der Voyageradresse vom ORB angefordert und die Methode `message(msg)` asynchron aufgerufen (Zeile 5).

Dieser Ausflug in die Benutzung des Voyager ORBs, der auch eine Einführung in einige Grundkonzepte von CORBA deutlich machen sollte, soll zunächst ausreichen. Zusammenfassend kann Voyager als Realisierung der IPMT Komponente gesehen werden, da es alle Anforderungen an die IPMT erfüllt. Die Verwendung von IIOP mit Voyager, um Nachrichten an andere FIPA Plattformen zu senden, soll der Übersichtlichkeit halber erst bei der Realisierung des ACC beschrieben werden.

7.2.2 FIPA Agenten

Jeder FIPA Agent benötigt, in gewissem Maße, gleiche Informationen und Funktionen, um zum Beispiel Nachrichten versenden und empfangen zukönnen. Da es wenig sinnvoll ist diese in jedem Agenten erneut zu implementieren, werden sie von der abstrakten Klasse `FipaAgent` gekapselt, die Basisklasse aller Agenten sein soll. Sie implementiert unter anderem das im letzten Abschnitt vorgestellte Interface `IFipaAgent`, das Exportieren der eigenen Instanz an Voyager, damit Nachrichten empfangen werden können, und das Erzeugen eines ACC Proxyobjektes, an das pauschal alle zu versendenden ACL Nachrichten geschickt werden. Dazu müssen Daten über Host und Port des ACC sowie über den eigenen Namen, Host und Port bekannt sein. Aus den eigenen Daten wird unter anderem die eigene Voyageradresse konstruiert, die dem ACC bekannt gemacht wird, damit dieser in der Lage ist ACL Nachrichten an diesen Agenten auszuliefern. Mehr dazu jedoch im nächsten Kapitel über den ACC. Die abstrakte Methode `message(String)`, aus dem `IFipaAgent` Interface, wird in dieser Klasse noch nicht implementiert, damit jeder Agent unterschiedliche auf ankommende Nachrichten reagieren kann. Detailliertere Informationen zur `FipaAgent` Klasse sind aus dem Quellcode zu nehmen, da eine genauere Beschreibung an dieser Stelle nur verwirrend wäre.

7.2.3 Agent Communication Channel

Alle ACL Nachrichten, die ein Agent mit Hilfe der Funktionen der eben vorgestellten Basisklasse an andere Agenten versenden möchte, werden an den ACC geschickt. Dieser leitet alle ankommenden ACL Nachrichten zum gewünschten Empfänger weiter. Befindet sich der Empfänger einer Nachricht auf derselben Plattform, so ist dessen IPMT

Adresse, im Fall dieser Plattform ist dies die Voyageradresse, vom ACC ausfindig zu machen und die Nachricht auszuliefern. Befindet sich der Empfänger auf einer anderen Plattform, so ist die Nachricht über IIOP an den ACC der Plattform weiterzuleiten. Im Design wurde festgelegt, dass die Voyageradressen aller Agenten und die Adressen anderer Plattformen vom ACC und nicht vom AMS verwaltet werden. Konsequenz daraus ist, dass jeder Agent seine Voyageradresse beim ACC bekannt machen muss, um Nachrichten empfangen zu können, und dem ACC die IIOP Adressen (IOR genannt) anderer Plattformen bekannt gemacht werden müssen. Wie dies genau geschieht, wird in den nächsten beiden Abschnitten zum plattforminternen und –externen Versand von Nachrichten beschrieben.

Der ACC wird durch eine gleichnamige Klasse abgebildet, die noch aus historischen Gründen von der `FipaAgent` Klasse abgeleitet ist. Im FIPA 97 Standard wurde der ACC noch als Agent verwirklicht. Bei der zukünftigen Entwicklung sollte die Implementierung der `ACC` Klasse an die Empfehlung des FIPA 2000 Standards angepasst werden, in dem neuerdings der ACC nicht mehr als Agent implementiert wird. Nachrichten, die weitergeleitet werden sollen, werden über die Methode `message(String)` erhalten. In ihr wird zwischen den zwei Funktionen der plattforminternen und –externen Weiterleitung differenziert.

7.2.3.1 Plattforminterne Kommunikation

Sender- und Empfängeradresse einer ACL Nachricht setzen sich im allgemeinen aus der Phrase <agent-name>@<home-agent-platform> zusammen, die einen Agenten eindeutig identifiziert. Als Sonderfall kann auch nur <agent-name> als Empfängeradresse genügen, um eine Nachricht korrekt ausliefern zu können. Dann wird davon ausgegangen, dass als Empfänger ein Agent auf derselben Plattform gemeint ist. Die Zusammensetzung der `home-agent-platform` (HAP) Adresse kann aus zwei unterschiedlichen Konstrukten gebildet werden. So ist `myAgent@iiop://fipaland.org:9000/acc` als auch `myAgent@fipaland` eine gültige Empfängeradresse, wobei die erstere die Protokolladresse des eigenen ACC darstellt und letztere den Namen der Plattform enthält. Der ACC prüft anhand der HAP Adresse des Empfängers einer ACL Nachricht, ob die Nachricht intern oder extern verschickt werden soll.

Anhand der möglichen Formate einer Empfängeradresse kann leider nicht immer auf die Voyageradresse des Empfängers geschlossen werden, die zur Auslieferung der Nachricht benötigt wird. Die FIPA löst dieses Problem, indem diese Informationen bei der Registrierung eines Agenten beim AMS mit angegeben werden. Der ACC fragt den AMS Agenten nach der AMS Registrierung des Empfängers, in der die gültige IPMT Adresse beschrieben steht. Da zunächst nur eigene Agenten auf der Plattform zu existieren brauchen und die Entscheidung getroffen wurde, den AMS nicht zu implementieren, werden nun folgende Voraussetzungen angenommen:

- Der Name eines Agenten ist eindeutig auf einer Plattform. (FIPA konform)
- Jeder Agent, der Nachrichten empfangen soll, hat Namen und Voyageradresse beim ACC zu registrieren (nicht FIPA konform)

Der ACC verwaltet diese Informationen in einem Dictionary und kann bei Bedarf auf die Adresse des Empfängers einer Nachricht schließen. Damit keine ungültige Adresse im Dictionary verbleibt, falls ein Agent beendet wird, stellt der ACC neben der Funktion zum Registrieren auch eine Funktion zum Deregistrieren bereit.

Die Antwort auf die Frage nach dem Sinn der Registrierung jedes Agenten beim ACC, wenn auch eine FIPA konforme Registrierung bei einem AMS Agenten hätte implementiert werden können, soll hier noch einmal zusammengefasst werden. Der AMS

Agent müsste, um FIPA konform zu sein, nach dem *FIPA Request* Protokoll Konversationen führen können, die Agent Management Ontologie verstehen können und würde wesentlich mehr Informationen verwalten, als momentan benötigt werden. Dass die Voyageradresse den ACC viel einfacher bekannt gemacht werden kann, als im FIPA konformen Ansatz, wird nun deutlich gemacht.

Das Registrieren und Deregistrieren kann simpel durch einen Voyager Aufruf geschehen. Dazu implementiert die `ACC` Klasse das `IACC` Interface (Abbildung 7-5), das vom `IFipaAgent` Interface erbt, und wird, genau wie ein `FipaAgent` Objekt aus Abschnitt 7.2.1, an Voyager unter dem Namen *acc* exportiert. Die Implementierungen der Methoden `register(String, String)` und `deregister(String)` der `ACC` Klasse fügen lediglich ein Name/Adress- Paar in ein einfaches Dictionary hinzu oder entfernen es.

```
public interface IACC
  extends com.agentscape.simon.fipa.util.agent.IFipaAgent
{
  public void setIOR(String ior);
  public void register(String name, String addr) throws AMSException;
  public void deregister(String name);
}
```

Abbildung 7-5: Öffentliches Interface des ACC

Registrierung und Deregistrierung der Agenten wird in der allgemeinen `FipaAgent` Basisklasse implementiert. Dort werden die Methoden `registerToAMS()` und `deregisterFromAMS()` (die Namen sind nicht glücklich gewählt) angeboten. Sie implementieren den einfachen Aufruf der entsprechenden Methoden auf dem ACC Proxyobjekt, wie Beispielhaft in Abbildung 7-6 zu sehen ist. Zeile 04 konstruiert die Voyageradresse des Agenten und Zeile 07 ruft die `register` Funktion des ACC auf.

```
01 public void registerToAMS()
02 {
03   try {
04     String bindAddress = "//"+host_+":"+port_+"/"+name_;
05     print("Registering with AMS at: " + bindAddress + " ......");
06     acc_.register(name_, bindAddress);
07     println("... done.");
08   } catch (AMSException e) {
09       println("... failed!\n" + e.toString()); }
10 }
```

Abbildung 7-6: Registrierung eines Agenten am ACC

Das Ausliefern einer ACL Nachricht an einen Agenten geschieht zunächst durch Zuordnung des Namens des Empfängers zur registrierten Voyageradresse. Anhand dieser Adresse wird das Proxyobjekt des Empfängers erzeugt und die Methode `message(String)` asynchron aufgerufen, wie schon in Abbildung 7-4 dargestellt.

Weitere Details zum internen Versand von Nachrichten können im Quellcode in der zentralen Methode `sendIntern(ACLMessage msg)` der ACC Klasse gefunden werden.

7.2.3.2 Plattform-externe Kommunikation

Soll eine ACL Nachricht an einen Agenten auf einer anderen Plattform weitergeleitet werden, genügt es, die Nachricht an den ACC der fremden Plattform zu schicken, dieser ist für die weitere Zustellung zum entsprechenden Agenten zuständig. Leider sind bei dieser Variante des Nachrichtenversands ähnliche Probleme zu bewältigen, wie im letzten Abschnitt. Bevor eine ACL Nachricht über IIOP an einen fremden ACC geschickt werden kann, muss dessen IIOP Adresse bekannt sein.

Eine Empfängeradresse hat das, schon aus dem letzten Kapitel bekannte, Format <agent-name>@<home-agent-platform>. Anhand des letzten Teils der Adresse entscheidet der ACC, ob die Nachricht über IIOP an eine andere Plattform gesendet wird oder nicht. Jedoch kann die HAP Adresse neben den beiden bekannten Formaten:

df@agentland (Plattformname),
df@iiop://agentland.com:9000/acc

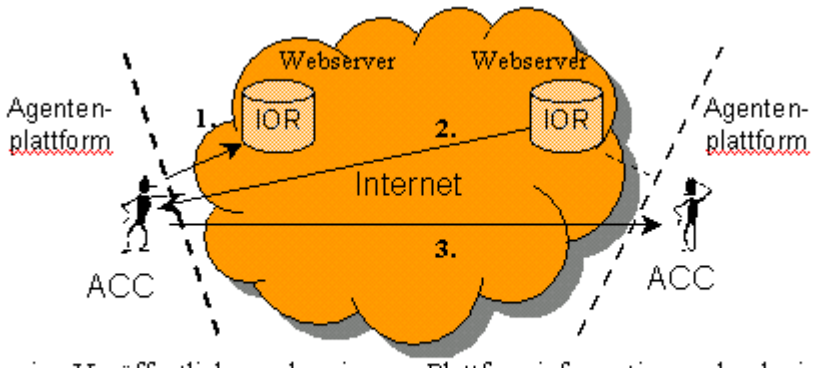
auch folgendes Format besitzen:

df@IOR:000000000000001149444c3a4649504

Leider unterstützt Voyager, im Gegensatz zu einigen anderen ORBs, das Format der mittleren Adresse nicht und kann Nachrichten nur versenden, wenn die IOR des Empfängers bekannt ist (das dritte Format). Da jedoch alle drei Formate gültig sind, ist eine Zuordnung dieser Formate, auf die IOR der Empfängerplattform unumgänglich.

Bevor auf die Zuordnung eingegangen wird, soll zunächst eine weitere offene Frage geklärt werden. Es wurde festgestellt, dass die IOR der Empfängerplattform bekannt sein muss, um Nachrichten an sie versenden zu können. Jedoch ist noch nicht geklärt, woher der ACC Kenntnis über die IOR bekommt und wie die eigene IOR verbreitet wird, damit die fremde Plattform Antworten zurückschicken kann.

Die FIPA schlägt zu diesem Thema vor IORs über Webserver zu verbreiten. Sie legt jedoch nicht fest, wie das Format einer solchen Webseite auszusehen hat und deckt mit dieser Lösung ein weiteres Problem nicht ab. Denn nach Neustart einer Plattform wird die alte IOR ungültig und eine neue erzeugt. Andere Plattformen werden über die neue nun aktuelle IOR nicht informiert. Sie müssten häufig prüfen, ob die ihnen bekannte IOR noch gültig ist. Diese Probleme können zur Zeit leider nicht FIPA konform gelöst werden, bevor jedoch eine eigene Lösung gefunden wird, was als Konsequenz hätte, dass diese Plattform nicht mehr kompatibel zu anderen Implementierungen wäre, wird die eigene Lösung zumindest an die Lösung der weit verbreiteten Plattform Fipa-OS angepasst. Folgende Abbildung verdeutlicht diese Lösung vorweg grafisch.



1. Passive Veröffentlichung der eigenen Plattforminformationen durch einen Webserver.
2. Besorgen der IOR einer bekannten Plattform, durch deren Webserver.
3. Aktive Bekanntmachung der eigenen IOR bei einer bekannten Plattform.

Abbildung 7-7: Publizierung der eigenen IOR

Das Format mit dem eine IOR über einen Webserver passiv verbreitet wird, hat folgendes Aussehen:

```
<platform-name>,<acc-guid>,<ior>
```

Eine Beispielplattform könnte durch folgende Informationen beschrieben werden:

```
agentland,iiop://agentland.com:9000/acc,IOR:000000001149444c3a4
```

Die aktive Veröffentlichung der eigenen IOR und das Besorgen der IORs anderer Plattformen, passiert nach folgendem Schema: Beim Start des ACC wird ein File ausgelesen, in dem URLs bekannter Plattformen stehen, deren IORs nach obigen Format unter diesen Adressen veröffentlicht werden. Die Informationen der Plattformen werden gelesen und in Objekten der Klasse `PlatformIdentifier` gespeichert.

Damit Nachrichten von fremden Plattformen an die eigene geschickt werden können, müssen diese die eigene IOR kennen. Theoretisch würde es genügen die eigene IOR im Sender Teil der ACL Nachricht mitzuschicken. Tests haben gezeigt, dass einige Plattformen, wie Fipa-OS, dieses Prinzip nicht unterstützen. Aus diesem Grund wird die eigene IOR zusätzlich, nach dem Vorbild von Fipa-OS, aktiv an bekannte Plattformen geschickt. Nachdem über Webserver alle IORs bekannter Plattformen ausfindig gemacht wurden, wird an jeden ACC der nun bekannten Plattformen eine `inform` Nachricht gesendet. Der Content dieser Nachricht enthält die eigene Plattforminformation in dem selben Format, das auch bei der Veröffentlichung via Webserver verwendet wird. Trifft eine solche Nachricht bei unserem ACC ein, wird die neue Information gespeichert. Der ACC ist dann in der Lage Nachrichten auch an diese Plattform zu schicken. Vorteil dieser (noch) nicht FIPA konformen Lösung ist, dass alle bekannten Plattformen beim einen Neustart der eigenen Plattform über die neue IOR informiert werden. Das Problem veralteter, ungültiger IORs kann somit nicht auftreten.

Kommen wir zurück auf das Mapping der HAP Adresse eines Empfängers einer ACL Nachricht zur entsprechenden IOR. Anhand der `PlatformIdentifier` Objekte, die jeweils eine bekannte Plattform repräsentieren, wird aus der jeweiligen HAP Adresse des Empfängers einer Nachricht die zu der Plattform gehörige IOR ausfindig gemacht. Bevor die ACL Nachricht an die Empfängerplattform verschickt wird, wird die eigene IOR im Sender der Nachricht eingetragen. Somit ist sicher gestellt, dass die Empfängerplattform in jedem Fall die aktuelle IOR dieser Plattform erhält. Für den genauen Ablauf, wie eine Nachricht über IIOP an einen anderen ACC schickt, möchte ich an dieser Stelle wieder auf den Quellcode, und zwar auf die Methode `sendExtern(ACLMessage)` der ACC Klasse, verweisen.

Während der Implementierung ist ein weiteres, unvorhergesehenes Problem aufgetreten. Es ist leider nicht möglich ein Objekt, in diesem Fall das `ACC` Objekt, für Voyager- und IIOP-Kommunikation an den Voyager ORB zu exportieren. Jedoch werden beide Kommunikationsarten benötigt, um Nachrichten von den Agenten der eigenen Plattform, sowie von Agenten anderer Plattformen zu erhalten. Gelöst wurde dieses Problem durch das zusätzliche `IIOPReceiver` Objekt, das Nachrichten von externen Plattformen über IIOP empfangen kann und an das `ACC` Objekt über Voyagerkommunikation weiterleitet. Beim Start des `IIOPReceivers` wird das Objekt FIPA konform an den Voyager ORB für IIOP Kommunikation exportiert, um Nachrichten empfangen zu können. Die dabei erzeugte IOR wird an den ACC geschickt, der diese IOR als die der Plattform ansieht und veröffentlicht. Aus diesem Grund implementiert das `IACC` Interface die zusätzliche Methode `setIOR(String)`, wie in Abbildung 7-5 zu sehen ist. Das Versenden von Nachrichten über Voyager und IIOP ist kein Problem und wird vom ACC durchgeführt.

Somit ist die Implementierung des ACC abgeschlossen und es können Agenten auf Basis der `FipaAgent` Klasse entworfen werden, deren ACL Nachrichten über den ACC an Empfänger weitergeleitet werden können.

7.2.4 Directory Facilitator Agent

Der DF Agent ist ein weiterer Bestandteil der FIPA-Plattform. Er bietet einen Katalogdienst an, in dem die Beschreibungen anderer Agenten, einschließlich den von ihnen angebotenen Diensten, nachgeschlagen, bzw. hinterlegt werden können. Auf jeder FIPA Plattform muss zumindest ein DF Agent existieren, der auch von Agenten fremder Plattformen nutzbar sein muss. Aus diesem Grund ist die Implementierung so weit wie möglich an den Vorgaben der FIPA gehalten und unterstützt alle spezifizierten Funktionen.

Der DF Agent führt Konversationen mit anderen Agenten, deren Ablauf über das *FIPA Request* Protokoll definiert wird. Er versteht Nachrichten, die durch die Agent Management Ontologie beschrieben werden und wird durch die Klasse `DFAgent` repräsentiert. Die `DFAgent` Klasse ist über die `DFRegistrableAgent` Klasse indirekt von der `FipaAgent` Klasse abgeleitet, und verfügt somit über allgemeine Fähigkeiten, wie das Senden und Empfangen von ACL Nachrichten. Auf die Funktionen der `DFRegistrableAgent` Klasse wird später noch einmal zurückgekommen. Bevor jedoch weiter auf die `DFAgent` Klasse eingegangen wird, soll zunächst die Nutzung und Abbildung des *FIPA Request* Protokolls vorgestellt werden. Mit Kenntnis über diesen zentralen Bestandteil des DF Agenten, ist die weitere Struktur recht einfach zu verstehen.

Konversation

Aus den Kapiteln 7.1.4 und 7.1.4.1, die eine allgemeine Abbildung der FIPA Protokolle und speziell die des *FIPA Request* Protokolls beschreiben, geht hervor, dass ein Protokoll jeweils durch zwei abstrakte Behaviourklassen dargestellt wird. Im Falle des *FIPA Request* Protokolls sind dies die Klassen `RequestInitiatorBehaviour` und `RequestResponderBehaviour`, die das Management mehrerer gleichzeitig geführter Konversationen und die Protokolllogik auf Seite des Initiators und Responders kapseln. Soll ein Dienst auf Basis dieses Protokolls angeboten werden, so ist jeweils von beiden Klassen eine weitere Klasse abzuleiten, die die Funktionalität des Dienstes implementiert. Auf Seite des DF Agenten, der den Responder des Dienstes darstellt, geschieht dies durch die Klasse `DFResponderBehaviour`, die lediglich die abstrakten Funktionen `checkRequest(ACLMessage)` und `handleRequest(ACLMessage)` der Basisklasse zu überschreiben braucht. Erstere von beiden wird von der Basisklasse aufgerufen, um zu prüfen, ob eine Anfrage verstanden wird und prinzipiell ausführbar ist. In diesem Fall genügt es, den Content der ACL Nachricht als gültigen Satz der Agent Management Ontologie zu interpretieren. Bei Erfolg wird eine `agree` Antwortnachricht erzeugt, die durch die Basisklasse an den Initiator zurück geschickt wird. Nach erfolgtem Versand der `agree` Nachricht wird die `checkRequest(ACLMessage)` Methode aufgerufen, die dem DF Agenten die eigentliche Arbeit ausführen lässt.

Kommen wir wieder auf die `DFAgent` Klasse zurück. Das `DFResponderBehaviour` wird genutzt, indem alle beim DF Agenten eingehenden Nachrichten an eine Instanz dieser Klasse weitergeleitet werden. Als Resultat wird vom `DFResponderBehaviour` einmal pro Request die `executeAction(AMAction)` Methode der `DFAgent` Klasse aufgerufen, die die angeforderte Aktion (Registrierung, Suche, ...) ausführt. Der Programmablauf dieser Methode soll an dieser Stelle nicht weiter beschrieben werden, weitere Details können auch hier wieder dem Quellcode entnommen werden.

Speicherung der Daten

Noch zu klären ist, in welcher Form die einzelnen Registrierungen gespeichert und verwaltet werden. Das Design gibt Gründe an, diese Objekte persistent zu speichern. Im Kapitel über die Agent Management Ontologie wurde beschrieben, dass solch eine Beschreibung durch die Klasse `DFAgentDescription` abgebildet wird. Es ist also eine

Lösung zu finden, in der dieses Java Objekt persistent gespeichert und komfortabel wiedergefunden werden kann. Prinzipiell gibt es etliche Varianten ein Objekt persistent zu speichern. Sie reichen von File basierten Ansätzen, wie die Umwandlung des Objektes in einen XML String oder die Speicherung eines *Java Serialized Objektes* in einem File, über die Nutzung von relationalen oder objektorientierten Datenbanken bis hin zum *Directory Service*.

Die Auswahl einer geeigneten Variante wird stark vereinfacht, durch eine genauere Betrachtung der Anforderungen zum Auffinden eines gespeicherten Eintrages. Eine Suchanfrage kann ein `DFAgentDescription` Objekt beinhalten, dessen vorhandene Parameter die Suche einschränken. Das Ergebnis besteht aus allen Registrierungen, die den vorhandenen Parametern entsprechen, sie dürfen jedoch weitere Parameter besitzen. Eine Suchanfrage kann zum Beispiel nur auf den Parameter `language` eingeschränkt werden und als Ergebnis sollten alle Registrierungen ausfindig gemacht werden, die zumindest dem Wert dieses Parameters entsprechen. Bei einer datei-basierten Speicherung der Daten, müssten alle Datensätze gelesen werden und manuell mit den Werten der Suchanfrage verglichen werden. Ein datenbank-basierter Ansatz würde eine Suchanfrage (*select* Statement) erzeugen, die die Suche in der Datenbank entsprechend einschränkt. Die Suchanfrage könnte je nach den Suchbedingungen sehr komplex werden. Ein Directory Service bietet in diesem Fall die komfortablere Möglichkeit, da die Suche und Auswahl vollständig vom Directory Service übernommen wird. Ein weiterer Grund für die Verwendung eines Directory Services ist, dass Einträge bestimmten Kontexten zugeordnet werden können. Dadurch wird eine Suche möglich, die alle Registrierungen eines Kontextes, oder zusätzlich alle Registrierungen aller Subkontexte zurückliefert. Der in Kapitel 6.3.2 vorgeschlagene Ansatz, einen Informationsagenten unter einen Kontext zu speichern, der das Themengebiet des Agenten beschreibt, kann so sehr einfach realisiert werden. Um FIPA konform zu bleiben, wird ein Ansatz realisiert, in dem grundsätzlich alle Subkontexte durchsucht werden Ist in einer Registrierung kein Kontext angegeben, so wird die Registrierung unter dem obersten Kontext gespeichert, ansonsten unter dem angegebenen Kontext. So wird sichergestellt, dass bei einer Standardanfrage alle registrierten Agenten mit in die Suche einbezogen werden, bei einer speziellen Anfrage jedoch nur Agenten, die unterhalb eines bestimmten Kontextes registriert sind.

Der Zugriff auf einen Directory Service wird von der Klasse `DFEngine` implementiert. Sie wird von der `DFAgent` Klasse benutzt und bietet ihr ein Interface an, dass das die vier DF Operationen abbildet.

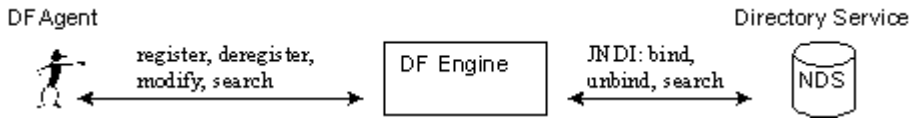


Abbildung 7-8: Struktur des DF Agenten

Der Zugriff aus einen Directory Service kann über eine herstellerspezifische API geschehen, über das *Lightweight Directory Access Protocol* (LDAP) [25], sofern der Directory Service LDAP kompatibel ist, oder über Suns JNDI Spezifikation, die eine einheitliche Schnittstelle definiert, über die sowohl auf LDAP basierende, als auch andere Directory Dienste genutzt werden können. Da die JNDI Schnittstelle recht einfach zu nutzen ist und den Vorteil hat, dass die Implementierung unabhängig vom verwendeten Directory Service ist, wird sie von der `DFEngine` Klasse genutzt. Als Directory Service wird der *Netscape Directory Server* (NDS) verwendet. Dieses Produkt unterstützt die JNDI Schnittstelle und ist für die Zwecke dieser Arbeit frei verfügbar.

Nutzung des DF Dienstes

Die Beschreibung des DF Agenten ist soweit abgeschlossen, beschränkt sich aber bisher nur auf den Anbieter des DF Dienstes. Nun soll auf die Benutzung des DF Dienstes eingegangen werden.

Oben wurde schon erklärt, dass zur Nutzung des *FIPA Request* Protokolls zwei abstrakte Klassen zur Verfügung stehen, von denen jeweils eine weitere Kasse erbt und dienstspezifische Details implementiert. Dies wurde bereits anhand des DF Agenten und der `DFResponderBehaviour` Klasse beschrieben, und soll nun auch für den Initiator geschehen.

Damit der DF Dienst von anderen Agenten genutzt werden kann, wurde die Klasse `DFInitiatorBehaviour` von dem `RequestInitiatorBehaviour` abgeleitet. Sie stellt eine recht komfortable Schnittstelle zum Start einer neuen Konversation zur Verfügung und implementiert alle abstrakten Methoden der Basisklasse. Der Agent kann durch dieses Behaviour mehrere Konversationen, auch mit unterschiedlichen DF Agenten, zur selben Zeit zu führen. Alle ACL Nachrichten, die beim Agenten ankommenden und zur Konversation mit dem DF Agenten gehören, müssen natürlich an das `DFInitiatorBehaviour` weitergereicht werden, damit Konversationen korrekt geführt werden können.

Der DF Dienst ist ein recht wichtiger Dienst und wird von vielen FIPA Agenten genutzt. Damit dies komfortabel geschehen kann und nur einmal implementiert werden muss, gibt es eine weitere abstrakte Agenten Klasse mit dem Namen `DFRegistrableAgent`. Sie ist im Paket *fipa.util.agent* zu finden und erbt von der `FipaAgent` Klasse. Sie nutzt das soeben beschriebene `RequestInitiatorBehaviour` und definiert die abstrakte Methode `getDFDescription`(), die von jedem abgeleiteten Agenten überschieben werden muss, damit eine individuelle DF Beschreibung beim jeweiligen DF Agenten registriert werden kann. Sie bietet allen abgeleiteten Klassen komfortable Methoden, wie `registerToDF`() oder `search(DFAgentDescription)`, zur Nutzung des DF Dienstes an.

Als Beispiel für einen Agenten, der von dieser Klasse ableitet, ist der DF Agent selbst zu nennen, denn um eine *Federated Search* durchführen zu können, muss er selbst als Initiator von Konversationen mit anderen DF Agenten auftreten. Wird eine Suchanfrage bearbeitet, deren Suchtiefe größer eins ist, so wird zuerst der eigene Directory Service nach Antworten durchsucht und dann die Suchanfrage an alle bekannten DF Agenten weitergeleitet. Dies geschieht am einfachsten durch die Funktionalität der `DFRegistrableAgent` Klasse. Sind alle Konversationen beendet, so kann das gemeinsame Suchergebnis an den Benutzer geliefert werden. Auch für die Registrierung eines DF Agenten bei DF Agenten anderer Plattformen bietet diese Basisklasse eine sehr nützliche Schnittstelle, der Agent kann somit bei der vereinten Suche anderer Agenten eingeschlossen werden.

7.2.5 TestAgent

Die Funktionalität der benötigten FIPA Plattform ist somit vollständig beschrieben. Zusätzlich soll ein weiterer Agent vorgestellt werden, der kein Bestandteil der Plattform ist, sondern die Plattform nutzt und ihre Funktionen testet.

Der Test Agent verfügt über eine grafische Oberfläche (siehe Abbildung 7-9), die Statusinformationen, ankommende und ausgehende Nachrichten darstellt. Er ist abgeleitet von der Klasse `DFRegistrableAgent` und hat somit die Fähigkeit den DF Dienst zu nutzen. Da die Basisklasse abstrakt ist, muss der `TestAgent` die abstrakte Methode `getDFDescription()`überladen, die seine eigene DF-Beschreibung erstellt. Über spezielle Textfelder der GUI kann die Adresse des DF Agenten angegeben werden,

mit denen der Test Agent kommunizieren soll. Somit ist es möglich, nicht nur die Funktionen der eigenen Plattform zu testen, sondern auch die anderer Plattformen. Dies können Instanzen der hier vorgestellten Plattform sein, aber auch andere Implementierungen, wie FIPA OS. In einem weiteren Textfeld kann der Pfad zu einem Textfile angegeben werden, im dem eine spezielle Suchanfrage im SL0 Format steht. Dieses wird ausgelesen und ermöglicht, durch Änderung der Suchanfrage, unterschiedliche Suchfunktionen, wie die *Federated Search* oder eine Begrenzung der Ergebnisanzahl, zu testen.

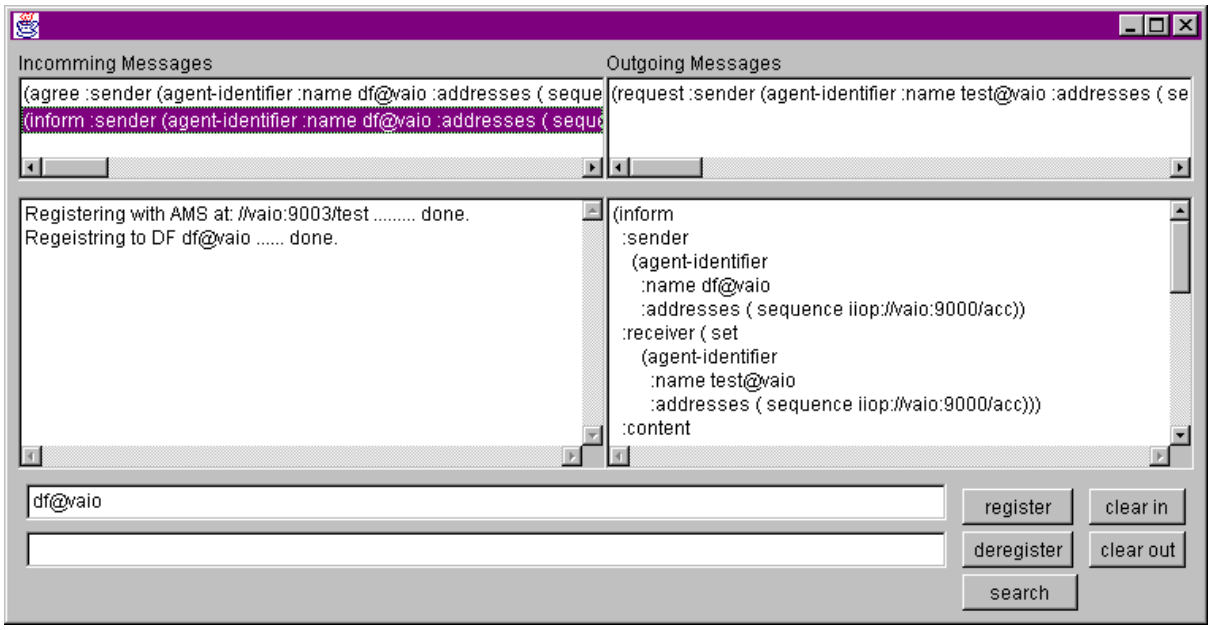


Abbildung 7-9: GUI des Test Agenten

Mit Abschluss der Vorstellung des Test Agenten ist die Beschreibung einer Plattform, die zumindest nach außen dem FIPA Standard entspricht, vollständig und der erste Teil dieser Arbeit abgeschlossen. Da mit einer Plattform und einem einfachen Test Agenten nicht viel anzufangen ist, ist es an der Zeit den ersten *richtigen* Agenten zu realisieren.

7.3 Informationsagent

Erste Überlegungen, wie ein Informationsdienst aussehen könnte, wurden schon in Kapitel 4 angestellt. Das Design hat gezeigt, dass diese Ansätze im allgemeinen zu realisieren sein müssten, indem viele noch nicht durchdachte Punkte aufgedeckt wurden und zu vorhersehbaren Problemen neue Lösungswege diskutiert worden. Nun gilt es, diese Ansätze von der Registrierung beim DF Agenten über die Gestalt des Contents der ACL Nachrichten, bis hin zum Ablauf eines Dialoges in Einklang mit der vorhandenen FIPA Plattform zu bringen.

Der Informationsagent unabhängig von der Art und Darstellung der auszutauschenden Information sein und wurde dementsprechend allgemein entworfen. Erst im nächstem Kapitel soll anhand eines Beispieles diese Flexibilität aufgegeben und ein spezieller Informationsdienst angeboten werden. Dieser Ansatz verhindert es, dass der Agent selbst die angebotenen Informationen kennt, da eine Zuordnung zwischen einem Begriff und der Information beliebig komplex sein und nicht allgemein abgebildet werden kann. Deshalb verhüllt er eine Applikation(engl. to wrap), die über Informationen und deren Semantik verfügt. Aus dieser Eigenschaft leitet sich der Klassename `WrapperAgent` des Informationsagenten ab.

Wie jeder Agent der Plattform, müsste auch dieser Agent von der Klasse `FipaAgent` erben. Da aber zum Auffinden anderer Wrapper Agenten und zum Veröffentlichen des eigenen Dienstes der Dienst des DF Agenten genutzt wird, ist die Klasse `WrapperAgent` von der Klasse `DFRegistrableAgent` abgeleitet. Somit wird die Funktionalität den DF Dienst zu nutzen und allgemeine Fähigkeiten, über die jeder FIPA Agent verfügen muss (z.B. Senden und Empfangen von Nachrichten), geerbt. Aus der obigen Beschreibung des Test Agenten sind die hieraus resultierenden zusätzlichen Aufwendungen schon bekannt und werden nicht noch einmal beschrieben.

Der Informationsagent tritt als Akteur in zwei unterschiedlichen Rollen auf. Auf der einen Seite übernimmt er die Rolle des Initiators, um Informationen zu suchen und auf der anderen Seite die Rolle des Responders, als Informationsanbieter. Als Responder wird sein Dienst von anderen Agenten recht einfach genutzt, indem ihm ACL Nachrichten geschickt werden. Etwas komplizierter ist es, in dieser Rolle die angefragten Informationen zu beschaffen, da, wie schon gesagt, der Wrapper Agent selbst über keine Informationen verfügen soll. Er muss in der Lage sein, eine externe Anwendung oder einen weiteren Agenten kontaktieren zu können. Da Beide nur eine spezielle Art von Informationen anbieten und sie auf unterschiedlichste Weisen kontaktiert werden können, kann an dieser Stelle die Art der Kontaktierung noch nicht festgelegt werden. Auf eine mögliche Art der Anbindung wird im nächsten Kapitel eingegangen.

In der Rolle des Initiators kehrt sich dieses Verhalten um. Informationen werden besorgt, indem mit anderen Agenten, die nun in der eben beschriebene Rolle des Responders agieren, über ACL Nachrichten kommuniziert wird. Am einfachsten in Anspruch genommen wird dieser Dienst über ein Interface, das seinen Benutzern zur Verfügung gestellt wird. Dieses Interface könnte auch über Voyager veröffentlicht werden, da dieser ORB auch schon zur Kommunikation mit der Plattform genutzt wird. Natürlich ist diese Art der Kommunikation ein Bruch in der Agentenkommunikation, doch würde der Wrapper Agent über ACL Nachrichten kontaktiert werden, müsste sein Benutzer zwingend ein Agent sein, der seine Anfrage in ACL und einem speziell definierten Content ausdrücken kann. Der Wrapper Agent soll aber auch von normalen Anwendungen genutzt werden können, weshalb an dieser Stelle einfache CORBA Kommunikation genutzt wird. Die Klasse `WrapperAgent` implementiert dazu das Interface `IWrapperAgent`, das lediglich die Funktion `requestInformation` bereitstellt. Als Parameter wird dieser Methode der gesuchte Begriff übergeben. Sie startet die Suche des Agenten und gibt das Ergebnis als String zurück. Um jedoch nicht schon an dieser Stelle zu sehr ins Detail zu gehen, wird später noch einmal an die Funktion dieser Methode angeknüpft.

Zusammengefasst kann der Informationsagent zu diesem Zeitpunkt, ähnlich wie der Test Agent, die Dienste der Plattform, einschließlich dem des DF Agenten, nutzen. Zusätzlich sind alle Rollen bekannt, die angenommen werden können, und die Schnittstellen, über die der Wrapper Agent genutzt werden kann oder sich Informationen beschafft. Bevor jedoch diese Rollen implementiert werden können, muss der gültige Content der ACL Nachrichten bekannt sein, die an einer Konversation beteiligt sind. Erst mit dieser Ontologie ist es möglich ein spezielles InformationsBehaviour, auf Basis des *FIPA Iterated Contract Net* Protokoll, und den Ablauf im Wrapper Agenten zu realisieren.

7.3.1 Ontologie

Im Design wurde schon ausgearbeitet, welche Semantik die einzelnen Nachrichten der Konversation ausdrücken soll. Es wurde festgelegt, dass das Ergebnis der Konversation, die eigentliche Information, nicht mit abgebildet werden braucht und die Beschreibung der gesuchten Information sowie die Vorschläge zu vorhandenen Informationen durch

einfache Begriffe beschrieben werden sollen. Die daraus hergeleitete Konsequenz war, dass eine sehr einfache Ontologie ausreichend ist, die mit der SL0 dargestellt werden kann. Der Vorteil in einer Darstellung des Contents in SL0 ist, dass bereits Parser und Composer vorhanden sind. Lediglich der Content selbst und entsprechende Interpreter und Composer aus und in die SL0 Darstellung werden noch benötigt.

In Anlehnung an den DF Agenten, könnte der Content durch Aktionen ausgedrückt werden. Von der Semantik her, die durch die Nachrichten ausgedrückt wird, würde der Anbieter aufgefordert eine Aktion auszuführen, nämlich nach Informationen oder Wissen nachzuschlagen bzw. zu versenden. Anders als beim DF Agenten geschieht dies nicht direkt, durch eine einfache Anforderung (`request`), sondern es werden zuerst Vorschläge (`call for proposal`) für mögliche Aktionen angefordert. Erst im nächsten Schritt wird dem Anbieter mitgeteilt, ob er die vorgeschlagene Aktionen ausführen, und somit die Information verschicken soll oder nicht. Aus dieser Betrachtungsweise ergibt sich, dass der Content aller ACL Nachrichten durch nur eine einzige Aktion ausgedrückt werden kann. In ihr werden die Begriffe abgebildet, die die auszutauschende Information beschreiben. Der Typ der ACL Nachricht reicht aus, um den jeweils ähnlichen Content der Nachrichten völlig unterschiedliche Bedeutungen zu geben. Folgende Aktion, dargestellt in SL0, soll die Semantik der Nachrichten abbilden:

```
(action
  (agent-identifier
    :name sport-agent@iiop://fipaland.org:9000/acc
    :addresses (sequence iiop://agentland.com:9000/acc))
  (knowledge
    :concept football
    :concept rugby
    :concept … ))
```

Abbildung 7-10: Beispiel - Content des Informationsagenten

Die Abbildung dieser Ontologie in die Programmiersprache Java, einschließlich Interpreter und Composer ist sehr einfach und ähnelt prinzipiell der Abbildung der ausführlich beschriebenen Agent Management Ontologie aus Kapitel 7.1.3. Details können deshalb den entsprechenden Klassen entnommen werden. Sie sind im Java Packet *fipa.ont.flm* zu finden.

Nachdem nun auch das Format der gültigen ACL Nachrichten definiert ist, kann nun weiter auf die Realisierung der Konversation eingegangen werden.

7.3.2 Anbieten von Informationen

In der Rolle des Responders muss die Klasse `WrapperAgent` die beiden Eigenschaften implementieren den angebotenen Dienst beim lokalen DF Agenten zu registrieren und eine der Rolle entsprechende Konversation nach dem FIPA ICN Protokoll führen zu können.

7.3.2.1 Publizierung des Dienstes

Die Fähigkeiten den DF Dienst nutzen zu können werden von der Klasse `DFRegistrableAgent` geerbt. Die Registrierung des eigenen Dienstes wird beim Start des Wrapper Agenten durchgeführt.

Der Kontext, zu dem Informationen angeboten werden, ist in der DF Beschreibung des eigenen Dienstes als zusätzliche Eigenschaft mit anzugeben, wie aus Abbildung 7-11 zu entnehmen ist. Dieser Wert ist abhängig von der angebotenen Information und wird

momentan aus einem Konfigurationsfile gelesen, welches in folgenden Versionen aber auch geschickter gelöst werden könnte.

```
(df-agent-description
  :name
    (agent-identifier
      :name  information-agent@agentland.com
      :addresses (sequence iiop://agentland.com:9000/acc))
  :language (set fipa-sl0)
  :ontology (set information-ontology)
  :protocol (set fipa-iterated-contract-net)
  :services (set
    (service-description
      :name information-service
      :type information
      :ontology information-ontology
      :properties (set
        (property
          :name context
          :value flirt)
        (property
          :name result-type
          :value xml)))))
```

Abbildung 7-11: Mögliche DF-Beschreibung eines Wrapper Agenten

In der aktuellen Implementierung des DF Agenten werden die Beschreibungen registrierter Agenten bekanntlich in einem Directory Service abgelegt, dessen Struktur einem Kontextbaum entspricht. Leider beinhaltet dies den Nachteil, dass die Struktur des Baums allen Agenten bekannt sein muss, damit sie sich unter einen Kontext registrieren können. Es steht nur eine begrenzte Auswahl an Kontexten zur Verfügung.

7.3.2.2 Ablauf der Verhandlung

Ist der Agent gestartet und sein Dienst am lokalen DF registriert, kann er von anderen Agenten genutzt werden. Als Kommunikationsbasis wird das *FIPA Iterated Contract Net* Protokoll (ICN) genutzt. Es werden nur Nachrichten verstanden, die der oben definierten Ontologie entsprechen.

Die Realisierung der Protokolllogik des ICN Protokolls wurde bereits in Kapitel 7.1.4.2 vorgestellt und wird somit als bekannt vorausgesetzt. Um Konversationen nach diesem Protokoll entsprechend der Rolle des Responders führen zu können, wird nach bekanntem Prinzip eine weitere Klasse von der `ICNResponderBehaviour` Klasse abgebildet. Dies geschieht durch die Klasse `WrapperResponderBehaviour`, die im wesentlichen in den zu überschreibenden abstrakten Methoden

- `handleCFP`
- `handleAcceptProposal`
- handleRejectProposal

die Logik des Informationsdienstes implementiert. Zur Erinnerung, diese Methoden werden bei Erhalt einer entsprechenden ACL Nachricht aufgerufen und sollen protokoll-unabhängige Funktionalität implementieren. Erst dieser speziellen Behaviourklasse ist die oben definierte Ontologie bekannt, so dass der Content der ACL Nachrichten auch erst durch diese Klasse interpretiert werden kann.

Bei Erhalt einer ACL Nachricht von Typ `cfp` wird zunächst vom Behaviour eine neue ICN Konversation angelegt, von der aus die Methode `handleCFP` aufgerufen wird. Der Content der ihr übergebenen ACL Nachricht wird hier zunächst interpretiert. Zu jedem darin enthaltenen Begriff wird über die `WrapperAgent` Klasse eine Anwendung, nach

Vorschlägen gefragt. Dazu bietet die dem Behaviour bekannte `WrapperAgent` Klasse die Methode `getProposals` an, die als Parameter einen Begriff erwartet. Das Ergebnis dieses Aufrufes sind Begriffe, die Vorschläge zu vorhandenen Informationen darstellen. Diese werden dem Initiator der Konversation geantwortet. Daraufhin wird eine Nachricht vom Typ `accept-proposal` oder `reject-proposal` erwartet, bevor die Konversation beendet wird. Die entsprechende Methode `handleRejectProposal` könnte theoretisch einige Aufräumarbeiten ausführen, im diesem Fall hat sie jedoch nichts zu tun. Die `handleAcceptProposal` Methode hat die Aufgabe die nun angeforderte Information auch wirklich zu beschaffen. Nachdem der Content dieser ACL Nachricht interpretiert wurde, übergibt sie den die angeforderte Information beschreibenden Begriff an den Wrapper Agenten, der die entsprechende Information von der Anwendung besorgt. Dazu wird eine `getInformation` Methode der `WrapperAgent` Klasse aufgerufen, die eine weitere Schnittstelle zur Anwendung repräsentiert.
Wie schon erwähnt, ist das Format der Information dem Wrapper Agenten egal, sie muss lediglich in einem String abbildbar sein. Nach Versand der Information an der Initiator, ist die Konversation erfolgreich beendet.

Damit obige ICN Konversationen nach diesem Ansatz geführt werden können, müssen alle zugehörigen ACL Nachrichten an die Responder-Behaviour Klasse weitergeleitet werden. Dies ist vom Wrapper Agenten nicht ganz einfach zu gewährleisen, da an dieser Stelle ein Agent zum ersten mal ankommende ACL Nachrichten an mehrere Behaviour aufteilen muss. Zu einem wird die `DFInitiatorBehaviour` Klasse genutzt, um den eigenen Dienst beim DF Agenten zu veröffentlichen, zu anderem die `ICNResponderBehaviour` Klasse, um den Informationsdienst anbieten zu können. Erschwerend kommen noch zwei weitere Behaviour dazu, da der Wrapper Agent auch nach weiteren Informationsagenten suchen und mit diesen verhandeln können soll. Eingehende ACL Nachrichten sind zunächst eindeutig einem Behaviour zuzuordnen, bevor sie weiterverarbeitet werden können. Um das Problem zu beheben, wird eine Helferklasse genutzt, an die der Agent alle eingehenden ACL Nachrichten weiterleitet. Ihr sind alle unterschiedlichen Behaviourinstanzen bekannt und sie leitet die Nachricht zum entsprechenden Behaviour weiter. Um eine Zuordnung treffen zu können, wird jedes Behaviour gefragt, ob es eine Konversation mit einer, in der ACL Nachricht übermittelten ID führt. Trifft dies zu, so wird die Nachricht an das entsprechende Behaviour weitergeleitet. Eine ACL Nachricht kann aber auch die Initialnachricht einer Konversation sein, die das Responder-Behaviour noch nicht kennt. In diesem Fall wird jedes Responder-Behaviour gefragt, ob dessen Initialnachricht dem Typ der ankommenden Nachricht entspricht. Bei Übereinstimmung wird die ACL Nachricht an dieses Behaviour weitergeleitet. Mit dieser zweiten Kondition ist ein Agent zur Zeit leider darauf beschränkt zu jedem ACL Typ nur ein Responder-Behaviour anbieten zu können, dessen Initialnachricht mit diesem Typ beginnt. Für die Zwecke dieser Arbeit ist diese Einschränkung genügend. In Zukunft könnten zusätzlich zum ACL Typ weitere Bedingungen geprüft werden, wie die Zugehörigkeit der Nachricht zu dem, vom Behaviour verwendeten Protokoll, der Contentsprache oder Ontologie.

Der Wrapper Agent ist nun in der Lage ACL Nachrichten dem richtigen Behaviour zuordnen zu können. Das `WrapperResponderBehaviour` interpretiert und verarbeitet alle ankommenden ACL Nachrichten und ruft Methoden der `WrapperAgent` Klasse auf und befragt damit den Agenten nach Vorschlägen zu einem Begriff bzw. über die eigentliche Information. Da die Information abhängig von der verwendeten Anwendung ist, werden die Methoden zunächst einmal abstrakt definiert. Im nächsten Kapitel werden sie überschrieben, um eine Anwendung zu kontaktieren.. Somit ist der Darstellung die

Responder-Rolle des Wrapper Agenten ausreichend beschrieben und es soll mit der Initiator-Rolle fortgefahren werden.

7.3.3 Auffinden von Informationen

Wird der Wrapper Agent in seiner Rolle als Initiator genutzt, ist dessen Methode `requestInformation` aufzurufen, die am Anfang dieses Abschnittes schon einmal erwähnt wurde. Sie wird über das Interface `IWrapperAgent` nach außen bereit gestellt und kann von externen Anwendungen genutzt werden. Sie führt im groben folgende fünf Schritte aus. Das Auffinden von Informationsanbietern, das Einholen von Vorschlägen, das Auswählen eines Vorschlags und Anfordern der Information und letztendlich die Rückgabe der gewünschten Information an den Benutzer. Der erste Schritt ist eine Suchanfrage beim DF Agenten. Die eigentliche Verhandlung findet bekanntlich über das *FIPA Iterated Contract Net* Protokoll, wobei in diesem Fall der Initiator-Teil des Behaviours zu nutzen ist.

7.3.3.1 Auffinden weiterer Informationsanbieter

Die Funktionalität, eine Suchanfrage an einen DF Agenten zu schicken, wird wie schon bei der Veröffentlichung des Dienstes von der Basisklasse geerbt. Als Parameter der Suche wird eine DF Beschreibung benötigt, die den zu nutzenden Informationsdienst beschreibt. Da jedoch einige Daten dieser Beschreibung, wie der Kontext, sich auf den speziellen Informationsdienst beziehen, kann die Klasse WrapperAgent diese Information nicht implementieren. Statt dessen wird eine abstrakte Methode bereitgestellt, in der ein spezieller Informationsagent die Beschreibung Agenten zu implementieren hat, die potentielle Informationsanbieter sein könnten. Eine mögliche DF Beschreibung für eine Suchanfrage wird in der folgenden Abbildung gezeigt.

```
(df-agent-description
  :language (set fipa-sl0)
  :services (set
    (service-description
      :name information-service
      :ontology knowledge
      :properties (set
        (property
          :name context
          :value leben.liebe.flirt)
        (property
          :name result-type
          :value xml-flirtmaschine)))))
```

Abbildung 7-12: Einschränkende Kriterien für eine Suchanfrage beim DF

Zum Ausführen der Suche wird lediglich die search Methode des DF Initiator-Behaviours aufgerufen. Ihr wird eine Beschreibung der gesuchten Agenten und einige Optionen übergeben, die besagen, dass die *Federated Search* (7.2.4) mit der festen Suchtiefe 3 genutzt werden soll. Nach erfolgter Suche wird das Ergebnis um die eventuelle eigene Registrierung bereinigt. Falls nun keine weiteren Agenten in der Liste der zu befragenden Agenten verbleiben, muss die Suche nach Informationen an dieser Stelle abgebrochen werden. In diesem Fall wir ein leerer String an den Benutzer zurückgegeben. Wurden Agenten gefunden, die möglicherweise Informationen zu dem Begriff anbieten können, wird nun mit ihnen eine Verhandlung gestartet.

7.3.3.2 Verhandlung um Informationen

Diese führt im wesentlichen die Klasse `WrapperInitiatorBehaviour`, die von der Klasse `ICNInitiatorBehaviour` abgeleitet ist. Dieses Behaviour ermöglicht es eine

Konversation mit mehreren Agenten über das FIPA Contract Net Protokoll zu führen. Details sind aus Kapitel 7.1.4.2 zu entnehmen. Das `WrapperInitiatorBehaviour` implementiert die abstrakten Methoden

- `createCFPContent` und
- `handleProposals`,

die die Funktionalität des Informationsdienstes implementieren.

Die `createCFPContent` Methode wird beim Start des Behaviours aufgerufen und erzeugt den Content der ACL Nachrichten, die an alle Teilnehmer geschickt werden. Ihr werden alle Begriffe übergeben, die die gesuchte Information beschreiben.

Nachdem die ACL Nachricht des Typs `cfp` erzeugt wurde, wird sie an alle teilnehmenden Agenten geschickt. Die Konversation befindet sich nun im Zustand `propose`, sammelt alle eingehenden Antworten und übergibt sie, nach Eingang der letzten Nachricht, an die `handleProposals` Methode. Diese interpretiert zunächst den Content jeder Antwort anhand der oben definierten Ontologie und speichert jeden einzelnen vorgeschlagenen Begriff zusammen mit dem Sender der Nachricht. Daraufhin wird ein Vorschlag ausgewählt, welches momentan zufällig passiert. In Zukunft wäre es sinnvoll die Vorschläge an den Agenten zurückzureichen, der eine Auswahl vornimmt, oder sie an den Benutzer weiterreicht, der dann eine Auswahl vornehmen kann.

Nach der Auswahl wird eine Menge von ACL Nachrichten erzeugt, die je nach Empfänger einen Vorschlag ablehnen oder annehmen, sprich den ACL Typ `accept-proposal` oder `reject-proposal` haben. Diese Nachrichten werden der Basisklasse zurückgegeben und nach Hinzufügen von weiteren Informationen, wie dem Protokoll, der Konversations-ID usw. versandt.

Die Konversation erwartet nun die angeforderte Information, von den Agenten, denen eine `accept-proposal` Nachricht geschickt wurde. Nach Erhalt der `inform` Nachricht(en) wird die Konversation beendet und die ganze ACL Nachricht an den Wrapper Agenten, oder genauer die `requestInformation` Methode, die die Konversation gestartet hatte, zurückgegeben. Die Suche ist somit abgeschlossen und der Content der ACL Nachricht wird an den Aufrufer zurückgegeben.

Somit ist auch die Beschreibung des Informationsagenten vollständig. Er ist in der Lage je nach Bedarf Informationen aufzufinden, die als String repräsentiert werden können. Das Format des Strings (HTML, XML ...) spielt dabei keine Rolle. Im Kapitel 8 wird auf Basis dieses abstrakten Wrapper Agenten ein kleines Anwendungsszenario beschrieben, in dem der Agent spezielle Informationen zum Thema Flirt anbietet und ausfindig macht. Da an dieser Stelle der praktische Teil zum größten Teil abgeschlossen ist, soll es an dieser Stelle schon einmal zu einer Zusammenfassung und Betrachtung der Realisierung kommen und auf problematische Stellen bei der Implementierung eingegangen werden.

7.4 Abschließende Betrachtung

Es wurde eine Agentenplattform realisiert, die den zuvor ausgearbeiteten Anforderungen eines Informationsagenten gerecht wird. Dies zeigt die recht problemlose Realisierung des Wrapper Agenten. Die Realisierung begann zu einer Zeit, in der die Version 97 – 99 des Standards aktuell waren. Diese waren jedoch in vielen Punkten nicht detailliert genug ausgeführt und in anderen Details wiederum zu theoretisch, was die Implementierung sehr erschwert hat. Mit Veröffentlichung der Version 2000 des Standards wurden viele dieser Probleme behoben, weshalb mitten in der Realisierung der zusätzliche Aufwand, einer Anpassung der bestehenden Plattform an die aktuellste Version des Standards,

betrieben wurde. Aus diesem Grund entspricht die Struktur der Plattform auch noch der Version 97. Leider hat auch die Version 2000 noch ihre Schwächen, wie die schon angesprochene mangelnde Beschreibung der Verbreitung der Adresse einer Plattform. Auch das Fehlermanagement, das bisher nicht angesprochen wurde, ist nicht ausreichend beschrieben. Zwar ist das Format der gültigen Fehler, die in der Agent Management Ontologie definiert werden, deutlicher beschrieben als in den Vorgängerversionen, doch fehlt immer noch eine genaue Zuordnung zwischen einer Handlung und den dabei auftretenden Fehlern.

Abgesehen von allerlei Kleinigkeiten, die dem Standard angekreidet werden können, ist aber auch positive Kritik zu äußern. Der FIPA Standard ist flexibel genug zusätzliche vom Standard abweichende Anforderungen abzubilden. Ich denke zum Beispiel an die Erweiterung der DF Registrierung um einen Kontext und den Typ der angebotenen Information. Konzeptuell ist der Standard gut durchdacht und als Basis einer Agentenplattform zu empfehlen. Allerdings deckt er so viele Aspekte ab, dass einiger Aufwand erforderlich ist, um sich ein Gesamtbild zu verschaffen und die, für die Anforderungen an eine Multiagentenplattform, relevanten Teile zu identifizieren. Die Entscheidung, den AMS Agenten nicht zu implementieren, darf nicht als Kritik verstanden werden, der FIPA Standard wäre zu komplex und an den realen Anforderungen einer Plattform vorbei spezifiziert werden. An anderen Stellen mag dies vielleicht gegeben sein. Doch auch rückblickend kann noch der Standpunkt vertreten werden, dass diese Komponente in dieser Form für die hiesigen Anforderungen überdimensioniert ist. Für andere Anforderungen wie z.B. Mobilität der Agenten ist der AMS Agent unumgänglich.

Bei der Realisierung der Plattform wurde Wert auf Kompatibilität zum FIPA Standard gelegt. Um zu zeigen, dass die eigene Plattform FIPA 2000 konform ist, wurden Tests mit einer der verbreiteten Plattform FIPA OS (siehe Kapitel 5.5) gemacht. Nachdem deren Variante der Veröffentlichung der Plattformadresse übernommen wurde, konnten im internen Netzwerk alle Aktionen des DF Dienstes erfolgreich getestet werden. Tests mit einer Instanz der Plattform im Internet, betrieben von der Firma Nortel, scheiterten leider an inkompatiblen IIOP Implementierungen. Die öffentliche FIPA-OS Plattform lief auf einer Java 1.2 Virtual Machine (VM) und verwendet das Sun CORBA, welches auf dieser Version der VM nicht mit dem IIOP vom Voyager kompatibel ist. Da die internen Tests mit FIPA OS, laufend auf einer Java 1.3 VM erfolgreich liefen, kann trotzdem von Kompatibilität der beiden Plattformen gesprochen werden.

Eindeutig positiv zu bewerten ist der Ansatz zur Abbildung der Protokolle, wie die Erfahrungen mit der Implementierung des DF Agenten und des Wrapper Agenten gezeigt haben. Es konnte die gesamte Protokolllogik in abstrakten Klassen gekapselt werden, die den Ablauf und die Verwaltung aller Konversationen übernimmt. Dienste auf Basis eines Interaktionsprotokolls zu realisieren, ist durch diesen Ansatz sehr stark vereinfacht worden. Das einzige Problem dieses Ansatzes lag auf Seite des Agenten und ist die Zuordnung der eingehenden ACL Nachrichten zu einem Behaviour. Bietet ein Agent mehrere Dienste an, die auf demselben FIPA Protokoll basieren und demselben Typ der initialen ACL Nachricht erwarten, ist eine einfache Zuordnung einer ACL Nachricht nicht mehr möglich. Im Falle der Agenten, die hier beschrieben wurden, tritt dieses Problem noch nicht auf und es reicht eine Prüfung des Typs der Nachricht aus. In Zukunft sollte jedoch noch einmal überlegt werden, an welchen Merkmalen eine ACL Nachricht einem Behaviour eindeutig zuzuordnen ist.

Der realisierte Informationsagent wurde sehr generisch implementiert, um unabhängig von der auszutauschenden Information zu sein. Im Prinzip bietet dieser Ansatz die Möglichkeit mit jeglichen Informationsanbieten auf Basis des Wrapper Agenten zu

kommunizieren. Ob die Information verstanden werden kann, fällt in den Zuständigkeitsbereich des Benutzers, der diesen Agenten für seine Bedürfnisse anpassen muss. Wie ein solcher Agent zu verwirklichen ist und eine mögliche Anwendung aussehen kann, soll nun im nun folgenden Kapitel beschrieben werden.

8 Szenario

Anhand eines realen Beispiels soll eine Anwendungsmöglichkeit des Wrapper Agenten gezeigt werden. Dabei werden mehrere Instanzen einer Anwendung mit unterschiedlichen Wissensbasen so erweitert, dass ihr gemeinsames Wissen genutzt werden kann.

Als Wissensbasis dient die Flirtmaschine [26] der Firma Popnet Agentscape [27], bei der diese Arbeit verwirklicht wurde. Die Flirtmaschine ist ein Agent, der dem Benutzer Informationen rund ums Thema Liebe und Flirt anbietet. Er ist über das Internet zugänglich und baut automatisch ein persönliches Profils des Nutzers auf, anhand dessen ihm interessante Informationen angeboten werden. Zur Kommunikation mit dem Nutzer existiert ein sogenannter CYB (Create Your Bot) dargestellt durch eine kleine Zeichentrickfigur (siehe Abbildung 8-1), der mit dem User *spricht*, passiv auf eingegebene Fragen antwortet, und pro-aktiv nach Meinungen und Vorlieben fragt. Der CYB ist in der Lage, abhängig vom Verhalten des Nutzers Emotionen darzustellen. Der CYB kann vom Benutzer gelobt und gerügt werden und ist schlecht gelaunt, wenn mit ihm lange nicht gesprochen wird.

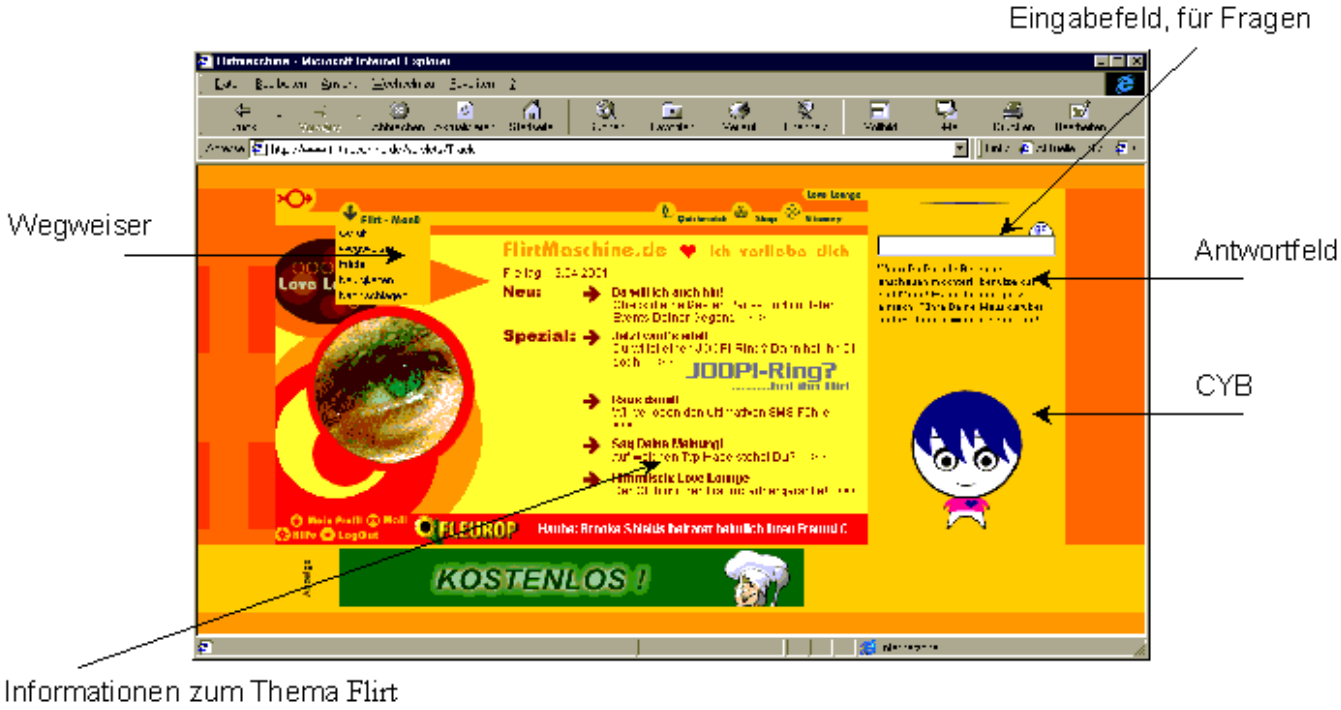


Abbildung 8-1: Oberfläche der Flirtmaschine

Anhand dieser Charakteristiken ist die Flirtmaschine im weiträumigen Bereich der Interface Agents aus Kapitel 2.2 einzuordnen, jedoch nicht zu den Informationsagenten zu zählen, da nicht eigenständig neue Informationen ausfindig gemacht werden.

Die Flirtmaschine soll durch Nutzung des Wrapper Agenten zu einem Informationsagenten erweitert werden. In einem ersten Ansatz soll dem CYB die Möglichkeit gegeben werden nach Informationen zu Fragen des Benutzers zu suchen, die durch die eigene Wissensbasis nicht beantwortet werden können. Dazu ist zunächst der verwendete Prozess der Spracherkennung zu verdeutlichen.

Dem CYB wird durch ein Eingabefeld eine Frage in deutscher Sprache gestellt. Durch *Pattern Matching*, einen Ansatz aus dem wissenschaftlichem Bereich *Speech Recognition / Natural Language Understandings* (NLU) [Allen 94], [Apostolico et al.

97] wird auf den Inhalt der Frage geschlossen und eine passende Antwort gesucht. Das *Pattern Matching* funktioniert im Prinzip so, dass verschiedenste Pattern definiert werden, die einen Satz auf ein bestimmtes Wort oder auf eine bestimmte Wort- und Zeichenfolge hin prüfen. Eine Vielzahl von Regeln wenden ein oder mehrere, zum Teil miteinander verknüpfte Pattern auf den eingegeben Satz an und liefern bei passender Bedingung vorgefertigte Antwortdialoge. Die Reihenfolge der Regeln ist so gewählt, dass anfangs sehr spezielle Regeln greifen, später aber sehr allgemeine Regen greifen, die als Antwortdialog eine ausweichende Aussage bestimmen, wie „Das habe ich gerade nicht verstanden.“.

Die Information, die ausgetauscht werden soll, ist in diesem Fall ein XML String, der einen Dialog beschreibt. Die Flirtmaschine wird also nur Informationen anderer Anbieter verstehen, die Dialoge in der selben Beschreibungssprache (definiert durch eine DTD) anbieten. Die Flirtmaschine basiert auf einem Framework genannt CyMON, mit dem ähnliche Agenten erstellt werden. Diese gemeinsame Basis ermöglicht es der Flirtmaschine Dialoge zu verstehen, die beispielweise von einem Reiseagenten stammen, der auch auf dem CyMON Framework basiert. Da die Beschreibung von Dialogen nicht standardisiert ist, ist dieser Ansatz auf die Kooperation von Agenten beschränkt, die auf dem CyMON Framework basieren.

Aus Gründen der Einfachheit, werden lediglich drei Flirtmaschinen so erweitert, dass bei nicht vorhandenem Wissen die anderen Instanzen zu Rat gezogen werden. Sie haben unterschiedliche Wissensbasen, die sich durch unterschiedliche Dialoge zu den Themen Wirtschaft, Politik und Tennis unterscheiden. Wird eine Frage zum Thema Wirtschaft an die Politik-Instanz gestellt, so sollen die anderen Instanzen um einen Antwortdialog gebeten werden.

8.1 FlirtAgent

Die Klasse Wrapper Agent repräsentiert einen generischen Informationsagenten. Lediglich die Darstellung der Information als Zeichenkette und deren Beschreibung durch eine eigens definierte Ontologie schränken den Informationsdienst auf syntaktischer Basis ein. Auf semantischer Ebene ist der Wrapper Agent unabhängig von der Art der anzubietenden Information. In Kapitel 4.2 wurde behauptet, dass ein Agent, der jegliche Art von Informationen ausfindig machen kann, gewisse Schwächen aufweist und eine Spezialisierung und Einschränkung des anzubietenden Informationsumfangs, ein interessanter Ansatz zur Lösung dieser Schwächen sein könnte. Die Vision ist, eine Vielzahl von Anbietern zu haben, von denen jeder Experte auf einem Gebiet ist und ganz spezielle Informationen anbieten kann.

Dieser Ansatz wird durch eine `FlirtAgent` Klasse verwirklicht, die von der Klasse `WrapperAgent` erbt. Die Informationen, die der `FlirtAgent` anbieten und auffinden soll, müssen einem Dialog der Flirtmaschine entsprechen, also einem XML String, der einer speziellen Sprache entspricht. Durch das Erben von der `WrapperAgent` Klasse, besitzt die `FlirtAgent` Klasse alle Funktionalität des Wrapper Agenten und muss lediglich alle Funktionen implementieren, die sich speziell auf den Flirtdienst beziehen. Neben der Anbindung der Flirtmaschine ist primär die eigene Beschreibung zur Registrierung beim DF Agenten, und die Beschreibung der zu suchenden Agenten zu implementieren, damit lediglich Agenten kontaktiert werden, die Informationen anbieten, die auch selbst verstanden werden. Zur Implementierung der eigenen DF Beschreibung muss der Kontext bekannt sein, unter dem sich der Flirtagent registriert. In diesem Szenario könnte dieser Kontext könnte für alle drei Flirtmaschinen gleich sein, in der Annahme, dass trotz der Unterschiede in ihrer Wissensbasis sie weiterhin Experten im Bereich Flirt sind. Ein anderer Ansatz wäre von drei unterschiedlichen Experten auszugehen, die in

unterschiedlichen Kontexten registriert sind. Bei jeder Suche müssten der oder die entsprechenden Agenten kontaktiert werden, die dem Kontext des zu suchenden Begriffs angehören. Dies ist bei einem sehr speziellen Begriff, wie *Bruttoinlandsprodukt* problematisch, da die Beziehung zum Kontext *Wirtschaft* für einen Sportexperten nicht einfach herzustellen ist.

Wegen dieser Probleme wird zunächst der einfache Ansatz gewählt, alle drei Agenten unter einem Kontext zu registrieren. Somit kann auch auf die Definition eines Kontextbaums verzichtet werden. Komplexe Szenarien sind mit diesem Ansatz allerdings schwer zu lösen.

Aus der DF Registrierung muss auch die Art des Dienstes bzw. die Art der angebotenen Information hervorgehen. Im Design in Kapitel 6.3.2 wurde hierfür ein zusätzliches `property` Konstrukt mit dem Namen `result-type` in der DF Beschreibung vorgesehen. Im Falle der Flirtmaschine könnte dieses den Wert *CyMON* haben. Es ist fragwürdig, inwiefern dieser einfache Ansatz auch in einer komplexeren Umgebung, mit sehr vielen Agenten und unterschiedlichsten Diensten, ausreichend ist. Für dieses Szenario soll er allerdings genügen.

Die Anbindung der `FlirtAgent` Klasse an die Flirtmaschine geschieht über Voyager. Um Dialoge oder Vorschläge zu einem Begriff nachzuschlagen, werden die Methoden zweier Komponenten der Flirtmaschine genutzt. Sie werden remote über die schon bekannte Voyagerkommunikation aufgerufen. Dazu implementiert der `FlirtAgent` die in Kapitel 7.3.2.2 schon vorgestellte, abstrakte Methode `getProposals` der `WrapperAgent` Klasse, um die Flirtmaschine nach Vorschlägen zu einem Begriff zu befragen. Eine andere Komponente der Flirtmaschine wird von der ehemals abstrakten `getInformation` Methode kontaktiert, um einen Antwortdialog zu besorgen.

Um nach unbekannten Dialogen zu suchen, kontaktiert die Flirtmaschine die `FlirtAgent` Klasse über Voyager, indem die das `IWrapperAgent` Interface nutzt.

Hiermit ist die Beschreibung des Flirt Agenten auch schon abgeschlossen, da keine weiteren Funktionen zu implementieren sind. Zum besseren Verständnis wird noch einmal auf die Flirtmaschine eingegangen und Erweiterungen beschrieben.

8.2 Erweiterungen an der Flirtmaschine

Zur Realisierung dieses Szenarios reicht es nicht aus nur den Flirt Agenten zu implementieren. Auch an der Flirtmaschine sind an einigen Stellen Anpassungen vorzunehmen, denn anstatt einen Ausweichdialog anzuzeigen, falls kein Dialog in der eigenen Wissensbasis vorhanden ist, soll bekanntlich der Flirt Agent kontaktiert werden. Diese Erweiterungen sollen jedoch auf konzeptioneller Ebene beschrieben werden, da eine Einführung in die beteiligten Komponenten der Flirtmaschine und deren Funktionsweise zu aufwändig wäre.

8.2.1 Auffinden eines Dialoges

Zunächst müssen drei unterschiedliche Wissensbasen bereitgestellt werden. Jede von ihnen wird um einige Dialoge bereichert bzw. beschränkt. Zusätzlich werden an Stellen, an denen die allgemeinen Regeln greifen, um ausweichende Dialoge bereitzustellen, zusätzliche Regeln definiert, die den Begriff isolieren, um den es in der Frage geht. So wird die Frage „Zeig mir etwas über Wirtschaft“ durch ein *ZeigMir* Pattern erkannt. Gibt es keine Regel, die das *ZeigMir* Pattern mit dem Begriff *Wirtschaft* in Verbindung bringt, soll nun die neue Regel greifen, die den Begriff in einen, hier nicht weiter wichtigen, Konstrukt speichert. Abhängig vom Ergebnis des Pattern Matchings, kann dem User schon jetzt ein Dialog gezeigt werden oder wird der `FlirtAgent` beauftragt, einen

Dialog zu dem isolierten Begriff zu suchen. Konnte ein neuer Dialog (z.B. „Wirtschaft ist ein spannendes Thema, was willst du wissen?“) gefunden werden, wird dieser dem Benutzer gezeigt. Ist auch die Suche erfolglos geblieben, wird ein Standard Dialog, wie „Tut mir leid, aber das Wort Wirtschaft kenn ich nicht“ ausgegeben.

8.2.2 Anbieten eines Dialoges

Zum Anbieten der Dialoge brauchen keine Anpassungen an der Flirtmaschine vorgenommen werden, da sie die benötigte Funktionalität schon anbietet. Wie in Kapitel 8.1 erwähnt, braucht die `FlirtAgent` Klasse diese Funktionalität lediglich ansprechen. Trotzdem soll noch einmal das Anbieten eines Dialogen aus Sicht der Flirtmaschine kurz beschrieben werden.

Während einer Konversation wird der `FlirtAgent` zunächst nach Dialogvorschlägen zu einem Begriff gefragt. Die Flirtmaschine verwaltet viele Dialoge Kontextbezogen, ähnlich des Prinzips der DF Registrierungen. Durch Suche im eigenen Kontextbaum, kann die Flirtmaschine Auskunft geben, ob Dialoge zu einem Begriff vorhanden sind. Die hierarchische Strukturierung ermöglicht es, weitere Begriffe vorzuschlagen, zu denen Dialoge mit ähnlichen Informationen existieren, indem alle Kind-Knoten des Kontextes, der ursprünglich gefunden wurde, überprüft werden. Die Kontextnamen aller gefundenen Knoten, zu denen Dialoge vorhanden sind, werden als Vorschläge an der `getProposals` Methode der `FlirtAgent` Klasse geantwortet.

Im nächsten Schritt wird anhand der gelieferten Vorschläge durch die `getInformation` Methode des Flirt Agenten ein Dialog angefordert. Da diese Funktionalität auch innerhalb der Flirtmaschine genutzt wird, um anhand eines Profils dem User Vorschläge zu interessanten Themen machen zu können, ist dieser Punkt mit wenig Aufwand verbunden. Einzig der Aufruf dieser Funktion, mit dem ausgewählten Begriff, ist in der `FlirtAgent` Klasse zu implementieren.

8.3 Zusammenfassung und Betrachtung

Durch die Anpassungen an die Flirtmaschine und anhand des Flirt Agenten können nun die drei Flirtmaschinen mit den unterschiedlichem Wissensbasen gestartet werden. Dem CYB können weiterhin Fragen gestellt werden, die er bestmöglichst zu beantworten versucht. Einziger vom Benutzer kaum gemerkter Unterschied ist, dass bei Fragen, die mit „Zeig mir ...“, Was weißt Du über ...“ oder „Was ist...“ beginnen und deren beinhalteter Begriff unbekannt ist, der Flirt Agent beauftragt wird eine passende Antwort zu finden.

Bei der Realisierung kam es zu dem Problem, dass die Flirtmaschine keine Auswahl der Dialogvorschläge anderer Agenten treffen konnte. Aus diesem Grund wird zur Zeit ein Vorschlag nach dem Zufallsprinzip ausgewählt. Wünschenswert wäre es gewesen, wenn die Flirtmaschine anhand des Profils des Benutzers einen geeigneten Vorschlag hätte auswählen können. Bei genauerer Betrachtung ist dieses Problem ein allgemeines Problem des hier verwirklichten Informationsagenten, denn die Anwendung, die genutzt wird um Informationen anzubieten, muss in der Lage sein, anhand eines Begriffs Vorschläge machen zu können und anhand eines Vorschlags die angeforderte Information bereitzustellen. Die Seite des Initiators muss es ermöglichen die gesuchte Information mit einem Begriff zu beschreiben, und wie in diesem Fall nicht möglich, einen Vorschlag auszuwählen. Glücklicherweise erfüllt die Flirtmaschine diese Bedingungen.

Dass ein Antwortdialog von der anfragenden Flirtmaschine verstanden und dem Benutzer angezeigt werden kann, liegt an dem Format der Antwort, das dem der eigenen Dialoge

entspricht. Einen Agenten ausfindig zu machen, dessen Information verstanden werden kann, hängt lediglich von der Einschränkung der Suchanfrage ab, die an dem DF Agenten geschickt wird. Das Format des Informationstyps wird momentan durch den einfachen Wert eines `property` Konstruktes in der DF Beschreibung beschrieben, wie in Abbildung 7-12 zu sehen ist. Dies ist für ein einfaches Szenario, wie diesem ausreichend, sollte zukünftig doch durch genaue Angabe (in diesem Fall: XML und Name der DTD) geschehen, damit auch andere Agenten die Möglichkeit haben diesen Dienst zu nutzen, falls ihnen das Format bekannt ist.

Alles in allem konnte dieses Szenario zeigen, dass der Ansatz eines Informationsagenten, dessen Informationen von anderen Agenten weiterverarbeitet werden kann, zu verwirklichen ist. Natürlich sind an der einen oder anderen Stelle ein paar Ecken und Kanten, die zukünftig aber geglättet werden können.

9 Zusammenfassung und Ausblick

Zum Abschluss soll die Arbeit noch einmal zusammengefasst und gegenüber gestellt werden, welches die Ziele waren und was davon erreicht werden konnte. Anschließend soll die Tragweite der gefunden Lösung aufgezeigt und ein Ausblick über eine mögliche Weiterentwicklung und eine kurze persönliche Bewertung gegeben werden.

9.1 Rückblick und Einschränkungen

In dieser Arbeit wurde die Problematik von Informationsagenten betrachtet und ein eigener Ansatz verwirklicht, der die Einschränkungen heutiger Informationsagenten nicht mehr aufweist. Die hier entworfenen Informationsagenten kooperieren miteinander und bauen einen intelligenten Informationsdienst auf. Zum Auffinden von Informationen wurden die Kommunikationsfähigkeiten von Agenten genutzt, indem Nachrichten untereinander ausgetauscht werden. Die Eigenständigkeit der Agenten ermöglicht es ohne Hilfe von außen Informationsanbieter ausfindig zu machen und mit ihnen Verhandlungen zu führen.

Ein solches Multiagentensystem benötigt einige Infrastrukturen und Regulierungen. Damit der hier entworfene Informationsagent zukünftig auch mit anderen Agenten kooperieren kann, wird der FIPA Standard verwendet. Zur Verwirklichung dieses Agenten waren folgende Funktionalitäten hilfreich. Der Standard definiert das Format der Nachrichten und stellt, durch die Agentenplattform, eine Grundlage zum Austausch von Nachrichten bereit. Zusätzlich wird Unterstützung beim Führen von Verhandlungen, durch Definition von Protokollen, geboten das Auffinden möglicher Kooperationspartner ermöglicht. Stärke und Schwäche des FIPA Standards ist seine Gesamtheit. Im Vergleich mit KQML und KIF, die lediglich Nachrichten beschreiben ist dies ein Verteil, da auch eine Plattform spezifiziert wird. Anderseits ist er sehr komplex und schwer zu entscheiden, welche Komponenten für eine Aufgabe benötigt werden. Technisch macht er, bis auf einige Kleinigkeiten, einen guten Gesamteindruck.

Zur Verwirklichung wurde einerseits eine Agentenplattform entwickelt, die äußerlich, für Agenten auf anderen Plattformen, FIPA 2000 konform ist. Auf die Implementierung des vollständigen Standards wurde aufgrund dessen Komplexität und nicht benötigter Funktionalitäten verzichtet. Anderseits wurde ein generischer Informationsagent entwickelt, der sich der Dienste dieser Plattform bedient. Er ist fähig Informationen anzubieten und nach Informationen zu suchen. Dieser Agent ist zunächst unabhängig von der anzubietenden und zu suchenden Information, indem gesuchte und vorgeschlagene Informationen lediglich durch einfache Begriffe beschrieben werden. Das Format der auszutauschenden Information braucht dem Agenten nicht bekannt sein. Durch diese einfache Beschreibungsmöglichkeit können keine komplexen Suchanfragen gemacht werden.

In einem einfachen Anwendungsbeispiel wurde ein Informationsagent verwirklicht, der XML Ausdrücke als Informationen anbietet. Ein Netz bestehend aus drei solcher Agenten, das die einzelnen Wissensbasen dieser Agenten zu einer gemeinsamen erweitert, konnte zeigen, dass der in dieser Arbeit vorgestellte Ansatz prinzipiell funktionsfähig ist.

Die Informationen, die in diesen beispielhaften Informationsdienst ausgetauscht werden, können aufgrund des Formats lediglich von bestimmten Anwendungen und Benutzern verstanden werden. Sollen zukünftig unterschiedliche Anwendungen auf Basis dieses Informationsagenten gemeinsam einen Dienst anbieten und nutzen, wird eine gemeinsames Format der Information benötigt. Zusätzlich müssten sich sie miteinander kooperieren Informationsagenten auch auf eine gemeinsame Ontologie einigen, die

Anfragen und Vorschläge beschreibt. Da Standardisierungen hierzu nicht bekannt sind, könnte die in dieser Arbeit entworfene Ontologie genutzt werden.

Genügen einfache Begriffe als Suchanfrage und zur Beschreibung von Vorschlägen, ist diese Ontologie ausreichend. Sollen komplexere Beschreibungen abgebildet werden, ist zu überlegen inwiefern eine Ontologie zur allgemeinen Beschreibung von Informationen noch zu verwirklichen ist. Spätestens wenn informations-spezifische Details in die Anfrage mit eingehen sollen, wie bei einer Anfrage nach dem TV Programm vom heutigen Abend, wird pro Informationsdienst eine spezielle Ontologie benötigt. Diese kann dann auch die auszutauschende Information mitbeschreiben, wie es in der Agent Management Ontologie auch geschieht.

Ein weiteres problematisches Szenario ist folgendes: Solange Informationen lediglich mit einem Begriff beschrieben werden können, kann es zu Mehrdeutigkeiten kommen. Informationen zum Begriff *Turm* könnten eine Schachfigur oder ein Gebäude beschreiben. Zur Vermeidung dieser Probleme sollte der Informationsagent den angebotenen Dienst unter einem Kontext registrieren und veröffentlichen. Technisch ist dies möglich, wie in Kapitel 7.2.4 gezeigt wurde, doch muss die Struktur des Kontextbaums allen Anbietern und Nutzern des Dienstes bekannt sein, damit sie sich unter den richtigen Kontext registrieren und mit den richtigen Informationsanbietern kooperieren können. In einer Laborumgebung oder einem abgegrenzten System mag dies einfach möglich sein, doch in einer offenen Umgebung, die aus fremden Agenten besteht, muss ein solcher Kontextbaum standardisiert sein. Ob ein Baum gefunden werden kann, der alle Kontexte unserer komplexen Welt präzise genug abbilden kann, ist zu bezweifeln. Wahrscheinlich würden sogar viele verschiedene Bäume benötigt werden. Inwiefern dieser Ansatz in einem offenen System zu verwirklichen ist, ist fraglich.

9.2 Erweiterungen

Wie das Anwendungsszenario gezeigt hat, konnte das Ziel erreicht werden einen kooperierenden Informationsagenten zu verwirklichen, der Experte auf einem begrenzten Themengebiet ist und detaillierte Informationen in einem speziellem Format anbietet. Kooperation ermöglicht es Informationen anzufordern, die über das eigene Wissen hinausgehen. Es wurde kein Informationsagent verwirklicht, der die von ihm angebotene Information versteht, um sehr präzise Anfragen stellen zu können. Es wurde schnell deutlich, dass zu diesem Ziel lediglich ein Dienst verwirklicht werden muss, dessen gesamte Informationen durch eine Ontologie beschrieben wird. Ein solcher Ansatz wird schon bei einer Suche des DF Agenten verwendet. Auf dieser Basis können auch Hotelinformationsdienste verwirklicht werden. Auch an diesem Ansatz ist anzumerken, dass sich auf eine Ontologie geeinigt werden muss, bevor fremde Agenten miteinander kooperieren können.

Viele obigen Einschränkungen sind leider nicht allein zu lösen, sondern hängen von Standardisierungen und einer generellen Annahme, der in dieser Arbeit umrissenen Problemstellung ab. Neben weiteren technischen Feinheiten an der Plattform, wie die Implementierung weiterer FIPA Spezifikationen, könnte ein weiterer nächster Schritt die Teilnahme an dem AgentCities Projekt sein, dass in Kapitel 5.7 vorgestellt wurde. Die Ziele dieses Projektes decken sich recht genau mit den Problemen und Einschränkungen dieser Arbeit. So nutzen die Teilnehmer dieses Projektes weltweit den FIPA Standard, um zu zeigen, dass er mächtig genug ist, über Laborlösungen hinaus eingesetzt zu werden und dessen Probleme im praktischen Einsatz zu erkennen und zu beseitigen. Auf dem Standard aufbauend sollen, ähnlich wie in dieser Arbeit, komplexe Dienste entwickelt werden, die weltweit von anderen Agenten zu nutzen sind. Die Probleme, die dabei auftreten, werden ähnlich denen der hier beschriebenen sein. Es wäre denkbar mit

Partnern des AgentCities Projekt einen gemeinsamen Hotel- oder Restaurantdienst aufzubauen.

9.3 Ausblick

Bis das einleitende Szenario aus Kapitel 1 Wirklichkeit wird und ein Agent die individuelle Reiseplanung für Menschen übernehmen kann, wird noch einige Zeit vergehen und viel Arbeit geleistet werden müssen.

Diese Arbeit hat sich lediglich mit den Grundlagen eines solchen Szenarios, der Informationsbeschaffung beschäftigt. Sie zeigt, dass dieses Problem prinzipiell, durch Erschaffung einer Vielzahl von Informationsdiensten verwirklichbar ist. Zugleich werden aber auch die dabei noch zu lösenden Probleme hervorgehoben.

Bei einer so komplexen Tätigkeit, wie die Planung einer Reise, treten neben der Informationsdarstellung und Informationsaustausch noch viele weitere Probleme auf, wie die Planung oder das Treffen von Entscheidungen, abhängig von den Vorlieben des Benutzers. Zu weiteren Problemen kommt es, wenn ein deutscher Agent mit einem spanischen Agenten kommuniziert, da sie ihre Informationen in unterschiedlichen (menschlichen) Sprachen beschreiben. Die Beschreibung eines Museums in Spanisch müsste zunächst ins Deutsche übersetzt werden, bevor eine Entscheidung getroffen werden kann, ob dieses Museum interessant ist oder nicht.

Ein wichtiger Punkt ist die Sicherheit, da für Buchungen und Käufe zum Teil sehr vertrauliche Daten ausgetauscht werden müssen, wie Namen und Adressen, Bankverbindungen oder Kreditkartennummern. Da jede Person Daten über sich unterschiedlich stark als vertraulich einstuft und mit unbekannten Agenten kommuniziert wird, ist Sicherheit ein bisher wenig beachtetes, aber äußerst wichtiges Thema.

Zur Verwirklichung des einleitenden Szenarios Agententechnologie zu verwenden, um komplexe Planungen anzustellen und Probleme zu lösen, ist meiner Meinung nach vielversprechend. Der Ansatz als Grundlage hierfür spezielle Dienste zu anzubieten, die Informationen über eine Stadt anbieten oder Hotel-, Theater- oder Kinoreservierungen ermöglichen, ist, denke ich, ein weiterer sinnvoller Schritt. Betrachtet man die heutige Forschung und Entwicklung in dem Bereich der Dienste, haben dies auch viele andere erkannt, gerade weil mit ihnen zukünftig viel Geld zu verdienen ist. Die wichtigste Voraussetzung dazu ist jedoch die Standardisierung von Ontologien zur Nutzung eines Dienstes. Wie diese Arbeit gezeigt hat, können Agententechnologie und Multiagentensysteme als Basistechnologie dieser Dienste dienen, sind dazu aber nicht zwingend erforderlich. Dies zeigt die zur Zeit sehr rasante Entwicklung der Web Services. Im Prinzip können mit Web Services [17], [28] dieselben Dienste über Plattform- und Programmiersprachengrenzen hinweg verwirklicht werden, wie mit der Agententechnologie auch, doch verbirgt sich hinter ihnen ein ganz anderes Paradigma. Durch beeindruckendes Interesse großer Firmen, wie Microsoft, IBM und Sun, scheinen zur Zeit eher Web Services als Agententechnologie als Basistechnologie dieser Dienste verwendet zu werden.

Die Zukunft wird zeigen, ob Ontologien spezieller Dienste standardisiert werden können, und welche Technologie sich als Grundlage der ersten komplexen (wahrscheinlich auf Agententechnologie basierenden) Dienste eingesetzt wird.

Anhang A Struktur des Quellcodes

An dieser Stelle soll ein Überblick über die Struktur des Quellcodes gegeben werden. Die Klassen, die in dieser Arbeit entwickelt wurden, gehören unterschiedlichen Javapaketen an. Einen Überblick über alle vorhandenen Pakete gibt folgende Liste:

```
com.agentscape.simon.fipa.acc
com.agentscape.simon.fipa.df
com.agentscape.simon.fipa.lang.acl
com.agentscape.simon.fipa.lang.sl0
com.agentscape.simon.fipa.ont
com.agentscape.simon.fipa.ont.ams
com.agentscape.simon.fipa.ont.flm
com.agentscape.simon.fipa.prot
com.agentscape.simon.fipa.util
com.agentscape.simon.fipa.util.agent
com.agentscape.simon.fipa.wrapper
```

Aus historischen Gründen sind die Paketnamen nicht immer glücklich gewählt. Auch die Zugehörigkeit einiger Klassen zu bestimmten Paketen sollte zukünftig überdacht werden.

`com.agentscape.simon.fipa.acc`
Dieses Paket enthält alle Klassen die zur Verwirklichung des ACC benötigt werden. Neben der ACC Klasse ist hier auch die GUI des ACC zu finden sowie die `PlatformIdentifier` Klasse und das `IACC` Interface.

`com.agentscape.simon.fipa.df`
Hier können alle Klassen gefunden werden, die an der Realisierung des DF Agenten beteiligt sind. Neben dem Agenten selbst und der `DFEngine` Klasse sind hier auch die Behaviourklassen zu finden, die den DF Dienst auf Basis des *FIPA Request* Protokolls verwirklichen.

`com.agentscape.simon.fipa.lang.acl`
In diesem Paket sind die Klassen zu finden, die die ACL abbilden sowie deren Parser und Composer.

`com.agentscape.simon.fipa.lang.sl0`
Die SL0 wird durch Klassen dieses Paketes in Java abgebildet. Parser- und Composerklassen sind hier ebenfalls zufinden wie auch Exceptions.

`com.agentscape.simon.fipa.ont`
Allgemeine Klassen, die zur Interpretation und zum Erzeugen der SL0 Darstellung benötigt werden sind hier zu finden.

`com.agentscape.simon.fipa.ont.ams`
Die Abbildung der Agent Management Ontologie in Java Klassen sowie deren Interpreter und Composer in/aus die SL0 Darstellung sind in diesem Paket zu finden.

com.agentscape.simon.fipa.ont.flm
Die Klassen der Ontologie, die der Informationsdienst nutzt, befinden sich in diesem Paket.

com.agentscape.simon.fipa.prot
Zu diesem Paket gehören alle Klassen, die zur Realisierung der verwendeten Protokolle benötigt werden. Neben den allgemeinen protokoll-unabhängigen Klassen, wie `Conversation` oder `InitiatorBehaviour` sind hier auch alle Klassen zu finden, die das FIPA Request und das *FIPA Iterated Contract Net* Protokoll Implementieren, wie die `ICNInitiatorConversation` oder `RequestResponderBehaviour`.

com.agentscape.simon.fipa.util
Dieses Paket enthält lediglich die `FIPATokenizer` Klasse, die zur lexikalischen Analyse der ACL und SL0 Strings genutzt wird.

com.agentscape.simon.fipa.util.agent
Neben dem Test Agenten und einiger GUI Klassen, sind hier auch die `FipaAgent` und `DFRegistrableAgent` Klassen zu finden.

com.agentscape.simon.fipa.wrapper
Dieses Paket enthält die Klassen, die zur Realisierung des Informationsagenten benötigt werden.

Anhang B ACL Grammatik

Entnommen aus [FIPA 2000a]

```
ACLCommunicativeAct     = Message.
Message                 = "(" MessageType MessageSlot* ")".
MessageType             = See [FIPA00037]
MessageSlot             = ":sender" AgentIdentifier
                        | ":receiver" AgentIdentifierSet
                        | ":content" String
                        | ":reply-with" Expression
                        | ":reply-by" DateTime
                        | ":in-reply-to" Expression
                        | ":reply-to" AgentIdentifierSet
                        | ":language" Expression
                        | ":encoding" Expression
                        | ":ontology" Expression
                        | ":protocol" Word
                        | ":conversation-id" Expression
                        | UserDefinedSlot Expression.
UserDefinedSlot         = Word.
Expression              = Word
                        | String
                        | Number
                        | DateTime
                        | "(" Expression* ")".
AgentIdentifier         = "(" "agent-identifier" ":name" word
                            [ ":addresses" URLSequence ]
                            [ ":resolvers" AgentIdentifierSequence ]
                            ( UserDefinedSlot Expression )* ")".
AgentIdentifierSequence= "(" "sequence" AgentIdentifier* ")".
AgentIdentifierSet      = "(" "set" AgentIdentifier* ")".
URLSequence             = "(" "sequence" URL* ")".
DateTime                = DateTimeToken.
URL                     = See [RFC2396]
```

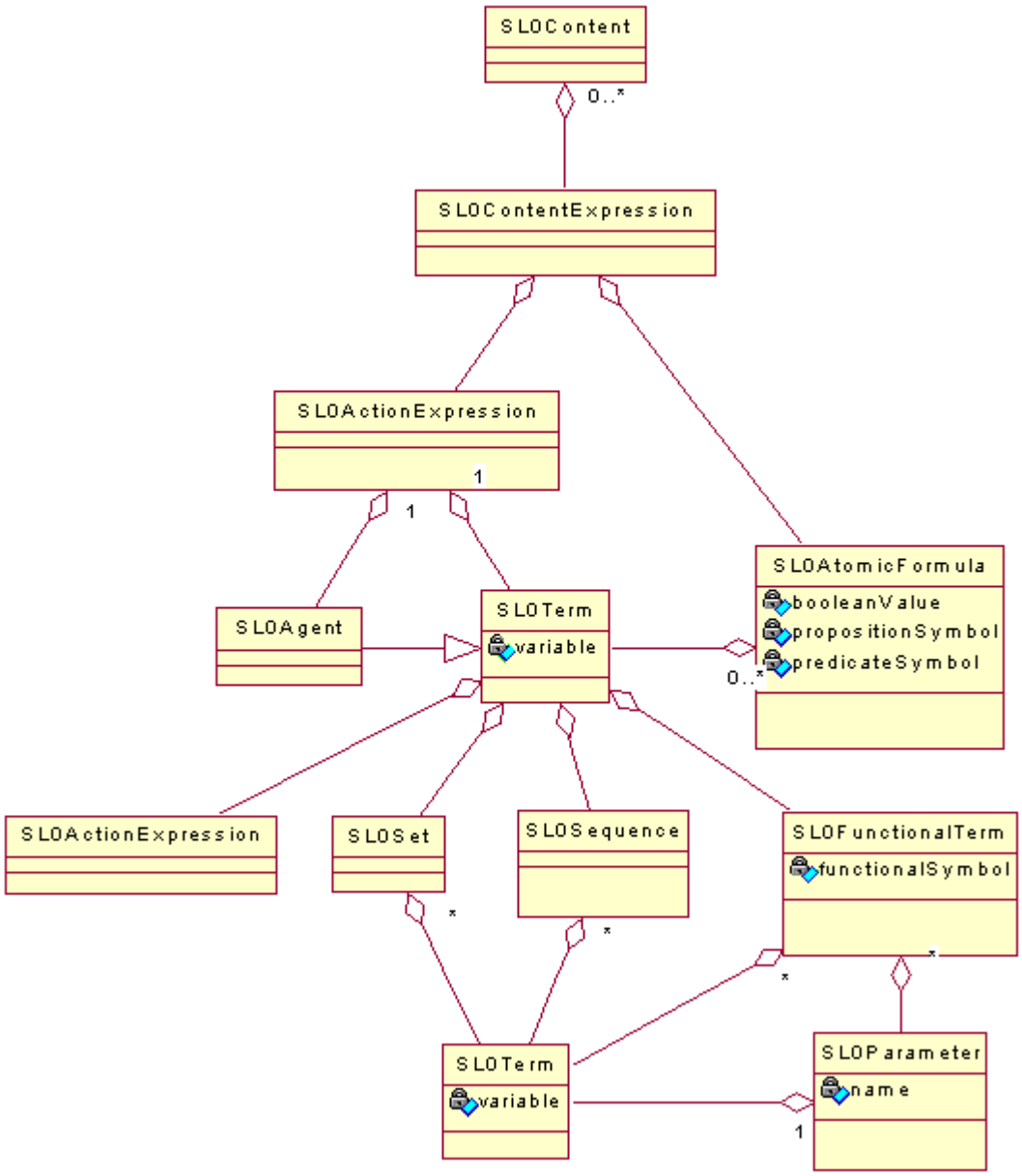
Anhang C SL Grammatik

Entnommen aus [FIPA 2000e]

```
Content             = "(" ContentExpression+ ")".
ContentExpression   = ActionExpression
                    | Proposition.
Proposition         = Wff.
Wff                 = AtomicFormula
                    | "(" ActionOp ActionExpression ")".
AtomicFormula       = PropositionSymbol
                    | "(" "result"        Term Term ")"
                    | "(" PredicateSymbol Term+ ")"
                    | "true"
                    | "false".
ActionOp            = "done".
Term                = Constant
                       | Set
                    | Sequence
                    | FunctionalTerm
                    | ActionExpression.
ActionExpression    = "(" "action" Agent Term ")".
FunctionalTerm      = "(" FunctionSymbol Term* ")"
                    | "(" FunctionSymbol Parameter* ")".
Parameter           =  ParameterName ParameterValue.
ParameterValue      = Term.
Agent               = Term.
FunctionSymbol      = String.
PropositionSymbol   = String.
PredicateSymbol     = String.
Constant            = NumericalConstant
                    | String
                    | DateTime.
Set                 = "(" "set" Term* ")".

Sequence            = "(" "sequence" Term* ")".
NumericalConstant   = Integer
                    | Float.
```

Abbildung der SL0 Grammatik in einen Java Objektbaum.



Anhang D DF Beispielregistrierung

1. Anforderung einer Registrierung, die an den DF Agenten geschickt wird.
2. Bestätigung des DF Agenten, dass die Anforderung ausgeführt werden kann.
3. Information des DF Agenten über erfolgreiche Registrierung.

```
(request
  :sender
    (agent-identifier
      :name myAgent@agentland.com
      :addresses (sequence iiop://agentland.com:9000/acc))
  :receiver(set
    (agent-identifier
      :name df@agentland.com
      :addresses (sequence iiop://agentland.com:9000/acc)))
  :content
    (action
      (agent-identifier
        :name df@agentland.com
        :addresses (sequence iiop://agentland.com:9000/acc))
      (register
        (df-agent-description
          (agent-identifier
            :name myAgent@agentland.com
            :addresses (sequence iiop://agentland.com:9000/acc))
          :ontology (set information-ontology)
          :language (set fipa-sl0)
          :services (set
            (service-description
              :name flirt-information-service
              :type information-service)))))
  :language sl0
  :ontology fipa-agent-management
  :protocol fipa-request)
```

```
(agree
  :sender
    (agent-identifier
      :name df@agentland.com
      :addresses (sequence iiop://agentland.com:9000/acc))
  :receiver(set
    (agent-identifier
      :name myAgent@agentland.com
      :addresses (sequence iiop://agentland.com:9000/acc)))
  :content
    (action
      (agent-identifier
        :name df@agentland.com
        :addresses (sequence iiop://agentland.com:9000/acc))
      (register
        (df-agent-description
          (agent-identifier
            :name myAgent@agentland.com
            :addresses (sequence iiop://agentland.com:9000/acc))
          :ontology (set information-ontology)
          :language (set fipa-sl0)
          :services (set
            (service-description
              :name flirt-information-service
              :type information-service)))))
  :language sl0
  :ontology fipa-agent-management
  :protocol fipa-request)
```

```
(inform
  :sender
    (agent-identifier
      :name df@agentland.com
      :addresses (sequence iiop://agentland.com:9000/acc))
  :receiver(set
    (agent-identifier
      :name myAgent@agentland.com
      :addresses (sequence iiop://agentland.com:9000/acc)))
  :content
   (done
      (action
        (agent-identifier
          :name df@agentland.com
          :addresses (sequence iiop://agentland.com:9000/acc))
        (register
          (df-agent-description
            (agent-identifier
              :name myAgent@agentland.com
              :addresses (sequence iiop://agentland.com:9000/acc))
            :ontology (set information-ontology)
            :language (set fipa-sl0)
            :services (set
              (service-description
                :name flirt-information-service
                :type information-service)))))))
  :language sl0
  :ontology fipa-agent-management
  :protocol fipa-request)
```

Abkürzungsverzeichnis

ACC	**A**gent **C**ommunication **C**hannel
ACL	**A**gent **C**ommunication **L**anguage
AMS	**A**gent **M**anagement **S**ystem
AP	**A**gent **P**latform oder **A**genten**P**lattform
API	**A**pplication **P**rogram **I**nterface
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture
CA	**C**ommunicative **A**ct
DF	**D**irectory **F**acilitator
DTD	**D**ocument **T**ype **D**efinition
EBNF	**E**rweiterte **B**ackus-**N**aur **F**orm
FIPA	**F**oundation for **I**ntelligent **P**hysical **A**gents
GUID	**G**lobal **U**nique **I**dentifier
HAP	**H**ome **A**gent **P**latform
HTTP	**H**ypertext **T**ransfer **P**rotocol
IIOP	**I**nternet **I**nter **O**RB **P**rotocol
IPMT	**I**nternal **P**latform **M**essage **T**ransport
JADE	**J**ava **A**gent **DE**velopment framework
JINI	**J**ava **I**ntelligent **N**etwork **I**nfrastructure
JMS	**J**ava **M**essaging **S**ervice
JNDI	**J**ava **N**aming and **D**irectory **I**nterface
KIF	**K**nowledge **I**nterchange **F**ormat
KQML	**K**nowledge **Q**uery **M**odelling **L**anguage
OMG	**O**bject **M**anagement **G**roup
ORB	**O**bject **R**equest **B**roker
RMI	**J**ava **R**emote **M**ethod **I**nvocation
RPC	**R**emote **P**rocedure **C**all
SL	**S**emantic **L**anguage
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
UDDI	**U**niversal **D**escription, **D**iscovery and **I**ntegration
UML	**U**nified **M**odelling **L**anguage
VM	**V**irtual **M**achine
XML	e**X**tensiblie **M**arkup **L**anguage

Glossar

Agent

In dieser Arbeit : siehe Kapitel 2.3

Agent Communication Channel (ACC)

Komponente einer FIPA Agentenplattform. Zuständig für die Auslieferung einer ACL Nachricht an ihren Empfänger.

Agent Communication Language (ACL)

Kommunikationssprache der FIPA Agenten.

Agent Management System (AMS)

Komponente einer FIPA Plattform. Sie ist für die Verwaltung der Agenten und die Benutzung des ACC zuständig.

Agentenplattform (AP)

Basis eines Multiagentensystems.

Application Program Interface (API)

Öffentliche Schnittstelle zu einem Programm, Framework oder Objekt.

Common Object Request Broker Architecture (CORBA)

Standardisierte Architektur mit der Objekte über Prozessgrenzen hinweg zugänglich und geteilt werden.

Communicative Act (CA)

Typ einer ACL Nachricht. Ein CA drückt den illokutionären Act einer Nachricht aus.

Content

Inhalt oder Semantik einer ACL Nachricht.

Contentsprache

Sprache mit der Content z.B. in einem String oder XML abgebildet werden kann.

Dialog

Folge von Nachrichten zwischen zwei Agenten.

Directory Facilitator (DF)

Komponente einer FIPA Plattform. Der DF ist ein Dienst indem Agenten Beschreibungen von sich und ihren Diensten registrieren und andere Agenten auffinden können.

Directory Service

Dienst in dem Datenstrukturen in hierarchischen Strukturen persistent hinterlegt werden können.

Document Type Definition (DTD)

Festlegungen über Aufbau, Struktur und gegenseitige Beziehungen von Elementen, Attributen und Entities von XML-Dokumenten

Erweiterte Backus-Naur Form (EBNF)

Darstellungsform von formalen Sprachen

Foundation for Intelligent Physical Agents (FIPA)

Organisation zur Erstellung von Softwarestandards für heterogene und kooperative Agenten und agentenbasierte Systeme.

Framework

Ein Framework ist ein wiederverwendbares, teilweise fertiggestelltes Softwaresystem.

Global Unique Identifier (GUID)

Identifiziert einen Agenten weltweit eindeutig.

Home Agent Platform (HAP)

Plattform auf der ein Agent zuerst erzeugt wurde. Sie ist verantwortlich für die Identifizierung eines Agenten.

Internet Inter ORB Protocol (IIOP)

Protokoll, das CORBA Implementierungen interoperabel macht.

Illokutionärer Akt

Beschreibt die Handlung, die mit einer Nachricht ausgedrückt werden soll.

Internal Platform Message Transport (IPMT)

Komponente einer FIPA Plattform. Sie garantiert physischen Austausch der ACL Nachrichten.

Java Serialized Objektes

Ein Ansatz von Java der Objekte in eine Darstellung umwandelt, in der sie in einer Datei gespeichert oder in einem Netzwerk ausgetauscht werden können.

Java Naming and Directory Interface (JNDI)

Java Spezifikation, die eine API für den gemeinsamen Zugriff auf unterschiedlichste Directory Services definiert.

Java Messaging Service (JMS)

Java Spezifikation, die eine API für den gemeinsamen Zugriff auf unterschiedlichste Messaging Systeme definiert.

Knowledge Interchange Format (KIF)

Contentsprache, ähnlich der SL.

Knowledge Interchange Format (KIF)

Sprache zur Abbildung von Informationen bzw. Content oder Semantik.

Knowledge Query Modelling Language (KQML)

Kommunikationssprache für Agenten, ähnlich der ACL.

Ontologie

Struktur zur Beschreibung eines Sachbereichs.

Object Request Broker (ORB)

Zentrale Instanz einer CORBA Implementierung. Sie verwaltet alle Objekte, die remote verfügbar sind.

Proxy(-objekt)

Ein Stellvertreter, der ein Original imitiert oder ersetzt.

Remote Message Invocation (RMI)

Java Funktionalität Objekte zwischen verschiedenen Prozessen auszutauschen. Basiert auf *Java serialized objects*.

Remote Procedure Call (RPC)

Kommunikationsform. Beim Aufruf einer Prozedur auf einem nicht lokalen Objekt wird der eigene Programmablauf solange unterbrochen bis die Ausführung der Prozedur beendet ist.

Satz

Instanz einer formalen Sprache. Eine ACL Nachricht ist ein Satz der ACL.

Simple Mail Transport Protocol (SMTP)

Protokoll über das Emails in Internet verschickt werden.

Semantic Language (SL)

Contentsprache definiert von der FIPA.

SL0

Kleinste Untermenge der SL.

Sprache (formal)

Eine von einer Grammatik erzeugte formale Sprache ist die Menge aller Sätze, die sich mit der Grammatik erzeugen lassen (bzw. die mittels der Grammatik analysierbar ist).

Sprechakttheorie (Speech Act Theorie)

Theorie, die die wichtigsten Merkmale von sprachlichen Handlungen beschreibt.

Token

Atomares Teil eines Strings oder eines Satzes.

Tokenizer

Zerlegt einen String oder einen Satz in einzelne Token.

Universal Description, Discovery and Integration (UDDI)

UDDI ist eine Spezifikation für verteilte web-basierte *Dienste-Kataloge*, die als Web Services zur Verfügung gestellt werden. Man kann es als eine Art *Gelbe Seiten* sehen, in denen Dienste registriert und aufgefunden werden können.

Voyager

Einer CORBA Implementierung

Extensible Markup Language (XML)

XML ist eine Metasprache die es erlaubt, eigene Textbeschreibungssprachen für beliebige Datenformate zu entwickeln.

Literatur

[Aho et al. 88] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman; “Compilers – Principles, Techniques and Tools”; Addison Wesley; 1988

[Allen 94] James F. Allen; “Natural Language Understanding” 2nd edition; Addison-Wesley Pub Co.; Janaur 1994

[Apostolico et al. 97] A. Apostolico, Z. Galil; “Pattern Matching Algorithms”; Oxford University Press; 1997

[Austin 75] J.L. Austin; “Zur Theorie der Sprechakte (How to Do Things with Words)”; 1975

[Bradshaw 96] Jeffrey M. Bradshaw; “Software Agents”; MIT Press; April 1997

[Brenner et al. 98] Walter Brenner, Rüdiger Zarnekow; Hartmut Wittig; „Intelligente Softwareagenten Grundlagen und Anwendungen“; Springer Verlag; 1998

[Brooks 90] R. A. Brooks; "Elephants Don't Play Chess”; In: “Robotics and Autonomous Systems” Vol. 6, 1990, pp. 3--15.

[Durfee et al. 89] E. H. Durfee, T. A. Montogomery, “MICE: A Flexible Testbed for Intelligent Coordination Experiments”; In: “*Proceedings of the 1989 Distributed Artificial Intelligence Worksho*p”; pp. 25-40; 1989.

[FIPA 97a] FIPA Dokument Nr.: OC00019; „FIPA 97 Specification, Part 1, Version 2.0, Agent Management“

[FIPA 99a] FIPA Dokument Nr.: OC00021; „Spec 13, Version 2.0, FIPA Developer's Guide”

[FIPA 2000a] FIPA Dokument Nr.: XC00070G; “FIPA ACL Message Representation in String Specification”

[FIPA 2000b] FIPA 2000: “FIPA Communicative Act Library Specification”

[FIPA 2000c] FIPA Dokument Nr.: XC00026F; “FIPA Request Interaction Protocol Specification”

[FIPA 2000d] FIPA Dokument Nr.: XC00030F; “FIPA Iterated Contract Net Interaction Protocol Specification”

[FIPA 2000e] FIPA Dokument Nr.:XC00008G; “FIPA SL Content Language Specification”

[FIPA 2000f] FIPA Dokument Nr.:XC00023H; „FIPA Agent Management Specification“

[Franklin 96] Stan Franklin, Art Graesser; “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents”; 1996

[Gilbert et al.95] D. Gilbert, et al.; „IBM Intelligent Agent Strategy“; IBM Cooperation; 1995

[Gamma et al. 96] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; „Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software “; Addison Wesley; 1996

[Görz et al. 00] Görz Günther, Rollinger Claus-Rainer, Schneeberger Josef; „Handbuch der Künstlichen Intelligenz“; 3. Auflage; Oldenburg Verlag; 2000

[Jennings et al. 98] N. R. Jennings, K. Sycara, M. Wooldridge; "A Roadmap of Agent Research and Development"; In: “Int Journal of Autonomous Agents and Multi-Agent Systems” 1 (1) 7-38; 1998

[Levine et al. 92] John R. Levine, Tony Mason, Doug Brown; „ Lex & Yacc “; O’Reilly; 1992

[Maes 95] Pattie Maes; "Artificial Life Meets Entertainment: Life like Autonomous Agents"; In: *Communications of the ACM*, Vol. 38, No. 11, 108-114 ; 1995

[McCabe 95] Clark McCabe; “April - Agent Process Interaction Language”; In: “Intelligent Agents, Lecture Notes in Artificial Intelligence”; M. J. Wooldridge, and N. R. Jennings; pp.324 - 340; Springer-Verlag, 1995.

[Nwana 96] H. S. Nwana; Software Agents: An Overview; in Knowledge Engineering Review, Vol. 11, No 3, pp.205-244, Sept. 1996; Cambridge University Press, 1996

[Oaks et al. 00] Scott Oaks, Henry Wong; “Jini in a Nutshell”; O'Reilly UK; März 2000

[Flanagan 99] David Flanagan; „Java in a Nutshell : A Desktop Quick Reference“; 3.Auflage; O'Reilly & Associates Inc.; November 1999

[Ruh et al. 99] William Ruh, Thomas Herron, Paul Klinker; „IIOP Complete“; Addison Wesley; Dezember 1999

[Searle 76] J.R. Searle; „A classification of illocutionary acts”; In: “Language in Society”, 5, 1-23; 1976

[Jung 00] Matthias Jung; Studienarbeit: „Agentenbasiertes Einsammeln verschiedenfarbiger Objekte“; Fachhochschule Hamburg; 2000

[Tannenbaum 96] Tannenbaum, A. S.; „Computernetzwerke“; 3. Auflage; Prentice Hall; 1998

[Wooldridge 99] M. Wooldridge; „Intelligent Agents“; In: G. Weiss; „Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence“; MIT Press; 1999

Links

Diese Verweise wurden zuletzt im Oktober 2001 auf Gültigkeit geprüft.

[1] http://www.mimitchi.com/html/furbn24.htm
[2] http://www.stanford.edu/~vivlee/aibo/
[3] http://mpfwww.jpl.nasa.gov/MPF/sitemap/rover.html
[4] http://www.informatik.uni-bonn.de/~rhino/
[5] http://www.robocup.org
[6] http://www.cs.umbc.edu/kqml/
[7] http://logic.stanford.edu/kif/kif.html
[8] http://www.fipa.org
[9] http://www.fipa.org/repository/bysubject.html
[10] http://www.emorphia.com/
[11] http://sharon.cselt.it/projects/jade/
[12] http://www.nar.fujitsulabs.com/april/index.html
[13] http://sourceforge.net/project/?group_id=3173
[14] http://www.agentcities.org
[15] http://www.jini.org
[16] http://www.uddi.org
[17] http://www.w3.org/TR/wsdl
[18] http://www-ksl.stanford.edu/kst/what-is-an-ontology.html
[19] http://www.altavista.com
[20] http://www.yahoo.com
[21] http://www.webgain.com/products/java_cc/
[22] http://www-4.ibm.com/software/ts/mqseries/messaging/
[23] http://www.sonicsoftware.com/
[24] http://www.corba.org/
[25] http://www.openldap.org/
[26] http://www.flirtmaschine.de
[27] http://www.agentscape.de
[28] http://shinka.de/library/library.phtml?file=web_services.inc

Weitere Informationen zu Agenten können unter:

http://www.agentlink.org

http://agents.umbc.edu/