Diplomarbeit

Multiagentensysteme:

Anbindung der petrinetzbasierten Plattform CAPA an das internationale Netzwerk Agentcities

Christine Reese

Betreuung
Dr. Daniel Moldt,
Prof. Dr. Winfried Lamersdorf

November 2003

Universität Hamburg, Fachbereich Informatik

Danksagung

An dieser Stelle möchte ich mich für die vielfältige Unterstützung bei der Durchführung der Diplomarbeit bedanken: den Korrekturlesern und -leserinnen Lawrence Cabac, Michael Duvigneau, Jan Ortmann, Daniel Moldt, Klaus Reese, Heiko Rölke, Jutta Schenck, Christina Theilmann und Svenja Willkomm. Mein Dank für eine Zeit voller Austausch an die Mit-Diplomanden Christina Theilmann, Lawrence Cabac und Jörn Schumacher. Vielen Dank an Daniel Moldt für Diskussion und Anregung und für seine inspirierende Begeisterung für Agenten, Agentcities und Siedler. Die Implementierungsarbeiten fanden z.T. in Zusammenarbeit mit Michael Duvigneau und Jörn Schumacher statt. Bei der Arbeit an den Rechnern des Fachbereichs Informatik an der Universität Hamburg war es segensreich, Reinhard Zierke als Administrator zu haben. Agentcities unterstützte die Reise zum iD3 in Barcelona, Spanien durch einen *Travel Grant*.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Diese Arbeit ist ein Beitrag zur Entwicklung von Multiagentensystemen. Die Agententechnologie bietet eine Möglichkeit, den wachsenden Anforderungen an Softwaresysteme und Software-Entwicklung gerecht zu werden, indem kleine aktive Einheiten, die Agenten, modelliert werden. Die agentenorientierte Software-Entwicklung kann als Erweiterung der objektorientierten Software-Entwicklung aufgefasst werden, wobei der Fokus bei Agenten zusätzlich zur reinen Kapselung von Daten und Funktionen auf der Autonomie und Aktivität der Komponenten liegt. Auch zur Entwicklung einer Vorgehensweise und als Abstraktionshilfe ist das Bild einer autonomen und aktiven Einheit in der Software-Entwicklung hilfreich. Agenten können zu Netzen zusammengeschlossen werden, um Angebote von verschiedenen Entwicklern und aus verschiedenen Anwendungsfeldern zu verbreiten.

Im Rahmen dieser Arbeit soll die Anbindung einer Umgebung für Agenten an ein internationales Agentennetz durchgeführt und dokumentiert werden.

Die Arbeit basiert auf vielfältigen Themengebieten. Abbildung 1.1 stellt die Themengebiete mit einigen der vielfältigen Verknüpfungen zusammenfassend dar. Alle Arbeiten fanden innerhalb einer Gruppe mit dem Arbeitsschwerpunkt Agentenorientierung statt, welche Teil des Arbeitsbereichs TGI (Theoretische Grundlagen der Informatik) an der Universität Hamburg ist.

Die Agentenplattform CAPA ist zentral für die Implementierungsarbeiten im Rahmen dieser Diplomarbeit. CAPA (*Concurrent Agent Platform Architecture*) wurde

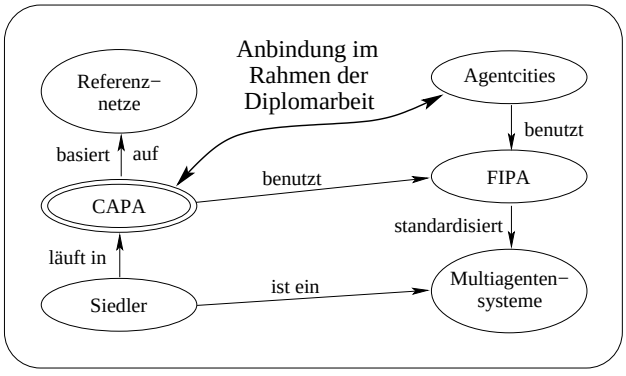


Abbildung 1.1: Die Themen der Diplomarbeit, repräsentiert mit je einem Stichwort

bei TGI aus dem allgemeinen Konzept MULAN (Multi-Agenten-Netze) mit Referenznetzen entwickelt. Mit Agenten in CAPA können Multiagentensysteme entwickelt werden. In einem Multiagentensystem wird eine komplexe Aufgabe von möglicherweise autonomen und mobilen Agenten gemeinsam bearbeitet. Ein Beispiel für ein Multiagentensystem ist Siedler, ein Spiel, das bei TGI als größere agentenorientierte Anwendung entwickelt wurde.

Das Standardisierungsgremium FIPA definiert Standards in Form von Spezifikationen für kommunizierende Software-Agenten. CAPA richtet sich nach diesen Spezifikationen, um Teil eines offenen Agentennetzes sein zu können. Agentcities ist eine europäische Initiative zur Entwicklung und Verbreitung der Agententechnologie und unterhält ein offenes Netz für Agenten im Internet (der Begriff Agentcities wird auch für das Agentennetz verwendet). In diesem Netz werden Plattformen, die „Agentenstädte“, verbunden.

Der Forschungsbereich zur Agententechnologie ist relevant, wie die Förderung von Agentcities in zwei EU-Projekten belegt; daneben existieren noch weitere EU-Projekte zum Thema Agenten (z.B. AgentLink). In Agentcities arbeiten an die hundert Forschungsgruppen zusammen. Auch in der erwähnten Arbeitsgruppe innerhalb von TGI an der Universität Hamburg wird seit mehreren Jahren an dem Thema gearbeitet und geforscht.

Der Forschungsbereich ist jung, das Standardisierungsgremium FIPA existiert seit 1996, Ende 2002 wurde der erste Satz von Spezifikationen zu ausgereiften Standards erklärt.

Die Anbindung von CAPA an ein Agentennetz ist nützlich für die Arbeitsgruppe, in deren Rahmen diese Arbeit durchgeführt wird. Die Entwicklungen dort zielen auf heterogene und offene Agentensysteme, bisher wurden jedoch keine Implementierungen für eine internationale Vernetzung durchgeführt. Die Standardisierung und Anbindung dieser Arbeiten an ein offenes internationales Netzwerk ist ein Meilenstein in der Entwicklungsarbeit dieser Gruppe und ermöglicht weitergehende Forschung sowie die Zusammenarbeit mit anderen Forschungsgruppen.

Die umfassende Darstellung der Themengebiete und die Begriffsbildung sind Voraussetzung für ein homogenes Verständnis innerhalb der Arbeitsgruppe. Insofern ist diese Arbeit zusammen mit anderen Veröffentlichungen und Arbeiten innerhalb der Arbeitsgruppe für diese relevant und nützlich.

Die Anbindung einer neu entwickelten, standardkonformen Agentenplattform an Agentcities ist nützlich für die FIPA und für Agentcities. Erst durch eine Vielfalt von Umsetzungen von Spezifikationen kann sich ein verbindlicher und praxistauglicher Standard bilden. Agentcities lebt als offenes und heterogenes Netzwerk davon, dass möglichst viele verschiedene Plattformen miteinander verbunden werden. Jede Plattform, die die technische Vielfalt vergrößert, ist ein Gewinn für Agentcities und für die Agentenforschungsgemeinde.

Ziel der Arbeit

Im Rahmen dieser Diplomarbeit soll die Anbindung der Agentenplattform CAPA an das von Agentcities betriebene internationale Netz von Agentenplattformen durchgeführt und dokumentiert werden.

Neben der technischen Umsetzung dieses Ziels dient diese Arbeit außerdem der Einarbeitung und der Darstellung der Themengebiete zum Nutzen zukünftiger Projektteilnehmer, da bisher nur wenig Dokumentation vorhanden ist. Auch die klare Darstellung von Begriffen durch Abgrenzung ist Teil dieser Diplomarbeit.

Die Aufgabe lässt sich wie folgt angehen: Zuerst muss geprüft werden, ob CAPA und Agentcities „gut genug“ zueinander passen; grundsätzlich ist das der Fall, da sich beide auf die Spezifikationen der FIPA beziehen. Gut genug bedeutet hier, dass die evtl. vorhandenen Differenzen durch Anpassung und Erweiterung in CAPA ausgeglichen werden können. So entsteht eine Liste von offenen Anforderungen an CAPA. Für jede Anforderung muss geprüft werden, mit welcher Methode sie erfüllt werden kann. Für CAPA kommen Anpassung, Parametrisierung, Erweiterung durch Konstruktion und Erweiterung durch Nutzung bestehender Software in Frage. Nach der Anforderungsermittlung und der Formulierung eines Lösungsansatzes müssen in einer Konstruktionsphase die notwendigen Schritte durchgeführt und dokumentiert werden.

Konkrete Randbedingungen

Neben dieser konzeptuellen Vorgehensweise waren bereits zum Beginn der Arbeit einige konkrete Randbedingungen bekannt: die technische Zielsetzung, die Ausgangssituation, die Aussicht auf eine konkrete Anwendung und die anvisierte externe Präsentation.

Technische Zielsetzung: Agentcities prüft die zugehörigen Agentenplattformen regelmäßig mit einer Agentenanfrage, die beantwortet werden muss. Dazu wird ein Ping-Agent benötigt, der auf standardkonforme Weise kommunizieren kann, genauer: der mit Hilfe von HTTP in der von der FIPA spezifizierten Agentenkommunikationssprache ACL („*Agent Communication Language*“) in XML-Darstellung kommunizieren kann. Eine solche Kommunikation ist das technische Ziel dieser Arbeit.

Zur Ausgangssituation: Die Implementierung der FIPA-Spezifikationen für CAPA wurde im Rahmen der Diplomarbeit von Michael Duvigneau bis Dezember 2002 weitgehend durchgeführt. Duvigneaus Arbeit bildet die Basis für die vorliegende Arbeit, deren Beginn im Oktober parallel zum Abschluss von Duvigneaus Arbeit lag.

Konkrete Anwendung und Fernziel: Im ersten Halbjahr dieser Arbeit wurde in einem studentischen Projekt eine agentenorientierte Anwendung mit CAPA realisiert: *Siedler*. Die Vorlage ist das Brettspiel „Die Siedler von Catan“. Das Fernziel, zu dem diese Arbeit nach dem Entwurf und der Implementierung des Spiels einen Schritt

beiträgt, ist die inhaltliche Anbindung von Siedler an Agentcities. Damit wird nicht nur die technische, sondern auch die inhaltliche Vielfalt von Agentcities vergrößert.

Externe Präsentation: Es gehörte zu den Aufgaben im Rahmen dieser Arbeit, CAPA und Siedler als potentielle Teile von Agentcities auf einer Ausstellung im Rahmen eines Wettbewerbs von Agentcities vorzustellen.

Aufbau der Arbeit

Kapitel 2, „*Referenznetze und* RENEW“, führt wichtige Begriffe aus der Programmierung mit Referenznetzen ein, weil CAPA zum großen Teil mit Referenznetzen implementiert ist. Referenznetze sind eine ausdrucksstarke Variante der Petrinetze mit einer engen Verbindung zur Programmiersprache Java[1].

Der große thematische Rahmen für diese Arbeit, die Technologie für Multiagentensysteme, ist noch kein weit verbreitetes Wissen. Kapitel 3, „*Agenten*“, führt die notwendigen Fachbegriffe ein, von denen in dieser Einleitung bereits einige intuitiv verwendet wurden. Die Begriffe *Agentennetz* und *Agentenplattform* spielen für diese Diplomarbeit eine zentrale Rolle. Nach der Einführung der Agententechnologie aus zwei Perspektiven wird das Agentenmodell MULAN vorgestellt, welches die Grundlage für CAPA bildet. Zuletzt beschreibt dieses Kapitel die FIPA-Spezifikationen mit dem Schwerpunkt Nachrichtentransport. Die technischen Begriffe, die für das weitere Verständnis der Arbeit wesentlich sind, werden in den drei hinteren Abschnitten dieses Kapitels eingeführt.

Kapitel 4 ist *Agentcities* gewidmet. Es beschreibt zunächst die Organisationsstruktur und Zielsetzung von Agentcities, dann wird der von Agentcities ausgeschriebene Wettbewerb vorgestellt. Schließlich wird das von Agentcities bereitgestellte Netz für Agentenplattformen beschrieben, an das CAPA im Rahmen dieser Arbeit angebunden wird. Aus dieser Beschreibung werden die Anforderungen zur Anbindung einer Plattform an Agentcities deutlich. Den Abschluss des Kapitels bildet eine Einordnung von Agentcities, u.a. bezüglich MULAN. Aus diesem Kapitel ist für das Verständnis der weiteren Arbeit vor allem der dritte Abschnitt wichtig.

Im fünften Kapitel werden nacheinander die Bestandteile von CAPA in drei Punkten erläutert: Agenten, Sprachbestandteile und Nachrichtentransport. Der vierte Abschnitt fasst Details zur Initialisierung und zur Inspektion zusammen, stellt eine Entwurfsmethode für Agenten in CAPA knapp vor und setzt CAPA in Bezug zur FIPA und zu Agentcities. Bis auf wenige kleinere Abschnitte sind alle Teile dieses Kapitels für das Verständnis der weiteren Arbeit notwendig.

Kapitel 6, „*Anbindung von* CAPA *an Agentcities*“ beschreibt die Konstruktionsphase. Hier sind die aus den Kapiteln 4 und 5 deutlich gewordenen, bisher nicht erfüllten Anforderungen an CAPA zusammengefasst und zusammen mit Lösungsansätzen dargestellt. Darauf folgt die Dokumentation der Anbindung von CAPA an

[1] Der Formalismus der Referenznetze samt dem Werkzeug RENEW zum Erstellen und Ausführen der Netze wurden in der Dissertation von Olaf Kummer (siehe [Kum02]) entwickelt.

Agentcities in zwei Phasen: Zuerst wird CAPA im zweiten und dritten Abschnitt dieses Kapitels um die beiden wesentlichen fehlenden Teile erweitert, danach werden die Details für die Realisierung der Anbindung anhand von Tests im vierten Abschnitt erarbeitet: Auf die Beschreibung der Testszenarien folgt die Beschreibung der dabei gefundenen Probleme und ihrer Lösung. Im letzten Abschnitt wird die erweiterte Plattform dargestellt und es werden einige Anregungen für die weitere Entwicklung von CAPA vorgestellt.

Kapitel 7, „*Eine Anwendung: Siedler*“, beschreibt die agentenbasierte Anwendung Siedler, die in einem studentischen Projekt entwickelt und im Rahmen dieser Diplomarbeit auf dem Wettbewerb von Agentcities präsentiert wurde. Das Spiel an sich wird vorgestellt, danach das Projekt und das resultierende Agentensystem. Abschließend wird Siedler in Bezug auf den Wettbewerb und die weitere Anbindung an Agentcities gesetzt.

Nach der Zusammenfassung und dem Ausblick in Kapitel 8 ist im Anhang unter anderem ein Glossar mit den wichtigsten Begriffen zur Verfügung gestellt.

2 Referenznetze und Renew

Zur Programmierarbeit mit Referenznetzen – zur Bearbeitung und Ausführung – steht die Entwicklungsumgebung Renew zur Verfügung. Renew bildet damit die grundlegende Laufzeitumgebung für diese Diplomarbeit. Da der verwendete Referenznetz-Formalismus nicht als allgemein bekannt vorausgesetzt werden kann, wird hier eine Einführung gegeben.

Vorgreifend soll hier auf die weitere Verwendung des Begriffes „Netz“ eingegangen werden, denn er wird in dieser Arbeit auf zwei verschiedene Weisen verwendet: Entweder ist damit ein Referenznetz gemeint (z.B. in dem Begriff „Protokollnetz“), oder es ist damit eine Vernetzung im Internet gemeint (ein Netz von Forschern, von Agentenplattformen oder von Agenten). Insbesondere ist der Begriff „Agentennetz“ zweideutig: Er kann sowohl ein Referenznetz zur Implementierung eines Agenten in Capa meinen als auch eine Vernetzung von Agenten im Internet. Beides wird erst in den folgenden Kapiteln benötigt, in diesem Kapitel geht es ausschließlich um Referenznetze.

Es folgt zunächst ein Überblick zum Thema Petrinetze und Referenznetze, danach werden die konkreten Bestandteile und Eigenschaften von Referenznetzen erläutert und im dritten Abschnitt wird Renew mit einigen Aspekten seiner Benutzung vorgestellt. Am Schluss werden einige für diese Arbeit relevante Aspekte der Verwendung von Referenznetzen erläutert.

2.1 Überblick

Referenznetze sind Petrinetze höherer Ordnung. Sie bestehen aus Netzelementen, einer Markierung sowie formalen und informalen Anschriften. Es gibt verschiedene Arten von Marken, insbesondere können Marken auch Netzreferenzen enthalten, daher der Name Referenznetze.

Zuerst wird die allgemeine Petrinetznotation (benannt nach Carl Adam Petri) eingeführt, dann folgt ein Überblick zu Renew und Referenznetzen.

Die folgende Darstellung richtet sich nach einer Einführung von David Poutakidis[1]. Ein Petrinetz besteht aus *Stellen* (als Kreise dargestellt) und *Transitionen* (als

[1]Der Text dieser beiden Absätze wurde übersetzt und teilweise ergänzt. Vgl. [PPW]

Rechtecke dargestellt); Stellen und Transitionen sind durch *Kanten* (Pfeile) verbunden. Zusätzlich können Stellen *Marken* enthalten (durch einen Punkt notiert). Die Bestückung des Netzes mit Marken ist seine *Markierung*, und die Ausführung eines Petrinetzes besteht in der Änderung der Markierung gemäß der Schaltregel; die Stellen, Transitionen und Kanten zwischen ihnen bleiben unverändert.

Eine Transition in einem Petrinetz ist *aktiviert*, wenn jede Eingangsstelle (also eine Stelle mit einem Pfeil zu der Transition) mindestens eine Marke enthält. Eine aktivierte Transition kann *schalten* oder *feuern*, indem von jeder Eingangsstelle eine Marke entfernt wird und in jede Ausgangsstelle (also jede Stelle, zu der eine Kante von der Transition aus hinzeigt) eine Marke plaziert wird. Die Transition im Bild 2.1 kann feuern, indem sie die Marken von **a** und **P** entfernt und eine Marke in **Q** legt. Danach ist sie nicht mehr aktiviert.

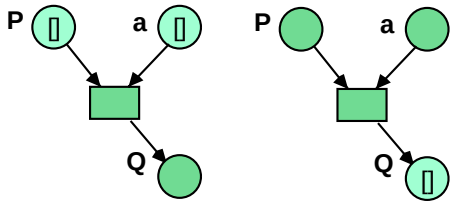


Abbildung 2.1: Transition, nach [PPW]

Diese Beschreibung trifft auf sog. S/T-Netze (Stellen-Transitionen-Netze) zu, einer einfachen Art von Petrinetzen. Eine erweiterte Art sind *gefärbte* Petrinetze, in denen Marken verschiedene Eigenschaften haben. Die Farbe einer Marke beschreibt den *Typ*, den ihr *Wert* annehmen kann. Statt Punkte in einer Stelle zu haben, können also Zahlen darin liegen oder Strings oder Werte von komplexeren Typen. In einem gefärbten Petrinetz kommen vielfältige Beschriftungsmöglichkeiten zum Aufbau des Netzes hinzu. Stellen und Kanten können getypt werden, an Transitionen können zusätzliche Bedingungen für die Aktivierung notiert werden.

Der nächste Schritt wird durch die *Referenznetze* gebildet. Referenznetze wurden 2002 von Olaf Kummer eingeführt (siehe [Kum02]). Referenznetze sind eng mit Renew verbunden, beides wird im Handbuch zu Renew ([DKW02]) ausführlich vorgestellt[2]. Im Vergleich zu den S/T-Netzen kommen neue Arten von Beschriftungen und Marken hinzu, sowie die Instanziierung von Netzen und eine starke Verbindung zu Java.

[2]Die jeweils aktuelle Version von Renew sowie Dokumentation und Artikel sind im Internet verfügbar – [ren03]

2.2 Elemente von Referenznetzen in Renew

Das Verhältnis von Netzen zu Netzinstanzen ist vergleichbar mit dem Verhältnis von Klassen zu Objekten in der objektorientierten Programmierung. Ein Netz enthält eine statische Struktur von Netzelementen, außerdem Beschriftungen und Initialisierungsausdrücke. Eine Netzinstanz enthält zusätzlich eine Markierung und ein (Schalt-) Verhalten. Eine Netzinstanz kann Instanzen von beliebigen Netzen erzeugen und die Referenzen darauf in Marken behalten. In dieser Arbeit wird in den meisten Fällen Netzinstanz ausgeschrieben, bei der Funktionsbeschreibung kann jedoch auch abkürzend der Begriff Netz verwendet werden. Insgesamt kann Netz also Referenznetz, eine Instanz eines Referenznetzes oder eine Vernetzung im Internet beschreiben.

Die Verbindung von Referenznetzen und Java ist auf zwei Ebenen möglich. Einerseits können Marken in Referenznetzen beliebige Java-Objekte sein; an Kanten- oder Transitionsanschriften können Methoden an solchen Objekten aufgerufen werden. Andererseits kann mit Hilfe von *Stub-Klassen* und *synchronen Kanälen* eine gleichberechtigte Kommunikation zwischen Java-Objekten und Netzinstanzen in beide Richtungen stattfinden.

Die im Überblick vorgestellten Netzelemente (Stellen, Transitionen, Kanten) werden bei Referenznetzen um einige Elemente erweitert. Die Angaben zur Darstellung der Elemente beziehen sich auf die Darstellung in Renew.

- Eine Stelle kann graphisch mehrfach dargestellt werden, um die Übersichtlichkeit eines Netzes zu verbessern (sog. *virtuelle Stellen*). Die Kopie ist durch einen doppelten Kreis erkennbar.
- Transitionen können in Renew für die Dauer eines Schaltvorgangs zusammengefasst werden. Dies geschieht mittels *synchroner Kanäle*.
- In Renew gibt es zusätzlich zur einfachen Kante mit einer Pfeilspitze weitere Kantenarten:
  - Eine *Reservierungskante* hat an jedem Ende eine Pfeilspitze und ist eine abkürzende Schreibweise für eine Eingangs- und eine Ausgangskante mit derselben Beschriftung.
  - *Testkanten* haben keine Pfeilspitzen. Durch Testkanten können Marken nebenläufig (gleichzeitig) auf Vorhandensein oder ihren Wert getestet werden.
  - Eine *flexible Kante* hat zwei Pfeilspitzen in einer Richtung. Mit einer flexiblen Kante kann eine Marke, die ein Java-Array enthält, in Marken für jedes Element des Arrays zerlegt werden. Andersherum können mehrere Marken zu einem Array bekannter Größe zusammengefasst werden.

2.2.1 Marken

Anonyme „schwarze“ Marken werden mit eckigen Klammern notiert. Alle anderen Marken sind *gefärbte* Marken. Eine gefärbte Marke kann ein beliebiges Java-Objekt sein und insbesondere auch eine Referenz auf eine Netzinstanz enthalten. Konzeptionell gesehen, ist dann ein Netz als Marke in einem anderen Netz enthalten. Dieser Mechanismus heißt *Netze in Netzen* und ist für den Entwurf komplexer Referenznetze sehr bedeutsam.

Mit einem *Tupel* können mehrere Werte (Java-Objekte und primitive Datentypen) zu einer Marke zusammengefasst werden. Ein Tupel wird mit eckigen Klammern umschlossen, die Elemente werden mit Kommas getrennt. Eine anonyme Marke ist in RENEW durch ein leeres Tupel notiert.

Im statischen Netz werden Marken durch Ausdrücke an Stellen dargestellt, die bei der Instanziierung zur Anfangsmarkierung ausgewertet werden. In Netzinstanzen werden Marken auf verschiedene Weise dargestellt: normalerweise nur die Anzahl der Marken in einer Stelle. Es ist immer auch möglich, den Inhalt darzustellen, entweder direkt in der Darstellung der Netzinstanz oder in einem eigenen Fenster, was für komplexe Marken praktisch ist. Der Inhalt einer Marke kann in String-Repräsentierung dargestellt werden oder als Java-Objekt in der erweiterten Tokendarstellung, dann ist es browsbar als Hierarchie von Objekten, die einander enthalten (siehe Abbildung 2.2). Dabei wird der jeweilige Klassenname als Rahmen angezeigt und in der Ebene darunter seine Instanzvariablen und parameterlosen Methoden. Auf diese Weise lassen sich komplexe Marken in einer Netzinstanz gezielt und typgenau inspizieren.

2.2.2 Synchrone Kanäle

Ein synchroner Kanal kann für die Dauer eines Schaltvorgangs zwei oder mehr Transitionen zu einer verschmelzen. Dazu werden diese Transitionen mit Kanalanschriften beschriftet. Auf der einen Seite gibt es den *Downlink*, der eine Netzreferenz angibt, einen Namen für den Kanal und eine Liste von Parametern. Auf der anderen Seite gibt es den *Uplink*, der nur den Namen und eine passende Parameterliste angibt. An einer einzelnen Transition darf höchstens *ein* Uplink stehen, weil Uplinks bei der Bindungssuche nicht verfolgt werden können; sie können nur „gefunden“ werden: Bei der Bindungssuche in einer Netzinstanz an einer Transition mit Downlink wird dieser Downlink zur referenzierten Netzinstanz verfolgt, dort wird dann die Bindungssuche fortgeführt.

Wie bereits erwähnt, kann über synchrone Kanäle auch Kommunikation und Datenaustausch mit Java-Objekten stattfinden. Solche Kanäle vermitteln zwischen einer Java-Beschreibung der Netzinstanz (eine sog. *Stub-Klasse*) und einer Kanaltransitionen.

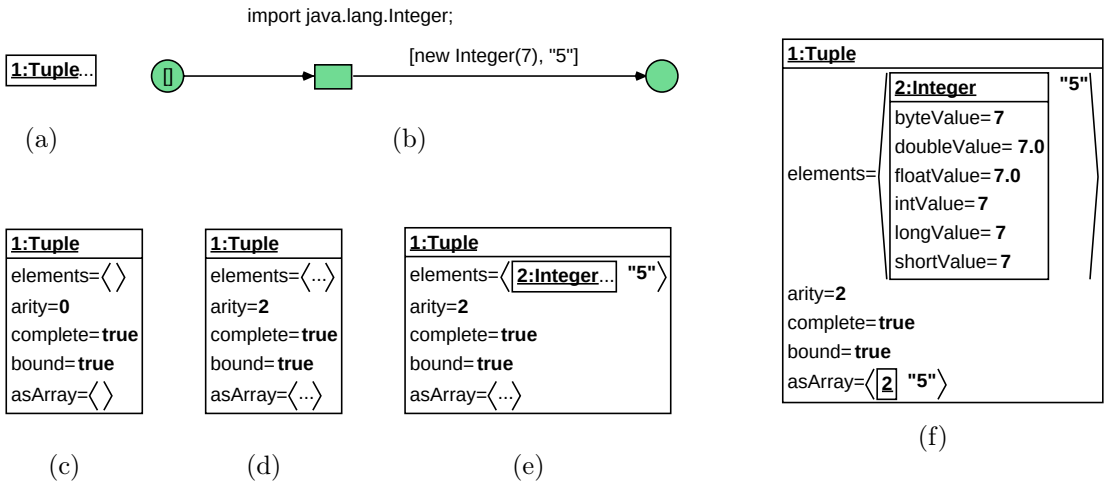


Abbildung 2.2: Erweiterte Tokendarstellung in Renew[3]. Dargestellt ist ein Netz, in dem vor und nach dem Schalten der Transition je eine Marke liegt (b). In der erweiterten Tokendarstellung erscheinen beide Marken zunächst als `Tuple` (a). Dieses kann erweitert werden: Das leere Tupel ist in (c), das gefüllte Tupel in (d) zu sehen. Teile (e) und (f) zeigen weitere Schritte bei der Erweiterung von (d).

2.2.3 Beschriftungen

Die Beschriftungssprache für Referenznetze in Renew ist Java sehr ähnlich, aber leicht angepasst an die Bedürfnisse zur Verwendung in Petrinetzen. Fast alle Befehle sind übernommen[4] bis auf das abkürzende UND, das abkürzende ODER, die bedingte Zuweisung und den Mechanismus für Ausnahmen (Exceptions). Hinzugefügt wurde das im Abschnitt 2.2.1 (*Marken*) beschriebene Konzept der Tupel, sowie die Schlüsselwörter `guard`, `action`, `manual` und `:new`, die an Transitionen vorkommen. Die Anschriften werden in den nächsten Absätzen erklärt. Einige Beispiele finden sich in Kapitel 2.4.2 (*Ein Beispielnetz*).

Jede Anschrift kann nachträglich auskommentiert werden, indem der Texttyp von `Inscription` auf `Label` geändert wird. Der Text wird dann kursiv dargestellt.

...an Stellen. Der *Name* ist eine informale Anschrift. Eine *Typbezeichnung* lässt nur Marken des angegebenen Typs auf dieser Stelle zu. *Initialisierungsausdrücke* werden zur Anfangsmarkierung einer Netzinstanz ausgewertet.

[3] Alle weiteren Bilder dieser Arbeit sind selbst erstellt, sofern nicht anders angegeben.
[4] Siehe [KW99, S. 14]

...an Kanten. Hier stehen Ausdrücke zur Bindungssuche und um Marken zu konstruieren, z.B. um Werte zu einem Tupel zusammenzufassen.

...an Transitionen. Hier sind vielfältige Beschriftungstypen möglich.
- Eine Anschrift mit dem reservierten Wort `guard` ist ein *Wächter*. Der darauf folgende Ausdruck muss zu `true` ausgewertet werden, damit die Transition aktiviert sein kann.
- Das reservierte Wort `action` leitet eine *action-expression* ein. Der Ausdruck dahinter wird beim Schalten einer Transition genau einmal ausgewertet.
- Das reservierte Wort `manual` kennzeichnet eine Transition mit einer zusätzlichen Bedingung zum Schalten. Wenn sie aktiviert ist, kann sie nicht automatisch schalten, sondern muss manuell geschaltet werden.
- Beim Schalten einer Transition kann eine neue Netzinstanz erzeugt werden. Eine *erzeugende Anschrift* besteht aus dem Schlüsselwort `:new` in Verbindung mit einem Variablennamen und einem Netznamen. Eine neue Instanz des angegebenen Netzes wird erzeugt und die Referenz darauf in der Variablen gespeichert.
- *Kanalanschriften* verbinden für einen Schaltvorgang zwei Transitionen zu einer oder ermöglichen Informationsaustausch mit Java-Objekten.
- Ausdrücke an Transitionen ohne einleitendes Schlüsselwort werden bei der Bindungssuche ausgewertet.
- Eine Transition kann einen *Namen* haben, der das Schaltverhalten nicht beeinflusst.

Der Deklarationsknoten ist keinem grafischen Netzelement zugeordnet. Ein Knoten dieser Art darf höchstens einmal pro Netz vorhanden sein. Hier werden die benötigten Java-Klassen importiert und Variablen deklariert. Ist der Knoten nicht vorhanden, ist das Netz komplett ungetypt. Sowie einer existiert, müssen *alle* in den sonstigen Anschriften verwendeten Variablen hier deklariert sein. Der Klassenpfad, in dem die angegebenen Klassen gesucht werden, wird beim Aufruf von RENEW angegeben.

2.2.4 Schaltregel

Die Schaltregel bestimmt, wann genau eine Transition aktiviert ist und was passiert, wenn sie schaltet. Der Schaltvorgang ist in jedem Falle atomar. Intuitiv gesehen, müssen genug „passende“ Marken für alle Eingangs- und Teststellen vorhanden sein. Alle Variablen in der Umgebung einer Transition müssen widerspruchsfrei gebunden werden können. Die Umgebung einer Transition besteht aus den Eingangs- und Ausgangsstellen und den verbindenden Kanten, bei synchronen Kanälen erweitert sich die Umgebung. Entsprechend der Ausgangskanten werden neue Marken in die Ausgangsstellen gelegt. Die Aktionen in den *action-expressions* der Transition werden beim Schalten ausgeführt.

2.3 Renew

Renew ist ein Editor für Referenznetze, außerdem ein Simulator und damit auch eine Testplattform für theoretische Betrachtungen von Referenznetzen und anderen Petrinetz-Varianten. Renew wird als integrierte Entwicklungsumgebung für die Anwendungsprogrammierung mit Referenznetzen verwendet und weiterentwickelt. Mit Renew wurden bereits größere Anwendungen entwickelt. Renew entstand zusammen mit dem Mechanismus der Referenznetze, so dass beide eng verwoben sind[5].

Um eine Simulation in Renew zu starten, muss zuerst Renew als Java-Programm mit dem richtigen Klassenpfad und Parametern gestartet werden. Dazu wird ein *Startskript* benötigt, in dem die nötigen Werte gesetzt werden. Die Simulation wird dann mit Bezug auf ein Netz gestartet (das *Aufrufnetz*), in dem weitere Einstellungen gesetzt werden. Dieses Aufrufnetz wird instanziiert und an den darin aktivierten Transitionen können Java-Objekte und weitere Netzinstanzen, insbesondere auch eine GUI erzeugt werden. Das Aufrufnetz kann entweder weiter zur Steuerung der Simulation verwendet oder nach „getaner Arbeit“ entfernt werden (wenn keine Transition darin mehr aktiviert werden kann, wird der Speicher vom *Garbage-Collector* in Java freigegeben).

Zur Ausführung werden die Netze von unnötiger Information und von der grafischen Darstellung getrennt. So eine „vorkompilierte“ Version kann auch einzeln gespeichert werden, das ist dann ein *Schattennetz*. Es ist möglich, eine ganze Reihe von Netzen in eine Datei „vorzukompilieren“, die dann vom Simulator ohne grafische Darstellung simuliert werden kann. So eine Datei enthält ein System von Schattennetzen, ein *Shadow Net System*[6]. Das ist sinnvoll, wenn besonders viele Netze an der Simulation beteiligt sind, die dann nicht alle in eigenen Fenstern angezeigt werden, sondern im Hintergrund geladen werden. Die Schattennetze haben den weiteren Vorteil, dass sie aus Jar-Dateien gelesen werden können. So kann eine ganze Anwendung in einer Datei enthalten sein und es wird möglich, Renew ohne grafische Oberfläche zu starten und eine Anwendung laufen zu lassen. Die Simulation kann dann über eine Eingabeaufforderung gesteuert werden.

Die aktuelle Entwicklung erlaubt es, in einer entfernten Simulation laufende Netzinstanzen lokal anzuzeigen und zu inspizieren und die Simulation zu beeinflussen (Thomas Jacob in [Jac02]). Um eine mit Schattennetzen gestartete Simulation zu inspizieren, startet man Renew mit GUI, verbindet sich mit der laufenden Simulation und kann dann alle Netzinstanzen anzeigen.

Zur Inspektion einer laufenden Simulation stehen verschiedene Möglichkeiten zur Verfügung, von denen einige bereits im vorigen Abschnitt über Referenznetze erklärt wurden. So lassen die Netzinstanzen eine eingehende und variantenreiche Inspektion der aktuellen Markierung zu. Einzelne Schritte können manuell geschaltet werden.

[5] Weitere Informationen: [DKW02] (Handbuch), [ren03] (Webseite mit der aktuellen Version)

[6] Der deutsche Begriff Schattennetz bezeichnet im Folgenden ein einzelnes vorkompiliertes Netz, der englische Begriff *Shadow Net System* ein lauffähiges System von Schattennetzen.

Weiter ist es möglich, die Simulation anzuhalten, entweder auf Knopfdruck oder an einem vorher definierten Haltepunkt, um in Ruhe zu inspizieren. Danach kann die Simulation schrittweise, manuell oder wie vorher wieder aufgenommen werden.

Zur Inspektion kann auch die programmierte Anwendung Möglichkeiten bereitstellen. So ist für das Agentenmodell MULAN ein MulanViewer als Plug-In für RENEW entwickelt worden, der die wesentlichen Vorgänge und Zustände anzeigt (Timo Carl in [Car03], der MulanViewer wird in Abschnitt 5.4.2 vorgestellt).

Grafische Darstellung in Renew

Neben den Elementen, mit denen die Funktion eines Netzes notiert wird (also Stellen, Transitionen, Kantenvarianten und Beschriftungen) können in RENEW beliebige Grafiken erzeugt werden. Diese sind Hilfen für den Leser und den Programmierer von Netzen. Neben grundlegenden Zeichen-Funktionen (Polygon, Linien, Flächen, Texte) stehen Verbindungswerkzeuge zur Verfügung: Texte können an andere Elemente angehängt und zwei Elemente mit verschiedenen Kantenstilen fest verbunden werden.

Alle Elemente (verhaltensbestimmende und verzierende) können nachträglich über Menübefehle beliebig in Farbe, Linienstil und ähnlichem angepasst werden. Stellen haben also keine feste Farbe oder Form (sie können zu Ellipsen beliebiger Größe gezogen werden), sondern sind nur deshalb Stellen, weil sie mit dem Stellen-Werkzeug erzeugt wurden.

Marken lassen sich in einer laufenden Simulation durch Bitmap-Grafiken darstellen. So ist eine besonders suggestive Darstellung einer Simulation möglich. Die Möglichkeiten der grafischen Gestaltung sind damit enorm. Sie bewegen sich zwischen den Extremen der aufwändigen grafischen (und lauffähigen) Präsentation und der rein funktionalen Spezifikation von Netzen.

2.4 Referenznetze in der Anwendung

Petrinetze finden immer weitere Verbreitung, sowohl in der Systemanalyse wie auch in der Programmierung (eine aktuelle und umfassende Darstellung geben Gireault und Valk in [GV02]). Bei der Programmierung mit Netzen besteht die gleiche Gefahr wie bei der Programmierung mit textuellen Programmiersprachen: Netze können unübersichtlich und unverständlich gestaltet sein und derselbe Sachverhalt kann auf verschiedene Weise erreicht werden. Die Bezeichnung „Spaghetti-Code“ kann bei Referenznetzen eine neue Bedeutung erhalten. Deshalb haben sich bei den Entwicklern von MULAN und CAPA Konventionen für die Darstellung gebildet: Die Voreinstellung für Stellen und Transitionen ist ein dunkles Grün. In hellem Lila sind die Stellen des Datenflusses gehalten. Dunkelgrüne Schrift dient für Kommentare oder Netzteile zur Fehlerdiagnose, also Teile, die keine Bedeutung für die Anwendung

haben. Die Netzkomponenten bilden eine eigene Art von Konvention, sie werden im nächsten Abschnitt kurz vorgestellt. Im zweiten Abschnitt wird ein Beispiel aus der Anwendungswelt vorgestellt.

2.4.1 Netzkomponenten

Lawrence Cabac stellt fest, dass die große Vielfalt der Erscheinungsformen von Petrinetzen im Allgemeinen von Vorteil sei, dabei aber die konforme Darstellung von ähnlichen Netzen innerhalb eines Einsatzgebietes wünschenswert sei. Dies sei der Fall bei der Anwendungsentwicklung mit Petrinetzen[7].

Für häufig vorkommende Muster hat Cabac deshalb das Konzept der *Netzkomponenten* entwickelt (siehe [Cab02]). Außerdem hat er beispielhaft Vorschläge für Netzkomponenten für die Anwendungsentwicklung mit Capa entwickelt, welche die häufigsten Aufgaben abdecken. Netzkomponenten erleichtern die Erstellung von Referenznetzen und sie geben eine strukturierte und durchdachte Vorlage für die grafische Anordnung von Netzelementen. Die entstehenden Netze sind ähnlich aufgebaut, das trägt erheblich zur Lesbarkeit der Referenznetze bei. Auf der diesjährigen Petrinetzkonferenz wurde ein Konzept zur Verwendung von Netzkomponenten bei der Entwicklung von Agenten in Capa vorgestellt (siehe [CMR03]).

2.4.2 Ein Beispielnetz

Zum Abschluss dieses Kapitels zu Referenznetzen soll hier noch ein ein zur Übersichtlichkeit gekürztes Netz aus der Anwendung Siedler (im Kapitel 7 vorgestellt) als Beispiel vorgestellt werden. Siedler ist mit der Agentenplattform Capa (siehe Kapitel 5) mit Hilfe von Netzkomponenten entwickelt worden. Das Verhalten der modellierten Agenten wurde dabei mit vielen einander sehr ähnlichen Netzen spezifiziert: den *Protokollnetzen*. Alle Protokollnetze haben genau einen Anfang und mindestens ein Ende, die Agenten kommunizieren untereinander mittels Nachrichten. Für diese wiederkehrenden Elemente wurden Netzkomponenten entworfen, die jeweils eine Funktion erfüllen und die leicht zu kombinieren sind. Die Netzkomponenten für Capa sind mit dunkelgrauen Transitionen und Stellen gestaltet und mit einer grauen Figur hinterlegt, die die geometrische Form betont. Diese Formen sind in den Referenznetzen besonders gut zu erkennen. Das Beispiel soll in diesem Abschnitt nicht im Bedeutungszusammenhang dargestellt werden, vielmehr sollen die grafischen Elemente benannt werden.

Im Bild 2.3 ist die Grobstruktur (die geometrische Form) dreier Netzkomponenten dargestellt. Sie heißen `IN`, `OUT` und `OUT-IN`. `OUT-IN` entspricht grafisch den übereinander geschobenen Netzkomponenten `OUT` und `IN` und bildet so eine abkürzende Schreibweise. Im zweiten Bild (2.4) sind die Komponenten in ihrer Erscheinungsform abgebildet. Über den Komponenten schwebt der *Datenfluss*, die untere Basislinie

[7] In [CMR03]

bildet den *Kontrollfluss* ab. Die Verbindung zwischen Datenfluss und Kontrollfluss geschieht über einzelne virtuelle Stellen an den Netzkomponenten.

Die erste Netzkomponente heißt `START (IN)`, in der folgende Elemente vorhanden sind: An der ersten Transition steht `:start()`. Das ist ein Uplink ohne Parameter, mit dem eine Marke in den Kontrollfluss gelegt wird. Diese Transition legt je eine anonyme Marke in die beiden nachfolgenden Stellen. Die obere Transition trägt wieder einen Uplink, dieser Kanal heißt `in` und hat einen Parameter `p`. Hier bleibt das Referenznetz stehen, bis tatsächlich ein anderes Netz mit einer Referenz auf diese Instanz eine Transition aktiviert, welche einen Downlink namens `in` mit einem Parameter hat, der dort an einen Wert gebunden werden kann. Nun können diese beiden Transitionen in einem Vorgang schalten. Die Information, die über den Parameter `p` übermittelt wurde, wird in die nachfolgende Stelle gelegt. Nun ist die zweite Transition im Kontrollfluss aktiviert, weil beide Eingangsstellen Marken enthalten. Beim Schalten wird eine Marke in die Stelle zwischen der ersten und der zweiten Netzkomponente gelegt, so dass im *Kontrollfluss* weiter gearbeitet wird.

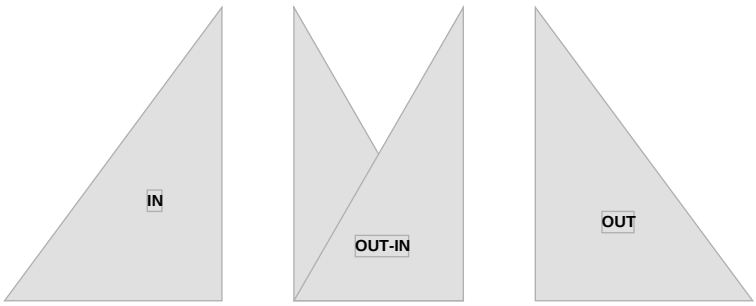


Abbildung 2.3: Formen einiger Netzkomponenten

Die erste Transition der zweiten Netzkomponente hat zwei Eingangsstellen. Die Kontrollfluss-Eingangsstelle ist belegt. Die andere Eingangsstelle ist eine virtuelle (doppelter Rand). Sie heißt `<SelfAid>`, ein *Hinweis* für den Leser, dass dies eine Kopie der Stelle gleichen Namens über der ersten Netzkomponente ist, der Name hat aber keine semantische Bedeutung (in Renew zeigt ein Doppelklick auf eine virtuelle Stelle ihre Kopiervorlage an). Die Kante zu dieser Stelle ist eine Testkante. Das bedeutet, die Stelle muss besetzt sein, kann aber während des Schaltvorgangs gleichzeitig von anderen Transitionen getestet werden. `<SelfAid>` bildet eine Verbindung zum Datenfluss. Sie enthält eine Marke, weil die Transition im Datenfluss (mit dem Kanal `wb:ask(...)`, der die Information für `<SelfAid>` bereitstellt) bereits schalten und eine Marke in die Stelle `<SelfAid>` legen konnte. An der so aktivierten Transition im Kontrollfluss steht eine *action-expression*. Hier wird eine Nachricht mit Hilfe eines Java-Methodenaufrufs erzeugt. Dazu wird der Inhalt der Stelle `<SelfAid>` benötigt. Das Ergebnis wird als Marke in die obere Ausgangsstelle gelegt. Es folgt ein

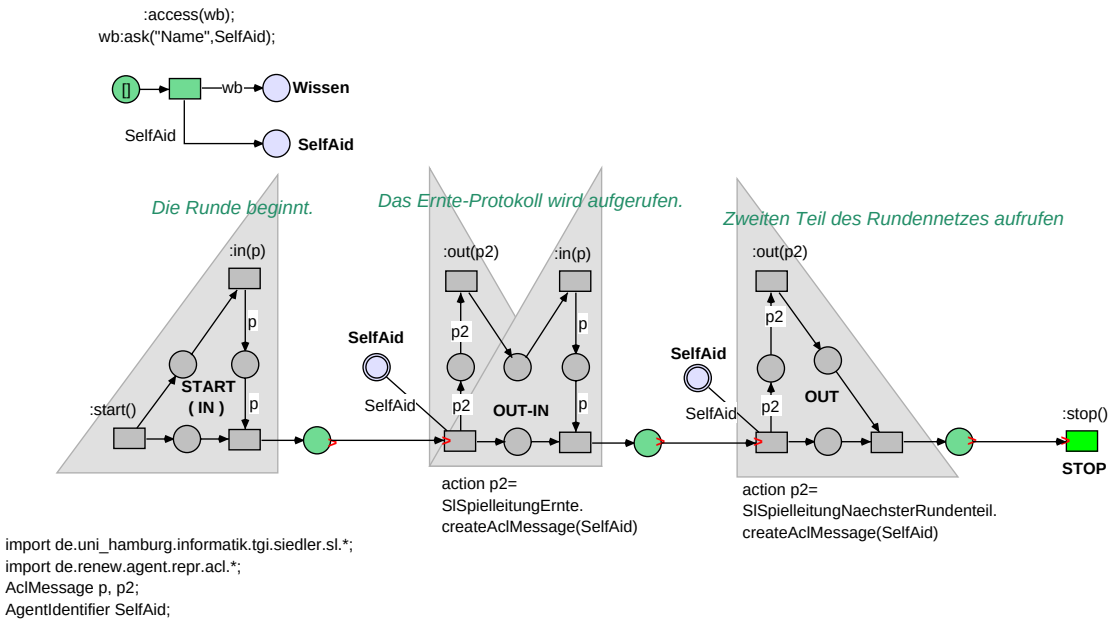


Abbildung 2.4: Funktionsfähige Netzkomponenten

Kanal namens `out` mit einem Parameter, welcher für je einen Schaltvorgang an den Inhalt von `p2` gebunden wird, so dass diese Information auf der anderen Kanalseite aufgenommen werden kann. Dann wartet das Netz darauf, dass der Kanal `in` schaltet. Die darin übertragene Information wird wie bei der ersten Netzkomponente an `p` gebunden und mit dem Schalten der nächsten Transition abgezogen und steht damit nicht mehr zur Verfügung. Die dritte Netzkomponente entspricht dem Anfang der zweiten Netzkomponente. In einer *action-expression* wird mit Hilfe von Daten wieder eine Information erzeugt und über den Kanal `out` an die übergeordnete Netzinstanz übergeben.

Die einzelne Transition `<STOP>` ist auch eine Netzkomponente. Sie hat eine Kanalanschrift für den Kanal `stop`. Wenn das Referenznetz bis zum Ende gelaufen ist, kann es auf diese Weise der umschließenden Netzinstanz mitteilen, dass es seine Arbeit abgeschlossen hat. Mit dieser Transition wird die letzte Marke aus dem Kontrollfluss des Netzes genommen, so dass es nicht mehr schalten kann. Wenn das umschließende Netz danach die Referenz auf dieses Netz verwirft, kann der *Garbage-Collector* von Java den Speicher der Netzinstanz freigeben. Das Referenznetz könnte statt dessen auch wieder aktiviert werden, indem der Kanal `start` ein zweites Mal schaltet.

Im Deklarationsknoten unter der ersten Netzkomponente werden die benötigten Java-Klassen importiert und die Variablen deklariert. In diesem Fall werden zwei Java-Packages importiert und drei Variablen deklariert.

3 Agenten

Die hier vorgestellten Agenten werden in der Agententechnologie verwendet, dabei handelt es sich um Software-Agenten. Unter dem Begriff Agententechnologie werden seit etwa zehn Jahren verschiedene Technologien zu einer neuen Abstraktionsweise für die Software-Entwicklung zusammengefasst. Die Wurzeln der Agententechnologie liegen vor allem in der *Künstlichen Intelligenz* (KI) und in der *Softwaretechnik*. In der KI sind Agenten „intelligente“ Einheiten, die Informationen verarbeiten können. In der Softwaretechnik sind Agenten ein Konzept zur Abstraktion und Kapselung für verteilte, offene oder anderweitig komplexe Systeme. Eng damit verwandt ist der Aspekt der Umsetzung von agentenorientierter Software.

Das Feld der Agententechnologie ist keinesfalls homogen. Durch die verschiedenen Ansätze und Zusammenhänge existieren innerhalb der Agententechnologie unterschiedliche Definitionen eines Agenten. Mittels Standardisierung werden ähnliche Agentenbegriffe zusammengefasst und gruppiert und gleichzeitig unterschiedliche Herangehensweisen stärker gegeneinander abgegrenzt. In den Abschnitten *Anwendungsorientierte Sicht auf Agenten* und *Technische Sicht auf Agenten* wird der Begriff Agent aus zwei Blickwinkeln erläutert und schließlich für diese Arbeit festgelegt. Dabei bezieht sich der erste Abschnitt eher auf das Verständnis und die Idee eines Agenten mit einem Beispiel für ein Multiagentensystem, während der zweite Abschnitt eher die in dieser Arbeit verwendeten Begriffe darstellt. Im dritten Abschnitt wird MULAN vorgestellt: ein in Petrinetzen realisiertes, umfassendes Modell für Agenten mit deren Umgebung und Funktionsweise. Um schon an dieser Stelle einen Überblick über die wesentlichen Begriffe zu erhalten, ist das Bild 3.1 auf Seite 32 gut geeignet. Im letzten Abschnitt dieses Kapitels werden die Arbeiten der FIPA vorgestellt, soweit sie für diese Arbeit benötigt werden.

3.1 Anwendungsorientierte Sicht auf Agenten

In dem Vortrag „Agent-Oriented Software Engineering“ der EASSS'03 (zur EASSS siehe *AgentLink* in Abschnitt 4.1) stellen Bergenti, Shehory und Sturm folgende Definitionen für Agent und Multiagentensystem als die stärkere von zwei möglichen Sichtweisen vor (vgl. [BSS03, S. 129][1]):

Definition 3.1 *KI-Sichtweise auf Agent und Multiagentensystem.*

3.1.1 (Agent) *„An Agent must be proactive, intelligent, and it must conversate instead of doing client-server computing.“*

3.1.2 (Multiagentensystem) *„A multiagent system is a society of individual (AI software agents) that interact by exchanging knowledge and by negotiating with each other to achieve either their own interest or some global goal.“*

Dies ist das aus der künstlichen Intelligenz übernommene Konzept von Agenten. Leider steht nur indirekt im zweiten Teil, dass es sich dabei um Software-Ageten handelt. Nach dieser Definition wird ein Softwareprogramm als *Agent* bezeichnet, wenn es Methoden der künstlichen Intelligenz nutzt und nicht Teil eines Client/Server-Systems ist, sondern an Konversationen teilnimmt. Ein *Multiagentensystem* ist eine Gesellschaft von Agenten, die untereinander Wissen austauschen und verhandeln, um ihre eigenen Interessen zu verfolgen oder um ein globales Ziel zu erreichen.

Intelligente Agenten treffen Entscheidungen aufgrund von Wahrnehmungen und ihres modellierten Wissens über ihre Umgebung. Geläufige Entscheidungsarchitekturen[2] sind die BDI-Architektur (*Beliefs-Desires-Intentions*) und die Subsumptionsarchitektur.

Bei der BDI-Architekur sind innerhalb des Agenten seine Informationen über den Zustand der Umgebung (*Beliefs*) modelliert. Der Agent hat außerdem Ziele und Wünsche über den Zustand der Umgebung (*Desires*). Zusammen mit dem Wissen um seine Handlungsmöglichkeiten kann ein Plan erstellt werden, in dem die Absichten (*Intentions*) festgelegt sind.

Die Subsumptionsarchitektur wurde von Brooks eingeführt (siehe [Bro86]). Der Agent (in Brooks Fall ein mobiler Roboter) verarbeitet Wahrnehmungen in mehreren Schichten. Wahrnehmungen können entweder von Sensoren oder über Nachrichten aufgenommen werden. Die am höchsten priorisierte Schicht reagiert zuerst auf eintreffende Reize, z.B. Kollisionsvermeidung bei einem mobilen Roboter. Die niedriger priorisierten Schichten werden erst aktiviert, wenn die hoch priorisierten Schichten keine Handlungen ausführen. Ein Agent mit einer Subsumptions-Entscheidungsarchitektur ist ein rein reaktiver Agent.

[1] Das Wort *conversate* gibt es im Englischen nicht, es ist zu verstehen als: *an Konversationen beteiligt sein.*

[2] Der Begriff der Architektur wurde differenziert für die vielfältigen Aspekte in der Agententechnologie (siehe Glossar)

Beispiel

Ein Beispiel für ein Multiagentensystem stellt Nicholas Jennings in seinem Vortrag „Applying Agent Technology“, ebenfalls auf der EASSS vor (siehe [Jen03, Seite 268]): Eine Fertigungsstraße in einer Autofabrik soll effizienter genutzt werden. Vor der Änderung gab es Maschinen für spezielle Aufgaben und ein Förderband für die Arbeitsstücke. Die Planung wurde jede Nacht für den nächsten Tag entwickelt und dann tagsüber mit häufigen Abweichungen so gut wie möglich umgesetzt.

In der Neuentwicklung wurden die Förderbänder und Maschinen flexibler gestaltet, so dass Arbeitsstücke beliebig zwischen den Maschinen ausgetauscht werden können. Dadurch wächst die Vielfalt der möglichen Wege eines Arbeitsstückes und entsprechend auch die Vielfalt der Arbeitsabläufe an einer Maschine beträchtlich. Ein Tagesplan wird nicht mehr statisch über Nacht entwickelt, sondern von einem Multiagentensystem dynamisch während der laufenden Produktion. In diesem Multiagentensystem gibt es im Wesentlichen zwei Sorten von Agenten: Arbeitsstück-Agenten und Maschinen-Agenten. Die Förderbänder werden auch von „Agenten“ gesteuert, die jedoch sehr einfach gehalten sind und nur nach der im nächsten Abschnitt vorgestellten software-orientierten Sicht als Agenten bezeichnet werden können.

Die Agenten interagieren untereinander, um ihr internes Ziel bestmöglich zu erreichen: Arbeitsstück-Agenten wollen alle nötigen Arbeitsschritte in der richtigen Reihenfolge und möglichst schnell durchlaufen, Maschinen-Agenten wollen eine gute Auslastung für ihre Maschine erreichen, und Förderband-Agenten bieten einen Dienst zum Transport der Arbeitsstücke an, der von den anderen Agenten in Anspruch genommen wird.

Die Planung findet nun nicht mehr zentral und „offline“ statt, sondern während der Produktion, als Teil des Vorgangs. Die einzelnen Agenten kennen jeweils ihre lokale Situation und ihr nächstes Ziel und erreichen durch Kommunikation und Kooperation eine gute Lösung der globalen Planungsaufgabe.

Jennings stellt dies als ein Beispiel einer gelungenen Umorientierung von einer statischen Tagesplanung zu einer dynamischen Planung dar. In seinem Vortrag ist Jennings nicht auf die Frage eingegangen, wie Verklemmungen in einer dynamischen Planung vermieden werden können[3]. Es gibt natürlich auch andere Ansätze als den agentenorientierten, um eine dynamische und verteilte Planung zu realisieren. Beim Entwurf und der Entwicklung ist diese Metapher jedoch von besonderem Wert.

[3] Ezpeleta et al betrachten die automatische Erzeugung von verklemmungsfreien Petrinetzen, mit denen dann flexible Fertigungssysteme modelliert und gesteuert werden können (siehe [EGVC98]).

Begriffe

Die Agenten, die hier behandelt werden, kommunizieren grundsätzlich mittels Nachrichten, diese Eigenschaft wird mit dem Begriff „kommunizierende Agenten“ ausgedrückt. Wenn diese Nachrichten getypt sind, kann man auch von „*typed-message-agents*“ sprechen. Wenn die Agenten zusätzlich nach den Spezifikationen der FIPA entworfen sind, können sie auch als „FIPA-Agenten“ oder „ACL-Agenten“ bezeichnet werden (ACL ist die von der FIPA definierte Agentenkommunikationssprache). Diese Begriffe betonen jeweils nur leicht unterschiedliche Aspekte desselben Prinzips. Wenn die Agenten nicht nur mittels Nachrichten kommunizieren, sondern auch über physikalische Sensoren und Effektoren verfügen, kann diese Eigenschaft mit dem Begriff „situierte Agenten“ betont werden.

Eine Ontologie definiert Begriffe, die zur Beschreibung und Repräsentierung eines Wissensgebiets verwendet werden können, sowie die Beziehungen und Eigenschaften der durch dieses Vokabular bezeichneten Objekte. Mit Hilfe von Ontologien kann man z.B. festlegen, wie der Inhalt von Nachrichten oder ein bestimmter Nachrichtentyp interpretiert werden soll.

Zusammen mit dem Agenten wird häufig auch seine Umgebung genannt, also das Umfeld, in dem er sich befindet und in dem er handelt. Die verschiedenen Eigenschaften von Umgebungen führen Wooldridge und Luck in ihrem Vortrag „Introduction to Agents“ auf der EASSS ein (siehe [WL03]):

- Eine Umgebung kann *vollständig einsehbar* sein, dann bilden die Wahrnehmungen des Agenten sicheres und vollständiges Wissen.
- In einer *deterministischen* Umgebung hat eine Handlung des Agenten eine vorhersagbare Wirkung – nicht unbedingt für den Agenten selbst: Wenn die Umgebung nicht vollständig einsehbar ist, kann eine deterministische Umgebung nicht-deterministisch auf den Agenten wirken.
- In einer *episodischen* Umgebung wirken sich Handlungen des Agenten nur auf die aktuelle Episode aus. Eine Episode ist ein Element aus einer Folge von unabhängigen Teilen.
- Eine *statische* Umgebung ändert ihren Zustand im Gegensatz zu einer dynamischen Umgebung nur durch Handlungen des Agenten.
- Eine *diskrete* Umgebung hat eine endliche Menge von Zuständen, in einer *kontinuierlichen* Umgebung sind immer Zwischenzustände möglich.

Die Eigenschaften einer konkreten Umwelt für Agenten hängen vom Anwendungsfeld ab. Sobald mehrere Agenten interagieren, bilden diese einen Teil der Umwelt, die dann nicht vollständig einsehbar ist, da jeder Agent eine eigenständige Einheit bildet.

3.2 Technische Sicht auf Agenten

In der Agentenorientierten Softwareentwicklung (AOSE) werden Agenten als Designmetapher verwendet.

> „Agents provide designers and developers with a way of structuring an application around autonomous, communicative elements“
> [LMP03, S. 10]

Einerseits bieten Agenten eine angemessene Art, komplexe Systeme mit vielfältigen unabhängigen Komponenten zu betrachten. Andererseits ermöglichen Agenten auch die Zusammenfassung von verschiedenen Funktionen, die vorher getrennt betrachtet wurden (wie Planung, Lernen und Koordination), zu einem konzeptuellen Ganzen (vgl. [LMP03, S. 2f]).

Bis hierher wurde die Idee des Agenten vorgestellt, im Folgenden geht es um die Konzepte zur technischen Umsetzung von agentenorientierter Software. Die folgenden Definitionen eines Agenten und eines Multiagentensystems stammen wie die Definitionen im vorigen Abschnitt aus dem Vortrag von Bergenti et al.:

Definition 3.2 *Softwaretechnik-orientierte Sichtweise auf Agent und Multiagentensystem.* [BSS03, S. 129][4]

3.2.1 (Agent) *„An agent is a software component with internal (either reactive or proactive) threads of execution, .. that can be engaged in complex and stateful interaction protocols.“*

3.2.2 (Multiagentensystem) *„A multiagent system is a software system made up of multiple independent and encapsulated loci of control (i.e., the agents) interacting with each other in the context of a specific application viewpoint.“*

Diese Definition betont die Aspekte, die sich auf die Software-Entwicklung auswirken; sie ist allgemeiner als die KI-orientierte Definition, weil Intelligenz als spezielle Form verteilter Kontrolle betrachtet werden kann und Konversation als eine spezielle Form von Interaktion. Die Elemente der Definition bedürfen der Erläuterung. In der folgenden Liste sind die wesentlichen Eigenschaften von Agenten zusammen mit ihrer softwaretechnischen Entsprechung aus der Definition aufgezählt:

- Ein Agent ist eine *Softwarekomponente*, also bezüglich Daten und Funktionen eine *gekapselte* Einheit.

[4] Im Original heißt es *interactions protocols* und *software systems*. Die beiden Punkte stehen für ein weggelassenes *and*.

- Ein Agent hat interne *Threads*. Dies ist eine Voraussetzung für *Autonomie* des Agenten im Kontrollfluss.
- Diese Threads können *flexibel* gestartet werden: Ein *reaktiver* Thread wird als Reaktion auf einen Reiz aus der Umwelt gestartet, ein *proaktiver* Thread wird gestartet, um ein intern formuliertes Ziel zu erreichen. In der Möglichkeit zu proaktivem Verhalten liegt eine erweiterte Autonomie.
- Ein Agent kann an *komplexen und statusbehafteten* Interaktionen teilhaben, die in *Interaktionsprotokollen* formuliert sind.
- Ein *Multiagentensystem* besteht aus mehreren unabhängigen und gekapselten Orten der Kontrolle (den Agenten).
- Der Begriff der *Umgebung* wird in der Definition nicht explizit genannt, der Agent jedoch als Komponente bezeichnet. Ein Agent ist außerdem in einem Multiagentensystem von anderen Agenten umgeben, das Multiagentensystem ist also ebenfalls Teil der Umgebung für Agenten.
- Ein spezielles Multiagentensystem ist über einen *Anwendungszusammenhang* definiert.

Zwischen den beiden bisher vorgestellten extremen Sichtweisen (orientiert an der künstlichen Intelligenz (Definition 3.1) bzw. an der Softwareentwicklung (Definition 3.2)) steht die sorgfältig formulierte und häufig zitierte Definition von Jennings und Wooldridge (siehe [WJ95]). Diese Definition hat meiner Ansicht nach einen angemessenen Abstraktionsgrad, um Agenten in der Praxis und im Konzept zu erfassen.

Definition 3.3 *Gemäßigte Sichtweise auf Konzept und Realität von Agenten.*

3.3.1 (Agent) *„An Agent is an encapsulated computer system, situated in some environment, and capable of flexible autonomous action in that environment in order to meet its design objectives.“* [Jen03, S. 266f, Folien 10–13]

In dieser Definition sind die Begriffe, die nach der Definition 3.2 ausformuliert wurden, direkt genannt. Ein Agent ist demnach ein gekapseltes Computersystem, das sich in einer Umgebung befindet und dem in dieser Umgebung flexible autonome Aktionen möglich sind, um die Entwurfsziele zu erfüllen.

Für die vorliegende Arbeit ist die Definition 3.2 maßgeblich. Sie ist die allgemeinste und betont die softwaretechnischen Aspekte. Das ist zur Arbeit an einer Infrastruktur für Agenten sinnvoll.

Zur Realisierung von agentenorientierter Software wird eine Infrastruktur geschaffen, die nicht anwendungsspezifische Aufgaben wie Nachrichtentransport und Agentenverwaltung übernimmt: die *Agentenplattform*. In dieser Arbeit ist die Agentenplattform CAPA von zentraler Bedeutung; sie wird im Kapitel 5 vorgestellt. Die Plattform bildet die unmittelbare Umgebung eines Agenten, ähnlich einem Browser für Applets. Mehrere Plattformen können vernetzt werden.

Der Begriff Multiagentensystem

Der Begriff Multiagentensystem wird in verschiedenen Zusammenhängen verwendet. Im Folgenden werden drei Benutzungsweisen identifiziert und dargestellt: Multiagentensysteme als Oberbegriff, in der anwendungsorientierten und in der technisch orientierten Sichtweise.

Als Oberbegriff dient *Multiagentensysteme* zur Abgrenzung der Agententechnologie gegen andere (evtl. verwandte) Technologien wie die Objektorientierung und die Technologien für verteilte Systeme[5].

In der anwendungsorientierten Sichtweise wird der Begriff Multiagentensystem für eine agentenorientierte Anwendung benutzt (so auch in den Definitionen 3.1 und 3.2). Ein Multiagentensystem in diesem Sinne sind das im vorigen Abschnitt vorgestellte Beispiel und die Anwendung Siedler, die im Kapitel 7 dieser Arbeit vorgestellt wird. Diese Sichtweise ist in dem Ursprung der Agententechnologie aus der künstlichen Intelligenz verankert, deshalb werden hier häufig intelligente Agenten assoziiert.

Die technische Sichtweise schließlich ist eher an der Softwaretechnik orientiert und stärker mit dem Umfeld der verteilten Systeme verwandt. Ein Multiagentensystem in diesem Sinne ist ein technisch abgegrenztes Netz von Agenten, z.B. eine Agentenplattform oder mehrere vernetzte Agentenplattformen. Um die technische Sichtweise von der anwendungsorientierten Sichtweise abzugrenzen, wird in dieser Arbeit für vernetzte Plattformen der Begriff *Agentennetz* gewählt. Alternative Bezeichnungen sind *Netz von Agentenplattformen* oder *Plattformnetz*, ich sehe jedoch die Agenten als die aktiven Elemente in einer Vernetzung und die Plattformen als Kommunikations- und Verwaltungsschicht darunter. Deshalb wird in dieser Arbeit der Begriff *Agentennetz* verwendet, obwohl dieser Begriff auch für ein Referenznetz in CAPA verwendet wird, welches einen Agenten repräsentiert.

Der Übergang von einem (anwendungsgebundenen) Multiagentensystem zu einem (technisch definierten) Agentennetz kann jedoch fließend sein: Wenn die Agenten einer spezifischen Agentenplattform miteinander kommunizieren müssen (z.B. um Ressourcen für die jeweilige Anwendung zu reservieren), kann dies bereits als Anwendungszusammenhang bezeichnet werden.

[5] In diesem Sinne ist der Titel dieser Arbeit zu verstehen.

Zusammenfassend ist hier die vorgeschlagene Verwendung der Begriffe dargestellt und mit Abbildung 3.1 veranschaulicht:

Agententechnologie und Multiagentensysteme zur Abgrenzung gegen andere Technologien wie Objektorientierung und verteilte Systeme.

Multiagentensystem bezeichnet ein System von Agenten, die in einem Anwendungszusammenhang interagieren. Die Agenten können verschiedenen Agentenplattformen zugeordnet sein.

Agentennetz bezeichnet ein System von Agenten, die technisch miteinander verbunden sind. Sie könnten untereinander kommunizieren. Eine Agentenplattform bildet einen Knoten in einem Agentennetz.

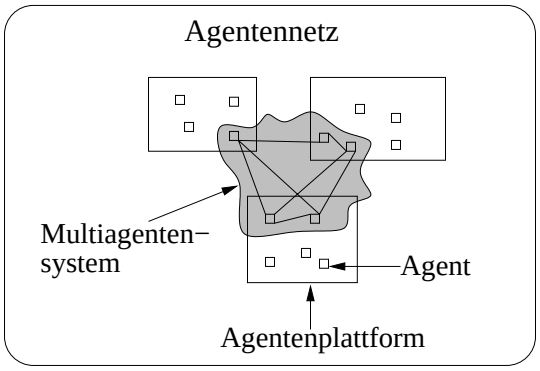


Abbildung 3.1: Multiagentensystem versus Agentennetz

In Abbildung 3.1 ist ein Agentennetz aus drei Agentenplattformen zu sehen. Den Plattformen sind Agenten zugeordnet, von denen einige ungeachtet der Plattformgrenzen untereinander kommunizieren und so ein Multiagentensystem bilden.

Damit bildet ein Agent einen Knoten in einem Multiagentensystem, eine Plattform bildet einen Knoten in einem Agentennetz. Insofern laufen die Grenzen eines Agentennetzes immer entlang von Plattformgrenzen, bei einem Multiagentensystem sind die Grenzen unabhängig von den beteiligten Plattformen. Die technische Infrastruktur einer Plattform kann von mehreren Anwendungen genutzt werden.

Das Referenzmodell

Die bisher dargestellte Information soll nun zu einem Referenzmodell zusammengefasst und ergänzt werden: *Agenten* sind (laut Definition 3.2) an zustandsbehafteten Konversationen beteiligt, deren Zustand und Nachrichten der Agent mit Hilfe von *Protokollen* kontrolliert. Eine Nachricht stellt die Handlung (einen Sprechakt) eines

Agenten dar und wird *Performativ* genannt[6]. Jeder Agent verwaltet sein Wissen in einer *Wissensbasis*. Die unmittelbare Umgebung der Agenten ist die *Plattform*, mehrere Plattformen können zu einem *Agentennetz* verbunden werden.

3.3 Mulan

Die bisher beschriebenen technischen Strukturen lassen sich in einer Hierarchie aus vier Ebenen einordnen: Agentennetz[7], Plattform, Agent und auf der untersten Ebene die Protokolle und die Wissensbasis. Mit dieser Hierarchie wird das Agentenkonzept umfassend abgebildet. Zur Modellierung dieser Hierarchie eignen sich Referenznetze, wobei der Mechanismus *Netze in Netzen* verwendet wird. Der vertikale Informationsaustausch kann über synchrone Kanäle stattfinden, etwa wenn in einem Protokoll eine Nachricht erzeugt wird, die der Agent zur Übermittlung an die Plattform weitergibt.

Mit MULAN (für „*MUL*ti*A*genten*N*etze“) liegt eine umfassende, lauffähige Modellierung dieser Verhältnisse mit Referenznetzen vor. MULAN wurde 1999 von Rölke eingeführt (in [Röl99]), die Entwicklung ist in [PN01, PN03] veröffentlicht. Die Architektur von MULAN ist in Abbildung 3.2 dargestellt.

Die oberste Ebene bildet vernetzte Plattformen ab. Sie ist im Bild als *Multi Agent System* beschriftet und mit einer beispielhaften Vernetzung dargestellt. In jeder Stelle ist eine Plattform enthalten (also liegt dort eine Referenz auf ein Plattformnetz), die Transitionen stellen die Nachrichtenkanäle zur plattformübergreifenden Kommunikation dar.

Alle Plattformen in MULAN haben die abgebildete Struktur. In der Stelle in der Mitte des Netzes sind alle Agenten der Plattform enthalten (d.h. es liegen dort Referenzen auf die Netzinstanzen der Agenten). Die Plattform verwaltet den Lebenszyklus der Agenten, kann also Agenten erzeugen und beenden; außerdem bietet die Plattform Kommunikationskanäle für interne und plattformübergreifende Nachrichten an. Weitere anwendungsunabhängige Dienste wie z.B. ein erweiterter Agentenlebenszyklus mit inaktivem Zustand sind möglich.

Der Aufbau der Agenten-Ebene enthält alle wesentlichen Funktionen für kommunizierende Agenten und braucht für einen konkreten Agenten nicht geändert werden. In den drei zentralen Stellen ist der Zustand des Agenten repräsentiert: Die obere Stelle enthält die Wissensbasis des Agenten (also eine Referenz auf die Netzinstanz

[6] Die Sprechakte gehen auf die Analyse der menschlichen Kommunikation von Searle (siehe [Sea69]) zurück. Danach enthält jeder kommunikative Akt Botschaften auf mehreren Ebenen: Worte werden zu Sätzen gereiht, Objekte referenziert, eine Handlung ausgeführt (behaupten, warnen, fragen etc.) und schließlich wird eine Wirkung beim Gesprächspartner hervorgerufen. Die Handlung ist der *performative* Akt.

[7] In diesem Abschnitt kollidieren die beiden Bedeutungen des Begriffs Agentennetz, deshalb wird hier im Weiteren „Vernetzung von Plattformen“ verwendet. So bedeutet Agentennetz in diesem Abschnitt ein Referenznetz oder Netzinstanz für einen Agenten.

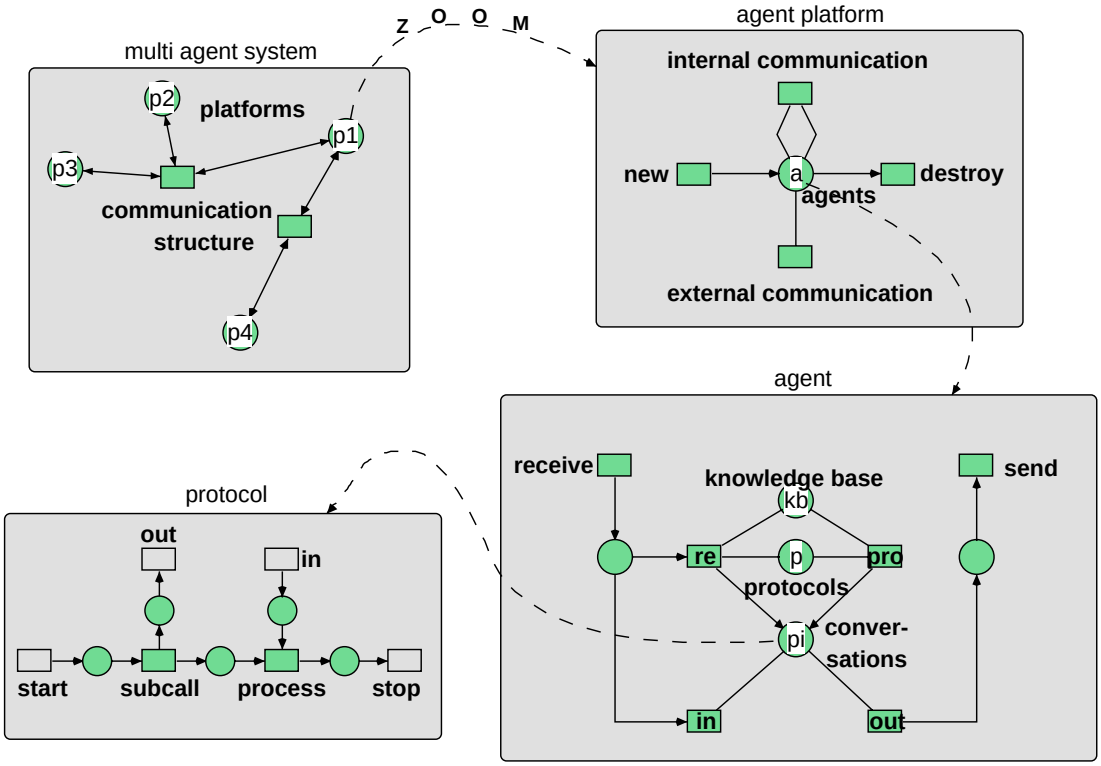


Abbildung 3.2: MULAN-Architektur. Hier aus [Duv02, S. 30]

der Wissensbasis), die mittlere enthält Verweise auf die dem Agenten bekannten Protokolle und die unterste enthält Konversationen (also Referenzen auf instanziierte Protokollnetze).

Über die Transition `<receive>` erhält der Agent Nachrichten von der Plattform. Verweist eine ankommende Nachricht auf eine laufende Konversation, kann die Transition `<in>` schalten, die die Nachricht an die entsprechende Protokollnetzinstanz weitergibt. Über die Transition `<out>` kann eine Nachricht aus dem Protokollnetz verschickt werden. Der Agent reicht diese Nachricht über die Transition `<send>` weiter an die Plattform. Schließlich sind die beiden Transitionen `<re>` und `<pro>` für die Instanziierung von Protokollen zuständig. Sie bilden so die Anfangspunkte für Handlungen des Agenten, die Namen `re` und `pro` stehen für *reaktive* und *proaktive* Handlung. Der Agent handelt proaktiv, wenn in der Wissensbasis die Voraussetzungen dafür gegeben sind. Eines der verfügbaren Protokolle wird dann instanziiert und in die Stelle `<conversations>` gelegt und Handlungen (insbesondere Nachrichtenaustausch) werden von dort ausgeführt. Der Agent handelt reaktiv, wenn eine Nachricht ohne einen Verweis auf eine existierende Konversation eintrifft. Mit Hilfe der Wissensbasis entscheidet der Agent, welches der verfügbaren Protokolle geeignet

ist, um auf die Nachricht zu reagieren. Dieses Protokoll wird wie bei der proaktiven Handlung instanziiert und in die Stelle `<conversations>` gelegt.

Schließlich ist beispielhaft ein Protokollnetz dargestellt[8]. Es hat eine Start- und eine Stopp-Transition und die Transitionen `<in>` und `<out>`, über die mit synchronen Kanälen Nachrichten mit den Transitionen `<in>` und `<out>` im Agentennetz ausgetauscht werden können. Protokollnetze werden individuell für die Aufgaben eines Agenten entworfen. Zusammen mit dem in der Wissensbasis verwalteten Wissen bestimmen die Protokollnetze das Verhalten eines Agenten.

Die Wissensbasis eines Agenten befindet sich auf derselben Ebene wie die Protokollnetze (die Darstellung des Netzes fehlt). Im Unterschied zu den Protokollnetzen wird die Wissensbasis nur einmal bei der Erzeugung eines Agenten instanziiert und bleibt für die gesamte Lebenszeit des Agenten erhalten. Der Inhalt der Wissensbasis ist individuell für einen Agenten zu einem Zeitpunkt. Das Anfangswissen wird dem Agenten beim Start mitgegeben. Aus den Protokollen heraus können Einträge in der Wissensbasis gelesen, hinzugefügt oder geändert werden.

In den Protokollnetzen kann die Mächtigkeit der Referenznetze voll zur Gestaltung von Agenten genutzt werden: MULAN kann sogar rekursiv betrachtet werden, denn ein Agent kann ein Protokoll enthalten, welches wiederum ein Agentennetz, eine Plattform oder einen Agenten darstellt.

Die in MULAN modellierten Agenten lassen sich ohne zusätzlichen Aufwand in der Infrastruktur adaptiv, mobil und autonom gestalten. Diese Thematik wird zur Zeit von Heiko Rölke im Rahmen einer Dissertation ([Röl04]) untersucht. Zur Mobilität von MULAN-Agenten wurde auf der Petrinetz-Konferenz im Jahr 2003 ein Konzept vorgestellt ([KMR03]). Darin wird vorgestellt, wie ein mobiler Agent während seiner Laufzeit von einer Plattform zu einer anderen wechseln kann. In diesem Fall wird mit Hilfe der Plattformen ein Ortskonzept verbunden. Bei einer solchen Bewegung wird die Wissensbasis eines Agenten übertragen, zusätzlich steht dem Agenten ein Kanal zur Verfügung, über den er anzeigen kann, ob er gerade tätig ist.

Mit Hilfe von speziellen Protokollen kann der Agent sein Verhalten im Laufe der Zeit aufgrund von Ereignissen anpassen, indem er z.B. Protokolleinträge in der Wissensbasis ändert. Auf diese Weise können MULAN-Agenten adaptiv sein.

Die Autonomie der MULAN-Agenten hat zwei Grundlagen: die nebenläufige Ausführung eines Agenten und die starke Kapselung, bei der nur mittels Nachrichten auf den Agenten Einfluss genommen werden kann.

Bei der Entstehung von MULAN um 1998 waren die noch jungen Spezifikationen der FIPA richtungsweisend. Das Referenzmodell der FIPA wurde für MULAN zugrunde gelegt, also die Konzepte, dass Agenten auf Plattformen existieren, dass Agenten mittels Nachrichten kommunizieren und darin ihre einzige Schnittstelle nach außen liegt. Im nächsten Abschnitt wird die FIPA mit den hier relevanten Spezifikationen vorgestellt.

[8]Ein Beispiel für ein Protokollnetz wurde bereits als Beispiel in Kapitel 2 vorgestellt.

3.4 Standardisierung nach FIPA

Neben der KI-orientierten, der Software-orientierten und der gemäßigten Agentendefinition (Definitionen 3.1, 3.2 und 3.3) wird durch jede Implementierung eines Agentensystems eine operationale Definition erzeugt. Wenn verschiedene Systeme zusammenarbeiten sollen, müssen insbesondere die dadurch definierten Schnittstellen eines Agenten übereinstimmen.

Die FIPA (*Foundation for Intelligent Physical Agents*) ist ein Gremium zur Standardisierung von Agentenkommunikation, das 1996 gegründet wurde und eng mit anderen Gruppen wie der OMG zusammenarbeitet, um relevante und verbreitete technische Standards zu schaffen.

Die Spezifikationen der FIPA beziehen sich auf die Außenwirkung von Agenten, auf deren Nachrichten und angebotenen Dienste und auf Schnittstellen. Das Agentenkonzept der FIPA wird in der sogenannten *Abstrakten Architektur* erarbeitet und in einer Sammlung von Spezifikationen unter dem Begriff *FIPA2000* konkretisiert. Die FIPA betont die Möglichkeit, sehr verschiedene Middlewaretechniken und Programmiersprachen zur Umsetzung der Abstrakten Architektur und auch der konkreteren Spezifikationen zu nutzen.

Die FIPA sieht fünf Zustände vor, in denen eine Spezifikation sich befinden kann. Der erste Vorschlag wird als vorläufig (*preliminary*) bezeichnet. Wenn dieser ausgearbeitet ist und brauchbar scheint, wird er als experimentell (*experimental*) gekennzeichnet. Hat sich die Spezifikation bewährt, wird sie zum Standard (*standard*) erklärt. Aus jedem dieser drei Zustände kann eine Spezifikation als nicht mehr empfehlenswert (*deprecated*) eingestuft werden und schließlich als hinfällig (*obsolete*). Am 6. Dezember 2002 wurde ein ganzer Satz von Spezifaktionen zu FIPA2000 aus dem experimentellen Status zu Standards erhoben, z.B. auch die zentrale Spezifikation [FIPA 23], die nach der Abstrakten Architektur im Abschnitt zur konkreten Architektur vorgestellt wird.

3.4.1 Die Abstrakte Architektur

Die Abstrakte Architektur definiert ein Referenzmodell für Agenten, welches auch Vorbild für die Entwicklung von MULAN war und bereits vorgestellt wurde. Weiter definiert die Abstrakte Architektur die wesentlichen Elemente einer Agentenumgebung, wie in Abbildung 3.3 dargestellt. Es muss demnach einen Nachrichtentransport geben, Verzeichnisse für Agenten und Dienste und eine gemeinsame Agentenkommunikationssprache.

Der Aufbau einer Nachricht wird in [FIPA 61] spezifiziert. Die so definierte Sprache heißt ACL (*Agent Communication Language*). Eine Nachricht enthält nach [FIPA 61, S. 2] die Bezeichnung des kommunikativen Aktes (also das Performativ, z.B. *request* oder *inform*), Informationen über die Teilnehmer der Konversation (Absender, Empfänger, Antwortadresse), den eigentlichen Nachrichteninhalt mit In-

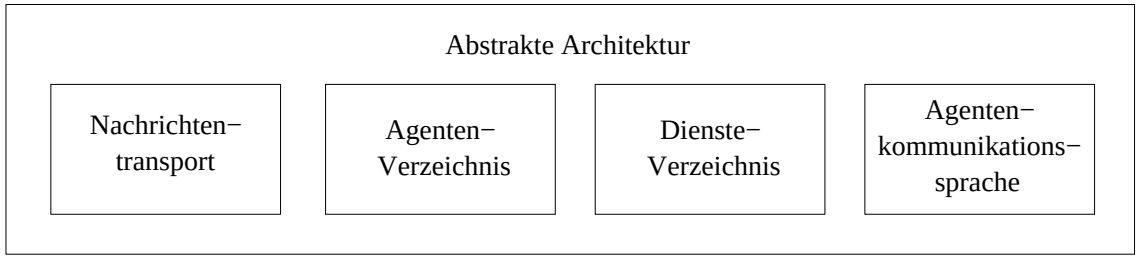


Abbildung 3.3: Elemente der Abstrakten Architektur. Nach [FIPA 01, S. 6]

formationen dazu (Inhaltssprache, Kodierung und Ontologie) und schließlich Informationen zur Konversationskontrolle (Protokoll, Konversations-ID, Nachrichten-ID, evtl. einen Verweis auf eine alte Nachricht und eine Zeitangabe). Alle Felder einer ACL-Nachricht sind in Abschnitt 3.4.4 auf Seite 42 dargestellt.

3.4.2 Die konkrete Architektur: FIPA2000

Die Spezifikationen, die unter FIPA2000 zusammengefasst sind, führen die Details zur Nachrichtendarstellung aus, geben den Elementen aus der Abstrakten Architektur Namen sowie konkrete Entsprechungen und füllen die von der Abstrakten Architektur nicht spezifizierten Teile aus (etwa die Zustände im Lebenszyklus eines Agenten). In dem Dokument [FIPA 23] wird der Bogen von der Abstrakten Architektur zu konkreten Spezifikationen geschlagen, in denen mögliche Implementationsweisen spezifiziert werden. Die Dokumentstruktur einiger FIPA2000-Spezifikationen ist in Abbildung 3.4 auf der nächsten Seite dargestellt.

Das **Referenzmodell** der FIPA2000-Architektur stimmt mit dem Referenzmodell der Abstrakten Architekur überein, dabei wurde das Konzept der Agentenplattform hinzugefügt. Die **Namen der Agenten** sollen die Form `lokalerName@Plattformname` haben. Die bereits in der Abstrakten Architektur verlangten **Dienste** werden hier benannt und konkretisiert: Der Dienste-Verzeichnisdienst wird vom `"DF"` (*Directory Facilitator*) erbracht, der Agenten-Verzeichnisdienst ist dem `"AMS"` (*Agent Management System*) zugeordnet, der Nachrichtentransportdienst soll vom `"ACC"` (*Agent Communication Channel*) erbracht werden. Dazu steht eine Definition eines Nachrichtenumschlags und mehrerer Transportmittel (*Message Transport Protocol*, `"MTP"`) zur Verfügung. Die **Plattform** betreffend wird spezifiziert, wie ein Agent sich auf einer Plattform registrieren soll und welche Zustände der Lebenszyklus eines Agenten hat. Die **Agentenkommunikationssprache** (`"ACL"`) wird bereits als Teil der Abstrakten Architektur definiert, die konkrete Syntax wird hier für drei verschiedene Repräsentierungen spezifiziert. Für den Inhalt von Nachrichten werden mehrere **Inhaltssprachen** definiert. Eine Teilmenge der Inhaltssprache SL mit der Bezeichnung SL0 wird zur Agentenverwaltung verwendet. Eine

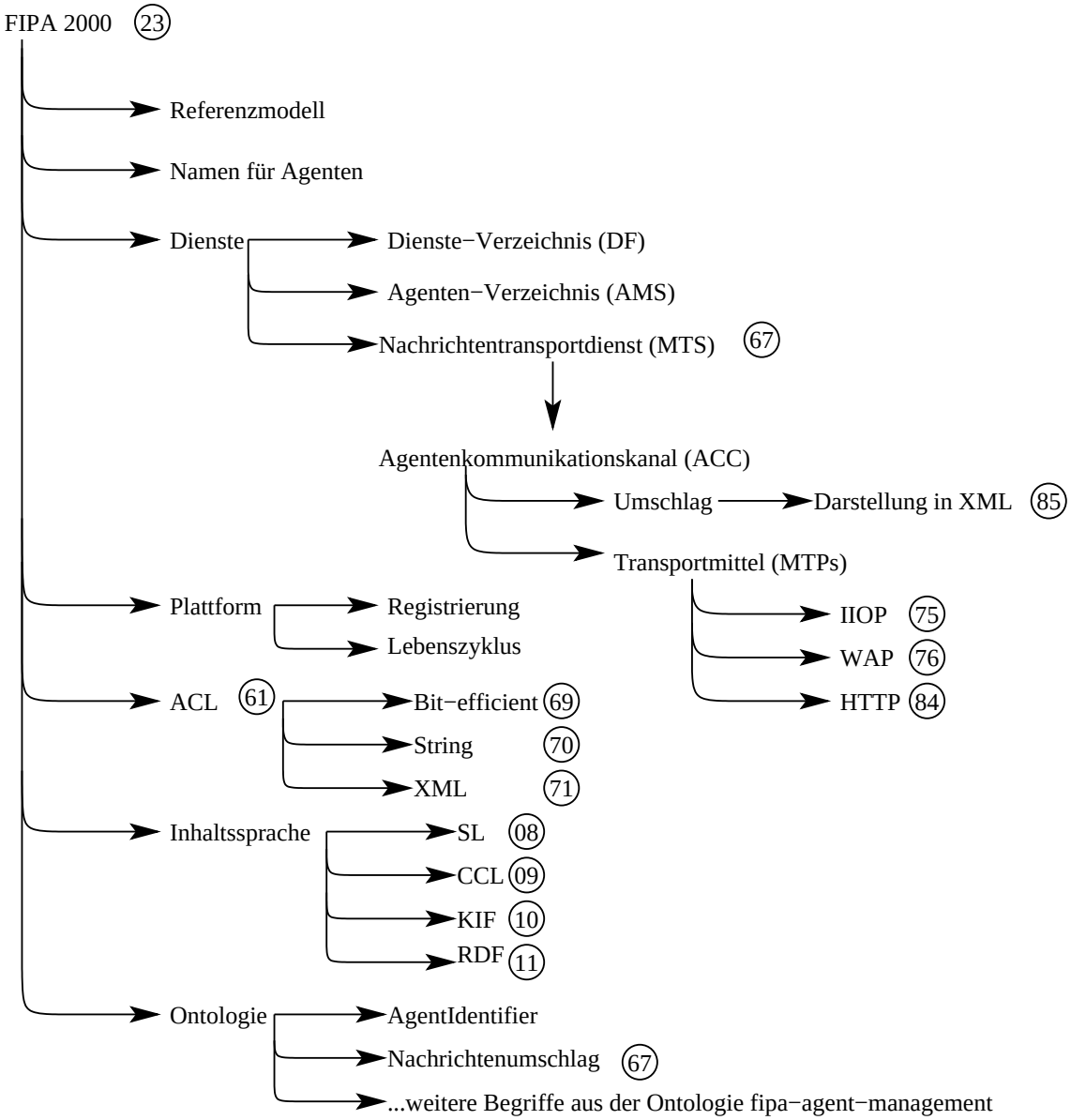


Abbildung 3.4: Dokumentstruktur der FIPA zur Spezifikation einer Plattform. In den Kreisen sind die Dokumente mit ihrer Nummer angegeben, wo der Inhalt nicht in [FIPA 23] spezifiziert wird.

große Bedeutung kommt der Definition von festen Begriffen in einer **Ontologie** zu. Die Ontologie definiert unter anderem Datenstrukturen zur Agentenidentifikation (`"agent-identifier"`) und für den Umschlag von Nachrichten (`"envelope"`).

Die Bestandteile einer Agentenplattform nach FIPA2000 sind im Überblick in Abbildung 3.5 dargestellt. Die **Sprachdefinitionen** umfassen die Agentenkommunikationssprache ACL, die Inhaltssprache SL0 und als Basisontologie die Agent Management Ontology. Die Elemente einer Nachricht können in einem internen Format dargestellt werden, müssen aber vor dem Versenden in eine der spezifizierten **Darstellungen** transformiert werden. Für den **Nachrichtentransportdienst** sind drei alternative Transportmittel definiert. Die **Verzeichnisdienste** werden von `AMS` und `DF` bereitgestellt, welche auf der Plattform wie Agenten ansprechbar sein müssen.

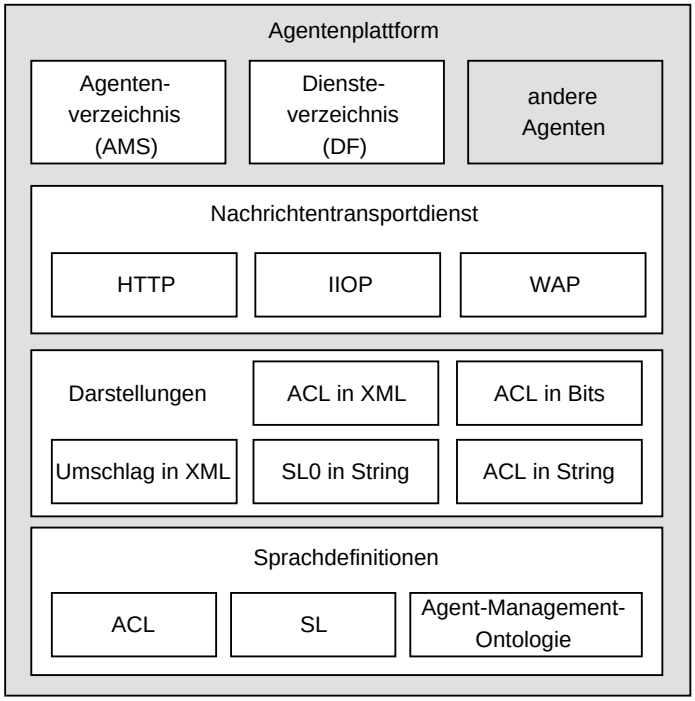


Abbildung 3.5: Elemente einer FIPA2000-Agentenplattform

Eine Nachricht ist in drei Schichten aufgebaut: Die Agentenkommunikationssprache ist die äußerste Schicht, die Inhaltssprache ist in der Mitte und die Ontologie definiert die darin verwendeten Begriffe. Zum Transport wird der Nachricht ein Umschlag mit den für den Transport relevanten Daten beigefügt, in dem unterwegs Informationen zum Verlauf der Nachricht protokolliert wird.

Die Darstellung der Inhaltssprache ist ihre Kodierung (`encoding`), die Darstellung

der ganzen Nachricht ihre Repräsentierung (`representation`). Diese Verwendung der Begriffe wird für den Rest des Textes beibehalten.

Jede Nachricht wird originalgetreu übertragen, es kann höchstens die Repräsentierung der Nachricht geändert werden (z.B. zwischen Bit- und String-Darstellung, wenn ein Transportmittel keinen Bit-Parser zur Verfügung hat).

Die Abfolge von Konversationen wird in Interaktionsprotokollen festgelegt. Interaktionsprotokolle werden mit Hilfe von AUML-Interaktionsdiagrammen spezifiziert. Diese werden im Anhang kurz vorgestellt, ein Beispiel für ein umfangreiches Diagramm ist in Abbildung 7.1 auf Seite 118 zu sehen. Hier bezeichnet der Begriff Protokoll den Ablauf der gesamten Kommunikation, während in MULAN die Steuerung innerhalb eines Agenten ebenfalls Protokoll (Protokollnetz) genannt wird.

3.4.3 Der Nachrichtentransport

Der Transportdienst (*Message Transport Service*, MTS) einer Agentenplattform wird vom Agentenkommunikationskanal (*Agent Communication Channel*, ACC) bereitgestellt. Ein solcher Kommunikationskanal kann ein oder mehrere Transportmittel (*Message Transport Protocol*, MTP) zur Verfügung haben. Jedes Transportmittel sorgt für die physikalische Informationsübertragung zwischen zwei Agentenplattformen oder zwischen dem ACC einer Plattform und den lokalen Agenten (dann ist es ein internes Transportmittel). Jedes Transportmittel benutzt genau ein Übertragungsprotokoll, das kann ein proprietäres internes Protokoll sein oder eines der in den Spezifikationen verwendeten Internet-Protokolle. Ein Transportmittel kann eine oder mehrere Repräsentierungen unterstützen.

Der ACC muss Nachrichten von einem Transportmittel empfangen und an ein geeignetes Transportmittel weiterleiten. Um festzustellen, ob ein Transportmittel für den Transport einer bestimmten Nachricht geeignet ist, muss der ACC zwei Dinge wissen: ob die Adresse für das Transportmittel erreichbar ist und ob die Repräsentierung von dem Transportmittel unterstützt wird. Ist Letzteres nicht der Fall, kann der ACC eine Konvertierung der Nachrichtendarstellung erwägen.

3.4.4 Datenstrukturen für Nachrichten

In diesem Abschnitt sollen die für diese Arbeit relevanten Datenstrukturen knapp angegeben werden: Die ersten beiden sind grundlegende Datenstrukturen für ACL und SL, die übrigen sind konkrete Strukturen für Nachrichtenelemente.

ValueTupel und KeyValueTupel

Das *ValueTupel* (`VT`) fasst mehrere Werte zu einem Tupel zusammen, das *KeyValueTupel* (`KVT`) fasst Schlüssel-Wert-Paare zu Tupeln zusammen. Das `VT` eignet sich vor

allem für mehrere Werte gleichen Typs und gleicher Bedeutung, das `KVT` eignet sich für stärker strukturierte Sammlungen von Werten. Sowohl `KVT`s als auch `VT`s müssen einen speziellen Namen (*Frame*-Bezeichner) tragen, der dann den spezielleren Typ bezeichnet (z.B. `acl-message`).

Die folgenden Relationen für den Vergleich von `KVT`s sind nicht der FIPA-Spezifikation entnommen, sondern aus der Implementierung von `KVT`s in Capa. Sie werden hier mit angegeben, weil sie zum verwendeten Datentyp `KVT` untrennbar hinzugehören. Sie gelten für den Datentyp `VT` entsprechend.

Äquivalenzrelation für KVTs: „Zwei KVTs sind gleich, wenn sie denselben Frame repräsentieren und die gleichen Schlüssel mit gleichen Werten enthalten.“ [Duv02, S. 121]

Subsumptionsrelation für KVTs: „Ein KVT-Objekt subsumiert ein anderes, wenn beide KVTs denselben Frame repräsentieren und alle Schlüssel aus dem ersten KVT im subsumierten KVT ebenfalls vertreten sind, wobei die beiden Werte zu einem Schlüssel sich ebenfalls (in derselben Richtung) subsumieren müssen.“ [Duv02, S. 121]

Die Subsumptionsrelation dient dem Vergleich zweier `KVT`s, der häufigste Fall ist der Vergleich von Nachrichtenmustern mit vollständigen Nachrichten. Das Muster enthält nur einige Felder, welche für die Bedeutung des Musters relevant sind. Wenn eine Nachricht von dem Muster subsumiert wird, hat die Nachricht mindestens die relevanten Eigenschaften.

Word und String

Der Typ `Word` wird von der FIPA bezüglich ACL und SL leicht verschieden definiert. Beide `Word`-Typen dürfen keine nicht druckbaren Sonderzeichen oder runde Klammern enthalten. In dem, was an erster Stelle erlaubt ist, unterscheiden sich die beiden `Word`-Typen: Ein ACL-`Word` darf (nach [FIPA 70, S. 3]) nicht mit einem der Zeichen `"#"`, `"-"`, `"@"` oder Ziffern beginnen. Bei einem SL-`Word` ist (laut [FIPA 08, S. 3]) das `"@"` am Beginn erlaubt, das `"?"` und `":"` sind jedoch zusätzlich verboten. Immer, wenn eines dieser Zeichen am Anfang stehen soll oder nichtdruckbare Sonderzeichen (z.B. auch das Leerzeichen) enthalten sein sollen, muss der Typ `String` verwendet werden. Ein `String` ist mit Anführungszeichen begrenzt. Wenn innerhalb eines `String`s wieder Anführungszeichen vorkommen sollen, die nicht den `String` beenden sollen, muss ihnen ein Backslash `"\"` vorangestellt werden.

Diese Definitionen zusammen sind nicht zweifelsfrei entscheidbar. So sind die Anführungszeichen in dem Typ `Word` an keiner Stelle verboten. Ein gültiges `Word`, das mit Anführungszeichen umschlossen wird, bleibt ein gültiges `Word`, wird damit aber auch zum gültigen `String`. Die Anführungszeichen gehören im ersten Fall zu den Daten hinzu, im zweiten Fall zum Datentyp!

Das macht es notwendig, bei einer Umsetzung einen pragmatischen Mittelweg zu wählen, z.B. die Anführungszeichen intern nicht im Typ Word zuzulassen.

Nachricht

Eine ACL-Nachricht ist ein spezielles `KVT` mit dem *Frame*-Bezeichner `acl-message`. Die vier Arten von Informationen wurden bereits bei der Beschreibung von ACL in der Abstrakten Architektur erwähnt: Information über die Teilnehmer der Konversation, den Nachrichteninhalt mit begleitender Information, Information zur Konversationskontrolle und die Bezeichnung der Handlung (das Performativ). Eine `acl-message` besteht aus folgenden Feldern (vgl. [FIPA 67]):

`performative`	Bezeichnung des performativen Akts, der mit dieser Nachricht ausgeführt wird.
`sender`	Dieses Feld enthält den `Agent-Identifier` des sendenden Agenten.
`receiver`	Ein oder mehrere `Agent-Identifier` von empfangenden Agenten.
`reply-to`	`Agent-Identifier`, an die eine Antwort gesendet werden soll, falls es nicht der `sender` ist.
`content`	Inhalt der Nachricht.
`language`	Inhaltssprache (z.B. `PlainText` oder `fipa-sl0`)
`encoding`	Kodierung der Inhaltssprache, sofern es Alternativen gibt (z.B. `"fipa.sl0.string.std"`).
`ontology`	Bezeichner der Ontologie(n), aus denen Begriffe im Nachrichteninhalt verwendet werden.
`protocol`	Bezeichner für das (standardisierte) Interaktionsprotokoll, zu dem die Nachricht gehört.
`conversation-id`	Eindeutige Bezeichnung, mit der die laufende Konversation referenziert wird.
`reply-with`	Ein Wert, mit dem in einer späteren Nachricht diese Nachricht referenziert werden kann.
`in-reply-to`	Der Wert aus dem `reply-with`-Feld einer früheren Nachricht, auf die diese Nachricht eine Antwort ist.
`reply-by`	Zeitangabe, bis wann eine Antwort auf diese Nachricht eintreffen muss.

In der Nachricht wird auf verschiedenen Ebenen referenziert: Das Feld `receiver` referenziert einen Agenten, das Feld `conversation-id` referenziert innerhalb eines Agenten die Konversation, und mit Hilfe der Felder `reply-with` und `in-reply-to` kann eine spezielle Nachricht referenziert werden.

Umschlag

Der Nachrichtenumschlag bildet die zentrale Datenstruktur für den Nachrichtentransport (ein `KVT` mit dem *Frame*-Bezeichner `envelope`). Die Bedeutung der Felder im Umschlag wird im Dokument [FIPA 67] definiert, die konkrete Darstellung in XML im Dokument [FIPA 85]; die beiden letzten Felder dieser Tabelle sind nur in [FIPA 85] aufgeführt.

`to`	Ein oder mehrere `Agent-Identifier`, an die diese Nachricht ausgeliefert werden soll. Der Inhalt sollte mit dem Inhalt des Feldes `receiver` aus der Nachricht übereinstimmen, es sei denn, es handelt sich um eine weitergeleitete Nachricht.
`from`	Ein `Agent-Identifier`, der mit dem Wert des Feldes `sender` aus der Nachricht übereinstimmen sollte, es sei denn, es handelt sich um eine Weiterleitung. Ansonsten ist dies die Adresse für Fehler- und Bestätigungsnachrichten.
`comments`	Dieses Feld hat keine Semantik bei der Auslieferung von Nachrichten.
`acl-representation`	Repräsentierung der eigentlichen Nachricht. Z.B. `"fipa.acl.rep.string.std"`.
`payload-length`	Diese Information ist zum effizienten Parsen nützlich, insbesondere bei der bitweisen Repräsentierung der Nachricht.
`payload-encoding`	Die Kodierung des Nachrichteninhaltes.
`date`	Absende-Datum der Nachricht.
`intended-receiver`	In diesem Feld ist je eine Gruppe von `Agent-Identifier` aus dem Feld `to` mit gleicher Internet-Adresse enthalten.
`received`	Jeder ACC, der die Nachricht befördert, fügt dem Umschlag ein Feld `received` hinzu. Es enthält mindestens einen Zeitstempel sowie die Adresse des empfangenden und sendenden ACCs. Optional ist eine für den sendenden ACC eindeutige Nachrichten-ID sowie die Art des benutzten Transportmittels.
`transport-behaviour`	Für spätere Zwecke reserviert.
`user-defined`	An dieser Stelle ist der Umschlag für spezielle Bedürfnisse erweiterbar.
`encrypted`	Für den Fall, dass die Nachricht verschlüsselt ist, kann in diesem Feld z.B. ein öffentlicher Schlüssel transportiert werden.

Plattformbeschreibung

Die Plattformbeschreibung (ein `KVT` mit dem *Frame*-Bezeichner `ap-description`) enthält den Namen der Plattform und die Beschreibung der verfügbaren Plattformdienste, zu denen z.B. die verfügbaren Transportmittel einer Plattform zählen. Jede Transportmittelbeschreibung enthält einen Namen, eine Typbezeichnung (z.B. `"fipa.mtp.http.std"`) und eine oder mehrere Adressen, unter denen das Transportmittel erreichbar ist. Vgl. [FIPA 23, S. 15f].

Agentenidentifikation

Der `Agent-Identifier` ist eine zentrale Datenstruktur zur Identifizierung von Agenten (ein `KVT` mit dem *Frame*-Bezeichner `agent-identifier`). Er enthält mindestens den Namen des Agenten, der sich unter keinen Umständen ändern sollte. Die optionalen Felder zu Adressen und Namensauflösung können dagegen in zwei `Agent-Identifiern` desselben Agenten verschieden gefüllt sein, z.B. vor und nach einer Migration des Agenten. Damit ist auf dreifache Weise die Erreichbarkeit eines Agenten unterstützt, jedes der Felder kann benutzt werden, um Informationen zu einem Agenten zu erhalten: Anhand des Namens ist ein Agent innerhalb einer Plattform erreichbar. Die Adressen können direkt zur Nachrichtenzustellung benutzt werden, und die Namensauflösungsdienste können die Adresse anhand des Namens ermitteln (vgl. [FIPA 23]).

`name`	Der symbolische Name des Agenten
`addresses`	Eine geordnete Folge von Adressen unter denen der Agent erreichbar ist. Die Reihenfolge gibt die Präferenz an.
`resolver`	Eine Reihe von `Agent-Identifiern`, unter denen der Name des Agenten aufgelöst werden kann. Die Reihenfolge gibt die Präferenz an.

Datums- und Zeitangaben

Ein Zeitpunkt wird in einer lesbaren Zahlenreihe dargestellt, z.B. wird der 28.7.2003 um 13:56 dargestellt als `20030728T135600000`. Für die Darstellung der Zeitzone UTC wird ein `Z` angehängt und für die Darstellung von Zeitspannen wird ein `+` oder `-` vorangestellt. Vgl. [FIPA 70, Seite 4].

4 Agentcities

Agentcities ist eine internationale Initiative mit dem Ziel, das kommerzielle und wissenschaftliche Potential von agentenbasierten Anwendungen zu erforschen und zu realisieren (vertreten im Internet unter [acs03b]). Die Forschung wird in einem internationalen Forschungsnetz durchgeführt und durch ein weltweit offenes Netz von Plattformen mit diversen agentenbasierten Diensten unterstützt. Diese sollen tatsächliche persönliche und geschäftliche Aufgaben erfüllen können. Einfache Dienste sollen von intelligenten Agenten auf dynamische und autonome Weise zu komplexeren Diensten komponiert werden. Zur Standardisierung der Kommunikation werden die Spezifikationen der FIPA verwendet (wie in Abschnitt 3.4 (*Standardisierung nach FIPA*) erläutert).

In den folgenden Abschnitten wird zunächst die Organisation Agentcities in ihrem Umfeld und mit ihren Teilprojekten beschrieben (also das Forschungsnetz), danach der von Agentcities ausgeschriebene Wettbewerb und zuletzt das technische Netz von Agenten. Im letzten Abschnitt dieses Kapitels werden die wesentlichen Elemente des Agentennetzes in ihren Zusammenhang eingeordnet.

4.1 Organisation mit Umfeld

Agentcities ist in ein großes Umfeld von Projekten und Organisationen zum Thema Agententechnologie eingebettet. *Agentcities* und *AgentLink* stehen als Schwesterprojekte nebeneinander. Die Teilprojekte von Agentcities sind: *Agentcities.NET* für Austausch und Vernetzung von Forschungsgruppen, *Agentcities.RTD* für die technische Vernetzung von Agentenplattformen, sowie eine ganze Reihe von nationalen Projekten. Die *ACTF* (*Agentcities Task Force*) ist eine Organisation zur übergreifenden Koordination der Teilprojekte und beteiligten Personen in Agentcities. Agentcities.NET, Agentcities.RTD und AgentLink werden als Projekte innerhalb des europäischen Forschungsschwerpunktes *IST* gefördert.

In den nächsten Unterabschnitten werden die fünf genannten Projekte vorgestellt: IST und das Schwesterprojekt AgentLink aus dem Umfeld von Agentcities sowie die Teilprojekte Agentcities.RTD, Agentcities.NET und die Agentcities Task Force.

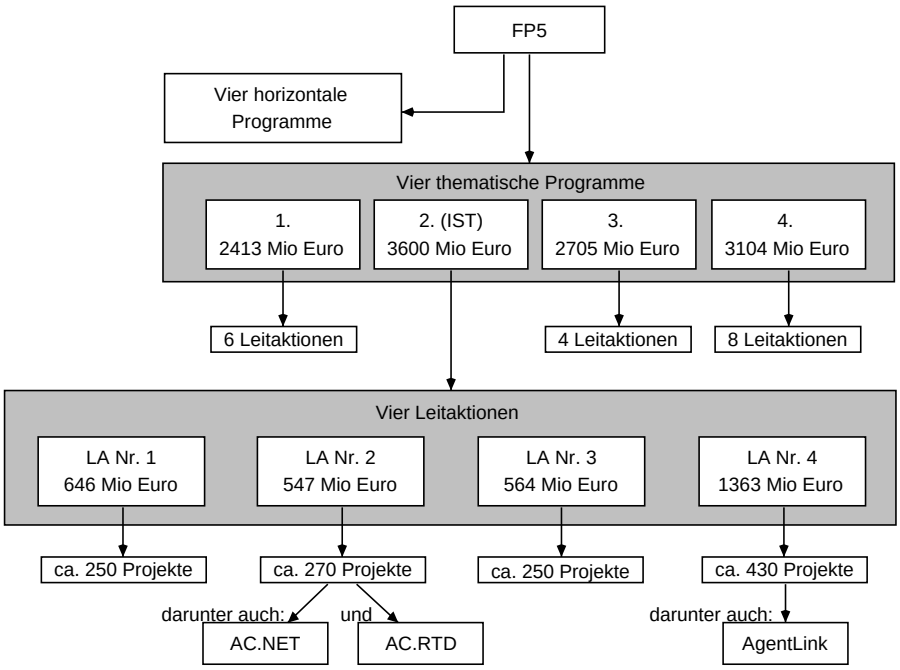


Abbildung 4.1: Die Projekte im FP5.

Information Society Technology (IST)

Die europäischen Forschungsaktivitäten sind in Rahmenprogrammen von je vier oder fünf Jahren Laufzeit organisiert, den *Framework Programmes*. Das fünfte Rahmenprogramm („FP5“) legt die Prioritäten für Forschung, technologische Entwicklung und Darstellung („RTD“) in Europa für die Jahre 1998 bis 2002 fest. Daraus ergibt sich der zeitliche Rahmen der hier vorgestellten Projekte. Agentcities und AgentLink bemühen sich um die Teilnahme am sechsten Rahmenprogramm der EU. Das sechste Rahmenprogramm („FP6“) legt die Forschungsprioritäten für den Zeitraum 2003 bis 2006 fest. Im fünften Rahmenprogramm gibt es thematische und horizontale Programme. Die horizontalen Programme betreffen jeweils alle thematischen Programme. Die vier thematischen Programme oder Schwerpunkte (siehe Abbildung 4.1, oberer Kasten) sind:

1. Lebensqualität und Management lebender Ressourcen
2. Benutzerfreundliche Informationsgesellschaft (IST)
3. Wettbewerbsorientiertes und nachhaltiges Wachstum
4. Energie, Umwelt und nachhaltige Entwicklung

In jedem der Schwerpunkte werden mehrere Leitaktionen verfolgt und in jeder Leitaktion werden einzelne Projekte gefördert. Die vier Leitaktionen der IST werden

hier mit ihrer Bezeichnung aufgeführt:

LA Nr. 1 Systeme und Dienste für den Bürger
LA Nr. 2 Neue Arbeitsmethoden und elektronischer Geschäftsverkehr
LA Nr. 3 Multimedia-Inhalte und -Werkzeuge
LA Nr. 4 Grundlegende Technologien und Infrastrukturen

Im Internet werden die thematischen und horizontalen Schwerpunkte des fünften Rahmenprogramms und alle (insgesamt 23) Leitaktionen anschaulich vorgestellt (siehe [eur03]), die Informationen über die geförderten Projekte sind auf einer separaten Datenbank-Webseite zugänglich (siehe [cor03a]). Jedes Projekt hat eine offizielle Projektseite im Internet (Agentcities.RTD: [acr03a]; Agentcities.NET: [acn03] und AgentLink: [agl03]).

Agentcities.RTD

RTD steht in Anlehnung an die Zielsetzung des FP5 für ***R**esearch, **T**echnological Development and **D**emonstration*. Alle Teilziele der Agentcities.RTD basieren auf der Realisierung einer zentralen Plattform für agentenbasierte Dienste als Forschungsumgebung.

Vierzehn Agentenplattformen haben das Ziel, von Projektbeginn an interessante und nützliche Beispieldienste anzubieten und so das anfängliche Netz zu bilden. Die zentrale Plattform ist offen für Anmeldungen von weiteren Agentenplattformen. Die angebotenen Dienste der vierzehn Starter-Plattformen und der weiteren angemeldeten Plattformen sind während der Projektlaufzeit öffentlich sichtbar. Damit entsteht eine offene, verteilte, heterogene und dynamische Umgebung für agentenbasierte Dienste, die als Agentennetz im Internet öffentlich zugänglich ist und damit auch eine fortlaufende Demonstration des Entwicklungsstands im Projekt darstellt.

Neu entwickelten Diensten und Plattformen dient das Agentennetz als Referenzumgebung, an der die Standards für Kommunikation und Darstellung getestet werden können. Insbesondere sind Versuche mit kommerziellen Diensten willkommen, da sie besonders gut mit der übergreifenden Leitaktion „Neue Arbeitsmethoden und elektronischer Geschäftsverkehr“ übereinstimmen. Die Agenten sollen über ihre Dienste verhandeln und so komponierte Dienste von erhöhtem Wert schaffen.

Drei weitere Aspekte der Modellierung von Diensten in offenen heterogenen Anwendungen können untersucht werden: offene Kommunikation, Dienstermittlung und Stabilität. Der Punkt *offene Kommunikation* betrifft die Beschreibung von Interaktion auf einer hohen Ebene, also den Entwurf und die Formulierung von Interaktionsprotokollen, wie sie bereits von der FIPA für einige allgemeine Fälle spezifiziert wurden. Der Punkt *Dienstermittlung* betrifft die Beschreibung von Dienstkomponenten und allgemein die Modellierung von Diensten in einer offenen, heterogenen, dynamischen und verteilten Umgebung. Der letzte Punkt, *Stabilität*, betrifft die Frage, wie ein Dienst von entfernten Bestandteilen abhängt.

Die Forschung am Agentennetz soll gewisse konkrete Resultate erbringen, die hier kurz zusammengefasst sind:

- Eine praxistaugliche Agentennetz-Architektur. Ziel ist ein offenes, skalierbares, stabiles und zuverlässiges Agentennetz, das es Agenten ermöglicht, einander zu finden, miteinander zu kommunizieren und einander Dienste anzubieten (siehe [act02a]).
- Dienste mit erhöhtem Wert durch Komposition. Idealerweise entstehen diese dynamisch durch Verhandlungen unter den Agenten.
- Methoden, Prototypen und Modelle
  - für die Integration von kommerziellen Diensten,
  - für offene Kommunikation, d.h. Formulierung von Inhalt und Protokollen und
  - für die Dienstermittlung, d.h. semantische Modelle und konkrete Ontologien.

Das Agentennetz dient außerdem als gemeinsame Ressource für Entwickler, die ihre Dienste und Agentensysteme verbinden wollen. Es bietet einerseits einen Maßstab für die Erfüllung von Standards, andererseits kann die FIPA Erfahrungen mit der Umsetzung ihrer Spezifikationen sammeln und dazu Rückmeldung bekommen.

Das von Agentcities.RTD geschaffene Netz ist durch zentrale Verzeichnisse im Internet zugänglich (siehe [acs03a]).

Agentcities.NET

Das Projekt Agentcities.NET unterstützt weltweit Forschungs- und Industriegruppen, die an der Entwicklung des Agentennetzes von Agentcities.RTD teilhaben wollen (Informationen von der Webseite [acn03]). Agentcities.NET ermöglicht außerdem Technlogietransfer, indem Schlüsselfiguren der angrenzenden Industrie (also potentielle Nutzer der Agententechnologie: Internetprogrammierer, Personen aus dem eCommerce und aus der Systemintegration) mit Forschern und Entwicklern in Kontakt kommen. Es unterstützt die Realisierung und weite Ausdehnung des Agentennetzes als eine offene dynamische Umgebung für agentenbasierte Dienste.

Die konkret durchgeführten Aktionen von Agentcities.NET sind hier aufgeführt:

- Drei Informationstage (*Information days*, „iD1“ bis „iD3“) sollen einen schnellen und einfachen Zugang zu den Entwicklungen in dem Agentennetz geben. Diese Informationstage fanden in Form von zweitägigen öffentlichen, kostenlosen Workshops in Lausanne, Lissabon und Barcelona statt. Zum Projektende folgte ein „iD4“ im Oktober 2003.
- Im Rahmen des *deployment support program* werden 30 mit Agentcities verwandte Mini-Projekte finanziell unterstützt.
- Mit einem auf dem iD3 ausgeschriebenen Wettbewerb mit einer gleichzeitig stattfindenden technologischen Ausstellung wurde die Dienstentwicklung gefördert und die Aufmerksamkeit auf die Agententechnologie gelenkt.

- Drei Sorten von Arbeitsgruppen bieten Diskussions- und Feedback-Kanäle für Nutzer, Entwickler, Anbieter und Beobachter: Technisch bezogene Arbeitsgruppen, anwendungsbezogene Arbeitsgruppen und Arbeitsgruppen, die sich mit der Infrastruktur des Agentennetzes befassen.
- Um die Zusammenarbeit in Projekten zu erleichtern und die Testumgebung verstärkt als Lernumgebung zu nutzen, unterstützt Agentcities.NET studentischen Austausch, indem z.B. Reisekosten getragen werden.

Zusammenfassend ist das Ziel die Vernetzung von Forschern, Entwicklern, Nutzern, Förderern und anderen Interessenten der Agententechnologie. Dazu wird das Agentennetz diesen Personen als Testumgebung mit der verwendeten Technologie verfügbar gemacht und als Ort der Zusammenarbeit für Forschung und Entwicklung gestaltet.

Agentcities Task Force

Die verschiedenen Projekte und Personen, die an Agentcities partizipieren, arbeiten zum Teil im Rahmen der Agentcities Task Force (ACTF) zusammen. Die ACTF ist eine internationale, unabhängige und gemeinnützige Körperschaft. Ihre Aufgaben sind:

- Die *Koordination* zwischen den Projekten und Aktivitäten von Agentcities,
- die *Veröffentlichung* gemeinsamer Ressourcen wie z.B. Verzeichnisdienste und Ontologien und
- *Werbung* für die Arbeiten im Rahmen von Agentcities,
  - zum einen, um existierende Standards zu unterstützen und
  - zum anderen, um ein allgemein vermehrtes Interesse am Agentenbereich zu wecken und um die Teilnahme und weitere Entwicklungen in diesem Bereich zu ermutigen.

Die Abbildung 4.2 zeigt Agentcities noch einmal im Überblick mit den Teilprojekten, deren Tätigkeiten und Verbindungen.

AgentLink

AgentLink ist wie Agentcities.NET und Agentcities.RTD ein Projekt der IST. Einige Forschungsgruppen und Universitäten sind sowohl in Agentcities als auch in AgentLink vertreten. Auch in ihrer Zielsetzung sind Agentcities und AgentLink verwandte Projekte. Dabei wird in AgentLink der Schwerpunkt noch stärker auf Vernetzung von Forschern gelegt und es findet im Gegensatz zu Agentcities keine technische Entwicklung statt. Der vollständige Name von AgentLink lautet *European Network of*

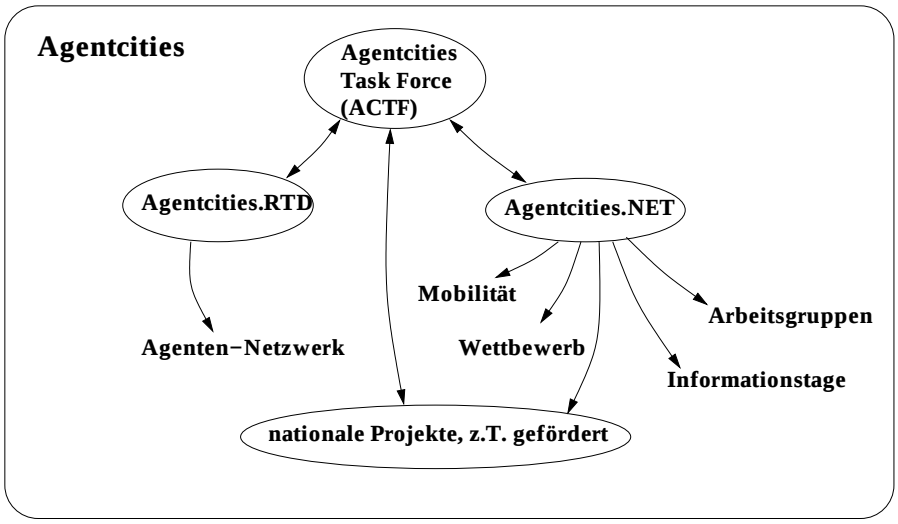


Abbildung 4.2: Die Tätigkeitsfelder von Agentcities

Excellence for Agent Based Computing. Die Tätigkeitsfelder von AgentLink werden in den folgenden Absätzen erläutert und in Abbildung 4.3 dargestellt.

Durch die **Interessengruppen** sollen Gemeinschaften von ähnlich interessierten Forschern gebildet werden. Jede Interessengruppe (*Special Interest Group*, kurz *SIG*) veröffentlicht im Laufe des Projekts eine *Roadmap*, also einen Überblick über die für das Gruppenthema relevanten Technologien und Entwicklungen. Es gibt Gruppen mit industriellem Schwerpunkt und solche mit Forschungsinteressen. Zur Zeit gibt es sieben aktive Interessengruppen, z.B. *MSEAS* (*Methodologies and Software Engineering for Agent Systems*) und *ABSS* (*Agent-Based Social Simulation*). Eine Übersicht über alle Gruppen findet sich auf der Webseite von AgentLink (siehe [agl03, unter /activities/sigs]).

Bei den **industriellen Aktivitäten** geht es um die Wahrnehmung agentenbasierter Technologie in der Wirtschaft; um Technologietransfer, um Standards und um die Vermittlung von Problemstellungen aus der Wirtschaft an die Forschungsgemeinde. AgentLink unterstützt die Bildung von Standards, indem den Mitgliedern aktuelle Informationen über Standards zur Verfügung gestellt werden und es ihnen ermöglicht wird, an Standardisierungsinitiativen teilzunehmen und Rückmeldungen zu den Standards zu geben. Der Technologie- und Wissenstransfer wird durch die weiter unten beschriebenen Datenbanken unterstützt.

Die Kommunikation zwischen Forschern und Personen aus der Industrie wird einerseits in den beschriebenen Interessengruppen gepflegt und andererseits durch Tagungen unterstützt, so gab es z.B. Anfang Februar 2003 eine eintägige Konferenz (die *Agent Technology Conference*) zum Thema *Agents for Commercial Applications* in Zusammenarbeit mit Agentcities.

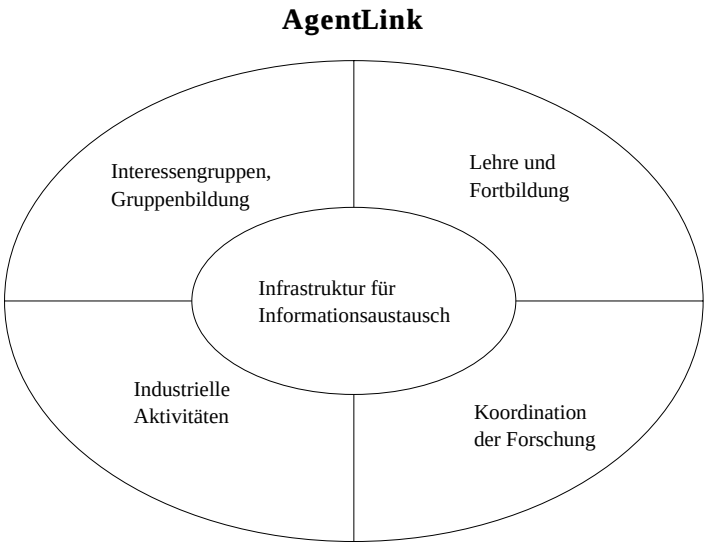


Abbildung 4.3: Die Tätigkeitsfelder von AgentLink

Zur **Koordination der Forschung** gibt es einerseits die oben erwähnten Interessengruppen und andererseits eine Reihe von Datenbanken, die eine Infrastruktur zur Veröffentlichung von Ergebnissen, Datensammlungen und Software bilden:

- *Agent Research Paper Clearing House*: Per Internet-Formular können Papiere mit einem Link und den bibliographischen Daten in eine thematische Gliederung eingeordnet werden.
- *Curricula Database*: Hier sind nach Ländern und Universitäten geordnete Einträge im Stil von Vorlesungsverzeichnis-Einträgen enthalten.
- *Agent-based Projects*: Dies ist eine einfache Liste mit Projekten. Jedes Projekt hat ein Akronym, eine kurze Beschreibung und einen Link zu einer Projektseite im Internet.
- *Member Research Profiles*: Pro Mitgliedsuniversität gibt es ein Datenblatt mit Informationen, z.B. zu Forschungsinteressen, zu Projekten und Kontaktinformation.
- *Agent Educational Software*: Dies ist eine Sammlung von einigen Links mit Beschreibungen zu Agentenplattformen, Sprachimplementationen und anderer Software aus dem Agenten-Umfeld.
- *Agents in the Press*: Diese Datenbank enthält Datum, Titel und Herausgeber von z.Zt. 50 Zeitungsartikeln und Nachrichten rund um Agenten. Auf der Webseite ist sies nicht unter dem Punkt *Databases*, sondern unter *Publications* eingeordnet.

Zur Förderung von **Lehre und Fortbildung** in der Agententechnologie gibt es die oben vorgestellte Datenbank *Curricula Database* und eine jährliche Schulung

zum Thema Agenten. Mit Hilfe der Datenbank können Lehrende und Lernende sich orientieren. Besonders wichtig für Lehre und Fortbildung ist jedoch die einwöchige Schulung, die seit 1999 jedes Jahr im Sommer oder Frühling stattfindet (*European Agent Systems Summer (Spring) School*, kurz „EASSS“)[1].

Das zentrale Ziel von AgentLink ist es, eine **Infrastruktur für Informationsaustausch** im Agentenbereich aufzubauen (auch im Bild zentral dargestellt). Gewissermaßen können die vier bisher genannten Punkte unter diesem zusammengefasst werden. Die Webseite von AgentLink ([agl03]) ist der zentrale Zugangspunkt für alle Informationen, insbesondere zu den vorgestellten Datenbanken. Weiter werden in einer moderierten und editierten Liste per E-mail aktuelle Informationen verschickt, wie z.B. ein Hinweis auf eine Datenbank, die durch vermehrte Nutzung größeren Wert für alle Beteiligten bekommt. AgentLink veröffentlicht regelmäßig einen Newsletter mit Fachartikeln, Buchbesprechungen und anderen Inhalten. Der Newsletter ist online in Form von pdf-Dateien verfügbar (siehe [agl03, unter /newsletter]).

4.2 Wettbewerb

Agentcities hat für den iD3 einen Wettbewerb zur Agententechnologie ausgeschrieben. Es wurden zwei Kategorien ausgeschrieben: In der Kategorie *Infrastruktur* waren unterstützende Konzepte oder Software gefragt[2], und in der Kategorie *Dienste und Anwendungen* war gefordert, dass eine vorgestellte Anwendung ihr Ziel außerhalb des Netzbetriebes haben und nicht die Agentenentwicklung an sich unterstützen soll. Unter den Bewertungskriterien war die Relevanz für offene Systeme und für Agentcities zentral.

Die Voraussetzungen für die Teilnahme waren möglichst niedrig gehalten, um viele Bewerber in dem noch jungen Feld zu ermutigen. So war die Anbindung an Agentcities oder die Kompatibilität mit den FIPA-Spezifikationen keine Voraussetzung zur Teilnahme, außereuropäische Projekte waren ebenso willkommen wie kommerzielle, kleine oder studentische Projekte wurden möglichst gleichgewichtig neben größere und finanziell geförderte Projekte gestellt, und es wurde eine Anzahl von großzügigen *Travel Grants* vergeben. Die einzige strenge Voraussetzung für eine Siegerehrung war die Präsentation der Bewerbung in der Finalrunde in Barcelona im Rahmen des iD3.

[1] Auf der EASSS 2003 waren die Vortragenden bekannte Personen aus dem Agenten-Umfeld. In der Regel gab es zwei vierstündige Vorträge an einem Tag. Die Themen reichten von allgemeinen Konzepten („Was ist ein Agent?“ in [WL03]) über konkrete Anwendungen („Intelligente Informations-Agenten“ in [Klu03]) bis zu tieferem theoretischen Wissen („Ein Kurs in Reinforcement Learning“ in [Wie03]).

[2] also Agentenlaufzeitumgebungen, Entwicklungswerkzeuge, Codegeneratoren, Verzeichnis- oder Namensdienste, Komponenten für Nachrichtenvermittlung oder Unterstützung für Ontologien oder Kommunikationssprachen

In jeder Kategorie wurden Preise für die besten drei Bewerber ausgeschrieben, zusätzlich gab es den *special prize* für die beste, tatsächlich in Agentcities verfügbare Bewerbung. Nachträglich wurden ein Preis für das beste studentische Projekt und ein *publicity*-Preis (vom Publikum vergeben) ausgeschrieben.

Der Wettbewerb hatte vier Phasen: erste Bewerbung, ausführliche Beschreibung des Beitrags, Testphase für den *special prize* und Präsentation im Rahmen der Ausstellung auf dem iD3. Die ausführliche Beschreibung des Beitrags war das wichtigste Dokument für den Wettbewerb, es ist im Anhang A beigefügt. Während der Testphase versuchte das Auswahlgremium, über Agentcities auf die für den *special prize* angemeldeten Systeme zuzugreifen. Auf der Ausstellung bewegten sich die Juroren erkennbar zwischen den Besuchern und ließen sich die Beiträge präsentieren.

Zum Wettbewerb meldeten sich 54 Interessenten, von denen 36 an der zweiten Phase teilnahmen. Dreizehn Finalisten wurden zur Ausstellung im Rahmen des iD3 eingeladen, wo die Schluss-Auswertung und Siegerehrung stattfand. Siedler erreichte die Finalrunde des Wettbewerbs, dazu wird im Kapitel 7 mehr berichtet. Die Preisträger des Wettbewerbs sind hier kurz charakterisiert.

WSDLTool WSDL steht für *Web Service Description Language* und ist als Standard zur Beschreibung von WebServices recht verbreitet. Die Eingabe für dieses Werkzeug ist eine WSDL-Beschreibungsdatei eines WebServices. Es kann dann vier verschiedene Ausgaben erzeugen: Eine Projektdatei für Protégé (Protégé ist ein frei verfügbarer Ontologie-Editor und ein Werkzeug für die Entwicklung von wissensbasierten Anwendungen, siehe [CFK+03]), ein RDF-Schema (das *Resource Description Framework* ist eine universale Sprache zur Darstellung von Wissen im Internet, siehe [BG03]) sowie Ontologie-Code und Agenten-Code für JADE in Java (JADE [BRP+03] ist eine frei verfügbare Agentenplattform, siehe Abschnitt 4.4 (*Einordnung*)). Das WSDLTool ist im Internet benutzbar (siehe [wsd03]).

X-Security X-Security ist eine Anwendung, um Agentennachrichten zu signieren und zu verschlüsseln. Sie besteht aus einer Zertifikat-Autoritätsstelle und aus den Agenten, die diese nutzen. Die Zertifikate und Schlüssel werden von der Autoritätsstelle generiert und ausgegeben. Die Agenten müssen dann so programmiert sein, dass sie die Zertifikate nutzen. Sowohl die Zentrale als auch eine Erweiterung für JADE-Agenten sind zum Download freigegeben (siehe [xse03]). Eine Zertifikat-Autoritätsstelle läuft auf der Plattform `prague.agentcities.net`. Eigene Autoritätsstellen und Zertifikate können dort registriert und abgefragt werden.

Taga Taga ist eine Reisemarkt-Simulation, in der Reisebuchungs-Agenten im Spiel gegeneinander antreten. Die Reisebuchungs-Agenten können Flüge und Hotelzimmer buchen, die sie auf einer Auktion ersteigern. Die reisewilligen Agenten kommunizieren mit den Reisebuchungs-Agenten über ein schwarzes Brett. Insgesamt gibt es

sechs Sorten von Agenten: die schon erwähnten Reisebuchungs-Agenten, die reisewilligen Agenten, den Agent für das schwarze Brett und weiter einen Auktions-Agenten, Anbieter-Agenten und einen zentralen Kontroll-Agenten („Bank“), der die Transaktionen des Auktions-Agenten und der reisewilligen Agenten protokolliert.

Die Plattform `taga.agentcities.net` ist online und enthält die beschriebenen Agenten (siehe auch [FZD+03]).

Ires Ires [ire03] ist ein Restaurant-Informations-Dienst. Es gibt drei Arten von Agenten: Restaurant-Server, persönliche Agenten und *Personal Facilitator*-Agenten. Die Restaurant-Server sammeln Informationen über Restaurants in einem bestimmten Gebiet, z.B. einer Stadt, persönliche Agenten haben Präferenzen, „Freunde“ (das sind andere persönliche Agenten) und geben Tipps über Restaurants an Interessenten. *Personal Facilitator*-Agenten suchen über einen `DF` Kontakte zwischen persönlichen Agenten. Es wurde eine Ontologie für Restaurantdienste erstellt und für die Tipps und die „Freundschaft“ wurde Trust-Technologie verwendet.

iBundler iBundler bietet verhandelnden Agenten Entscheidungshilfe. Das Verhandlungsszenario enthält vielfache Anbieter, die konfigurierbare Ware zu unterschiedlichen Bedingungen anbieten wie etwa Mengenrabatt oder Bündel. Juan Rodriguez schreibt in seinem Abstract (siehe [Rod03]), der Dienst sei unter Agentcities zugänglich, gibt jedoch keine konkrete Referenz an, so dass diese Angabe nicht prüfbar ist.

Consorts Consorts soll den Zugang zu intelligenten Informationsdiensten erleichtern. Der Ansatz will Möglichkeiten und Techniken des *Ubiquitous Computing* mit denen des *Semantic Web* verbinden. Konkret wurde ein Museums-Szenario implementiert, in dem ein Besucher-Agent, der sich einem Bild nähert (durch Sensoren für Raum und Zeit aus dem *Ubiquitous Computing* ausgelöst), zunächst Basis-Informationen zu dem Bild auf einem Display erhält. Fordert er weitere Informationen an, wird automatisch z.B. eine Google-Suche gestartet (also Methoden aus *Semantic Web* genutzt). Auf der Plattform `tokyo.agentcities.net` findet sich ein „GoogleAgent“; auf der Webseite (siehe [KSI+03]) findet sich ein Demo-Programm, das Besucher in einem Museum simuliert.

Orcas Orcas [Gom03] soll aus einer Menge von angebotenen Diensten und einer komplexeren Problembeschreibung ein Multiagentensystem (ein „Team“ von Agenten) bilden, die dieses Problem lösen. Es gibt Anbieter-Agenten, die einzelne Dienste anbieten, Nachfrage-Agenten, die Probleme formulieren, und es gibt vermittelnde Agenten. Zu den vermittelnden Agenten gehören Bibliothekare, die über die Fähigkeiten (*Capabilities*) der Anbieter Bescheid wissen, weiter gibt es *Capability-*

Brokers, die für eine Problembeschreibung die passenden Fähigkeiten zusammenstellen und *Team-Brokers*, die zu den nötigen Fähigkeiten die passenden Anbieter finden und diese durch Instruktion zu einem Team verbinden.

Grusma Grusma ist der Entwurf eines umfassenden medizinischen Informationsdienstes. Innerhalb des Systems kann ein Benutzer seine medizinischen Berichte aus einer zentralen Datenbank einsehen, Informationen über Kliniken und Praxen eines Gebietes einholen, oder konkrete Termine mit Ärzten vereinbaren. Der Arzt kann während einer Untersuchung oder Behandlung die medizinischen Berichte des Patienten einsehen und vervollständigen oder aktualisieren. Es gibt auch ein Konzept zur Sicherung der zentralen Daten mittels Verschlüsselung, Authentisierung und festen Kommunikationskanälen.

Das System besteht aus dem Datenbank-Wrapper, dem persönlichen Broker mit dem äußeren Zugriff, der z.B. über einen persönlichen Agenten genutzt werden kann. Weiter gibt es für jeden Arzt, jede Station und jedes Krankenhaus je einen Agenten, die in dieser Hierarchie miteinander kommunizieren (siehe [Mor03]).

4.3 Agentennetz

Nach diesem Weg über das Forschungsnetz der IST, AgentLink und den Wettbewerb soll nun das Agentennetz von Agentcities vorgestellt werden. Es besteht aus Agentenplattformen mit Agenten, welche Dienste anbieten. Die angebotenen Dienste sind entweder für *Personen* oder für *Agenten* benutzbar oder für beides geeignet. Das Agentennetz besteht (folgend der Dokumentation [act02a]) aus den folgenden wesentlichen Elementen:

- eine Domäne (`agentcities.net`);
- eine Sammlung von Forschungsgruppen;
- eine Sammlung von Agentenplattformen in der Domäne, von denen jede einer Forschungsgruppe zugeordnet ist;
- eine Sammlung von Agenten, von denen jeder einzelne auf einer Plattform läuft;
- eine Sammlung von Diensten, von denen jeder von einem oder mehreren Agenten im Netz angeboten wird. Ein Teil der Dienste ist zum *technischen* Aufbau des Agentennetzes selbst notwendig, der andere Teil der Dienste dagegen gehören zum *inhaltlichen* Aspekt des Agentennetzes.
- ein zentraler Knoten, der Verzeichnisdienste für das gesamte Netz anbietet.

Die *Forschungsgruppen* repräsentieren Gruppen an Universitäten und anderen Organisationen mit Interesse an Agentcities. Eine Forschungsgruppe kann ihre Plattformen verwalten, die Zuordnung von Plattformen zu Forschungsgruppen hat technisch

keine weitergehende Bedeutung. Eine Forschungsgruppe kann ihren Forschungsschwerpunkt beschreiben und Kontaktinformationen und Links bekannt geben.

Eine *Plattform* kann verschiedene Zustände im Netz haben:

- Mit einem Namen allein kann eine Plattform bereits *angemeldet* sein.
- Sie ist *aktiv*, wenn sie regelmäßig eine einfache Anfrage des Verzeichnisdienstes beantwortet. Diese Anfrage (Ping) hat im Agentcities-Agentennetz eine zentrale Bedeutung,
- ...weil der aktive Zustand die Voraussetzung für die weitere Annbindung der Agenten und der Dienste der Plattform ist, bei der die Agenten und Dienste zentral angezeigt werden (*volle Anbindung*).

Unabhängig von den bisher genannten Zuständen (angemeldet, aktiv, voll angebunden sowie Zwischenstadien) kann eine Plattform als *bereit* (`ready`) markiert sein, Nachrichten von dritten zu empfangen.

Der Zustand einer Plattform ist eng mit ihrer Repräsentierung in den zentralen Verzeichnissen verwoben. Die Verzeichnisse sind im Abschnitt *Verzeichnisdienste* auf Seite 59 genauer erläutert. Die technischen Dienste sind im Abschnitt 4.3.2 erläutert, darunter fallen auch die Verzeichnisdienste. Einige inhaltliche Dienste wurden im Abschnitt 4.2 (*Wettbewerb*) bereits vorgestellt.

4.3.1 Standards

Um eine Agentenplattform an Agentcities anzubinden, müssen die in Agentcities verwendeten Standards in einem gewissen minimalen Umfang implementiert werden. Um eine Plattform voll einzubinden, sind die Spezifikationen der FIPA maßgeblich, wie sie im Abschnitt 3.4 vorgestellt wurden. Eine solche Plattform muss einen `AMS` und einen `DF` zur Verfügung stellen. Die FIPA-Agent-Management-Ontologie stellt die einheitliche Interpretation der Begriffe und Anweisungen sicher. Die Plattform muss Nachrichten in der Agentenkommunikationssprache ACL empfangen und versenden können, die sie in der String-Darstellung verarbeiten muss. Nachrichten werden mit dem zentralen Knoten von Agentcities ausschließlich über das HTTP-Transportmittel (HTTP-MTP) versendet und empfangen. Die Felder des Nachrichtenumschlags müssen richtig interpretiert werden. Der Nachrichtenumschlag muss in XML dargestellt sein. In der folgenden Liste sind die für eine vollständige Anbindung an Agentcities notwendigen FIPA-Spezifikationen zusammengefasst (der letzte Punkt ist in der von Agentcities in [act02a] angegebenen Liste nicht enthalten):

- **Agent Management H** [FIPA 23]: Referenzmodell, `AMS`, `DF`, Teile der Basisontologie
- **ACL Message Structure E** [FIPA 61]: Nachrichtenstruktur (Bedeutung der Felder einer Nachricht), Teile der Basisontologie.

- **ACL Message Representation in String G** [FIPA 70]: Syntax (Repräsentierung) von Nachrichten.
- **Agent Message Transport Service D** [FIPA 67]: Bedeutung und Handhabung des Nachrichtenumschlags, Teile der Basisontologie.
- **Agent Message Transport Envelope Representation in XML H** [FIPA 85]: Syntax des Nachrichtenumschlags.
- **Agent Message Transport Protocol for HTTP D** [FIPA 84]: Transportmittel des zentralen Knotens.
- **SL Content Language** [FIPA 08]: Definition der Inhaltssprache für `AMS` und `DF`.

Eine Plattform muss diese Standards implementieren, um *voll* an Agentcities angebunden zu werden. Eine Plattform kann auch nur *angemeldet* sein, dann braucht allerdings überhaupt keine Implementation dahinter zu stehen. Dazwischen sind Übergangsstadien von Standardkonformität möglich.

Die FIPA-Spezifikationen sind jung und werden in Agentcities getestet, implementiert und referenziert, wodurch die FIPA Rückmeldung zu ihrer Arbeit erhält. Indem neue Agentenplattformen sich an die Gegebenheiten in Agentcities anpassen, wird das Teilziel von Agentcities erreicht, als Referenzumgebung zu gelten. Agentcities wird zum de-facto-Standard, wenn genügend verschiedene Plattform-Implementationen vertreten sind.

4.3.2 Technische Dienste

Die *inhaltlichen* Dienste nutzen die technische Infrastruktur und bauen darauf auf, im Gegensatz zu den *technischen* Diensten, die hier genauer beschrieben werden.

Zum Betrieb des Netzes sind vier Basisfunktionen notwendig: Kommunikation, Identität, die zentralen Verzeichnisse und Verwaltung der Plattform-Anbindung. Die Informationen stammen z.T. aus [act02a] und wurden nach eigener Anschauung ergänzt.

Kommunikation

Der Kommunikationsdienst wird nicht zentral angeboten, sondern von jeder Plattform selbst. Je nach Grad der Anbindung (angemeldet, aktiv oder vollständig) bedeutet das, dass die Plattform die Standards mehr oder weniger weitgehend implementiert (Zusammenfassung in 4.3.1 (*Standards*), Ausführung in 3.4 (*Standardisierung nach FIPA*)). Dadurch ist die Übereinstimmung auf der technisch-syntaktischen Ebene stufenweise sichergestellt. Die regelmäßige Anfrage mit Ping stellt die grundlegende Kommunikation sicher. Darauf aufbauend können der `AMS` und der `DF`

einer Plattform regelmäßig angesprochen werden, um weitergehende Eigenschaften der Kommunikation sicherzustellen: Teile der Basisontologie und die Inhaltssprache SL0. Die vollständig angebundenen Plattformen verfügen also über zusammenpassende Kommunikationsdienste und können so prinzipiell Nachrichten untereinander versenden. Die zentralen Verzeichnisdienste vermitteln die notwendige Kenntnis der Adressen, wodurch eine logisch vollständige Vernetzung der Plattformen entsteht.

Identität

Das Ziel des Identitäts-Dienstes ist es, weltweit alle Namen von Elementen im Agentcities-Agentennetz eindeutig zu qualifizieren. Der hierarchische Aufbau des Netzes kann für die Namensgebung benutzt werden; Internet, Domäne, Plattform, Agent und Dienst bilden eine Hierarchie, die der konzeptuellen Hierarchie für Agentensysteme im Abschnitt 3.3 (MULAN) entspricht. Eine Zusammenfassung der Konstruktion von Namen ist in Abbildung 4.4 dargestellt. Im Folgenden werden die einzelnen Namensverfahren vorgestellt.

Bereich: / Element:	Weltweit	Agentcities−weit	Innerhalb einer PF	Innerhalb eines Agenten
Plattformname	AC hängt Domänen−Name an	beim Registrieren im Formular überprüft (AC sorgt selbst dafür)	ist bereits eindeutig	ist bereits eindeutig
Agentenname	h.K.	h.K. und Plattform−Name angehängt	Aufgabe der Plattform selbst	ist bereits eindeutig
Dienstname	nicht einzeln ansprechbar	nicht einzeln ansprechbar	h.K.	der Agent sorgt selbst dafür

h.K. = hierarchische Kombination

Abbildung 4.4: Eindeutige Namen in Agentcities.

Die Eindeutigkeit von *Domänennamen* wird von Namensdiensten im Internet sichergestellt, hier handelt es sich um die Domäne `agentcities.net`.

Plattformnamen bestehen aus zwei untrennbaren Teilen. Der erste Teil muss innerhalb von Agentcities eindeutig sein, daran wird der Domänenname mit einem Punkt angehängt. Im Folgenden wird mit dem Begriff Plattformname der vollständige Name gemeint, es sei denn, der Text bezieht sich ausdrücklich auf den ersten Teil des Namens.

Agentennamen müssen innerhalb ihrer Plattform eindeutig sein. Wie die Plattform dafür sorgt, ist nicht festgelegt. CAPA erweitert den Agentennamen z.B. durch eine laufende Nummer. Der Plattformname wird mit einem @ an den Agentennamen

angehängt. Der erste Teil des Agentennamens bis zum @ wird im Folgenden als der „einfache“ oder „unqualifizierte“ Agentenname bezeichnet. Der vollständige Agentenname ist schon auf Plattform-Ebene weltweit eindeutig, weil die Domäne Teil der Namen ist. Diese Namensbildung stellt die Anforderung an die Heimatplattform, so geformte Agentennamen zu ermöglichen. Ob die Heimatplattform eines Agenten auch Nachrichten zustellt, in denen der Agentenname unqualifiziert angegeben ist, bleibt der konkreten Implementation überlassen, für interne Nachrichten ist dies möglicherweise erwünscht.

Bei *Dienstnamen* ist eine Eindeutigkeit vor allem bezüglich ihrer Eigenschaften erwünscht. Dieses wird durch implizite oder explizite, standardisierte oder proprietäre Ontologien erreicht und ist nicht Teil des Identitätsdienstes von Agentcities.

Ein Dienst kann nur über den ihn anbietenden Agenten angesprochen werden. Ist der Agent wie oben beschrieben adressiert, wird der Dienst als Teil des Nachrichteninhalts angesprochen.

Verzeichnisdienste

Der zentrale Knoten im Agentennetz bietet Dienste für drei Verzeichnisse an: ein Plattform-, ein Agenten- und ein Diensteverzeichnis, abgekürzt APD (*Agent Platform Directory*), `AMS` und `DF`. Das Plattform-Verzeichnis ist aufgeteilt in ein Verzeichnis der aktiven Plattformen und ein umfassenderes Verzeichnis aller angemeldeten Plattformen. Jeder dieser Verzeichnisdienste hat eine Agentenschnittstelle und eine Schnittstelle für menschliche Nutzer, wie in Abbildung 4.5 dargestellt.

Schnittstelle: / Sprachelement:	Mensch	Agent
Darstellungssprache	HTML (JSP)	FIPA−ACL
Inhaltssprache	Englisch	FIPA−SL
Ontologie	Begriffe aus dem Agentcities−Projekt, FIPA, Informatik	FIPA Agent Management Ontology

Abbildung 4.5: Schnittstellen der Verzeichnisdienste für Mensch und Agent

Die den drei Verzeichnisdiensten gemeinsame generelle Funktionsweise ist in Abbildung 4.6 dargestellt. Jeder Verzeichnisdienst hat eine Quelle in Form eines Speichers, aus der ein Anfrage-Agent die Information nimmt, welche Plattformen wie zu befragen sind (oberer Speicher im Bild). Die gewonnenen Informationen schreibt der Anfrage-Agent in einen Speicher, der nicht notwendig mit der Datenquelle identisch ist (deshalb ist der Ausgabespeicher unten gesondert, aber mit der gleichen Bezeichnung dargestellt). Aus dem Ausgabespeicher werden Anfragen an den Ver-

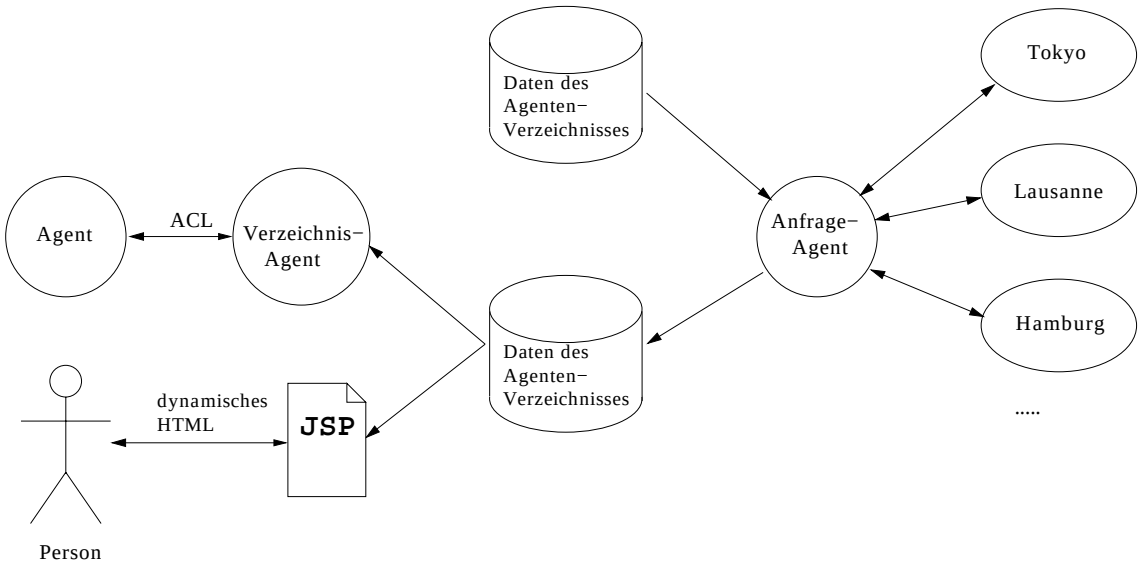


Abbildung 4.6: Allgemeine Funktionsweise der Verzeichnisdienste in Agentcities. Nach [acr03b, S. 10].

zeichnisdienst von Agenten und von Personen beantwortet (Abbildung 4.6 links, wie in Abbildung 4.5 tabellarisch dargestellt).

Abbildung 4.7 verdeutlicht die gegenseitige Abhängigkeit der Verzeichnisdienste. Im Gegensatz zur vorigen Abbildung sind die Speicher in dieser Abbildung konkret benannt und jedes Speichersymbol steht für einen tatsächlichen Speicher. Der zeitliche Ablauf zur Grafik ist wie folgt: Eine Person trägt über ein HTML-Formular Informationen in den Speicher „Daten des Plattformverzeichnisses“ ein (Anmeldung und Konfiguration einer Plattform). Der Plattform-Anfrage-Agent fragt aufgrund dieser Informationen die eingetragenen Plattformen ab (rechts in der allgemeineren Abbildung 4.6, in Abbildung 4.7 sind die angebundenen Plattformen nicht dargestellt). Das Ergebnis (aktuelle Informationen zur Erreichbarkeit der Plattformen) trägt der Plattform-Anfrage-Agent in denselben Speicher ein. Der Agenten-Anfrage-Agent und der Dienste-Anfrage-Agent fragen beide nur die als erreichbar gekennzeichneten Plattformen nach ihren Agenten bzw. Diensten (die Plattformen sind wiederum nicht im Bild 4.7 dargestellt) und tragen die gewonnenen Informationen in jeweils eigene Speicher ein. Aus den drei Speichern werden die Informationen über Plattformen, Agenten und Dienste den Agenten und Personen zugänglich gemacht (unten und rechts oben dargestellt).

Es gibt vier JSP-Seiten zum Zugriff auf die Verzeichnisse: In das Verzeichnis der angemeldeten Plattformen wird jede von einer Person eingetragene Plattform aufgenommen. In das Verzeichnis der aktiven Plattformen werden diejenigen Plattformen

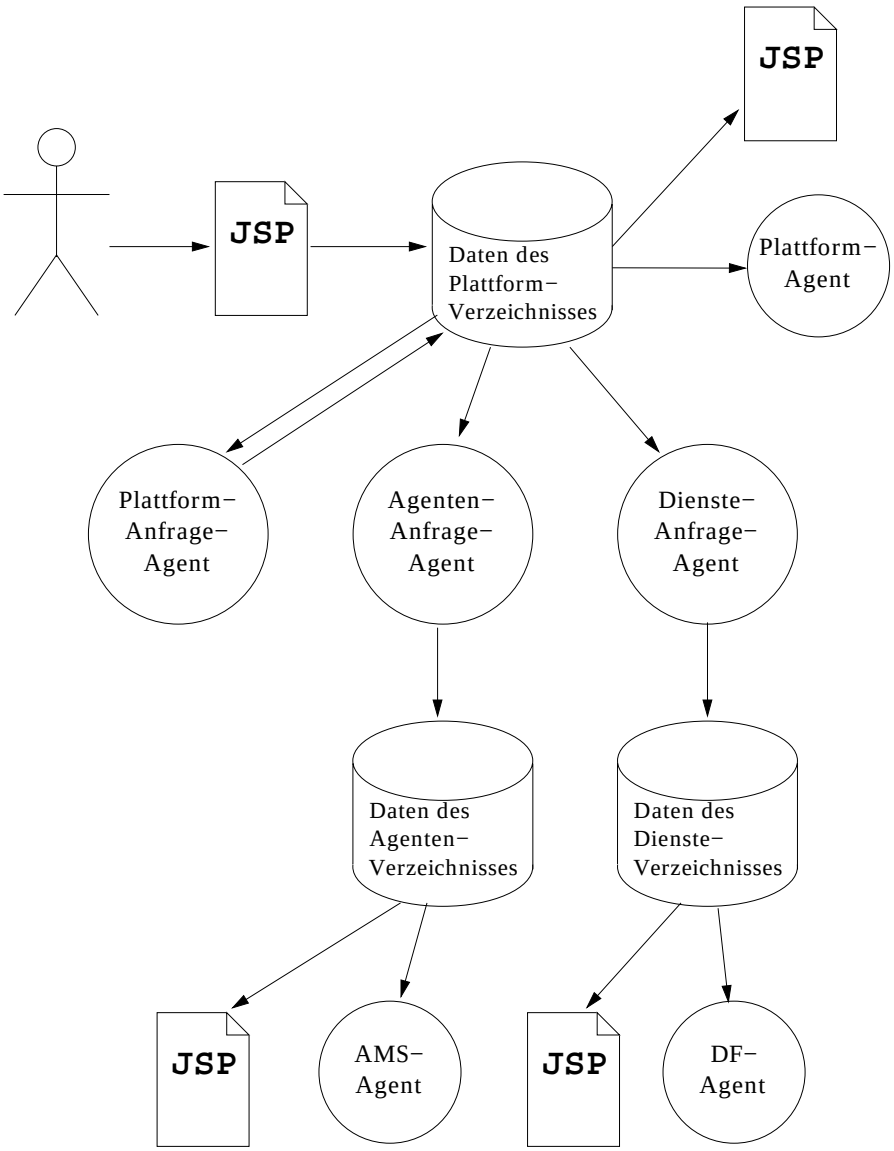


Abbildung 4.7: Zusammenspiel und konkrete Funktionsweise der Verzeichnisdienste in Agentcities. Nach [acr03b, S. 11].

aufgenommen, die in einem Zeitraum von sechs Wochen mindestens einmal auf die Ping-Anfrage geantwortet haben. Innerhalb dieses Verzeichnisses von aktiven Plattformen werden die Plattformen als `up` gekennzeichnet, die die letzte Ping-Anfrage beantwortet haben, die anderen als `down`. Die Plattformen, die als `up` gekennzeichnet sind, werden vom Agenten-Anfrage-Agent und vom Dienste-Anfrage-gent nach ihren Agenten bzw. Diensten gefragt, sofern der Eintrag zur Plattform von einer Person bei der Anmeldung so konfiguriert wurde.

Verwaltung

Die Verwaltung geschieht grundsätzlich manuell über die HTML-Schnittstelle. Sie bezieht sich auf die Forschungsgruppen, auf deren Kontaktpersonen und auf die angemeldeten Plattformen.

Vier Arten von Informationen sind nach außen über die HTML-Schnittstelle des Plattformverzeichnisses zugänglich:

- Allgemeine Informationen zur Forschungsgruppe
  - Eigene Beschreibung, z.B. zum Forschungsschwerpunkt
  - Gruppenmitglieder mit Kontaktinformationen
  - Übersicht der angemeldeten Plattformen mit ihrem Zustand
- Plattform-bezogene Informationen zur Forschungsgruppe
  - Zur Organisation, die die Plattform betreibt
  - Zum finanziellen Hintergrund der Plattform
  - Zum Ort der Plattform (samt Längen- und Breitengrad für die graphische Darstellung des Agentennetzes)
- Technische Informationen zur Plattform
  - Plattformname
  - Plattformbeschreibung nach FIPA
  - URLs von Log-Dateien und Kontaktinformation für Support
  - Information zur Infrastruktur der Plattform (zur Implementation)
- Informationen zur Konfiguration
  - prinzipielle Bereitschaft, Nachrichten von anderen Plattformen zu empfangen (ja/nein)
  - Name des Ping-Agenten
  - regelmäßige Ping-Anfrage (ja/nein) mit zeitlichem Abstand zwischen 40 Sekunden und fünf Minuten
  - `AMS`-Anfrage (ja/nein)
  - `DF`-Anfrage (ja/nein)

4.3.3 Anforderungen zur Anbindung einer Plattform

Die Agentcities Taskforce hat eine konkrete Anleitung zur Anbindung einer Plattform an Agentcities erstellt ([act02b]). Der folgende Text stellt die maßgeblichen Anforderungen von Agentcities an eine anzubindende Plattform dar.

Voraussetzungen Man braucht einen Rechner mit Internetzugang, vorzugsweise mit einer festen IP-Adresse und einem DNS-Namen. Für den Rechner wird die Erlaubnis des Netzwerkadministrators im lokalen Netz benötigt, an einem Port zu lauschen und Nachrichten zu verschicken.

Eine Plattform bereitstellen Es muss eine Plattform installiert und konfiguriert werden, die die im Agentennetz gültigen Spezifikationen (siehe Abschnitt 4.3.1) implementiert. Solche Plattformen existieren bereits oder werden selbst entwickelt (das ist bei CAPA der Fall).

Die zitierte Dokumentation listet sieben frei verfügbare und in Agentcities benutzte Plattformen auf (siehe Abschnitt 5.4.5). Eine solche Plattform muss man installieren und so konfigurieren, dass der Name der Plattform die Form `<stadt>.agentcities.net` hat. Die Stadt repräsentiert dabei (notfalls symbolisch) den Ort der Plattform (es heißt ja „Agent*cities*“).

In Abbildung 4.8 sind die Elemente hervorgehoben, die zur aktiven Anbindung einer Plattform notwendig sind.

Spezifikation des Ping-Agenten Der Ping-Agent muss in regelmäßigen Abständen eine Nachricht korrekt beantworten. Die ankommende Nachricht (siehe Abbildung 4.9) ist eine ACL-Nachricht mit dem `PlainText`-Inhalt `"ping"`. Die Ontologie ist nicht benannt. Das Performativ ist ein `"query-ref"`, also eine Anfrage. Damit sind die fünf Felder der Anfragenachricht gegeben: Das Performativ ist `query-ref`, das Feld `sender` enthält den Absender, `receiver` enthält den `Agent-Identifier` des Ping-Agenten auf der angebundenen Plattform (für CAPA ist das z.B. der Agent `Ping#1`). Das Feld `language` enthält `PlainText` und das Feld `content` enthält `"ping"`.

Die Anwortnachricht, die der Ping-Agent darauf geben muss, enthält ebenfalls die Felder `sender` und `receiver`, das Feld `language` bleibt `PlainText` und das Feld `content` enthält den String `"alive"`.

Dieser Dienst kann bei einem `DF` unter dem Namen `ACLPing` mit dem Dienst-Typ `ping_acl_alpha_v1.0` veröffentlicht werden.

Eine Gruppe registrieren Um sich einloggen zu können und dann Plattformen anzumelden, muss man sich als Forschungsgruppe registrieren. Um eine neue Gruppe anzulegen, gibt man auf der Webseite [acs03a, unter /create.jsp] Informationen zur

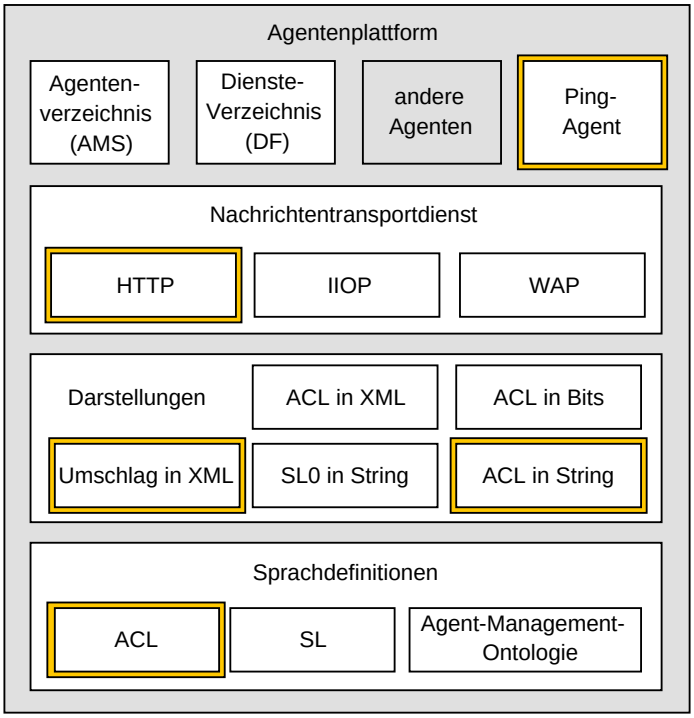


Abbildung 4.8: Elemente einer Agentenplattform in Agentcities. Vergleiche mit Abbildung 3.5 auf Seite 39.

Gruppe und zu einem ersten Mitglied an. Weitere Mitglieder können nach dem Einloggen einer Gruppe über eine Verwaltungsseite hinzugefügt werden.

Eine Plattform registrieren Um eine Plattform zu registrieren, müssen Informationen in ein von der Gruppenseite aus erreichbares HTML-Formular eingetragen werden. Diese Informationen wurden im Abschnitt 4.3.2 (*Technische Dienste*), im Unterabschnitt *Verwaltung* zusammengefasst.

Für die reine Anmeldung der Plattform reicht ein innerhalb von Agentcities eindeutiger Plattformname, für die aktive Anbindung müssen außerdem die Plattformbeschreibung und der unqualifizierte Name des Ping-Agenten angegeben, sowie die regelmäßige Ping-Anfrage aktiviert werden. Für die vollständige Anbindung muss die Anfrage an den `DF` und den `AMS` ebenfalls aktiviert sein.

Für die Verwaltung einer bereits registrierten Plattform wird dasselbe HTML-Formular wie für die Anmeldung verwendet.

Die Plattform erscheint nach der Anmeldung im globalen Plattformverzeichnis auf der Webseite [acs03a, unter /platform-all.jsp].

```
(QUERY-REF
   :sender (agent-identifier
    :name pingstat@webap.lausanne.agentcities.net
    :addresses (sequence http://srv02.lausanne.agentcities.net:7710/acc ))
  :receiver (set (agent-identifier
    :name Ping#1@Test.Hamburg.agentcities.net
    :addresses (sequence http://rzcspc13.informatik.uni-hamburg.de:3297/acc )) )
  :content "ping"
  :language PlainText)

(INFORM
  :reply-with Ping#1@Test.Hamburg.agentcities.net#458
  :receiver (set (agent-identifier
    :name pingstat@webap.lausanne.agentcities.net
    :addresses (sequence http://srv02.lausanne.agentcities.net:7710/acc)))
  :sender (agent-identifier
    :name Ping#1@Test.Hamburg.agentcities.net
    :addresses (sequence
      http://rzcspc13.informatik.uni-hamburg.de:3297/acc
      gopher://rzcspc13.informatik.uni-hamburg.de:4711/))
  :content "alive"
  :language PlainText)
```

Abbildung 4.9: Beispiel für eine Ping-Kommunikation

4.4 Einordnung

Zur Einordnung von Agentcities wird das Agentenkonzept MULAN mit Agentcities in Verbindung gebracht und ein Konzept zur rekursiven Anwendung von Agentcities-ähnlichen Einrichtungen erwähnt.

Innerhalb von Agentcities ist Ping ein zentrales Konzept. Diesem Konzept ist der darauf folgende Abschnitt gewidmet.

4.4.1 Agentcities als Multiagentensystem

Das von Agentcities betriebene Agentennetz ist zunächst technischer Natur. Es stellt eine Infrastruktur mit zentralen Verzeichnisdiensten und den Anfrage-Agenten bereit; auf dieser Infrastruktur können agentenbasierte Anwendungen (also *Multiagentensysteme* im Sinne der Definitionen 3.1 und 3.2) aufbauen. Man kann jedoch auch Agentcities selbst als Multiagentensystem auffassen, weil am Aufbau der zentralen Verzeichnisse mehrere Verzeichnis-Agenten beteiligt sind, sowie auf jeder aktiven Plattform ein Ping-Agent, evtl. noch `AMS` und `DF`. Insofern stellt Agentcities ein in beachtlichem Maße offenes und dynamisches Multiagentensystem dar, das seinen Zweck in der dadurch ermöglichten Forschungs- und Entwicklungsarbeit sowie in der Präsentation der beteiligten Plattformen findet.

Agentcities ist die einzige Vernetzung von FIPA-kompatiblen Agentenplattformen, auf die die FIPA selbst verweist (vgl. [fip03, unter /resources/platforms.html]). Anfang 1999 hatte die FIPA bereits Versuche mit einer offenen Struktur für FIPA-kompatible Produkte gestartet, die nun im Rahmen von Agentcities fortgeführt werden. Innerhalb von Agentcities bilden sich jedoch ähnlich geartete Projekte, die das Agentennetz von Agentcities als Einstiegspunkt nutzen, um eine ähnliche Struktur unterhalb von Agentcities aufzubauen. Dort sind jeweils nur inhaltlich verwandte Projekte miteinander vernetzt. Bei einer Anfrage wird so die umfangreiche irrelevante Information über alle Agenten in Agentcities stark verringert. Dieser Ansatz kann zur Skalierbarkeit eines solchen offenen Agentennetzes beitragen. Ein solches Subnetz funktioniert auch ohne Agentcities, sofern es unter einer festen Adresse im Internet erreichbar ist. Damit steht es sowohl neben als auch unterhalb des Agentennetzes von Agentcities.

Mit Mulan kann das Agentennetz von Agentcities modelliert werden, indem die Struktur der obersten Ebene angepasst wird. Dabei kann man sich entscheiden, ob die tatsächlichen Kommunikationswege abgebildet werden sollen oder die möglichen. Im zweiten Fall sind alle Plattformen vollständig untereinander vernetzt, der zentrale Knoten würde nicht herausragen. Im ersten Fall wären die zentralen Verzeichnisse und Anfrage-Agenten von Agentcities mit allen anderen Plattformen vernetzt; einige zusätzliche Verbindungen würden bestehen. Über die Zeit müsste die Netzstruktur sich verändern. Nun sind die Referenznetze in Renew prinzipiell statisch, nur ihre Markierung kann sich während der laufenden Simulation ändern. Wenn jedoch ein Agent in Mulan erzeugt wird oder ein Agent ein Protokollnetz startet, so wird eine weitere Instanz eines Netzes erzeugt und die Referenz darauf als Marke gehalten. In gewissem Sinne entsteht damit eine dynamische Struktur von Referenznetzen, weil die Elemente der neuen Netzinstanz in Form einer Marke Teil des umschließenden Netzes sind. Wendet man nun das Konzept von Mulan rekursiv an und steckt die Funktionalität der Mulan-Hierarchie in das Protokoll eines Agenten, so entsteht eine dynamische Struktur, mit der die Struktur von Agentcities nachgebildet werden kann.

4.4.2 Ping

Ping ist ursprünglich Teil einer relativ niedrigen Kommunikationsschicht, der Internet-Schicht des TCP/IP-Referenzmodells. In dieser Schicht werden verschiedene Internet-Kontrollprotokolle verwendet, so auch das *Internet Control Message Protocol* (ICMP). Es definiert verschiedene Nachrichtentypen, mit denen unerwartete Ereignisse berichtet werden und mit denen die Funktionsfähigkeit und Performanz in Computernetzen getestet werden kann. Mit den Nachrichten ECHO (ping) und ECHO REPLY kann geprüft werden, ob ein Rechner läuft (siehe [Tan03, S. 449f]).

Ein funktionierendes Ping stellt sicher, dass die beteiligten Rechner laufen, dass ein Betriebssystem installiert ist, dass die Kabel richtig angeschlossen sind, und dass

Ports und Firewalls zumindest ein Ping erlauben. Auch das Routing im Computernetz funktioniert dann. Diese Form von Ping wird eher im Fehlerfall zur Prüfung der genannten Fälle verwendet.

In Agentcities wird dauerhaft ein regelmäßiges Ping verwendet. Hier kommt dem Ping eine zentrale Bedeutung zu. Es bildet die Voraussetzung für jede aktive Teilnahme einer Plattform am Agentennetz. Es enthält das Konzept, dass eine Plattform kurzfristig nicht mehr erreichbar sein kann und dass Plattformen kurzfristig hinzukommen. Also ist das regelmäßige Ping die Voraussetzung für eine möglichst aktuelle Darstellung der verbundenen Plattformen in einem potientiell hoch dynamischen System.

Ping stellt außerdem die Kompatibilität der Plattformen auf der technischen Kommunikationsebene sicher, indem alle Instanzen der Kommunikation genutzt werden. Die Kommunikation von Agent zu Agent funktioniert damit auf jeden Fall zwischen der Plattform und dem zentralen Knoten. Daraus folgt, dass auch die Plattformen untereinander auf diese grundlegende Weise kommunizieren können. Zur vollständigen Anbindung werden zusätzliche Elemente benötigt, wie die Verwendung einer Inhaltssprache, alternativer Repräsentierungen, komplexerer Ontologien und alternativer Transportmittel.

5 Capa

Die Agentenplattform Capa, die im Rahmen dieser Arbeit erweitert werden soll, ist aus Mulan entstanden und daher eng mit Mulan verwoben. Die Entwicklung von Capa reicht von frühen Arbeiten zur Agentenmodellierung mit Referenznetzen [Wie96] zur grundlegenden Architektur von Mulan [Röl99] mit Veröffentlichung [KMR01], über die Umsetzung der FIPA-Kommunikations-Architektur [Duv02] und deren Veröffentlichung [DMR02] bis hin zu aktuellen Arbeiten zu Details wie der Mobilität von Agenten in [KMR03].

Die Elemente von Capa sind in Abbildung 5.1 dargestellt. Die Pfeile von rechts deuten die Erweiterbarkeit von Capa auf den verschiedenen Ebenen an. Die Agentenschicht ist im Unterschied zu den beiden bisherigen Architekturbildern (Abbildung 3.5 auf Seite 39 und Abbildung 4.8 auf Seite 64) in zwei Schichten dargestellt. Die Elemente der Plattform werden in den ersten drei Abschnitten dieses Kapitels näher erläutert.

Im ersten Abschnitt dieses Kapitels werden der Aufbau und die Funktionsweise von Agenten in Capa beschrieben sowie die zur Plattform gehörenden Agenten, der Abschnitt bezieht sich damit auf die beiden oberen Schichten in Abbildung 5.1. Der zweite Abschnitt stellt die von Capa unterstützten Sprachen und ihre Darstellungen vor, damit bezieht er sich auf die beiden unteren Schichten im Bild. Im dritten Abschnitt werden die Funktionsweise und der Aufbau des Nachrichtentransports in Capa vorgestellt, das entspricht der mittleren Schicht im Bild. Der letzte Abschnitt erläutert Allgemeines zur Konfiguration von Capa und zu Inspektionsmöglichkeiten. Es wird eine Entwurfsmethode für Agenten in Capa erläutert und schließlich wird Capa mit FIPA und Agentcities in Verbindung gebracht.

5.1 Agenten in Capa

Die Agenten in Capa entsprechen den Agenten in Mulan (Abschnitt 3.3 (Mulan)). Die Modellierung der Agenten in Mulan erlaubt bereits funktionsfähige kommunizierende Agenten. Mit Capa wird Mulan zu einer FIPA-konformen Agentenplattform ausgebaut, indem die vorhandene Entscheidungsarchitektur für Agenten um die von der FIPA spezifizierte Kommunikationsarchitektur ergänzt wurde. Die Plattformarchitektur von Capa setzt sich aus Mulan- und FIPA-Elementen zusammen[1].

[1] Der hier verwendete Architekturbegriff wird in [Duv02] definiert und im Glossar am Ende dieser Arbeit vorgestellt (→*Architektur*).

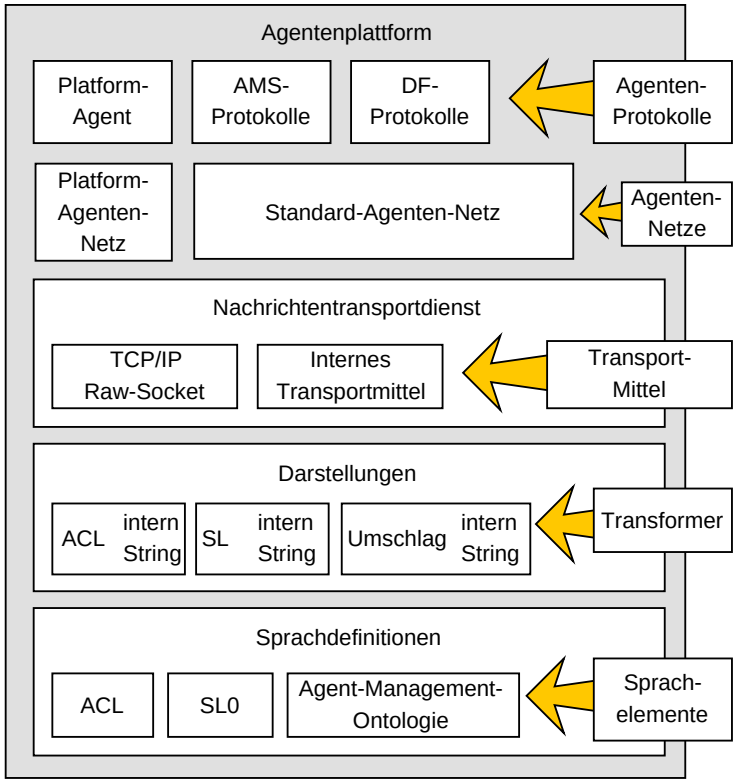


Abbildung 5.1: Elemente der Agentenplattform CAPA (Ausgangssituation). Entwurfsgrundlage: [Duv02, Seite 72].

In den folgenden Unterabschnitten wird zunächst der Aufbau der Agenten in CAPA beschrieben, dann deren Funktionsweise. Die zur Plattform fest zugehörigen Agenten werden im dritten Teil dieses Abschnitts beschrieben.

5.1.1 Aufbau

Ein Agent in CAPA besteht aus dem *Agentennetz*[2], einem Referenznetz für die Verwaltung der *Wissensbasis* mit dem aktuellen Wissen sowie den *Protokollnetzen* und einem Referenznetz *Protokollfabrik*. Eine Instanz des Agentennetzes repräsentiert einen Agenten. Das Agentennetz ist in der Hierarchie aus Referenznetzen zwischen der Plattform (nach oben) und der Wissensbasis, der Protokollfabrik und den Protokollen nach unten angeordnet. Die grobe Struktur des Agentennetzes wurde im Abschnitt MULAN beschrieben, eine eingehende Beschreibung dieser Netze ist in

[2]In diesem Kapitel bezeichnet der Begriff Agentennetz durchgehend das Referenznetz eines Agenten in CAPA, es sei denn, es wird ausdrücklich auf Agentcities Bezug genommen.

[Duv02, Seite 86–90] zu finden.

Das Agentennetz des Plattform-Agenten ist an die besonderen Aufgaben (Initialisierung der Plattform) angepasst. Der Pfeil von rechts im Bild 5.1 auf der vorherigen Seite deutet an, dass weitere Anpassungen des Agentennetzes denkbar sind. Für die Erweiterung von Capa an dieser Stelle existiert jedoch keine parametrisierte Schnittstelle.

Wissensbasis

Das Wissen eines Agenten in Capa ist in Schlüssel-Wert-Paaren gespeichert. Die Schlüssel sind Strings, die Werte können beliebige Java-Objekte sein. Der Agent kann während einer Konversation lesend und schreibend auf das Wissen zugreifen, bei maximaler Nebenläufigkeit.

Es gibt verschiedene Arten von Einträgen einer Wissensbasis:

- Die Beschreibung der eigenen Dienste enthält Bezeichner aus einer Ontologie.
- Die Beschreibung der benötigten Dienste enthält ebenfalls Bezeichner aus einer Ontologie.
- Protokolleinträge beschreiben Nachrichtenmuster und bezeichnen die Protokolle, die zur Bearbeitung eines Nachrichtentyps geeignet sind.
- Ein Eintrag mit dem Schlüsselwort `proaktiv` bezeichnet die Protokolle, die ein Agent ohne Anstoß von außen instanziiert.
- Weitere Einträge können das Fachwissen des Agenten in Form von beliebigen Java-Objekten enthalten.

Der Zugriff auf das Wissen wird von dem Referenznetz *Wisssensbasis* bereitgestellt. Die Zugriffsmöglichkeiten teilen sich in synchronisierte (schreibende) und nebenläufige (lesende) Zugriffe. Die lesenden Zugriffe sind Anfragen nach der Existenz von Schlüsseln oder der Existenz von Protokolleinträgen zu einer gegebenen Nachricht, sowie Fragen nach dem Wert zu einem Schlüssel bzw. nach einem Protokollnamen zu einer gegebenen Nachricht. Mit den schreibenden Aktionen kann ein Wert zu einem Schlüssel geändert oder ersetzt werden und es kann ein Eintrag neu hinzugefügt werden. Schließlich kann die Wissensbasis das gesamte aktuelle Wissen in Form eines Strings ausgeben, was zum Abspeichern und zur Migration verwendet werden kann. Das Anfangswissen der Agenten wird in einer Textdatei angegeben. Als ein Beispiel einer solchen Datei kann die in Abbildung 6.1 auf Seite 95 dargestellte und dort erläuterte Wissensbasis dienen.

Protokollfabrik und Protokollnetze

Wie bereits im Abschnitt 3.3 (Mulan) erläutert, werden in Capa Protokolle zur Auswertung und Verarbeitung von Nachrichten verwendet. Die Protokollfabrik entscheidet aufgrund des Wissens, welches Protokoll für eine ankommende Nachricht

geeignet ist und initialisiert dann das passende Protokollnetz. In jeder Protokollnetzinstanz wird dann eine Konversation verwaltet.

In einem Protokoll hat der Agent folgende Handlungsmöglichkeiten:

- Er hat Zugriff auf die Wissensbasis.
- Er kann den Inhalt von ankommenden Nachrichten verarbeiten.
- Er kann Nachrichten versenden.
- Er kann die Konversation beenden, so dass ankommende Nachrichten nicht mehr zugeordnet werden können.
- Bei all diesen Möglichkeiten steht die volle Ausdruckskraft von Java und Referenznetzen zur Verfügung.

Die Schnittstelle eines Protokollnetzes zum Agentennetz besteht aus den synchronen Kanälen `start` und `stop` zur Steuerung der Konversation, `in` und `out` für eingehende und ausgehende Nachrichten sowie `access` für den Zugriff auf die Wissensbasis.

Wenn einem Agenten ein Protokoll zur Verfügung gestellt werden soll, so geschieht das über einen Eintrag in die Wissensbasis des Agenten. Die Schnittstelle für neue Protokolle ist damit der Inhalt der Wissensbasis eines Agenten, denn ein Eintrag in der Wissensbasis ist die Voraussetzung für die Instanziierung eines Protokollnetzes.

In dem Architekturbild von Capa (Abbildung 5.1 auf Seite 69) sind die einzelnen Protokollnetze und die Protokollfabrik nicht dargestellt, weil sie zu detailliert sind bzw. für jeden Agenten gleich. Die Protokollnetze sind pro Agent zu einem Kasten zusammengefasst in der obersten Schicht repräsentiert. Neue Agenten können mit je einem Kasten einbezogen werden, wobei zu unterscheiden ist zwischen solchen Agenten, die zur Plattform gehören und solchen Agenten, welche die Plattform nutzen.

5.1.2 Funktionsweise

In diesem Unterabschnitt wird die Funktionsweise eines Agenten in Capa erläutert. Zunächst wird die Initialisierung eines Agenten beschrieben und danach das Zusammenspiel von Wissen und Protokollen noch einmal aufgegriffen. Schließlich folgen Anmerkungen zur Synchronisierung mehrerer paralleler Konversationen und zuletzt wird der Zustand und das Verhalten eines Agenten erläutert.

Initialisierung eines Agenten

Zur Erzeugung eines Agenten wird eine Nachricht mit einer Aufforderung an den `AMS` der Plattform geschickt. Die Nachricht enthält den gewünschten Agentennamen, einen Verweis auf das Referenznetz, das zur Verwaltung des Wissens verwendet werden soll und das Anfangswissen des Agenten in String-Darstellung. An den Agentennamen wird bei der Erzeugung eine laufende Nummer angehängt, um Na-

menskonflikte zu vermeiden. Das Netz für die Wissensbasis bleibt meistens gleich, denn das Standardnetz stellt alle grundlegenden Funktionen bereit.

Beim Start eines Agenten wird dieser automatisch bei der Plattform angemeldet, damit für ihn ankommende Nachrichten ausgeliefert werden können. Die Anmeldung beim `AMS` und `DF` der Plattform muss der Agent selbst erledigen. Dazu existiert ein Protokoll (`generalAgentSetup`), das als `proaktiv` in die Wissensbasis eingetragen werden kann. Für die Anmeldung beim `DF` benutzt es die Beschreibung der eigenen Dienste aus der Wissensbasis und den Agentennamen. Danach sucht es beim `DF` nach den von diesem Agent benötigten Diensten. Das Ergebnis der Anfrage wird dann zusammen mit dem vollständigen Agentennamen (mit laufender Nummer und `"@Plattformname"`) in die Wissensbasis eingetragen. Zum Abschluss schickt das Protokoll `generalAgentSetup` eine Nachricht mit dem Inhalt `"start"` an den Agenten selbst. In der Wissensbasis kann dann (muss aber nicht) ein weiterer Eintrag mit einem spezifischen Initialisierungsprotokoll stehen, das von der `"start"`-Nachricht aktiviert wird.

Zusammenspiel von Wissen und Protokollen

Das Zusammenspiel von Wissen und Protokollen im Agenten ist ein wesentlicher Aspekt in Mulan und damit auch in Capa. Hier wird definiert, wie die statischen Aspekte (also die Protokollimplementationen) und die dynamischen Aspekte (also der Inhalt der Wissensbasis und die ankommenden Nachrichten) zusammenspielen.

Die Wissensbasis enthält in den Protokolleinträgen Muster von Nachrichten, welche je einem Protokollnamen zugeordnet sind[3]. Die Protokollfabrik vergleicht eine ankommende Nachricht mit den Mustern in der Wissensbasis. Sie initialisiert das Protokoll mit dem genauesten Muster[4] und übergibt die Nachricht dem Protokoll zur Auswertung. Die Protokolle haben Zugriff auf die Wissensbasis des Agenten, so können die Protokolleinträge dynamisch angepasst werden.

Die Protokolle, die in der Wissensbasis als proaktiv gekennzeichnet sind, werden sofort nach dem Starten des Agenten initialisiert.

Um eine Nachricht einer laufenden Konversation zuzuordnen, wird der Wert des `in-reply-to`-Feldes der Nachricht verwendet. Nachrichten, die das `in-reply-to`-Feld gesetzt haben, werden direkt an das darin referenzierte Protokollnetz weitergeleitet. Falls weder in der Nachricht eine laufende Konversation referenziert wird (d.h. falls das Feld `in-reply-to` leer ist) noch ein zur Nachricht passender Eintrag in der

[3] Der Protokollname wird als ausreichender Verweis auf die tatsächliche Implementation verwendet. Wird die Implementation ausgetauscht (z.B. bei einem mobilen Agenten), kann auf interessante Weise das Verhalten des Agenten seiner Umgebung angepasst werden.

[4] Die zum Vergleich von Muster und Nachricht verwendete Relation ist in Abschnitt 3.4.4 auf Seite 40 dargestellt.

Wissensbasis existiert, wird die Nachricht in der Protokollfabrik als nicht verstanden (`not-understood`) aufbewahrt[5].

Synchronisation der Konversationen

Hat ein Agent mehrere Konversationen parallel laufen, kann eine evtl. notwendige Synchronisation entweder über die Wissensbasis geschehen oder über Nachrichten, die der Agent sich selbst schickt. Das Protokoll `generalAgentSetup` nutzt diese zweite Möglichkeit, indem es am Ende eine `"start"`-Nachricht abschickt. Die logische Abfolge mehrerer Konversationen wird ebenfalls über Nachrichten koordiniert.

Zustand und Verhalten eines Agenten

Der Zustand eines Agenten (wie auch der ganzen Plattform) wird streng genommen durch die Markierung der beteiligten Netze vollständig beschrieben (Markierung hier im Sinne von Netzen in Netzen, siehe Kapitel 2). Für einen Agenten ist sind die beteiligten Netze das Agentennetz sowie die als Marken darin enthaltenen Netze: Protokollfabrik, Wissensbasis und die instanziierten und verfügbaren Protokolle. Um den Plattformzustand vollständig zu beschreiben, ist das Netz des Plattform-Agenten und die darin enthaltenen Netzreferenzen auf den ACC und auf alle Agentennetze notwendig. Zur Inspektion des Plattformzustands gibt es den MulanViewer, der in Abschnitt 5.4.2 auf Seite 81 beschrieben ist.

Die Markierung enthält demnach alle Informationen über laufende Konversationen, das Wissen der Agenten und über Nachrichten, die gerade ankommen oder versendet werden. Das Agentennetz verfügt über eine Zählvorrichtung mit drei Zählern, in der die Nachrichten gezählt werden, die gerade ankommen (1) oder versendet werden (2) sowie die aktiven Protokolle(3). Stehen alle drei Zähler auf Null, ist der Agent untätig (also *idle*). Nur in diesem Fall ist sein Zustand vollständig durch den Inhalt der Wissensbasis bestimmt (die verfügbaren Protokolle sind durch namentliche Verweise in der Wissensbasis enthalten).

Das Verhalten eines Agenten wird durch das anfängliche Wissen und durch die darin referenzierten Protokolle bestimmt. Da in den Protokollen potentiell voller Zugriff auf die Wissensbasis besteht, können zur Laufzeit weitere Protokollinformationen zum Wissen hinzugefügt werden, vorausgesetzt, es gibt anfangs ein Protokoll, das diese Anpassung vornehmen kann.

[5] Die FIPA schreibt vor, unverständliche Nachrichten mit einem Hinweis zu beantworten. Die Sammlung `not-understood` kann dazu benutzt werden, sie fängt jedoch nur einen Fehlerfall ab, bei dem kein passendes Protokollnetz gefunden wird. Weitere Fehlerfälle sind: Nachrichten mit einer ungültigen Referenz auf eine laufende Konversation und innerhalb eines Protokolls unverständliche Nachrichten.

5.1.3 Zur Plattform gehörende Agenten

Zu jeder Capa-Plattform gehören drei Agenten fest hinzu: der `AMS`, der `DF` und der Plattform-Agent. Der Plattform-Agent bildet zusammen mit dem Plattform-Objekt die Grundstruktur der Plattform.

Die von der FIPA spezifizierten Verzeichnisdienste für Agenten und deren Dienste wurden in Capa als Agenten implementiert. Die Agenten `AMS` und `DF` werden beim Start der Plattform erzeugt. Sie werden so angesprochen, wie es in [FIPA 23] spezifiziert wird: Es werden ACL-Nachrichten mit SL-Inhalten mit Begriffen aus der `fipa-agent-management`-Ontologie versendet, um Agenten in den Verzeichnisdiensten zu registrieren und um Anfragen an die Verzeichnisdienste zu formulieren. Die Art und Abfolge der Nachrichten ist in Interaktionsprotokollen festgelegt.

Der Lebenszyklus von Agenten in Capa (also vor allem das Erzeugen und Terminieren von Agenten) wird vom `AMS` und vom Plattform-Agenten in Verbindung mit dem internen Transportmittel gemeinsam bearbeitet. Hier wird deutlich, dass dieses die Kernelemente von Capa sind.

5.2 Sprachen und Repräsentierungen

Die Sprachen und deren Darstellungen sind in dem Architekturbild zu Capa (Abbildung 5.1 auf Seite 69) in den unteren beiden Schichten dargestellt. In Capa gibt es eine Implementierung mit Java-Klassen für den Nachrichtenumschlag, für ACL und für SL0 sowie eine implizite Umsetzung der FIPA-Agent-Management-Ontologie.

Diese Schicht ist erweiterbar durch weitere Ontologien, Inhalts- und Nachrichtensprachen, sowie um weitere Darstellungen und Transformer.

Die Struktur einiger Sprach-Elemente wurde bereits im Abschnitt *Datenstrukturen für Nachrichten* auf Seite 40 vorgestellt. In den beiden folgenden Abschnitten werden ACL und SL vorgestellt, wie sie in Capa implementiert wurden.

5.2.1 ACL

Die ACL-Implementierung enthält:

- Klassen, die je ein Element der Sprache darstellen
- Eine Parser-Spezifikation
- Eine Transformer-Klasse, die zwischen der internen Java-Repräsentierung und der offiziellen String-Darstellung umwandeln kann.

Die ACL-Implementierung stellt neben den Konstruktoren für die Elemente einer Nachricht weitere Hilfsmethoden zur Verfügung, insbesondere ist die `reply()`-Methode von Bedeutung. Sie belegt die Felder einer neuen Nachricht mit Werten aus der zu beantwortenden Nachricht, so dass nur die neuen Daten angegeben werden müssen.

Die Stringdarstellung von ACL wird durch aufeinander aufbauende `toString()`-Methoden erzeugt, so dass eine vollständige Nachricht durch einen Aufruf dieser Methode rekursiv als String dargestellt wird.

Die Unterscheidung der Typen `Word` und `String` ist wie bereits im Abschnitt 3.4.4 (*Datenstrukturen für Nachrichten*, Seite 40) erläutert nicht zweifelsfrei möglich, deshalb wurde eine pragmatische Lösung gewählt. Bei dieser Lösung wird der Typ `Word` von dem Typ `String` nicht unterschieden, sondern immer ein `String` gesendet, wenn dies erlaubt ist und ein `Word` nur dann, wenn `String` nicht erlaubt ist.

Bei verschiedenen Möglichkeiten zur Darstellung eines Feldes entsteht aus dieser Entscheidung das Problem in der Praxis, dass nach dem Parsen die ursprüngliche Darstellung (z.B. `Word`) nicht mehr bekannt ist. Wird beim Versand einer Antwort eine andere Darstellung gewählt (in CAPA grundsätzlich `String`), kann sie der Empfänger nicht lesen, falls er nur die von ihm ursprünglich gesendete Darstellung toleriert[6].

Beim Parsen einer Nachricht werden in CAPA die alternativen Typangaben für einzelne Nachrichtenfelder berücksichtigt. Die verschiedenen Teile der ACL-Nachricht werden beim Parsen nicht gesondert behandelt, sondern möglichst gleichartig als *KeyValue-Tupel* oder als *Value-Tupel*, dadurch bleibt der Parser einfach.

5.2.2 SL

Ähnlich wie die ACL-Implementierung für Nachrichten stellt die SL-Implementierung Konstruktoren für die Elemente eines SL-Nachrichteninhalts bereit.

Die Konvertierung der Inhaltskodierung aus der internen Darstellung in die String-Darstellung erfolgt zusammen mit der Konvertierung der Nachricht über die Methode `toString()` der Nachricht. Bei der Umwandlung muss das Feld `encoding` in der Nachrichtenstruktur aktualisiert werden (vgl. zum Text Abbildung 5.2)[7]. Die tiefer geschachtelte `toString()`-Methode des Nachrichteninhalts hat auf das Feld in der umschließenden Nachricht keinen Zugriff, weshalb das Feld `encoding` nicht aktualisiert wird. Dabei entstehen Nachrichten, deren Inhalts-Darstellung nicht mit der Angabe in dem Feld `encoding` übereinstimmt. Bei dem Paar *Darstellung des Inhalts* und `encoding` handelt es sich um nebeneinander liegende Felder derselben Datenstruktur, bei dem Paar *Darstellung der ACL-Nachricht* und `acl-representation` handelt es sich um Teile zweier verschiedener Datenstrukturen. Deshalb führt die Verwendung der `toString()`-Umwandlung für ACL nicht zu einer inkonsistenten Nachricht, für SL aber schon. Nachrichten, bei denen das Feld `encoding` nicht mit der tatsächlichen Darstellung des Inhalts übereinstimmt, können nicht geparst werden. Aus diesem Grund wurde bisher auf die Verwendung des

[6] Vorgriff: Dies war bei der `AMS`-Anfrage von Agentcities der Fall.

[7] Zur Erinnerung: Darstellung ist der Oberbegriff für Repräsentierung (Nachricht) und Kodierung (Inhalt).

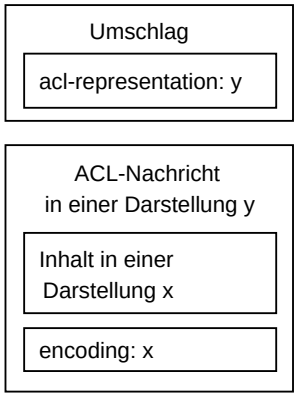


Abbildung 5.2: Die Felder `acl-representation` und `encoding` im Zusammenhang mit den Datenstrukturen `acl-message` und `envelope`.

Feldes `encoding` einer Nachricht verzichtet, da die Kodierung eines Nachrichteninhalts in Capa immer dieselbe ist wie die Repräsentierung der Nachricht und so das Feld `acl-representation` ausreicht.

Beim Entwurf neuer Multiagentensysteme werden insbesondere die Abfolge und die Inhalte von Nachrichten entworfen und damit auch eine implizite Ontologie geschaffen.

Um das Generieren und Parsen solcher Nachrichten zentral zu implementieren, existiert eine Umgebung, in der für jeden neuen Nachrichtentyp eine SL-Klasse entworfen wird, die unterstützende und insbesondere typsichere Methoden anbietet. Diese Klassen können sowohl vom Absender als auch vom Empfänger einer Nachricht genutzt werden, sofern beides Capa-Agenten sind[8].

Die SL-Implementierung enthält dieselben Arten von Klassen wie die ACL-Implementierung:

- Klassen, die je ein Element der Sprache darstellen
- Eine Parser-Spezifikation
- Eine Transformer-Klasse, die zwischen der internen Java-Repräsentierung und der offiziellen String-Darstellung umwandeln kann.

5.2.3 Ontologie

Eine Ontologie definiert Begriffe, die zur Beschreibung und Repräsentierung eines Wissensgebiets verwendet werden können, sowie die Beziehungen und Eigenschaften der durch dieses Vokabular bezeichneten Objekte. In der FIPA-Agentenkom-

[8]Prinzipiell ist die Unterstützung für SL kapselbar und kann als Java-Bibliothek veröffentlicht werden.

munikation werden Ontologien herangezogen, um festzulegen, wie der Inhalt von Nachrichten interpretiert werden soll.

In CAPA wurde eine Anbindung für Prozess-Ontologien umgesetzt (von Jan Ortmann in [Ort03]). Diese erlaubt es, die Auswirkungen einer Nachricht bezüglich eines Dienstes zu explizieren. Die Hauptarbeit in der Nutzung einer solchen Anbindung besteht jedoch in der *Erstellung* von anwendungsbezogenen Ontologien. Zur Zeit sind die für Anwendungen verwendeten Ontologien noch implizit festgelegt. Dieses geschieht in CAPA in drei Bereichen: in ACL- und SL-Hilfsklassen, in Protokolleinträgen der Wissensbasen von Agenten sowie innerhalb von Protokollnetzen. Die Programmierung der oben erwähnten SL-Hilfsklassen für Nachrichtentypen stellt bereits eine gewisse Zentralisierung dieser Informationen dar.

5.3 Nachrichtentransport

Der Nachrichtentransport in CAPA ist der FIPA folgend in zwei Bereiche geteilt, die eng zusammenarbeiten: einerseits der Transportdienst (*Message Transport Service*, MTS) und andererseits die Transportmittel (*Message Transport Protocol*, MTP). Sie sind in CAPA mit Java-Interfaces spezifiziert: `TransportService` und `Transport`. Die Bezeichnung `Transport` lehnt sich hier an die Bezeichnung der Abstrakten Architektur an (siehe [FIPA 01, S. 12 und 42]).

Die Interfaces für den Nachrichtentransport arbeiten zusammen bei der Initialisierung, beim Nachrichtenempfang und -versand und bei der Suche eines passenden Transportmittels für eine bestimmte Nachricht.

Im Folgenden wird das Zusammenspiel der Interfaces beschrieben, im nächsten Abschnitt deren Implementationen in CAPA: der ACC und die Transportmittel.

5.3.1 Die Interfaces `TransportService` und `Transport`

Das Interface `TransportService` wird in CAPA einmal implementiert, das Interface `Transport` wird für jedes Transportmittel in CAPA implementiert. Es folgt eine Liste der Methoden dieser Interfaces und dann eine Beschreibung der Zusammenarbeit.

TransportService Das Interface `TransportService` spezifiziert die Methoden:

Typ	Name	Parameter
`ApTransportDescription`	`getDescription`	`()`
`void`	`addTransport`	`(Transport transport)`
`void`	`removeTransport`	`(Transport transport)`
`void`	`transportMessage`	`(MessageEnvelope envelope, Object message, URL from, Transport transport)`

Transport Das Interface `Transport` spezifiziert die Methoden:

Typ	Name	Parameter
`boolean`	`canReach`	`(URL address)`
`String[]`	`getAcceptedRepresentations`	`()`
`MtpDescription`	`getDescription`	`()`
`void`	`setTransportService`	`(TransportService service)`
`String`	`transportMessage`	`(URL to, MessageEnvelope envelope, Object message)`

Initialisierung Bei der Initialisierung der Plattform werden (fest codiert) die Konstruktoren der bekannten Transportmittel aufgerufen.

In einem zweiten Schritt wird jedes erzeugte Transportmittel mit der Methode `TransportService.addTransport(Transport)` einer Liste von verfügbaren Transportmitteln hinzugefügt. Dabei wird auch die Transportbeschreibung der Plattform um das Transportmittel erweitert (`Transport.getDescription()`) und dem Transportmittel wird der zentrale Transportdienst bekannt gemacht (mit dem Aufruf `Transport.setTransportService((TransportService)this)`).

Nachrichtenempfang und -versand Wenn ein Transportmittel eine Nachricht empfängt (von einem Agenten bei einem internen Transportmittel oder von einem Transportmittel einer anderen Agentenplattform) muss es die Nachricht an den ACC mit `TransportService.transportMessage(envelope,message,from,this)` weiterleiten. Die Parameter `envelope` und `message` bilden die empfangene Nachricht, die Parameter `from` und `this` dienen dem ACC zur Erstellung eines neuen Feldes `received` in der Nachricht.

Die Aufforderung eine Nachricht zu versenden teilt der ACC dem Transportmittel durch die Methode `Transport.transportMessage(URL to, MessageEnvelope envelope, Object message)` mit.

Suche eines passenden Transportmittels Um festzustellen, ob ein Transportmittel für den Transport einer bestimmten Nachricht geeignet ist, muss der ACC zwei Dinge wissen: ob die Adresse für das Transportmittel erreichbar ist und ob die Repräsentierung der Nachricht von dem Transportmittel unterstützt wird. Ist letzteres nicht der Fall, kann der ACC eine Konvertierung anhand der verfügbaren Transformer erwägen. Die Methoden für diese Fragen sind `Transport.canReach(URL address)` und `Transport.getAcceptedRepresentations()`.

Terminierung Die Schnittstellen müssen bei der Terminierung des Transportdienstes zusammenarbeiten, damit die Transportmittel die Ports ordnungsgemäß schlie-

ßen können. Da allgemein noch kein Mechanismus zur Terminierung der Plattform existiert, ist diese Funktion in der Schnittstelle zwischen ACC und Transportmittel nicht vorgesehen.

5.3.2 Der Agentenkommunikationskanal (ACC)

Der ACC ist die Implementierung des Interfaces `TransportService` als Referenznetz, so dass mit möglichst geringem Programmieraufwand eine voll funktionsfähige und nebenläufige Version möglich war.

Die Schritte, die für jede ankommende Nachricht durchgeführt werden, enthalten die oben beschriebenen Schritte zum Nachrichtenempfang und -versand und der Suche nach einem passenden Transportmittel:

- Der Nachrichtenumschlag wird bei mehreren Empfängern nach Adressengruppen aufgeteilt und die Nachricht entsprechend oft kopiert. Diese Nachrichten werden danach einzeln behandelt.
- Aus dem Umschlag werden die Adressen der Empfänger gelesen.
- Für die erste Adresse wird ein passendes Transportmittel ausgewählt.
- Die Repräsentierung der Nachricht wird mit Hilfe eines Transformers für den aktuell probierten Transport angepasst.
- Die Nachricht wird dem Transportmittel übergeben. Falls es ein internes Transportmittel ist, werden die einzelnen Empfänger getrennt übergeben.

5.3.3 Die Transportmittel (MTPs)

In CAPA sind zwei Transportmittel implementiert: ein Transportmittel für die plattforminterne Nachrichtenzustellung und ein nicht standardisiertes Transportmittel für die plattformübergreifende Kommunikation im Computernetz.

Das interne Transportmittel vermittelt Nachrichten in der internen Java-ACL-Darstellung direkt über synchrone Kanäle zwischen dem ACC und den Agenten. Es ist aus der Implementierung von MULAN entstanden. Das Transportmittel für plattformübergreifende Kommunikation benutzt eine Socket-Verbindung zwischen Rechnern, um Nachrichten in String-Darstellung zu übermitteln. Als Adressen dienen URLs mit dem Protokollbezeichner `gopher`, um die Klasse `URL` von Java benutzen zu können.

Beide Transportmittel sind als Netze implementiert, deren Stub-Klassen das Interface `Transport` implementieren.

5.4 Allgemeines zu Capa

Dieser Abschnitt gibt einen Überblick über CAPA, nachdem die Elemente der Plattform in den vorigen Abschnitten bezogen auf das Architekturbild von CAPA am

Anfang des Kapitels (Seite 69) beschrieben wurden. Es werden die Konfiguration der Plattform und der MulanViewer vorgestellt, danach eine Entwurfsmethode für Multiagentensysteme in Capa. Zum Ende werden in einer knappen Liste die in Capa berücksichtigten Spezifikationen der FIPA zusammengestellt und alternative Agentenplattformen in Verbindung mit Agentcities.

5.4.1 Initialisierung und Konfiguration

Capa ist in Renew als Entwicklungs- und Laufzeitumgebung eingebunden. Die Startprozedur durchläuft verschiedene Etappen mit jeweiligen Konfigurationsmöglichkeiten. Abgesehen von einer agentenorientierten Anwendung, die auf der Plattform läuft, besteht vollständiger Lauf von Capa aus folgenden Schritten:

- Renew mit dem Startskript starten
- Die Simulation mit dem Aufrufnetz starten
  - Start der Plattform
  - Start von Agenten
    (agentenbasierte Anwendung läuft)
  - evtl. Terminierung und Neustart von Agenten
- Terminierung der Plattform zusammen mit Renew

Im Startskript werden Pfade und Optionen gesetzt, die z.T. auch über die Kommandozeile übergeben werden können. Im Startskript wird z.B. festgelegt, ob die Renew-GUI angezeigt werden soll oder ob Renew nur eine textuelle Eingabeaufforderung anzeigen soll.

Im Aufrufnetz wird zunächst der Plattformname festgelegt und die Plattform mit `AMS`, `DF` und Plattform-Agent erzeugt, dann wird eine Nachricht an den `AMS` geschickt, in der die ersten Agenten mit ihrem Namen und ihrem Anfangswissen auf der Plattform erzeugt werden. Ein Agent kann dann die Erzeugung weiterer Agenten veranlassen. Häufig bietet das Aufrufnetz auch eine Möglichkeit, mit einer Nachricht an den `AMS` die anwendungsbezogenen Agenten auf der Plattform zu beenden. Danach können sie mit derselben Aufrufnetzinstanz wieder erzeugt werden, z.B. mit modifiziertem Anfangswissen.

Die Plattform kann nicht zufriedenstellend terminiert werden, ohne Renew zu beenden, weil für die Transportmittel keine Benachrichtigung beim Beenden vorgesehen ist. Der Port, an dem das TCP-Transportmittel lauscht, wird deshalb erst beim Beenden von Renew geschlossen. Ein Neustart der Plattform ist nicht möglich, wenn das Transportmittel nicht erfolgreich initialisiert werden kann (d.h. es kann nicht an dem schon geöffneten Port lauschen).

Die bestehende Initialisierung des Transportdienstes erlaubt eine Konfiguration nur direkt im Quellcode (siehe Abschnitt *Initialisierung* auf Seite 78).

Zur Konfiguration gibt es neben dem Startskript, seinen Kommandozeilenparametern und dem Aufrufnetz noch eine *Properties*-Datei für Renew, in der das

allgemeine Verhalten von RENEW gesteuert werden kann (z.B. Vorbelegungen für Kommandozeilenparameter oder Pfadangaben zu Plug-Ins für RENEW).

5.4.2 Der MulanViewer

Der MulanViewer wurde nach ersten Erfahrungen mit der Anwendungsentwicklung mit CAPA speziell zur Beobachtung der Plattform entwickelt (von Timo Carl, siehe [Car03]). Die Abbildung 5.3 zeigt einen *Screenshot* des MulanViewers. Zu sehen ist links eine Liste mit den Agenten auf der Plattform (`AMS`, `DF`, `Ping#1`

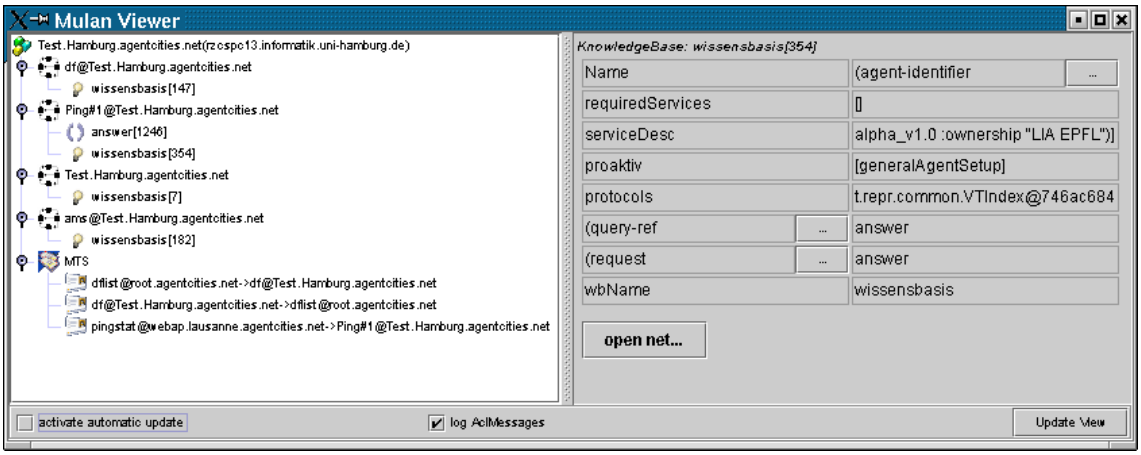


Abbildung 5.3: Der MulanViewer

und der Plattform-Agent `Test.Hamburg.agentcities.net`), deren Wissensbasen (mit den Nummern der Netzinstanzen in eckigen Klammern dahinter), ein laufendes Protokoll (`answer` beim Ping-Agent) sowie einige Nachrichten (in der Form `Absender`→`Empfänger`).

Im rechten Fensterteil kann Detailinformation zu einem ausgewählten Punkt der linken Seite angezeigt werden. Zu sehen ist das Wissen des Ping-Agenten in einer Schlüssel-Wert-Tabelle (links ist die Schlüsselseite, rechts die Werte-Seite). Der erste Eintrag (`Name`) enthält einen mehrzeiligen `Agent-Identifier` in String-Darstellung. Der volle Eintrag wird sichtbar, wenn man auf das Feld rechts klickt. Der Eintrag `requiredServices` (benötigte Dienste) zeigt einen leeren Vektor, das Feld für die angebotenen Dienste (`serviceDesc`) enthält die von Agentcities vorgeschriebene Beschreibung des Ping-Dienstes. Das Protokoll `generalAgentSetup` wurde beim Start des Agenten proaktiv ausgeführt. Die verfügbaren Protokolle sind im nächsten Eintrag gespeichert und in den beiden nachfolgenden Einträgen lesbar dargestellt: Der Ping-Agent wird das Protokoll `answer` auf zwei verschiedene Nachrichtentypen verwenden. Die vollständigen Muster werden mit Hilfe der Knöpfe rechts im Schlüsselfeld angezeigt. Schließlich ist der Name der Wissensbasis eingetragen. Un-

ter der Darstellung des Wissens ist über den Button `open net...` die Netzinstanz der Wissensbasis zugänglich. Um das laufende Protokoll zu betrachten, wählt man `answer[1246]` aus und kann dann über den Button `open net...` die Netzinstanz öffen, welche den Zustand der Konversation zeigt.

Die Details zu den Elementen der Plattform sind:

- für Agenten: Zustand (meistens aktiv) und Zugriff auf die Agentennetzinstanz
- für Wissensbasen: Zustand wie im Screenshot rechts gezeigt (aktuelles Wissen des Ping-Agenten) und Zugriff auf die Netzinstanz
- für Protokolle: Konversations-Identifikation und Zugriff auf die Protokollnetzinstanz, d.h. auf den Zustand der Konversation
- für Nachrichten: Inhalte aller Felder einer Nachricht.

Mit den Netzinstanzen stehen dann die vielfältigen Inspektionsmöglichkeiten von Renew uneingeschränkt zur Verfügung (siehe Abschnitt 2.3 (Renew) und Abschnitt 2.2.1 auf Seite 17). Insbesondere kann durch die erweiterte Tokendarstellung der Java-Aufbau einer Marke direkt inspiziert werden (ein Beispiel ist in Abbildung 2.2 auf Seite 18 dargestellt).

5.4.3 Entwurf von Multiagentensystemen

Der Entwurf eines Multiagentensystems besteht wesentlich aus Interaktionsdesign und Informationsdesign. Zur Realisierung in Capa müssen die Protokolle der Agenten implementiert und der Inhalt der Wissensbasen programmiert werden. Die Abbildung 5.4 gibt einen Überblick über die Vorgehensweise, die bei der Entwicklung des Siedler-Spiels (siehe Kapitel 7 (*Eine Anwendung: Siedler*)) verwendet wurde. In diesem Fall gab es die Originalvorlage des Brettspiels und es lagen bereits Ergebnisse eines vorigen Projektlaufs vor; die Anforderungen wurden in Form einer Liste von Anwendungsfällen (Use-Cases) angegeben.

Um danach ein Agentensystem zu entwerfen, werden funktionale Rollen mit Hilfe der Anwendungsfälle gegeneinander abgegrenzt. Die Rollen werden konzeptuell zu Agenten zusammengefasst. Pro Agent muss eine Wissensbasis entworfen werden und pro Anwendungsfall in der Regel ein AUML-Interaktionsdiagramm. Die Interaktionsdiagramme werden verfeinert und schließlich mit Hilfe von Netzkomponenten in Netzstrukturen implementiert. Zuletzt werden die Anschriften des Protokollnetzes angepasst. Für jedes Protokoll muss ein passender Eintrag in die Wissensbasis hinzugefügt werden. Einzelne Rollen können kann anderen Agenten zugeordnet werden, indem der entsprechende Teil der Wissensbasis übertragen wird.

Bei diesem Prozess spielen die Netzkomponenten eine wichtige Rolle. Das Konzept der Netzkomponenten wurde im Abschnitt 2.4.2 auf Seite 22 bereits angedeutet. Die Protokollnetze für Capa bilden die Anwendungsdomäne, anhand derer Lawrence Cabac das Konzept der Netzkomponenten entwickelt hat (siehe [Cab02]). Die in jedem

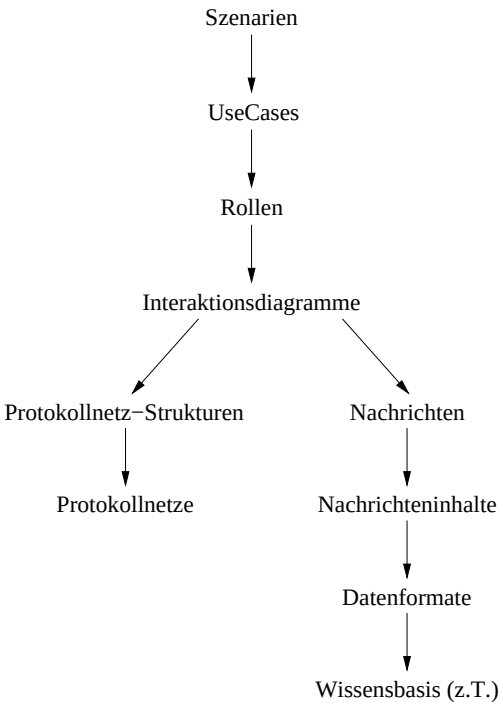


Abbildung 5.4: Eine Entwurfsmethode für Agenten

Protokoll wiederkehrenden Aufgaben (Beginn und Ende der Konversation, Nachrichtenempfang und -versand, Warten auf Antwortnachrichten und Zugriff auf die Wissensbasis) werden mit Netzkomponenten entworfen. Sie erleichtern die Programmierung von Protokollnetzen und verbessern die Lesbarkeit. Die Netzkomponenten bilden jedoch vor allem auch eine enge Verbindung zwischen Interaktionsdiagrammen und Protokollnetzen (diese Verbindung wird in [CMR03] hergestellt). Daraus ergibt sich als großer Vorteil dieser Entwurfsmethode ein fließender Übergang von der Designphase zur Implementierungsphase.

Beim Entwurf einer Wissensbasis müssen die Schlüssel und Datenformate sowie Abhängigkeiten in der Wissensbasis auf die Anforderungen der verschiedenen Interaktionen abgestimmt sein. Die Wissensbasen müssen getrennt von den Protokollen entwickelt werden, weil der Formalismus der Interaktionsdiagramme keine Möglichkeit bietet, Vorgänge innerhalb eines Agenten zu beschreiben. So kann der eben angedeutete Vorteil für die Wissensbasen nicht genutzt werden. Die Interaktionsdiagramme könnten zu diesem Zweck erweitert werden, allerdings muss die Modellierungsebene klar sein (also der Übergang zwischen *Inter*-Aktion und *Intra*-Aktion)[9].

Falls besondere Anforderungen an die Verarbeitung des Wissens eines Agenten

[9]Z.B. könnte die Wissensbasis als Rolle in das Interaktionsdiagramm eingetragen werden. Das hängt eng mit dem Verhältnis zwischen Agent und Rolle zusammen.

gestellt werden, die über die oben beschriebenen Zugriffsmöglichkeiten hinausgehen, kann ein modifiziertes Netz für die Wissensbasis entworfen und beim Start des Agenten angegeben werden (z.B. kann die Standard-Wissensbasis nicht auf dem vorhandenen Wissen schließen[10]).

Abbildung 5.4 zeigt zusammengefasst die Schritte beim Entwurf eines Multiagentensystems. Indem diese Schritte ausgearbeitet und untersucht werden und die Werkzeugunterstützung weiter ausgebaut wird, kann eine umfassende und intuitive Entwurfsmethode für Multiagentensysteme entstehen. Das soll jedoch nicht im Rahmen dieser Diplomarbeit geschehen.

5.4.4 Capa und FIPA

Bei der Realisierung von Capa fanden folgende Spezifikationen der FIPA Anwendung (Informationen entnommen und zusammengefasst aus [Duv02, S. 136f]):

[FIPA 61] Nachrichtenaufbau

[FIPA 70] Nachrichtendarstellung

[FIPA 23] Teile der Ontologie, Agentenverwaltung und Kommunikation dazu

[FIPA 67] Teile der Ontologie und Nachrichtenversand

[FIPA 08] SL zur Kommunikation für die Agentenverwaltung

[FIPA 01] Gestaltung der Transformer, Klassenstruktur des Transportdienstes

[FIPA 37] Verwaltungsfunktionen nutzen die hier spezifizierten Performative

[FIPA 26] Anfragen an Verwaltungsfunktionen folgen dem *Request*-Protokoll

[FIPA 87] Mobilität hat die Plattformimplementierung nur gering beeinflusst.

5.4.5 Capa und Agentcities

Agentcities verbindet bereits eine Vielfalt von FIPA-kompatiblen Agentenplattformen, von denen einige frei verfügbar sind.

Die unten aufgeführten Plattformen werden (bzw. wurden) alle im Agentcities-Agentennetz verwendet und sind frei verfügbar (vgl. [act02b, S. 9]). Der Vergleich der Vorzüge und Schwächen der einzelnen Plattformen und Capa ist nicht Bestandteil dieser Arbeit, eine solche Liste kann jedoch der Ausgangspunkt für einen umfassenden Vergleich von Plattformen sein. Capa gehört nach der erfolgreichen Anbindung an Agentcities ebenfalls in diese Liste.

[10] Im Sinne der Logikprogrammierung

April Agent Platform (AAP)

- Entwickler: Fujitsu Laboratories of America, California, USA
- Im Netz verfügbar: `http://www.sourceforge.net/projects/networkagent/`
- April ist eine Programmiersprache zur Agentenkonstruktion. AAP läuft auf Mac und Linux.

BlueJade

- Entwickler: Hewlett Packard Laboratories, California, USA
- Im Netz verfügbar: `http://www.hpl.hp.com/org/stl/maas/`
- Besonders betont wird die Mobilitätsforschung

Comtec Agent Platform

- Entwickler: Communication Technologies, Sendai, Japan
- Im Netz verfügbar: `http://ias.comtec.co.jp/ap/`
- Implementiert die FIPA-Spezifikationen von 1997 und 1998. Im Januar und Februar 1999 fanden erste Versuche zur Verbindung verschiedener Plattformen in einem Netz statt.

FIPA-OS

- Entwickler: Emorphia, Harlow, UK
- Im Netz verfügbar: `http://fipa-os.sourceforge.net/`
- Die *Agent Technology Group* trat 1997 der FIPA bei und präsentierte 1999 das erste FIPA-kompatible Produkt.

Java Agent Development Environment (JADE)

- Entwickler: Telecom Italia Laboratories, Turin, Italien
- Im Netz verfügbar: `http://jade.cselt.it/`
- Heute die bekannteste Agentenplattform, die bei Agentcities große Verbreitung gefunden hat. Sie hat einen Ping-Agenten, einen Dummy-Agenten zum manuellen Empfangen und Versenden von Nachrichten und einen Sniffer-Agenten, der die Kommunikation visualisieren kann.

Lightweight Extensible Agent Platform (LEAP)

- Entwickler: EU IST LEAP Project Consortium
- Im Netz verfügbar: `http://leap.crm-paris.com/`
- Die Betonung liegt auf der Lauffähigkeit auf kleinen Geräten wie z.B. Handys.

Zeus

- Entwickler: BTexact Technlologies, Adastral Park, UK
- Im Netz verfügbar: `http://www.btexact.com/projects/agents/zeus/`
- Entwurf von intelligenten Agenten mit visueller Unterstützung.

6 Anbindung von Capa an Agentcities

Nachdem in den vorigen Kapiteln alle Grundlagen und Voraussetzungen vorgestellt wurden, wird in diesem Kapitel die Anbindung von Capa an Agentcities beschrieben. Es soll also eine bisher proprietäre und nicht in Zusammenarbeit mit anderen Plattformen getestete Agentenplattform – Capa – an das internationale Agentennetz Agentcities angebunden werden. Die Grundlagen zur Agententechnologie wurden in Kapitel 3 (*Agenten*) beschrieben, Agentcities wurde im Kapitel 4 vorgestellt und Capa im Kapitel 5. Da Capa mit Referenznetzen in Renew implementiert wurde und auch die nötigen Erweiterungen z.T. in Referenznetzen durchgeführt werden, wurden diese Grundlagen im Kapitel 2 (*Referenznetze und* Renew) eingeführt.

Zunächst werden in diesem Kapitel die Anforderungen zusammengetragen, die sich aus den Kapiteln 4 und 5 ergeben. In den zwei darauf folgenden Abschnitten werden die beiden zentralen Teile der Realisierung eingehend vorgestellt: das HTTP-Transportmittel und der Ping-Agent. Danach wird in einem dritten Abschnitt beschrieben, wie mit diesen beiden Bausteinen die Anbindung von Capa an Agentcities gelang. Im letzten Abschnitt dieses Kapitels sind unter der Überschrift *Zusammenfassung und Ausblick* die Ergebnisse zusammengefasst und bewertet.

6.1 Anforderungen und Lösungsansätze

Aus der Kombination der Informationen aus den vorhergehenden Kapiteln können die Anforderungen für diese Arbeit abgeleitet werden. In der Einleitung wurde ein Ziel angedeutet, welches zunächst konkretisiert wird, danach werden die Anforderungen zusammengestellt. Zu den einzelnen Anforderungen wird das Prinzip der Lösung jeweils mit angegeben.

6.1.1 Zielsetzung

Das Ziel der Implementierungen im Rahmen dieser Arbeit ist die *aktive* Anbindung von Capa an Agentcities. Das bedeutet, dass eine Plattform im Verzeichnis der aktiven Plattformen auf der Webseite [acn03, /globalapd.jsp] erscheint. Die *vollständige* Anbindung wird bei der Umsetzung und Anforderungsermittlung einbezogen, um auch diese möglichst umzusetzen.

6.1.2 Anforderungen mit Lösungsansatz

Den Ausgangspunkt für die Sammlung der Anforderungen bildet die Liste von Anforderungen, die Agentcities an eine anzubindende Agentenplattform stellt. Diese werden ergänzt um die konkreten Spezifikationen der FIPA zu den einzelnen Punkten und reduziert um die Eigenschaften, die CAPA und die Arbeitsumgebung am Informatik-Rechenzentrum der Universität Hamburg bereits erfüllen.

Die Gliederung dieses Abschnitts folgt der Gliederung des Abschnitts 4.3.3 (*Anforderungen zur Anbindung einer Plattform*). Zur Ergänzung werden die in Abschnitt 3.4 erläuterten Spezifikationen der FIPA verwendet und durch einen Vergleich mit der Beschreibung zu CAPA werden die zu erfüllenden Anforderungen ermittelt.

Allgemeine Voraussetzungen

Auf jedem Arbeitsplatz am Rechenzentrum der Hamburger Informatik gibt es Internetzugang mit einer festen IP-Adresse und einem `DNS`-Namen sowie die Erlaubnis, Nachrichten zu verschicken. An einem Port auf eintreffende Nachrichten zu lauschen ist jedoch nur innerhalb des Informatik-Rechnernetzes erlaubt. Mit einem Antrag kann die Erlaubnis für einen bestimmten Port auf einem bestimmten Rechner eingeholt werden. Das bedeutet, dass für CAPA in Agentcities vorerst immer derselbe Rechner mit immer demselben Port verwendet werden wird.

Diese Anforderung zieht eine weitere Anforderung nach sich, die hier aufgegriffen wird: Die Plattform muss dauerhaft im Hintergrund als Serverprozess laufen, weil die Rechner von wechselnden Personen genutzt werden. Die Voraussetzung für die Erfüllung dieser Anforderung waren Weiterentwicklungen von RENEW parallel zu dieser Diplomarbeit, die es erstens ermöglichen, eine in RENEW laufende Simulation von einem entfernten Rechner aus mit RENEW zu inspizieren[1], zweitens erlauben die Weiterentwicklungen, RENEW komplett ohne grafische Oberfläche zu starten. Nur eine textuelle Eingabeaufforderung bietet in diesem Fall lokalen Zugriff auf eine laufende Simulation, der Zugriff über die grafische Oberfläche kann dank der zuerst genannten Weiterentwicklung mit einem zweiten RENEW durchgeführt werden.

Damit sind die wesentlichen Voraussetzungen erfüllt, die es ermöglichen, RENEW im Hintergrund als Serverprozess laufen zu lassen. Vom Rechenzentrum aus muss es daneben noch die Möglichkeit geben, einen Prozess ohne Zeitbeschränkung und ohne Beschränkung der Rechenleistung auch nach dem Ausloggen laufen zu lassen. Die Beschränkung für die Laufzeit und die Rechenleistung kann vor dem Aufruf von RENEW mit dem Unix-Befehl `ulimit` aufgehoben werden. Damit der Prozess nach dem Ausloggen weiterläuft, wird `screen` benutzt, ein Manager für virtuelle Konsolen. Ein laufender Prozess kann mit `screen` von der Konsole abgekoppelt und nach dem Einloggen wieder mit einer sichtbaren Konsole verbunden werden. Erst

[1] Die Grundlage dafür ist die Arbeit von Thomas Jacob, siehe [Jac02].

die Kombination der vier genannten Punkte (Renew ohne GUI, Renew remote inspizierbar, Rechenzeit und Rechenleistungsbeschränkung aufhebbar und Prozess abkoppelbar von einer sichtbaren Konsole) ermöglicht einen „Dauerlauf“ von Capa.

Bereitstellung einer Plattform

In Abschnitt 5.4.4 sind einige frei verfügbare Agentenplattformimplementationen vorgestellt, die die Anforderungen an eine Plattform erfüllen und in Agentcities bereits eingesetzt werden. Da es hier um die Anbindung einer neuen Plattform an Agentcities geht, müssen die Anforderungen näher beleuchtet werden. Zunächst müssen die Sichtweisen von Agentcities und Capa darüber, was eine Agentenplattform ist, konzeptuell zueinander passen. Nur wenn die Konzepte sich ergänzen oder übereinstimmen, ist eine weitere Betrachtung sinnvoll.

Agentcities definiert eine Plattform über die Spezifikationen der FIPA. Die Plattformen werden von Forschungsgruppen verwaltet. Die Kommunikation ist der FIPA folgend mit Protokolldiagrammen in AUML spezifiziert. Angemeldete Plattformen werden regelmäßig mit einem `Ping` angefragt, ob sie aktiv sind. Damit gibt Agentcities im Wesentlichen eine Kommunikationsarchitektur für die aktiven Plattformen vor.

Capa gibt für die Agenten eine Entscheidungsarchitektur vor. Die Handlungsmöglichkeiten eines Agenten sind mit Protokollnetzen spezifiziert, die über die Wissensbasis koordiniert werden. Zentrale Teile von Capa sind mit Petrinetzen implementiert. Den Spezifikationen der FIPA folgend besitzt Capa einen zentralen Agentenkommunikationskanal (ACC).

Diese beiden Teile überschneiden sich, indem sich beide auf die FIPA beziehen. Sie ergänzen sich, weil Capa die Entscheidungsarchitektur für Agenten liefert, und Agentcities einen Agenten (`Ping`) für den regelmäßigen Kontakt fordert. Die in Agentcities verwendeten Spezifikationen der FIPA sind im Abschnitt 4.3.1 auf Seite 56 aufgelistet und im Abschnitt 3.4.2 auf Seite 37 einzeln vorgestellt, soweit sie hier benötigt werden. Bei der Standardisierung von Capa, wie sie von Duvigneau durchgeführt wurde, fanden die in Abschnitt 5.4.4 (Capa *und FIPA*) aufgelisteten Nummern der FIPA-Spezifikationen Anwendung.

Im Vergleich zur Liste der in Agentcities benötigten Standards in Abschnitt 4.3.1 fehlen noch [FIPA 84] für das HTTP-Transportmittel und [FIPA 85] für die Umschlagsdarstellung in XML. [FIPA 75] (Transportmittel für IIOP) wird zusätzlich genannt, ist aber optional und braucht deshalb hier nicht berücksichtigt zu werden.

Es bleiben also das Transportmittel für HTTP und die Darstellung des Nachrichtenumschlags in XML als zu realisierende Standards übrig. Es steht zu erwarten, dass die Anbindung mit der Implementierung der Spezifikationen noch nicht funktionsfähig ist, sondern CAPA noch an die konkreten Bedingungen in Agentcities angepasst werden muss. Duvigneau vermutet bereits bei der Umsetzung der Standards, „dass die Zusammenarbeit mit anderen FIPA-konformen Plattformen auch bei vorhandenem Transportprotokoll nicht reibungslos klappt. Eine Abstimmung der beteiligten Plattformen auf einen gemeinsamen 'faktischen Standard' dürfte nötig werden." [Duv02, S. 137]. Das ist jedoch erst mit einer minimalen Kommunikationsmöglichkeit testbar. Deshalb hat das HTTP-Transportmittel die erste Priorität bei der Umsetzung im Rahmen dieser Arbeit. Die dazu notwendigen Informationen und die Umsetzungen sind im nächsten Abschnitt 6.2 (*Das HTTP-Transportmittel*) beschrieben. Die Dateien für das HTTP-MTP sollen zunächst Teil von CAPA sein. Später ist es wünschenswert, sie in einer Komponente zusammenzufassen. Momentan ist dies nicht notwendig, weil bei der Initialisierung der Plattform alle Transportmittel fest im Code einprogrammiert sind und es deshalb keine modulare Startkonfiguration gibt, wie es bereits im Abschnitt 5.3.1 (*Die Interfaces `TransportService` und `Transport`*) beschrieben wurde.

Entwurf und Installation eines Ping-Agenten

Mit Hilfe der Bausteine für einen neuen Agenten in CAPA soll ein Ping-Agent entworfen werden, der den im Folgenden beschriebenen Anforderungen genügt.

Agentcities Der Ping-Agent muss auf eine empfangene Nachricht korrekt antworten. Die Anfrage und die Antwort darauf sind in Abschnitt *Spezifikation des Ping-Agenten* auf Seite 63 spezifiziert.

Bei der Anmeldung einer Plattform in Agentcities muss der unqualifizierte Name des Ping-Agenten mit angegeben werden. Agentcities konstruiert den Namen des Ping-Agenten für die Anfrage dann aus diesem unqualifizierten Namen und dem Plattformnamen, die Adresse wird aus der Plattformbeschreibung entnommen.

Capa Nach der Beschreibung in Abschnitt 5.1 auf Seite 68 muss ein Agent in CAPA eine Wissensbasis und Protokolle haben. Die Wissensbasis des Ping-Agenten muss neben der Initialisierung (anmelden beim lokalen `AMS` und `DF`) ein Muster der Nachricht enthalten, auf die er antworten soll. In dem dann aufgerufenen Protokoll muss der Ping-Agent auf die empfangene Nachricht korrekt antworten. Zur Umsetzung wird ein Interaktionsdiagramm formuliert.

Agentennamen werden in CAPA mit einer laufenden Nummer versehen, um Namenskonflikte zu vermeiden. Die Anmeldung und Konstruktion des Agentennamen unter Agentcities macht hier eine Anpassung oder sorgfältige Startkonfiguration von CAPA notwendig.

Das Ergebnis des Ping-Entwurfs wird aus insgesamt vier Dateien bestehen: die Wissensbasis und das Protokollnetz des Agenten sowie ein Startskript und ein Aufrufnetz zur Konfiguration der Plattform. Zunächst soll der Ping-Agent Teil der Plattform sein, später als wesentlicher Bestandteil von ACE abkoppelbar sein und damit austauschbar gegen Infrastrukturen für andere offene Plattform-Netze. Die Entwicklung und das Ergebnis sind nach der Beschreibung des HTTP-MTP (Abschnitt 6.2) in Abschnitt 6.3 (*Der Ping-Agent*) dargestellt.

Anmeldung einer Gruppe und einer Plattform in Agentcities

Um eine Gruppe und eine Plattform in Agentcities zu registrieren, müssen zwei Formularseiten in Agentcities ausgefüllt werden. Die Gruppe bekommt den Namen TGI und repräsentiert damit den Arbeitsbereich „Technische Grundlagen der Informatik“ der Universität Hamburg, in dessen Rahmen diese Arbeit und die Entwicklung von Capa stattfanden. Die unmittelbar beteiligten Personen werden als Gruppenmitglieder eingetragen. Die Plattform soll den Namen `Hamburg.agentcities.net` erhalten. Die zur Anbindung wesentliche Plattformbeschreibung wird innerhalb von Capa nicht automatisch erstellt, deshalb wird sie manuell aus der FIPA-Spezifikation und aus den Daten des dann realisierten HTTP-MTPs erstellt und in die Formularseiten eingetragen.

6.2 Das HTTP-Transportmittel

Es soll in diesem Abschnitt ein Transportmittel für HTTP nach FIPA zu Capa hinzugefügt werden. Dabei sind verschiedene Herangehensweisen möglich:

Ein Transportmittel programmieren Ein neues Element in Capa kann entweder hauptsächlich mit Referenznetzen oder in Java implementiert werden. Duvigneau merkt an, Netze seien zur Implementierung von Transportmitteln nur geeignet, weil die Beobachtung der Marken ein komfortables Debugging ermöglicht. Aus Sicht der Effizienz sei Java vorzuziehen und aus Sicht der Nebenläufigkeit sei Java in diesem übersichtlichen Fall zu vertreten (vgl. [Duv02, S. 116]). Zur Implementierung einer HTTP-Verbindung mit Java kommt die API in `java.net` in Frage, die z.B. die Klasse `HttpURLConnection` anbietet.

Einen Webserver anpassen Es exisitieren andererseits eine ganze Reihe von frei verfügbaren Webservern, z.B. bietet SUN einen technischen Artikel mit einem kleinen Webserver an (siehe [Bro02]). Bei einem Webserver ist die Benutzung von HTTP bereits realisiert, alles Weitere muss programmiert werden.

Eine vorgefertigte Komponente nutzen Die Recherche ergab weiter, dass fertige Transportmittel speziell für Agentenplattformen nach FIPA frei verfügbar

sind. Insbesondere bot Agentcities eine modulare Implementation eines `ACC`s unter dem Namen *Java Agent Message Router* (JAMR) an, deren Transportmittel für HTTP und IIOP auch einzeln benutzbar sind.

Aufgrund des engen Zeitrahmens und der Situation im Arbeitsteam schien es sinnvoll, eine möglichst zeitsparende Variante zu wählen.

Ich habe mich entschieden, das HTTP-Transportmittel von JAMR zu nutzen, weil die Spezifikationen bereits berücksichtigt sind, weil es gebrauchsfertig geliefert wird und weil dieses Transportmittel Nachrichten mit JADE-Plattformen austauschen kann (dies ist wichtig, weil der Ping-Dienst von Agentcities ebenfalls eine JADE-Plattform nutzt). Dabei sind Umfang und Funktionalität exakt auf die zu erledigende Aufgabe zugeschnitten, nämlich FIPA-konforme Nachrichten zwischen Agentenplattformen auszutauschen. Insbesondere umfasst es bereits die Darstellung des Nachrichtenumschlags in XML, ist also in der Lage, beide fehlenden Spezifikationen abzudecken ([FIPA 84] und [FIPA 85]). Es ist frei verfügbar und wurde angeboten, um neuen Plattformen den Zutritt zu Agentcities zu erleichtern. Allerdings wird auf diese Weise eine zweite Implementation des Nachrichtenumschlags und ACL zu Capa hinzugefügt. Die zusätzliche Konvertierung bewirkt Reibungsverluste, die jedoch durchaus tolerierbar sind, solange nur einzelne und relativ kleine Anwendungen in Capa laufen.

6.2.1 Das HTTP-MTP von JAMR

Die aus dem Netz geladene Datei (siehe [jam02]) enthält ein gepacktes Verzeichnis mit Quellen, eine kurze Dokumentation, Lizenzbestimmungen und Angaben in Listenform zur Version und zu enthaltenen Paketen.

Bei der ersten Kompilierung stellte sich heraus, dass mehr Klassen geliefert wurden, als für das HTTP-MTP notwendig sind, weil das HTTP-MTP nur einen Teil der gesamten Implementation von JAMR ausmacht. Statt die Klassen vor der Auslieferung auseinander zu sortieren, wurde in der Dokumentation angegeben, nur drei relevante Pakete zu kompilieren. Aus den anderen Paketen werden nur einige Klassen benötigt. Diese benötigten Klassen werden vom Java-Compiler aufgrund von Klassenabhängigkeiten kompiliert. Die übrigen, nicht benötigten Klassen *können* nicht kompiliert werden, weil sie ihrerseits von weiteren Klassen von JAMR abhängen, die nicht mitgeliefert sind. Die benötigten Dateien lassen sich nach dem Kompilieren anhand der generierten Klassen identifizieren. Das ist wichtig, weil die Quellen von Capa von dem Produktionsmanagement-Werkzeug `ant` vollständig übersetzt werden. Aus diesem Grund dürfen keine nicht-kompilierbaren Klassen zu Capa hinzugefügt werden. Die benötigten Dateien wurden mit Hilfe eines im Rahmen dieser Arbeit entwickelten Shellskripts von den unbenötigten Dateien getrennt. Dieses Shellskript steht für eine neuere Version des Transportmittels weiter zur Verfügung.

Nachdem die benötigten Klassen von den übrigen getrennt wurden, bleiben Quellen in sieben wesentlichen Paketen, welche ich im folgenden kurz erläutere:

- `jamr.fipa` enthält Interfaces zur Darstellung einer FIPA-konformen Nachricht und deren Bestandteilen (also `Agent-Identifier`, Umschlag etc.), sowie Basisimplementationen dieser Interfaces.
- `javax.jms` enthält Interfaces aus der JMS-Java-API von Sun. JMS steht für *Java Messaging Service* und ist Teil der J2EE (Java 2 Enterprise Edition, Release 1.3).
- `jamr.jms.basic` implementiert die Interfaces aus `javax.jms`.
- `jamr.jms.codec` enthält eine Klasse, die mit XML, HTTP und FIPA-Darstellungen hantiert.
- `jamr.jms.http` enthält Klassen für den Verbindungsaufbau mit HTTP (Server, Clients, Verbindungspool etc.)
- `jamr.util` enthält einige wenige Hilfsklassen.
- `jamr.jms.test` enthält vier Klassen zum Testen des Transportdienstes.

Die Testklassen bilden die Dokumentation des Quellcodes. In der beigelegten Dokumentation werden nur die Kompilation und der Aufruf der Testklassen beispielhaft dargestellt. Innerhalb der Testklassen wird die Benutzung des Transportmittels an einem kleinen Beispiel dargestellt.

Die vier Testklassen bilden zwei Paare mit je einem Sender und einem Empfänger. Mit der einfachen Version lässt sich die Verbindung prüfen (für jede Nachricht wird ein Punkt auf die Konsole geschrieben), die andere Version stellt eine GUI für den Sender und eine für den Empfänger bereit, in der die Nachrichten angezeigt werden.

Nutzung des HTTP-MTPs von JAMR

Aus den Testklassen von JAMR konnte die Benutzung des HTTP-MTPs abgeleitet werden. Zum Start muss ein lokal verfügbarer XML-Reader als Java-Kommandozeilenparameter (`org.xml.sax.parser`) mit angegeben werden (in diesem Fall steht der Parser `org.apache.xerces.parsers.SAXParser` in RENEW zur Verfügung). Zum Verbindungsaufbau werden die Interfaces aus `java.jms` genutzt, die den Entwurfsmustern *Fabrik* und *Listener* folgen. Zur Erstellung einer Nachricht werden die Klassen und Interfaces aus `jamr.fipa` genutzt.

In der Klasse `SenderTest` wird eine Nachricht mit Umschlag erstellt, die dann abgeschickt wird. Der Nutzer hat die Wahl zwischen der Generierung von nummerierten Nachrichten, die in Zufallsabständen in einem eigenen Thread abgeschickt werden und der manuellen Verschickung von Nachrichten mit einem Inhalt, der über die GUI eingegeben wird. Die Klasse `SenderTest` liefert also die Anleitung für die Erstellung einer Nachricht mit Umschlag. Die Klasse `ReceiverTest` liefert die Anleitung für den Zugriff auf eine Nachricht. Anleitung ist hier schon fast zuviel gesagt,

da der Code keinerlei Kommentare enthält, dafür aber leichte Fehler, z.B. wird nach der Erzeugung einer Nachricht diese zweimal in derselben Weise umgewandelt.

6.2.2 Einbindung des HTTP-MTPs in Capa

Um das neue Transportmittel in Capa einzubinden, muss eine Klasse programmiert werden, die beide Schnittstellen erfüllt: das *Interface* `Transport` von Capa (vorgestellt in Abschnitt 5.3.1 (*Die Interfaces `TransportService` und `Transport`*)) und das *Interface* `MessageListener` des JAMR-Transportmittels für lauschende Klassen. In beide Richtungen muss zwischen der internen Darstellung in Capa und der internen Darstellung im HTTP-MTP eine Umwandlung des Nachrichtenformats durchgeführt werden. Die Konversion zwischen der externen Darstellung und dem internen JAMR-Format übernimmt das HTTP-MTP. Solange nur die String-Repräsentierung der Nachricht selbst unterstützt wird, braucht nur eine Umwandlung des Umschlags vorgenommen werden.

Bei der Konvertierung der Umschlags-Formate wurde vorerst auf die Umsetzung der nicht gebräuchlichen Felder (`Payload Encoding`, `Payload Length`, `Transport Behaviour` und `UserDefined`) sowie die Darstellung alter Informationen verzichtet.

Die implementierten Methoden stammen zumeist aus dem Interface `Transport`, nur die Methode `onMessage` im vorletzten Punkt dieser Liste stammt aus dem Interface `MessageListener` von JAMR. Der letzte Punkt fasst die lokalen Hilfsmethoden zusammen.

- `canReach` Diese Methode prüft, ob das Protokoll der angegebenen URL HTTP ist und ob die URL nicht die eigene Adresse enthält. Der Ausschluss der eigenen Adresse dient dazu, Nachrichten mit lokalen Adressen abzulehnen. Bei diesem Vergleich bleiben relative URLs und die Pfadangabe der URL unberücksichtigt. Beim Versuch, solche abgekürzten Adressen zu nutzen, wird die Nachricht endlos „zugestellt“, ohne je anzukommen.

- `getAcceptedRepresentations` Diese Methode gibt die korrekten Bezeichnungen der beherrschten Nachrichtendarstellungen zurück, in diesem Fall ist das `fipa.acl.rep.string.std`.

- `getDescription` Zur Erzeugung der Transpormittelbeschreibung steht in Capa der Konstruktor `MtpDescription("fipa.mts.mtp.http.std", _url)` zur Verfügung.

- `setTransportService` Diese Methode dient der Bekanntmachung des übergeordneten `ACC`s, an den ankommende Nachrichten weitergeleitet werden. Aufgerufen wird sie bei der Initialisierung der Plattform vom `ACC`.

- `transportMessage` Diese Methode wird vom `ACC` aufgerufen, wenn eine Nachricht versendet werden soll. Als Eingabeparameter bekommt sie die Nachricht

als `Object`, den Nachrichtenumschlag im internen Format sowie die `URL`, zu der die Verbindung aufgebaut werden soll. Bei der Übergabe einer Nachricht zum HTTP-MTP von JAMR muss sie eine `javax.jms.TextMessage` zur Verfügung stellen. Vorher muss sie eine Verbindung zur angegebenen `URL` erstellen. Die Methode arbeitet in vier Schritten: Verbindungsaufbau, Umschlag übersetzen, Inhalt hinzufügen und Nachricht abschicken. Der Verbindungsaufbau richtet sich nach der Vorgabe der JAMR-Testklassen. Der Umschlag im JAMR-Format muss aus dem Umschlag im Capa-Format erstellt werden. Dabei sind einige Felder typgleich repräsentiert, wie z.B. die Kommentare und `acl-representation` (als `String`), andere lassen sich leicht konvertieren (Capa liefert ein `Date`-Objekt, das in JAMR als Parameter für einen Konstruktor verwendet werden kann). Diejenigen Felder des Umschlags, die in einer Liste mehrere Werte enthalten (`encrypted`, `receiver` und `intended receiver`), werden stufenweise umgewandelt.

- `onMessage` Diese Methode wird aufgerufen, wenn über HTTP eine Nachricht empfangen wurde. Sie erhält eine `javax.jms.TextMessage`, aus der sie einen Capa-Umschlag generiert und die Nachricht als `String` extrahiert. Außerdem stellt sie die `URL` bereit, über die die Nachricht empfangen wurde. Da aus der `TextMessage` die absendende `URL` nicht ersichtlich ist, wird als Herkunfts-`URL` eine Adresse aus dem Feld `from` des Umschlags genommen. Bei der Übergabe der Nachricht an den lokalen `ACC` muss sich das Transportmittel identifizieren. Diese Methode konvertiert im Wesentlichen den JAMR-Umschlag in einen Capa-Umschlag.

- Als private Methoden wurden zwei Methoden zur Konvertierung der `Agent-Identifier` erstellt, da diese einen rekursiven Aufbau haben.

Das Transportmittel umfasst also neben den Klassen von JAMR nur die hier beschriebene Klasse `HttpMtp`. Streng genommen gehört auch die XML-Readerklasse dazu. In Renew ist XML bereits integriert und kann deshalb genutzt werden. Die Klasse `HttpMtp` ist in die Klassenhierarchie von Capa im Paket `de.renew.agent.transport.http` eingegliedert. Die Klassen von JAMR wurden zum Quellcode von Capa hinzugefügt. Wie bereits in den Anforderungen angedeutet, sollte die Einbindung von Transportmitteln in Capa grundsätzlich eher modular und konfigurierbar gestaltet werden. Durch den fest codierten Aufruf der Transportmittel bei der Plattforminitialisierung scheitert der Start der Plattform, wenn ein Transportmittel nicht erfolgreich initialisiert werden konnte (z.B. weil ein Port nicht mehr verfügbar ist). Beim internen Transportmittel ist dieses Verhalten erwünscht, ein externes, dynamisch ladbares Transportmittel sollte jedoch die laufende Plattform nicht aufhalten. Eine solche Kapselung setzt allerdings einen Mechanismus zur dynamischen Konfiguration von Capa voraus, der nicht im Rahmen dieser Diplomarbeit entwickelt wird.

Dennoch braucht das Transportmittel eine Konfigurationsmöglichkeit, um den Port, auf dem es lauschen soll und den Pfad-Teil der `URL` festzulegen. In CAPA fehlt bisher ein übergreifendes Konzept zur Parametrisierung. So wurde im ersten Entwicklungsschritt die Adresse (`URL`), an der das Transportmittel lauschen soll, fest im Code integriert. Im Laufe der Entwicklung wurde diese Information in eine Parameter-Datei (`http.properties`) ausgelagert. Falls keine Parameter-Datei vorhanden ist, wird ein im Code festgeschriebener Standard-Port verwendet.

6.3 Der Ping-Agent

Die Teile des implementierten Ping-Agenten werden in den folgenden Abschnitten detailliert angegeben.

6.3.1 Die Wissensbasis

Die verschiedenen Arten von Einträgen in einer Wissensbasis sind in Abschnitt 5.1 (Unterabschnitt *Wissensbasis*) erklärt. Abbildung 6.1 zeigt den Inhalt der Datei `Ping.wis`, aus der das Anfangswissen des Ping-Agenten gelesen wird. Der Ping-Agent startet das Protokollnetz `answer`, wenn er eine Nachricht empfängt, die auf das Muster passt. Das Protokoll `generalAgentSetup` wird proaktiv aufgerufen und es werden keine Dienste benötigt. Die Beschreibung des angebotenen Dienstes wurde aus der Spezifikation des Ping-Agenten in der Anleitung zur Anbindung von Agentcities ([act02b, S. 10]) entnommen. Diese Dienst-Beschreibung ist in der Abbildung zur besseren Lesbarkeit in mehreren Zeilen umgebrochen. Das Wissen verändert sich während der Laufzeit des Ping-Agenten nicht.

```
proaktiv=[generalAgentSetup]
%%%
serviceDesc=[(service-description
                              :name ACLPing
                              :type ping_acl_alpha_v1.0
                              :ownership "LIA EPFL")]
%%%
requiredServices=[]
%%%
protocol answer=(query-ref :content "ping" :language "PlainText")
```

Abbildung 6.1: Die Wissensbasis des Ping-Agenten

6.3.2 Das Protokollnetz

Das Interaktionsdiagramm für den Ping-Agenten wird aus den in Abschnitt *Spezifikation des Ping-Agenten* (S. 63) dargestellten Informationen entworfen (siehe Abbildung 6.2). Der Plattform-Anfrage-Agent `pingstat` von Agentcities sendet die Anfrage an den Agenten `ping#1`, worauf dieser antwortet. Das einzige Protokollnetz (`answer`) wird aus dem Interaktionsdiagramm entworfen (siehe Abbildung 6.3). Das Protokoll besteht aus drei Netzkomponenten: `START (IN)`, `OUT` und `STOP`. Der Deklarationsknoten ist unter der Start-Netzkomponente angeordnet. Die Antwortnachricht wird an der ersten Transition der `OUT`-Netzkomponente erzeugt, dabei werden alle Felder der Antwortnachricht außer dem Performativ und dem Inhalt von der Methode `AclMessage.reply()` ausgefüllt.

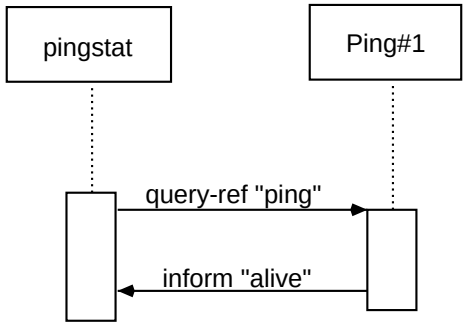


Abbildung 6.2: Das Interaktionsdiagramm für das Protokollnetz des Ping-Agenten

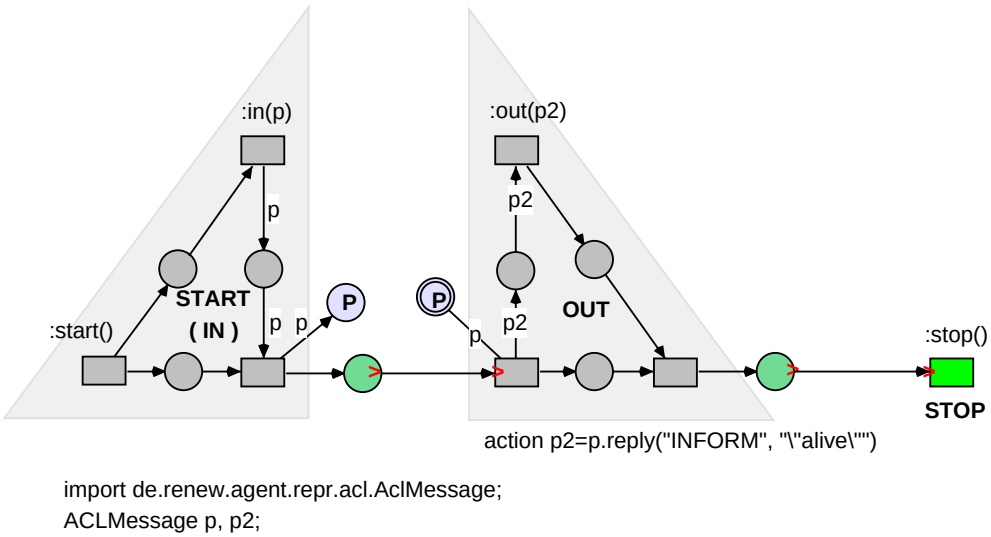


Abbildung 6.3: Das Protokollnetz `answer` des Ping-Agenten

6.3.3 Das Startskript

Das Startskript bildet zusammen mit dem Aufrufnetz die Initialisierung der Plattform und der darauf lauffähigen Agenten. Die konkretere Beschreibung in diesem Abschnitt ergänzt die Darstellung der groben Funktionsweise und der beteiligten Dateien im Abschnitt 5.4.1 zur Initialisierung und Konfiguration von Capa (Seite 80).

Das Startskript `pingStart.sh` enthält folgende Abschnitte:

- *Locations*: Pfade zu Renew, den Plattformklassen und -quellen, zu den Dateien des Ping-Agenten und den Renew-Tools.
- *Classpath*: Hier wird der Pfad zum Renew-Loader angegeben, der die Klassen für Capa enthält.
- *Netpath*: Im Klassenpfad wird nach `.sns`-Dateien (Schattennetzen) gesucht, in den Quellpfaden nach `.rnw`-Dateien (unkompilierte Dateien mit grafischer Darstellung). Diese werden dem Netzpfad hinzugefügt.
- *Parameters*: Eine Benutzungsanleitung. Im Wesentlichen geht es darum, dass z.B. für einen Serverprozess eine Simulation auch ohne eine GUI gestartet werden kann.
- *Debug Output*: Startzeit, Klassen- und Netzpfad werden in eine Logdatei geschrieben.
- *Shadow Net System Check*: Wenn die Parameter oben entsprechend gesetzt sind, sucht das Startskript nach einem *Shadow Net System*[2], das es als Simulation starten kann. Findet es keines, startet es die GUI mit dem oben angegebenen Aufrufnetz und fordert den Benutzer auf, ein *Shadow Net System* zu erzeugen.
- *Start Simulation*: Der Java-Aufruf von Renew mit den gesetzten Pfaden und Parametern. Die Simulation wird mit dem vorhandenen oder eben generierten *Shadow Net System* gestartet.

6.3.4 Das Aufrufnetz

In Abbildung 6.4 auf der nächsten Seite ist der zweite Teil der Initialisierung, das Aufrufnetz dargestellt.

Start der Plattform im Aufrufnetz Zunächst wird ein `Agent-Identifier` für den Plattform-Agenten erzeugt, dieser dient der Namensgebung der Plattform. Dann wird die Plattform an der Transition `<Erzeugung der Plattform>` erzeugt. Bei der Initialisierung des Plattform-Agenten werden auch der `AMS` und der `DF` erzeugt.

[2]Schattennetz: einzelnes Netz, *Shadow Net System*: ausführbare Datei mit Netzen. Siehe S. 20

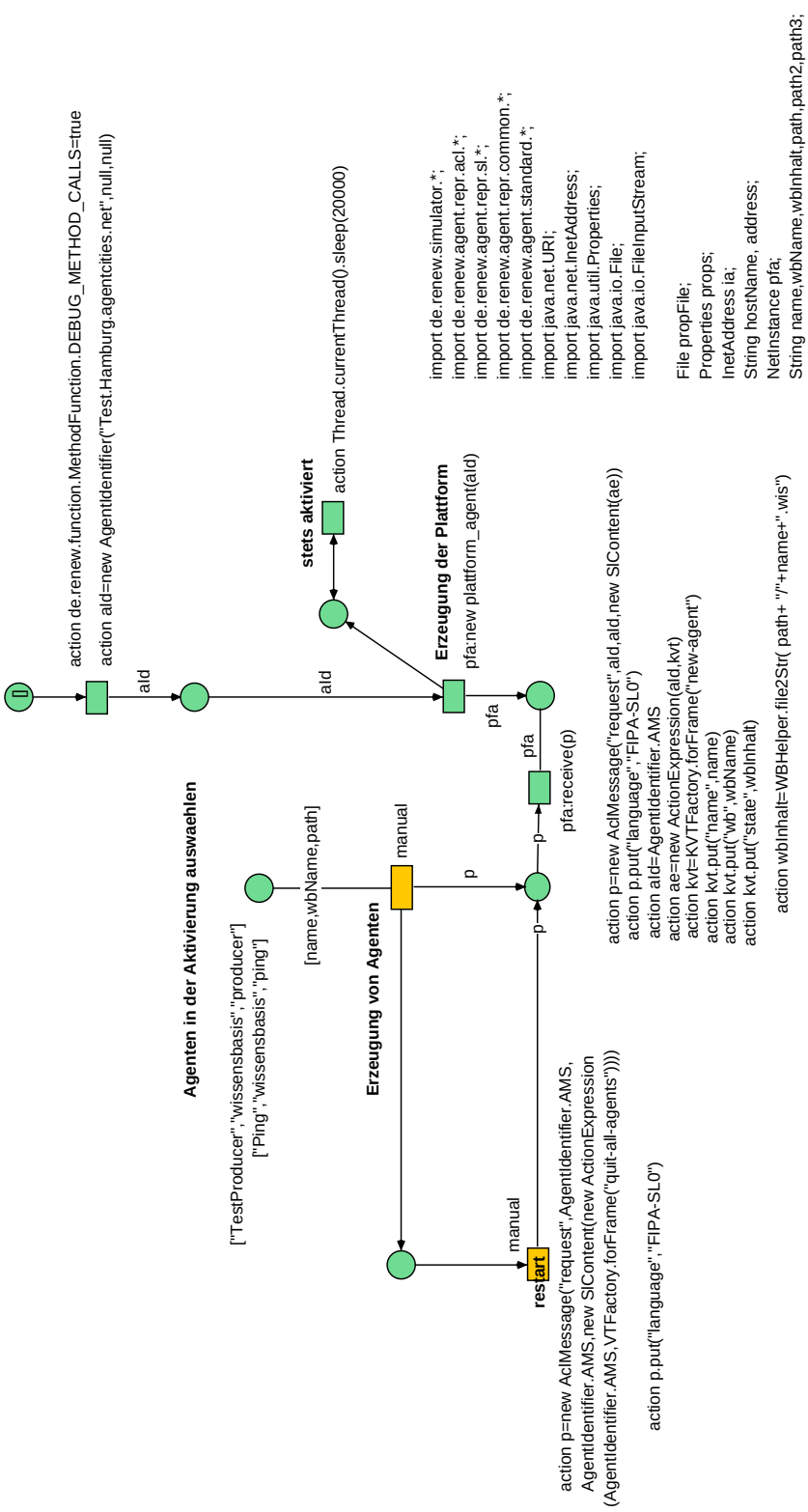


Abbildung 6.4: Das Aufrufnetz, das die Plattform, den Ping-Agenten und den Testagenten starten kann.

Start von Agenten im Aufrufnetz In der Stelle am Beginn werden die Namen und die Pfadstücke zu den Dateien mit dem anfänglichen Wissen der zu startenden Agenten zur Verfügung gestellt, in diesem Falle der Ping-Agent und ein Test-Agent namens `TestProducer`. Die manuell zu schaltende Transition `<Erzeugung von Agenten>` muss warten, bis der `AMS`-Agent existiert. Beim Schalten erzeugt diese Transition eine Nachricht an den `AMS`-Agenten mit den Befehlen, die unten im Bild mittig angeordnet sind. Sie enthält die Aufforderung, den Agenten unter dem gegebenen Namen und mit der in String-Form mitgelieferten Wissensbasis auf dieser Plattform zu erzeugen. Die Transition muss deshalb manuell geschaltet werden, weil der Nachrichtendienst auf der Plattform so funktioniert, dass nicht zustellbare Nachrichten mit einer Fehlermeldung beantwortet werden. Deshalb darf die Aufforderung an den `AMS` erst dann erfolgen, wenn dieser bereits als Agent exisitert.

An der Transition `<restart>` wird eine Nachricht an den `AMS` formuliert mit der Aufforderung, alle laufenden Agenten zu beenden (die zur Plattform gehörenden Agenten – `AMS`, `DF` und der Plattform-Agent – bleiben dabei erhalten). Man kann dann das Anfangswissen eines Agenten ändern und diesen Agenten mit der manuellen Transition wieder laden.

Im Aufrufnetz muss beachtet werden, dass RENEW für die Anbindung an Agentcities ohne GUI im Hintergrund laufen muss. In einer laufenden Simulation in RENEW wird eine Netzinstanz, in der keine Transition mehr schalten kann und auf die keine Referenz von einer anderen Netzinstanz oder von der GUI aus besteht, vom *Garbage-Collector* in Java eingesammelt. Dies ist sinnvoll und notwendig, denn allein für jede Ping-Anfrage wird das Protokollnetz `answer` instanziiert. Der Speicher wäre auf diese Weise schnell verbraucht. Das Aufrufnetz enthält nun wie oben beschrieben eine Transition, die nur manuell geschaltet werden kann, damit die Nachricht zur Erzeugung von Agenten erst dann an den `AMS` geschickt wird, wenn der `AMS` bereits existiert. Wird nun RENEW ohne GUI gestartet, existiert keine Referenz auf das Aufrufnetz und es wird vom *Garbage Collector* eingesammelt, sobald die Plattform erzeugt wurde. Dann kann die manuelle Transition nicht mehr geschaltet werden.

Dieses Problem lässt sich auf verschiedene Weisen lösen:

- Die Entscheidung, wann in einem Netz „keine Transition mehr schalten“ kann, also wann alle Transitionen des Netzes *tot* sind, kann den Sonderfall der manuellen Transition berücksichtigen. Auf diesen Punkt kann im Rahmen dieser Diplomarbeit kein Einfluss genommen werden.

- Das Aufrufnetz eines *Shadow Net Systems* kann speziell behandelt werden, so dass eine Referenz darauf beim Start der Simulation mit RENEW gehalten wird. Auch diese Idee entzieht sich dem Rahmen dieser Diplomarbeit.

- Die Initialisierung des Plattform-Agenten oder des `AMS`-Agenten kann erweitert werden, so dass die Erzeugung von anwendungsbezogenen Agenten angestoßen

wird. Da es um ein Synchronisationsproblem geht, bei dem der Zeitpunkt der Erzeugung des `AMS`-Agenten wesentlich ist, ist dies eine naheliegende Lösung. Innerhalb dieser Idee sind vielfältige Ausprägungen möglich, die dann auf den Entwurf von agentenbasierten Anwendungen zurückwirken.

- Die bisher manuelle Transition kann mit einem *Timeout* verzögert werden. Diese Methode birgt das Risiko in sich, dass die Simulation nicht wie beabsichtigt startet, wenn z.B. der Rechner einer relativ hohen Belastung ausgesetzt ist. Die Ursache ist dann für den Benutzer schlecht nachvollziehbar.

- Die Nachricht zur Erzeugung eines neuen Agenten zum Start einer agentenbasierten Anwendung wird von außen an den `AMS` der Plattform geschickt. Dies ist z.B. mit Hilfe einer Klasse ähnlich der Testklasse `SenderTest` von JAMR möglich. Dann hätte eine agentenbasierte Anwendung zwei Java-Befehle, die nacheinander aufgerufen werden müssten. Wie bei der ursprünglichen Lösung mit der manuellen Transition bleibt es dem Benutzer überlassen, den zweiten Aufruf nicht zu früh zu starten.

- Wenn im Aufrufnetz eine Transition existiert, die immer aktiviert bleibt, wird es nicht vom *Garbage Collector* eingesammelt. Der Zugriff auf das Aufrufnetz zum Schalten der manuellen Transition kann dann über einen zweiten Aufruf von RENEW geschehen. In dem zweiten Aufruf wird nur die GUI von RENEW gestartet und man bindet sich an die *remote* laufende Simulation an[3]. Es ist möglich, an einer entfernt laufenden Simulation die Agenten mit dem Mulan-Viewer anzuzeigen sowie alle Netzinstanzen in einer Liste anzeigen zu lassen, um so Zugriff auf alle Netzinstanzen und deren Marken zu haben.

Die letzte Variante ist hier gewählt worden, um eine schnelle und zuverlässige Lösung bereitzustellen. Der Start eines zweiten RENEW, mit dessen Hilfe die Simulation inspiziert werden kann, ist sowieso sinnvoll. Langfristig bevorzuge ich jedoch die Erweiterung der Initialisierungsphase der Plattform, weil eine agentenbasierte Anwendung dann alleine mit dem Aufruf des Startskripts gestartet werden kann.

Im Aufrufnetz in Abbildung 6.4 gibt es deshalb eine Transistion, die immer aktiviert ist und so verhindert, dass das Aufrufnetz zu früh vom *Garbage Collector* aufgesammelt wird.

6.3.5 Der Test-Agent

Zu Testzwecken wurde dem Ping-Agenten ein Test-Agent zugesellt, der Nachrichten abschickt und die Antwort darauf empfängt. Der Test-Agent besteht aus einer Wissensbasis (`Producer.wis`) und einem Protokollnetz (`producer.rnw`), in dem Nachrichten erzeugt und abgeschickt werden und die Antwort abgewartet wird. Dieses wird endlos fortgesetzt.

[3] Diese Funktionalität von RENEW wurde parallel zu dieser Arbeit entwickelt und im wurde bereits Hinblick auf eine Simulation ohne GUI als wesentlich erkannt.

6.3.6 Einbindung des Ping-Agenten in Capa

In den Anforderungen wurde bereits auf den Konflikt zwischen den dynamisch generierten Agentennamen in Capa und dem festen Namenseintrag in Agentcities hingewiesen. Deshalb muss der Ping-Agent nach dem `AMS` und `DF` als erstes auf der Plattform gestartet werden, damit er immer den gleichen einfachen Namen (`Ping#1`) zugewiesen bekommt, welcher zur Anmeldung in Agentcities verwendet werden kann.

Die Dateien für den Ping-Agenten und den Test-Agenten sind in dem Verzeichnis `Mulan/Platform/src/de/renew/agent/ace/infrastructure/` in je einem Verzeichnis (`ping` und `producer`) in die Klassen- und Netzhierarchie von Capa eingefügt. Das Startskript, das Aufrufnetz und die Logdatei des jeweils letzten Laufs liegen direkt in diesem Verzeichnis.

Spezielle Startkonfigurationen für Anwendungen, die ACE benutzen sollen, können in Verzeichnissen neben dem Verzeichnis `infrastructure` untergebracht werden. Weiteres zum Thema Startkonfiguration ist am Ende dieses Kapitels im Abschnit 6.5 (*Zusammenfassung und Ausblick*) zu finden.

6.4 Anbindung

Nachdem nun die beiden fehlenden Spezifikationen der FIPA mit dem HTTP-MTP von JAMR abgedeckt sind und der Ping-Agent realisiert wurde, kann die Anmeldung einer Plattform in Agentcities geschehen. Auf dem Weg zur aktiven Plattform in Agentcities gab es viele Testläufe und Testszenarien. Sie sind im nächsten Unterabschnitt beschrieben. Die Tests ergaben weitere Punkte, an denen Capa an die Bedingungen in Agentcities angepasst werden musste. Diese sind im zweiten Unterabschnitt aufgeführt.

6.4.1 Testen

Im Laufe der Entwicklung waren immer wieder Tests nötig, um das Entwickelte zu validieren und Fehler zu entdecken. Die Tests sind hier nach den verwendeten Testapplikationen in sechs Kategorien unterteilt. Einige Tests wurden nur mit ein oder zwei Plattformen von Capa durchgeführt, andere zusätzlich mit den Testklassen des HTTP-MTPs. Eine JADE-Plattform wurde ebenfalls zu Testzwecken installiert. Agentcities bietet eine umfangreiche *Testsuite* an, mit der einzelne Funktionen zum Transportdienst und zu den Verwaltungs- und Verzeichnisdiensten (also `AMS` und `DF`) getestet werden können. Eine hartnäckige Inkompatibilität ließ sich nur mit einem *Netz-Sniffer* lokalisieren. Zu guter Letzt ist die Anbindung an Agentcities selbst auch den Testläufen zuzurechnen.

Tests innerhalb der Capa-Plattform Für diese Art von Test ist der Test-Agent (*Producer*) da. Vor dem Start kann man im Protokollnetz des Test-Agenten angeben, an welche Adresse das Protokollnetz die Nachrichten schicken soll. Mit Hilfe der Capa-Plattform wurden drei Arten von Tests durchgeführt: *Producer zu Ping innerhalb einer Plattform*, *Producer zu Ping auf zwei Plattformen innerhalb der Informatik-Domäne* und *Producer zu Ping auf zwei Plattformen, nur eine innerhalb der Informatik*.

Producer zu Ping innerhalb einer Plattform. Hier wird die Funktion der Wissensbasis und des `answer`-Protokolls getestet. Die beteiligten Elemente sind der Test-Agent, der Ping-Agent, der `ACC` und das interne Transportmittel. Das HTTP-Transportmittel wird nur insofern getestet, als es nicht für interne Nachrichten benutzt werden soll.

Producer zu Ping auf zwei Plattformen innerhalb der Informatik-Domäne. Die beiden Plattformen können auf demselben Rechner oder auf verschiedenen Rechnern laufen. Technisch sollte das keinen Unterschied machen. Getestet wird damit vor allem die Funktion des HTTP-MTPs. Der `ACC` der Plattform muss das HTTP-MTP als das angemessene Transportmittel erkennen, das HTTP-MTP muss die Umschlagdarstellung umwandeln und die Nachricht über HTTP verschicken. Das HTTP-MTP muss auch Nachrichten empfangen, richtig umwandeln und dem `ACC` zur Weiterleitung übermitteln.

Producer zu Ping auf zwei Plattformen, nur eine innerhalb der Informatik. Hier wurde eine Capa-Plattform komplett außerhalb des Universitäts-Rechnernetzes installiert. Damit wird getestet, ob der Port freigegeben wurde und ob HTTP-Nachrichten die *Firewall* des Rechnernetzes in beide Richtungen durchqueren können.

Testapplikationen des HTTP-MTPs Die Testklassen des HTTP-MTPs konnten in den Tests vielfältig eingesetzt werden. Bei jedem der bisher beschriebenen Tests mit zwei Plattformen kann eine davon durch die Sender/Empfänger-Kombination der Testklassen ersetzt werden. Sie sind jedoch ein wenig umständlicher zu gebrauchen als der Test-Agent, weil sie keine GUI zum Editieren verschiedener Nachrichtenfelder anbieten, sondern diese Informationen im Code einprogrammiert sind. Für eine „schnelle“ Änderung musste so immer neu kompiliert und gestartet werden, während für geänderte Netze nur die Simulation neu gestartet werden muss. Dies gilt allerdings nur, wenn die Simulation nicht in Form eines *Shadow Net System* gestartet wurde. Agenten mit häufig zu änderndem Verhalten sollten dieses besser aus der Wissensbasis entnehmen, da so nur der Agent neu gestartet werden braucht.

Tests mit einer JADE-Plattform Hier wird eine CAPA-Plattform mit einem Ping-Agenten gestartet und eine JADE-Plattform aufgesetzt. Die JADE-Plattform bietet einen Dummy-Agenten, mit dessen GUI der Nutzer beliebige Nachrichten verschicken kann. Hier wird getestet, ob die Kommunikation zwischen dem HTTP-MTP von CAPA und einer JADE-Plattform funktioniert (die zentralen Verzeichnisse von Agentcities benutzen JADE.

Die Testsuite von Agentcities Für Agentcities existiert eine *Testsuite* [acs03c]. Man gibt dort den Namen und die Adresse der zu testenden Plattform an sowie das verwendete Protokoll (HTTP oder IIOP) und den unqualifizierten Namen des Ping-Agenten, der automatisch mit dem Plattformnamen vervollständigt wird.

Mit der *Testsuite* können umfangreiche inhaltliche Tests durchgeführt werden. Die Tests sind in sechs Bereiche zusammengefasst: Tests für den Transportdienst, Tests für das *Request*-Protokoll, je eine Gruppe von Tests für den `AMS` und den `DF` sowie die beiden letzten Gruppen von Tests, die das Verhalten der Plattform für den Fall testen, dass Nachrichten mit nicht übereinstimmenden Angaben im Umschlag und in der ACL-Nachricht empfangen werden. Diese letzten beiden Gruppen von Tests werden *Security*-Tests genannt. Die Reaktion darauf kann nicht technisch richtig oder falsch sein, vielmehr hängt das von der Auffassung der Entwickler der jeweiligen Plattform ab. In der Spezifikation der *Testsuite* (siehe [acs03c]) werden die einzelnen Tests beschrieben sowie die Entscheidung, wann einer der Sicherheitstests als „bestanden“ angezeigt wird.

Netz-Sniffer Mit dem Netz-Sniffer `ethereal` und einer besonderen Erlaubnis des Rechenzentrums ließen sich die gesendeten und empfangenen Netzpakete exakt rekonstruieren. Dieses half die Vermutung zu entkräften, das HTTP-MTP könnte am Inhalt der Nachricht vor dem Abschicken noch etwas ändern.

Das Feld `conversation-id` hatte in der Ping-Nachricht keine Anführungszeichen, nach dem Absenden aber schon. Letztlich lag das an dem im nächsten Abschnitt beschriebenen Umgang mit den Typen `Word` und `String`.

Die eigentliche Anbindung Natürlich ist auch die Anbindung ein Test – Sie hat auch nicht sofort geklappt, obwohl alle anderen Tests erfolgreich waren, z.B. wird der Nachrichteninhalt bei der *Testsuite* und bei den Verzeichnis-Agenten von Agentcities unterschiedlich geprüft.

6.4.2 Capa-Einstellungen für Agentcities

Dieser Abschnitt beschreibt die Änderungen, die neben den Erweiterungen um das HTTP-MTP und den Ping-Agenten nötig waren. Der Bedarf für die einzelnen Punkte zeigte sich nach und nach bei den eben beschriebenen Tests.

Name der Plattform

Für Agentcities muss der Plattformname konfigurierbar sein. Am Beginn der Arbeit war der Plattformname in einem internen Referenznetz der Plattform fest eingetragen und wurde mit Hilfe des Rechnernamens eindeutig qualifiziert. Der Name der Datei mit dem Anfangswissen des Plattform-Agenten war an diese Art der Namensgebung gekoppelt. Der Name der Plattform wird nun im Aufrufnetz festgelegt und die Kopplung des Plattformnamens an den Dateinamen wurde aufgelöst, dadurch wird der Plattformname frei wählbar.

Agentennamen

Die langen Agentennamen (mit `@Plattformname`) mussten erst ermöglicht werden. Sie werden im Netz `newAgent.rnw` erzeugt. Die Bildung des Plattformnamens kann in dem Initialisierungsnetz `PlatformSetup` beeinflusst werden. Die Änderung der Agentennamen zog eine Anpassung bei der Zustellung von Nachrichten nach sich. Agenten sind nun unabhängig von Großbuchstaben mit ihrem voll qualifizierten oder unqualifizierten Namen erreichbar.

Das Typsystem

Bei der Anbindung von Capa an Agentcities traten einige Unstimmigkeiten zwischen den Typsystemen auf. Zum Teil ist das auf die unvollständige Umsetzung der FIPA-Spezifikationen (auf beiden Seiten) zurückzuführen, zum Teil auch auf eine unterschiedliche Umsetzung von „Details“.

Die ersten beiden Punkte der folgenden Liste sind eng miteinander verwandt. Der konkretere zweite Punkt musste für die aktive Anbindung bearbeitet werden. Die weiteren Punkte betreffen die vollständige Anbindung von Capa an Agentcities. Sie sind jetzt alle zumindest provisorisch gelöst.

1. In Capa geht die Typinformation `Word` – `String` beim Parsen verloren. Die FIPA-Definitionen sind nicht entscheidbar (wie im Abschnitt *Word und String* auf Seite 41), deshalb hat Capa den Typ `Word` übersprungen. Capa kann ein `Word` parsen, speichert die Typinformationen jedoch nicht und sendet die Informationen als `String`, wo dies erlaubt ist.

   Diese Unvollständigkeit in der Umsetzung ist als solche nicht regelwidrig. Erst wenn ein Kommunikationspartner ebenfalls unvollständig implementiert, indem er in einem Feld nur noch den Typ `Word` parsen kann, tritt ein Fehler auf.

2. Nach der Spezifikation der FIPA kann jedes `Word` auch als `String` geparst werden (siehe wieder die Beschreibung im Abschnitt *Word und String*). Zusammen mit dem ersten Punkt entsteht eine fehlerträchtige Mischung: Das

Feld `conversation-id` einer Antwortnachricht erlaubt sowohl `Word` als auch `String`. Damit eine ankommende Nachricht einer bestehenden Konversation zugeordnet werden kann, muss die `conversation-id` exakt übereinstimmen. Wenn sie als `Word` gesendet wird und als `String` zurückkommt, ist sie um die Anführungszeichen erweitert und kann nicht erkannt werden.

Gelöst wird dieses Problem bei der Umwandlung einer Nachricht vom internen Format in die String-Darstellung durch eine Prüfung jedes Strings auf Zeichen, die in einem `Word` nicht erlaubt sind. Als *nicht erlaubt* werden alle Zeichen angesehen, die in der Definition eines `Word` in ACL *oder* in SL als *nicht erlaubt* angegeben wurden, weil eine Nachricht in Capa *zusammen* mit ihrem Inhalt in String umgewandelt wird. Sind nach dieser Regel keine unerlaubten Zeichen enthalten und ist in dem betreffenden Feld der Nachricht ein `Word` erlaubt, wird ein solches gesendet.

3. Die von Agentcities im Feld `ontology` gesendete Information konnte in Capa nicht geparst werden. Capa erwartete im Feld `ontology` ein `VTSet`, Agentcities sendet jedoch ein `Word`, dieses wird in Capa geparst und als `String` abgelegt. Folgende pragmatische Lösung wurde umgesetzt: Beim Parsen eines Ontologie-Feldes wird der geparste Typ abgefragt. Ein `String` wird dann in ein `VTSet` verpackt und kann der geparsten Nachricht hinzugefügt werden.

4. Grundsätzlich lassen sich leere `VT`s und `KVT`s nicht zuordnen, weil der Parser dann nicht aus der Struktur (Doppelpunkt-Notation bei `KVT`s, einfache Werte bei `VT`s) schließen kann, welche der beiden Datenstrukturen vorliegt. Bei Anfragen kommt es vor, dass leere `VT`s und `KVT`s als auszufüllende Muster empfangen werden; der Parser vermutet dann zunächst ein `VT`. Folgende lokale Lösung wurde umgesetzt: In das Netz `parseContent` wurde eine zusätzliche Abfrage eingebaut, die nötigenfalls ein leeres `KVT` erzeugt.

5. Die bisher nicht vollständige Implementierung der Typprüfung beim SL-Parser ist relativ starr. Für die Registrierung eines Agenten beim `DF` verwendet die *Testsuite* Strings, die so nicht geparst werden konnten. Um das Parsen dennoch zu ermöglichen, wird eine Sonderbehandlung für die Typen `DFAgentDescription` und `ServiceDescription` umgesetzt. Langfristig sollte die regelgerechte Typprüfung beim SL-Parser nachgerüstet werden.

6. Das Feld `encoding` konnte in Nachrichten von Agentcities in Capa nicht geparst werden, weil die dort angegebene Kodierung im Parser nicht bekannt war. In der FIPA-Spezifikation zu SL [FIPA 08] ist die korrekte Bezeichnung für die Stringdarstellung von SL nicht angegeben. Bei der Umsetzung in Capa wurde deshalb an dieser Stelle die Bezeichnung eingesetzt, die auch die Sprache selbst bezeichnet (also das Feld `language`: `"FIPA-SL0"`). Die von Agentcities gesendete Kodierung lautet jedoch `"fipa.sl.rep.string.std"`. Nach

der Umstellung im SL-Transformer konnte die Agentcities-Nachricht korrekt geparst werden.

7. Agentcities kann Capa-Nachrichten mit dem Feld `encoding` nicht parsen, da die tatsächliche Kodierung und die Angabe der Kodierung nicht übereinstimmen und außerdem die angegebene Kodierung in Agentcities nicht bekannt ist. Wie bereits in Abschnitt 5.2.2 beschrieben, entsteht diese ungünstige Situation bei der Transformierung von ACL und SL zur Stringdarstellung über geschachtelte `toString`-Methoden. Die Kodierung des Nachrichteninhalts und die Angabe der Kodierung stimmen dann nicht mehr überein. Für die Ping-Nachrichten wird nur die „Sprache“ `PlainText` verwendet, das Feld `encoding` wird ausschließlich für SL-Inhalte benötigt, deshalb war es erst für die volle Anbindung nötig, hier Abhilfe zu schaffen. Dazu wird vor der Umwandlung einer Nachricht in einen String grundsätzlich das Feld `encoding` umgesetzt. Dies wird in dem Moment zu weiteren Problemen führen, wenn weitere Kodierungen des Inhalts zur Verfügung stehen sollen (z.B. XML). Spätestens dann ist eine grundsätzliche Lösung des Problems notwendig.

6.5 Zusammenfassung und Ausblick

In diesem Abschnitt soll das Ergebnis der Umsetzungsphase kurz zusammengefasst werden. In dem darauf folgenden Ausblick werden einige Möglichkeiten und Anregungen zur technischen Weiterentwicklung vorgestellt.

Die realisierte Anbindungsumgebung

Unter der Bezeichnung Anbindungsumgebung sind die Elemente zusammengefasst, die spezifisch für die Anbindung an Agentcities sind. Das ist bis jetzt nur der Ping-Agent. Später können für agentenbasierte Anwendungen Konfigurations- und Startdateien hinzukommen und zusammenfassend der Name ACE (für *Agentcities Environment*) verwendet werden.

Die erweiterte Plattform Capa kann nun wieder als Architekturbild dargestellt werden. Im Vergleich zur Darstellung am Anfang des Kapitels zu Capa (Abbildung 5.1 auf Seite 69) zeigt Abbildung 6.5 auf der nächsten Seite die Elemente von Capa nach der Erweiterung im Rahmen dieser Arbeit. Die für die aktive Anbindung an Agentcities notwendigen Elemente sind hervorgehoben, wie bereits in dem Architekturbild zu Agentcities (Abbildung 4.8 auf Seite 64). Hinzugekommen sind die Darstellung des Umschlags in XML[4], das HTTP-Transportmittel und die Ping-Protokolle.

[4]Nach wie vor gibt es keinen Transformer für die Darstellung des Umschlags, die Darstellung hängt von dem benutzten Transportmittel ab.

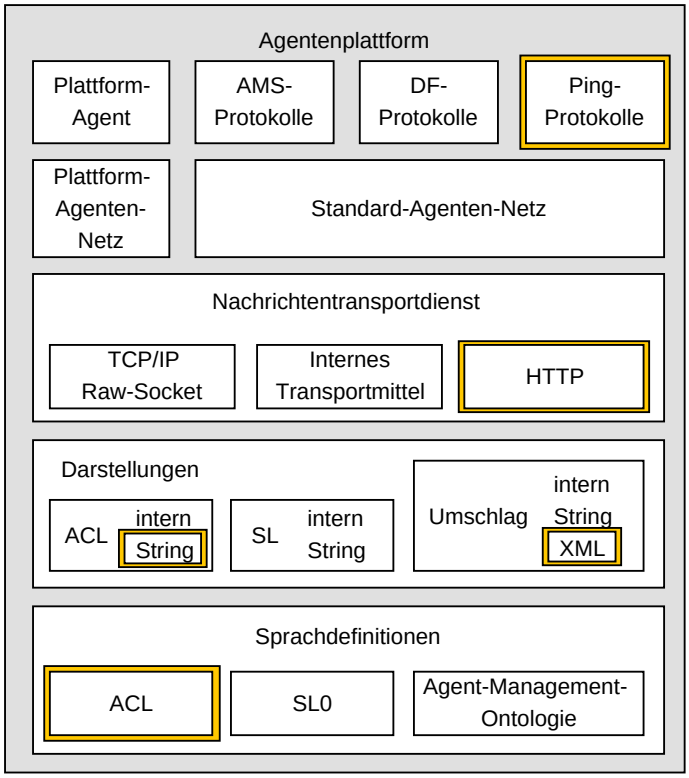


Abbildung 6.5: Die Elemente der Agentenplattform CAPA nach der Durchführung dieser Diplomarbeit.

Ausblick

An dieser Stelle sind einige Punkte aufgeführt, die sich für eine Umsetzung anbieten.

Startkonfiguration Bis jetzt werden in einem Aufrufnetz Agenten für eine Simulation erzeugt. Sollen Agenten aus verschiedenen Anwendungszusammenhängen auf einer Plattform laufen, werden sie bei der aktuellen Arbeitsweise in einem Aufrufnetz gemischt. Besser wäre es, eine konfigurierbare Startmöglichkeit für CAPA zu haben, bei welcher der Start und die Konfiguration der Plattform vom Start der agentenbasierten Anwendungen getrennt ist. Denkbar ist z.B. eine Infrastruktur, bei der pro Anwendung ein Start-Agent programmiert wird, der die anwendungsspezifischen Agenten startet (dies ist bei Siedler bereits der Fall) und über dessen Wissensbasis evtl. die Anwendung konfigurierbar ist. Die Start-Agenten könnten über eine erweiterte und konfigurierbare Initialisierungsphase der Plattform gestartet werden, indem z.B. der `AMS` über seine Wissensbasis oder über andere Parameter die Informationen zum Start eines oder mehrerer Agenten bekommt. Ein anwendungsspezifischer Start-Agent

startet die übrigen anwendungsbezogenen Agenten und initialisiert die Anwendung. Dabei sollten alle wichtigen Parameter möglichst in Wissensbasen untergebracht sein, weil Agenten auf der laufenden Plattform mit verändertem Anfangswissen neu gestartet werden können.

Agentennummern bei Bedarf Aufgrund der automatischen Nummerierung von Agenten in Capa können Agenten über mehrere Plattform-Starts hinweg nur durch ihre angebotenen Dienste statt über Namen identifiziert werden. Ich halte diese Vorgehensweise einerseits für sinnvoll, weil Dienstnamen in Ontologien spezifiziert werden, Agentennamen hingegen nicht. Andererseits ist die Identität eines Agenten bei der FIPA an den Namen gekoppelt (siehe *Agentenidentifikation* in Abschnitt 3.4.4).

Abgesehen von der konzeptionellen Frage nach der Agentenidentifikation muss der Ping-Agent immer zuerst auf der Plattform gestartet werden. Deshalb schlage ich vor, Agenten nur im Konfliktfall mit einer Nummer zu versehen. Dies trifft weitgehend das Konzept der FIPA und Konfliktfälle lassen sich vermeiden.

Kapselung der Transportmittel Die Kapselung der verfügbaren Transportmittel und ihre dynamische Einbindung und Konfigurierung in Capa wurden bereits in Duvigneaus Arbeit als erstrebenswertes Ziel formuliert (siehe [Duv02, Seite 119]), bis jetzt wurde dies jedoch nicht durchgeführt. Die Voraussetzung für die dynamische Einbindung und Konfigurierung ist die Kapselung, die einen getrennten Start und Terminierung ermöglicht. Ein Terminierungskonzept für Capa im Allgemeinen wäre der erste Schritt, damit die Ports der Transportmittel beim Beenden der Simulation in Renew wieder freigegeben werden. In einem weiteren Schritt muss die Initialisierung der Transportmittel dynamisch gestaltet werden, indem z.B. eine Parameter-Datei `capa.properties` verwendet wird, in der die zu ladenden Transportmittel mit einer zu instanziierenden Klasse und mit weiteren Parametern (z.B. der Adresse) angegeben werden.

Die am HTTP-Transportmittel beteiligten Klassen können leicht zu einer `JAR`-Datei zusammengefasst werden, welche als Komponente einer dynamischen Transportkonfiguration verwendet werden kann.

7 Eine Anwendung: Siedler

Mit Hilfe von Capa wird am Fachbereich Informatik der Universität Hamburg agentenorientierte Softwareentwicklung in studentischen Projekten betrieben. Daran können das Vorgehensmodell und der agentenorientierte Ansatz gelehrt, beobachtet und verbessert werden. Dazu wird ein Anwendungsbeispiel benötigt, das ausreichend komplex für den verteilten, agentenorientierten Ansatz und gleichzeitig einfach genug für ein 14-wöchiges studentisches Projekt ist, in dem die Teilnehmer die verwendeten Konzepte und Werkzeuge erst kennenlernen. Bei der gewählten Beispiel-Anwendung handelt es sich um eine agentenorientierte Implementierung des Brettspiels „Die Siedler von Catan“. Für diese Diplomarbeit ist Siedler deshalb von Bedeutung, weil anhand dieses Beispiels die geplante Anbindung von Capa an Agentcities demonstriert wurde (im Rahmen der Konferenz von Agentcities im Februar 2003, siehe Kapitel 4.2 (*Wettbewerb*)).

Der folgende Abschnitt enthält eine grobe Beschreibung des Spiels, die sich eng an der gelungenen Darstellung von Duvigneau orientiert (vgl. [Duv02, S. 139f]). Danach folgt ein Bericht aus dem Projekt. Das daraus entstandene Agentensystem ist schließlich in Kapitel 7.3 beschrieben.

7.1 Das Spiel

Das Gesellschaftsspiel „Die Siedler von Catan“ wurde 1995 in Deutschland herausgegeben und erfreut sich seitdem großer Beliebtheit (siehe [Teu03]). Auf die Grundversion folgten viele Varianten und Erweiterungen, so dass heute unter dieser Bezeichnung eine Reihe von Computerspielen, Kartenspielen und Brettspielen in großer Variation existiert. Die Grundversion des Spiels, die hier als Vorlage diente, ist ein rundenbasiertes Brettspiel für drei bis sechs Spieler.

Das Spielfeld wird für jedes Spiel neu aus 40 oder mehr sechseckigen Waben zusammengesetzt, so dass die Anfangsbedingungen stark variieren. Die Waben repräsentieren fünf verschiedene Landschaftstypen, in denen entsprechend fünf verschiedene Rohstoffsorten abgebaut bzw. produziert werden können: Gebirge – Erz, Ackerland – Getreide, Weide – Schafswolle, Hügel – Lehmziegel sowie Wald – Holz. Den Waben werden beim Spielfeldaufbau Würfelwerte zugeordnet, die mit unterschiedlich hohen Wahrscheinlichkeiten fallen können – dadurch kann die Verfügbarkeit bestimmter Rohstoffe von Spiel zu Spiel deutlich schwanken. Den Rand der Insel bilden Meer-Waben, an denen keine Rohstoffe abgebaut werden, einige bieten aber Handelsverbindungen in Form eines Hafens an.

Ein Spieler kann Siedlungen und Städte auf den Knotenpunkten zwischen den Waben bauen, die durch Straßen auf den Wabenkanten verbunden werden. Wenn die Würfelzahl einer Wabe fällt, ernten alle Spieler, die Siedlungen oder Städte an die Wabe angrenzend gebaut haben, den Rohstoff der Wabe. Die Rohstoffe werden benötigt, um weitere Siedlungen, Städte und Straßen zu bauen. Beim Bau dieser Elemente müssen das Straßen- und Siedlungsnetz jedes Spielers zusammenhängend bleiben und Mindestabstände zwischen Siedlungen und Städten aller Spieler eingehalten werden. Dadurch sind die zur Verfügung stehenden Bauplätze auf der Insel eine knappe Ressource, um die sich ein Wettbewerb zwischen den Spielern entwickelt.

Aus den Erträgen der Siedlungen an den Feldern kann ein Spieler nur selten die passenden Rohstoffe für seine weitere Expansion sammeln. Er kann dann auf zweierlei Wegen Rohstoffe tauschen. Zum einen besteht eine generelle, in den Spielregeln festgelegte, aber auch teure Tauschmöglichkeit zu festen Kursen mit der „Bank“. Hat ein Spieler Siedlungen oder Städte an Hafen-Waben gebaut, werden einzelne Tauschkurse bei der Bank günstiger. Zum anderen dürfen Spieler Rohstoffe untereinander tauschen, wobei das Tauschverhältnis individuell verhandelt werden kann. Diese Tauschmöglichkeit verleiht dem Spiel seinen besonderen Reiz, weil die Wertschätzung der Rohstoffsorten von Spieler zu Spieler und von Spiel zu Spiel sehr unterschiedlich ausfällt.

Ziel des Spiels ist das Erreichen einer vorbestimmten Summe an „Siegpunkten“. Siegpunkte gibt es vor allem für Siedlungen und Städte, sie können aber auch durch eine besonders lange Straße oder – mit etwas Glück – durch den Kauf von „Entwicklungskarten“ erworben werden.

Eignung für Agentcities

Der variable Spielfeldaufbau, die unterschiedlich gute Erreichbarkeit einzelner Rohstoffsorten für die verschienenen Spieler (abhängig von den ersten Siedlungen), die vielfältigen Tauschmöglichkeiten und die alternativen Siegpunktquellen ergeben einen komplexen Spielablauf, in dem viele unvergleichbare Strategien zum Erfolg führen können. Damit ist das Spiel besonders für den Einsatz planender, verhandelnder, kooperierender und konkurrierender Agenten geeignet.

Gleichzeitig sind die möglichen Zustände und Aktionen im Spiel-Szenario übersichtlich genug für eine Forschungsanwendung.

7.2 Das Projekt

Dieser Abschnitt gibt eine Beschreibung des Projekts bezüglich der Teilnehmer, Werkzeuge usw., im zweiten Teil dann einen ausführlichen Vergleich mit dem vorherigen Projekt, wobei zum Vergleich die Beschreibung des vorigen Projekts aus [Duv02] verwendet wird.

7.2.1 Beschreibung

Der agentenorientierte Entwurf des eben beschriebenen Spiels mit CAPA wurde an der Universität Hamburg in zwei aufeinander folgenden studentischen Projekten durchgeführt. Das erste Projekt (im Wintersemester 2001/2002) war der erste Einsatz von MULAN als Entwicklungsumgebung. Zeitgleich entstand CAPA und wurde schrittweise in die Anwendung integriert. Die Projektleitung konnte ebenso wie die Teilnehmer Erfahrungen mit agentenorientierter Software-Entwicklung sammeln. Im zweiten Projekt (im Wintersemester 2002/2003) wurden die gesammelten Erfahrungen für Verbesserungen genutzt und neue Vorgehensweisen und Werkzeuge eingesetzt.

Beteiligte Am Projekt nahmen 30 bis 40 Studenten teil, von denen ca. sieben bereits im ersten Durchlauf Erfahrungen gesammelt hatten. Für die übrigen Teilnehmer war eine umfassende Einführung in die verwendeten Konzepte und Werkzeuge notwendig. Ausgesprochen positiv war die stetige Teilnahme der Beteiligten.

Organisation Während der Vorlesungszeit (ein Zeitraum von 14 Wochen) fanden zwei gemeinsame Treffen pro Woche statt. In dieser Zeit wurden die Werkzeuge eingeführt und im Plenum die Alternativen im Entwurf diskutiert. Es gab betreute Gruppenarbeit, d.h. drei bis vier Betreuer waren anwesend und für Fragen bereit.

Die große Anzahl der Teilnehmer machte es unmöglich, die einzelnen Arbeitsergebnisse zentral zu begutachten. Statt dessen wurden Gruppen gebildet, die durch Informationen im Plenum in die Lage versetzt wurden, ihre Arbeitsergebnisse nach und nach selbst zu verfeinern und zu vervollständigen.

Konzepte und Werkzeuge Folgende Konzepte und Werkzeuge wurden im Projekt vorgegeben:

- Konzepte aus der Agentenorientierung
- Entwicklungs- und Laufzeitumgebung: RENEW
- Modellierung der Interaktionen mit AUML-Interaktionsdiagrammen
- Werkzeug dazu: RENEW-Diagrammtool
- Agentenrealisierung mit MULAN/CAPA
- Protokollnetze mit Netzkomponenten aus den Interaktionsdiagrammen entwerfen
- Nachrichtengestaltung mit SL- und ACL-Unterstützung von CAPA
- Beobachtung des System mit dem MulanViewer
- Editoren für Java-Code und Wissensbasen: Eclipse, Emacs
- Versionskontrolle und Kommunikationsschnittstelle: cvs (*Concurrent Versions System* unter `http://www.cvshome.org/`)
- Java-basiertes *build*-Werkzeug: ant (unter `http://ant.apache.org/`)

Schritt für Schritt wurden die Techniken im Projekt erläutert, so dass die Entwürfe jeder Gruppe bis zur fertigen Protokollimplementierung reifen konnten.

Das Vorgehen bei dem Entwurf und bei der Implementierung war experimentell und höchst parallel. Es ist bereits im Abschnitt 5.4.3 auf Seite 82 (*Entwurf von Multiagentensystemen*) allgemein beschrieben worden.

Anforderungen Die Anforderungen an das zu implementierende Spiel lagen den Betreuern in Form des alten Projektergebnisses vor, die Teilnehmer einigten sich darüber hinaus, möglichst nah an der Brettspiel-Vorlage zu bleiben. Es war beabsichtigt, von den Agentenimplementationen aus dem ersten Projekt möglichst viel Code (Referenznetze, Wissensbasen und Java-Klassen) zu übernehmen, also stand als Ausgangspunkt eine komplette Verzeichnisstruktur zur Verfügung. Aus diesen Randbedingungen wurden vor dem Beginn des Projekts von den Betreuern `UseCases` (Anwendungsfälle) formuliert. Die Kenntnis der Spielvorlage erleichterte außerdem die Identifikation von Akteuren und Rollen.

Methoden Die Anwendungsfälle waren mit je ein bis zwei Zeilen Text beschrieben. Jeweils drei bis vier Teilnehmer wurden zufällig je einem Anwendungsfall zugeordnet. Sie formulierten die Anwendungsfälle aus und vervollständigten sie. Die verfeinerten Anwendungsfälle wurden dann mit Interaktionsdiagrammen beschrieben und mit AUML-Interaktionsdiagrammen konkretisiert. Parallel zur Verfeinerung und Konkretisierung wurden Rollen zu Agenten zusammengefasst.

Die Verantwortung im Projekt war nach Anwendungsfällen und nach Agenten auf die Gruppen verteilt. Für die Interaktionsdiagramme und deren Implementierung in Sammlungen von zueinander passenden Protokollnetzen waren die Anwendungsfall-Gruppen zuständig. Die Agentengruppen waren für die Koordination zwischen den Anwendungsfällen je eines Agenten und für die Informationsverwaltung zuständig (also für die Wissensbasis-Einträge, für die interne Darstellung von Informationen und für die internen Protokolle). Die Alternativen bei Entscheidungen, die das Spielgeschehen beeinflussen, wurden im Plenum ausführlich diskutiert und (z.T. per Abstimmung) gemeinsam entschieden. Die Koordination zwischen den Gruppen bezüglich der Nachrichten- und Datenformate war nicht formal festgelegt.

7.2.2 Diskussion und Vergleich

In der Diplomarbeit von Duvigneau ([Duv02, Kapitel 6]) wird das erste Siedler-Projekt beschrieben, so dass ein Vergleich anhand dieses Textes möglich ist. Viele der Schwierigkeiten konnten im hier beschriebenen Projektlauf mit Hilfe von verbesserten Werkzeugen, einer verbesserten Agentenplattform und einer verbesserten Vorgehensweise umgangen werden. Einige wesentliche Punkte der Verbesserungen seit dem letzten Projekt sind hier aufgeführt:

- **Verbesserte Inspektionsmöglichkeiten** Der MulanViewer ist gegenüber dem letzten Projekt hinzugekommen. Er zeigt wichtige Informationen zur laufenden Plattform und ihren Agenten an und erleichtert den Zugang zu den Netzinstanzen der laufenden Agenten erheblich. Der Zugang zu den Netzinstanzen war zwar auch vorher möglich, indem aus dem Aufrufnetz das Plattform-Javaobjekt mit den von Renew zur Verfügung gestellten Möglichkeiten inspiziert wurde, aber die übersichtliche und leicht zugängliche Darstellung der relevanten Informationen im MulanViewer ist heute ein unverzichtbarer Bestandteil von Capa.
- **Verbesserte Interaktionsdiagramme** Die Interaktionsdiagramme wurden wesentlich aussagekräftiger durch die konsequente Nutzung von AUML und durch die enge Kopplung der Interaktionsdiagramme mit der Implementierung von Protokollnetzen. Inzwischen wurde ein Werkzeug für Renew entwickelt, mit dem AUML-Interaktionsdiagramme sehr komfortabel erstellt werden können.
- **Verbesserte Protokollnetze** Die Gestaltung der Protokolle wurde gegenüber dem letzten Projekt drastisch verbessert und vereinheitlicht, indem überall Netzkomponenten benutzt wurden. Die Netzkomponenten für Capa-Agenten wurden von Cabac als Beispielanwendung für sein neues Konzept der Netzkomponenten entwickelt. Cabac arbeitet aktuell an der automatischen Umsetzung von Interaktionsdiagrammen in Protokollnetze für Capa-Agenten (Konzept der Netzkomponenten: siehe [Cab02], Umwandlung von Interaktionsdiagrammen in Protokollnetz-Strukturen in [CMR03]).
- **Verbessertes Vorgehensmodell** Durch das ablauforientierte Vorgehensmodell wurde eine Struktur möglich, die selbst bei der großen Teilnehmerzahl eine hoch produktive Entwicklung erlaubte. Die Gruppeneinteilung bewirkte ein extrem paralleles Vorgehen, so dass relativ schnell eine recht komplexe Anwendung entstand, welche zum Ende des Projekts in großen Zügen lauffähig war.
- **Verbesserte Sprachunterstützung** Die Nachrichteninhalte wurden in SL0 statt in dem proprietären Datenformat des letzten Projekts verschickt. SL0 wird inzwischen von Capa gut unterstützt, so dass insbesondere das Parsen und Generieren solcher Nachrichten in Klassen zusammengefasst wird und nicht mehr in vielen Protokollen wiederholt implementiert werden muss. Die damit erreichte Typsicherheit hat die Zusammenarbeit erheblich erleichtert. Außerdem eignet sich das damit verwendete `KVT`-Konzept besonders gut zur Übermittlung von Informationen.

Es blieben jedoch durchaus Schwierigkeiten zu meistern, von denen hier einige genannt werden sollen, um im nächsten Projekt eine Verbesserung in diesen Punkten zu motivieren:

- **Die entstehende Ontologie sollte besser kommuniziert werden.** Durch die Implementierung von Protokollnetzen entsteht implizit eine Siedler-Ontologie. Die allgemeinen Regeln, nach denen Begriffe festgelegt werden, könnten noch besser kommuniziert werden: Verben vs. Nomen, englische vs. deutsche Begriffe und Verbindlichkeit für einmal festgelegte Bezeichner. Verbindlich steht hier für zwei Ausprägungen: Verbindlichkeit bezüglich der Schreibweise (Bindestrich, Groß-Kleinschreibung etc.) und Verbindlichkeit bezüglich alternativer Bezeichnungen: „Holz“ sollte immer „Holz“ sein und nicht an anderer Stelle als „Baum“ geführt werden.
- **Schattennetze verwenden oder nicht?** Ohne Schattennetze werden beim Start von CAPA sehr viele Fenster mit geladenen Netzen geöffnet. Das ist unübersichtlich.
  Die Verwendung von Schattennetzen ist jedoch generell bei der Entwicklung komplexer Systeme riskant, weil dann zu jedem Netz zwei Dateien existieren, von denen grundsätzlich nur die `.rnw`-Datei angezeigt werden kann, da die `.sns`-Datei keine graphischen Informationen enthält[1]. Die Suchreihenfolge beim Laden von Netzen ist dreistufig: Zuerst werden die geöffneten Netze durchsucht, dann die Schattennetze im Netzpfad und erst danach die `.rnw`-Dateien im Netzpfad.
  Die Lösung besteht meines Erachtens darin, einen sorgfältig ausgearbeiteten Mechanismus zur Aktualisierung der Schattennetze bereitzustellen.
- **Die Koordination der Gruppen ist ein besonders wichtiger Punkt.** Da die Koordination zwischen den Gruppen nicht geregelt war, fand ein guter Teil der nötigen Absprachen und Diskussionen auf dem Flur statt, der die drei bis vier Rechnerräume verbindet. Pro Raum gab es sechs bis 14 Rechnerarbeitsplätze, an denen z.T. sieben Leute als Gruppe zusammenarbeiteten. Infolge der relativ großen Gruppen war oft das Kontextwissen unvollständig, weil sich bei den beschriebenen Absprachen meistens nur zwei bis drei Personen trafen; den übrigen Gruppenmitgliedern fehlte dann Information aus der Absprache. Die Details der Implementierung waren selten allen Gruppenmitgliedern bekannt.
  Sobald die Teilnehmerzahl sinkt, wird das Kommunikationsproblem entschärft sein. Dennoch ist für eine große Gruppe eine explizite Kommunikationsstruktur notwendig, wie sie durch die zentral zur Verfügung gestellten Interaktionsdiagramme z.T. schon verfügbar war.

[1] Man kann ein Schattennetz anzeigen, dabei geht jedoch die Anordnung der Elemente und jede informale Anschrift verloren. Mit der Funktion *Automatic Net Layout* kann in RENEW die verklumpte Darstellung der Netzelemente entzerrt werden.

- **Der vorgegebene Code sorgte für Verwirrung.** Die Vermischung von Code aus dem ersten Projekt mit den Entwürfen des neuen Projekts war verwirrend, weil damit das Abgucken und Nachmachen nicht funktionierte. Dies ist normalerweise ein effizienter Weg, neuen Code zu erzeugen, wenn der Entwickler nicht Spezialist auf dem Gebiet ist. Wenn z.B. ein Agent erstellt werden soll und bereits Agenten vorhanden sind, wird der vorhandene Agent als Vorlage benutzt. In diesem Fall waren die Agenten des alten Projekts im Verzeichnisbaum eine Ebene höher eingeordnet als die Agenten des neuen Projekts und deshalb leichter zugänglich. So wurden diese als Ausgangspunkt für Veränderungen und Erweiterungen benutzt, was zu einer Vermischung von Code führte, so dass das alte Siedler nicht mehr lauffähig war.
  Auch in den extra für CAPA und Siedler erstellten Netzkomponenten wurden standardmäßig Klassen aus dem alten Projekt importiert, was nicht gleich bemerkt wurde, weil die Klassennamen gleich waren, und deshalb zu Fehlern führte.
  Also sollte das, was an Code im Quellverzeichnis oder als Netzkomponenten vorgegeben wird, entweder sorgfältig eingeführt und bekannt gemacht werden, oder so selbstverständlich funktionieren, dass kein Detailwissen dazu benötigt wird.
- **Die parallele Entwicklung von Laufzeitumgebung, Agentenumgebung und Agentenanwendung hat Vor- und Nachteile.** Die Entwicklung von RENEW, CAPA und Siedler lief zeitweise parallel. Das hat den Vorteil, dass für Weiterentwicklungen in RENEW und CAPA sofort Rückmeldungen zur Verfügung standen, weil sie im Projekt intensiv genutzt wurden. Außerdem konnten im Projekt Verbesserungsvorschläge gemacht und auftretende Bedürfnisse direkt umgesetzt werden.
  Die Quellen lagen an verschiedenen Orten verteilt: Für jedes der Projekte (RENEW, CAPA und Siedler) gab es ein `cvs`-Repository, in dem der offizielle Entwicklungsstand zentral gespeichert war. Die *Entwickler* eines Projekts besaßen je eine lokale Kopie von den für sie relevanten Quellen (für die Projektteilnehmer waren das die Siedler-Quellen). Für die *Nutzer* eines Projekts wurde eine zentral verfügbare Kopie der Quellen bereitgestellt (für die Teilnehmer des Projekts waren das RENEW und CAPA).
  Eine Fehlerquelle bei einer solchen parallelen Entwicklung ist der mehrstufige Weg bis zur Verfügbarkeit der Neuentwicklung. So wurde z.B. ein schwierig zu lokalisierender Fehler bei der Initalisierung eines `Agent-Identifier`s im Projekt bemerkt: das stetig wiederkehrende „`AgentIdentifier.AddAddress`“-Problem. Dieser Fehler wurde drei Mal korrigiert: Beim ersten Mal vergaß der Entwickler, die lokal korrigierte Version einzuchecken, beim zweiten Mal wurde die korrigierte Version eingecheckt, nicht aber an der zentralen Stelle ausgecheckt, von der die Projektteilnehmer CAPA benutzten. Nach der dritten Korrektur war endlich allen die korrigierte Version verfügbar.

- **Die Eignung neuer Begriffe für die Ontologie hängt vom allgemeinen Verständnis ab.** Die im letzten Projekt gewählten Aktionsbezeichner waren verwirrend und lösten Fehler durch Missverständnisse aus, weil sie nicht zentral definiert wurden und die Semantik nicht vorher festgelegt wurde (eine Aufforderung zum Bauen kann also beliebig mit *baue*, *bauen* oder *Bau* eingeleitet werden). Durch die oben beschriebene Verwendung von SL0 ist die Fehleranfälligkeit gesunken. Die gewählten Bezeichner wurden so für alle Protokolle in einer SL-Klasse definiert und nur noch in den Wissensbasen (eine pro Agent) extra definiert. Auch dies kann noch zusammengefasst werden, wenn explizite Ontologien in CAPA eingebunden werden können.
  Mit SL0 ist außerdem die Semantik der Nachrichtenteile offiziell festgelegt, sie ist jedoch nicht allen Teilnehmern klar geworden. So kommt es, dass an mehreren Stellen Nachrichten mit dem simplen Inhalt `"ack"` verschickt werden, die durch die automatisch hinzugefügte Information (`conversation-id`) zwar richtig zugeordnet werden können, aber in manchen Fällen nicht eindeutig genug sind.
  Demnach sollte ein Aktionsbezeichner stets ein Verb sein. Die Antwort auf eine Aufforderung (und auf jedwede andere Nachricht auch) sollte immer mit der `reply`-Methode erzeugt werden, um den Bezug zur Aufforderung eindeutig herzustellen.

7.3 Das Agentensystem

Nach der Beschreibung und Auswertung des Projekts bezüglich seiner Teilnehmer und Vorgehensweisen im letzten Abschnitt wird in diesem Abschnitt das resultierende Agentensystem unter verschiedenen Aspekten betrachtet und bewertet, nachdem es zunächst grob beschrieben wird.

7.3.1 Beschreibung

Es gibt vier administrative Agenten: die Bank, die Börse, die Spielleitung und die Insel. Die Insel bildet den Knotenpunkt zum Bauen während des Spiels, ähnlich dem Spielbrett in der Vorlage. Die Spielleitung verwaltet die Phasen einer Spielrunde und sorgt für die Initialisierung, die Bank verwaltet die Ressourcen der Spieler, die Börse schließlich kann Spieler zum Handeln vermitteln und Transaktionen verwalten.

Des weiteren gibt es beliebig viele Spieler-Agenten. Es wurde ein Spieler-Agent mit GUI entworfen, der mit den administrativen Agenten und mit anderen Spieler-Agenten kommuniziert. Daneben wurden zwei planende Spieler-Agenten entworfen, einer auf Prolog-Basis und einer auf Java-Basis. Statt die Entscheidung, wie in einer Spielsituation zu reagieren ist, einem menschlichen GUI-Benutzer zu überlassen, repräsentieren diese planenden Spieler-Agenten umfangreiches Wissen über die

Spielsituation in ihrer Wissensbasis und treffen aufgrund dieses Wissens eigene Entscheidungen. Aus Sicht der administrativen Agenten besteht keinerlei Unterschied zwischen den planenden Spieler-Agenten und einer Person, die den GUI-Spieler-Agenten benutzt.

Weitere planende Spieler-Agenten können implementiert werden. Sie müssen dann nach den AUML-Interaktionsdiagrammen die Protokolle implementieren, die für einen Spieler-Agenten notwendig sind. Dabei gibt es an verschiedenen Stellen im Spielverlauf Mechanismen, nach denen geregelt ist, ob ein Spieler-Agent auf eine Anfrage zwingend antworten muss oder ob es eine Zeitbeschränkung gibt, nach deren Ablauf das Spiel fortgeführt wird. Ein Spieler-Agent muss z.B. nicht zwingend bauen oder auch nur angeben, dass er nicht bauen möchte. Wird jedoch ein Spieler aufgefordert, den Räuber zu versetzen, so steht das Spiel still, bis eine gültige Antwort angekommen ist. Daraus ergibt sich eine Liste von minimalen Anforderungen an einen Spieler-Agenten, die für einen kompletten Spieldurchlauf notwendig ist. Diese Liste ist jedoch nicht im Rahmen des Siedler-Projekts erstellt worden.

Nachrichtenformate Es wurde auf die FIPA-konforme Nachrichtengestaltung mit SL0 als Inhaltssprache geachtet. Um den fehlerarmen und typsicheren Nachrichtenaustausch zu erleichtern, wurde für jeden Nachrichtentyp eine SL-Klasse geschrieben (siehe Abschnitt 5.2.2 auf Seite 75), dies war eine Aufgabe der jeweiligen Anwendungsfall-Gruppen. Jede dieser SL-Klassen kann zwischen dem internen Format und der Stringdarstellung eines bestimmten Nachrichtentyps konvertieren.

Spielstart Zum Spielstart wurde ein Agent mit dem Namen `SiedlerSpiel` entworfen, der ein Spiel starten kann, wenn er dazu aufgefordert wird (je nach Konfiguration auch proaktiv). Dazu fordert er den `AMS` auf, die administrativen Agenten auf der Plattform zu erzeugen und verbindet sie dann zu einem Spiel, indem er jedem der administrativen Agenten die aktuellen Namen und Adressen der übrigen administrativen Agenten bekannt gibt. Vor dem Start kann dem Agenten `SiedlerSpiel` angegeben werden, wie viele und welche Spieler-Agenten außerdem gestartet werden sollen. Auf anderen Plattformen (und damit auf anderen Rechnern) können zusätzliche Spieler-Agenten gestartet werden.

Jeder gestartete Spieler-Agent meldet sich bei der Spielleitung an. Die entfernten Agenten bekommen als Parameter bei ihrem Start die Adresse einer Plattform, bei deren `DF` sie die Adresse der Spielleitung erfragen können. Sind bei der Spielleitung genügend Spieler angemeldet, wird das Spiel mit der "initialen Bauphase" gestartet.

Der Rundenablauf Das Brettspiel ist rundenbasiert, die Teilnehmer des Projekts entschieden sich, dieses beizubehalten (zur Diskussion standen z.B. parallele Bauphasen, in denen die GUI-Agenten den Vorteil der allgemeinen menschlichen Kognition haben und die planenden Agenten den Vorteil der Geschwindigkeit ausnutzen können). Abbildung 7.1 zeigt das Interaktionsdiagramm, das den Rundenablauf

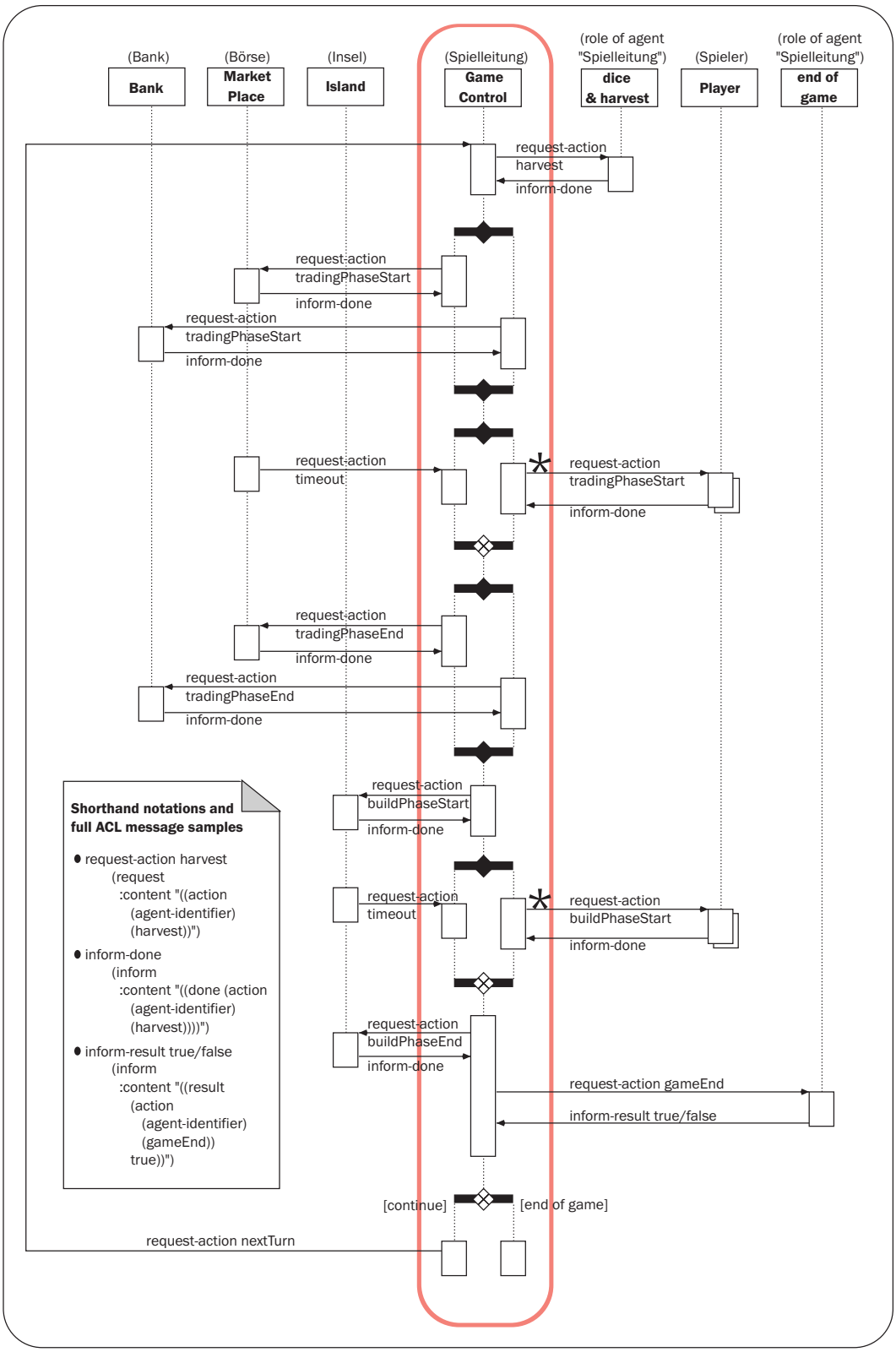


Abbildung 7.1: Interaktionsdiagramm: Runde bei der Spielleitung

steuert. Die Spielleitung ist umrandet in der Mitte dargestellt, links die übrigen administrativen Agenten und rechts zwei Rollen der Spielleitung und die Spieler, die zu einem Strang zusammengefasst sind. Zuerst wird in jeder Runde die Aktion *würfeln und ernten* ausgeführt, indem die Spielleitung sich selbst eine Nachricht schickt (Würfeln und Ernten ist eine Rolle der Spielleitung). Nachdem das Würfelergebnis bekannt ist und die direkt folgenden Ereignisse stattgefunden haben (dies ist in einem spezielleren Interaktionsdiagramm zum Anwendungsfall Würfeln und Ernten dargestellt), informiert die Spielleitung parallel die Bank und die Börse, dass die Handelsphase beginnt. Erst, wenn beide sich zurückgemeldet haben, werden alle Spieler informiert und gleichzeitig ein *Timeout* gestartet. Die Verhandlungen der Spieler sind in spezielleren Interaktionsdiagrammen spezifiziert. Wenn der *Timeout* abgelaufen ist oder wenn alle Spieler melden, dass sie nicht mehr handeln wollen, wird bei der Bank und bei der Börse das Ende der Handelsphase bekannt gegeben. Nach der Rückmeldung von beiden wird die Insel vom Start der Bauphase informiert. Ähnlich wie beim Handeln werden danach die Spieler informiert und gleichzeitig ein *Timeout* gestartet. Nach dem Ende der Bauphase wird die Rolle Spielende der Spielleitung zur Auswertung aufgefordert. Je nach Antwort wird der Sieger bekannt gegeben oder eine neue Runde begonnen.

7.3.2 Diskussion und Ausblick

Die in diesem Abschnitt dargestellte Diskussion knüpft an die Einschätzung zur Eignung von Siedler für eine agentenorientierte Umsetzung (auf Seite 110) an. Es werden nacheinander die Eigenschaften einzelner Agenten und des Agentensystems betrachtet und jeweils Erweiterungsmöglichkeiten aufgezeigt. Die Auswahl der betrachteten Eigenschaften hat keinen Anspruch auf Vollständigkeit, vielmehr werden einzelne interessante Eigenschaften betrachtet, die sich z.T. aus Gesprächen ergaben. So war z.B. eine Anmerkung, die ein Besucher auf der Ausstellung des iD3 zu Siedler machte, der Ausgangspunkt für den Punkt „Stabilität“.

Eigenschaften einzelner Agenten

Es werden die Eigenschaften Intelligenz, Autonomie und Mobilität betrachtet. Intelligenz im Sinne der Benutzung von Methoden aus der Künstlichen Intelligenz kann in planenden Spieler-Agenten umgesetzt werden. Dazu laufen im Arbeitsbereich TGI zur Zeit zwei Diplomarbeiten sowie ein Projektseminar. Die administrativen Agenten benötigen in ihrer jetzigen Form keinerlei Autonomie. Sie bieten ihre Dienste an und verhandeln nicht über Einzelheiten. Die Spieler-Agenten sind teilweise autonom; sie können *unter Beachtung der Spielregeln* verschiedene Strategien verfolgen. Um mobile Spieler-Agenten teilnehmen zu lassen, können die Spielregeln erweitert werden für „Reisen“ zwischen mehreren Inselflecken.

Eigenschaften des Agentensystems

Für das gesamte Multiagentensystem werden die Eigenschaften Komplexität, Offenheit, Stabilität und Sicherheit betrachtet.

Komplexität Die Komplexität von Siedler kann durch die Regeln gesteuert werden. Wenn kein Tausch von Rohstoffen möglich ist, sinkt die Komplexität der möglichen Handlungen. Sie steigt, wenn sehr viele Spieler-Agenten beteiligt sind, denkbare Erweiterungen für erhöhte Komplexität sind mehrere Inseln und sozial interagierende Gruppen von Spieler-Agenten sowie weitreichende Tausch- und Handelsmöglichkeiten.

Offenheit Siedler bildet kein offenes System in dem Sinne, als dass Komponenten von anderen Programmierteams eingebunden werden können. Die beteiligten Agenten sind bis auf die Spieler-Agenten statisch festgelegt. Siedler kann jedoch aufgrund der FIPA-konformen Heimatplattform Capa als Multiagentensystem in einem offenen Agentennetz teilhaben.

Um Komponenten von anderen Projektteams einbinden zu können, müssen die Schnittstellen veröffentlicht werden, insbesondere die Schnittstelle des Spieler-Agenten. Dabei kann entweder die Agentenschnittstelle veröffentlicht werden (das betont die Nutzung von Agenten verschiedener Projektteams) oder die Planerschnittstelle (das betont die Nutzung von Methoden aus der Künstlichen Intelligenz). Die Planerschnittstelle könnte mit Spieler-Agent inklusive Laufzeitumgebung (also Renew und Capa) und GUI veröffentlicht werden.

Eine Art von Offenheit ist die dynamische Zusammensetzung der beteiligten Agenten nach dem Spielstart. Dazu können die Spielregeln erweitert werden. Eine erste Möglichkeit bieten die Spieler-Agenten, dazu muss eine Regel zu den Anfangsbedingungen für einen hinzukommenden Spieler entworfen werden. Ein weitere Möglichkeit ist die dynamische Auswahl einer Bank. Wenn die Bank pro Runde oder alle paar Runden Rohstoffe „kostet“, können Spieler-Agenten dynamisch (evtl. zunächst statisch) zwischen verschiedenen Kostenmodellen wählen und im laufenden Spiel die Bank wechseln (das Konto transferieren).

Stabilität Siedler ist nicht stabil in dem Sinne, dass bei einem verteilten System der Ausfall einer Komponoente nicht das ganze System anhalten sollte. Siedler hängt von den administrativen Agenten ab; aber auch die Spieler-Agenten dürfen nach Beginn des Spiels nicht ausfallen, sonst kann das Spiel nicht zu Ende geführt werden. Das liegt unter anderem an dem rundenbasierten Spielablauf und an der statischen Zusammensetzung der Agenten zu einem Spiel. In diesem Punkt unterscheidet sich Siedler z.B. von der Reisemarkt-Simulation *Taga* (siehe Seite 53), in der es zwar auch zentral notwendige Agenten gibt, diese jedoch nicht von den Reisebuchungs-Agenten (also den Spieler-Agenten) abhängig sind.

Sicherheit Zur allgemeinen Sicherung der Agentenkommunikation gibt es das Projekt XSecurity (siehe Seite 53). Zur Verschlüsselung von entfernten Zugriffen auf die Laufzeitumgebung RENEW hat Thomas Jacob eine Arbeit verfasst (siehe [Jac02]).

Eine grundsätzlich andere Möglichkeit ist es, die direkte Kommunikation mit einer Agentenplattform zu unterbinden. Dies kann durch eine Indirektion zwischen Siedler und Agentcities realisiert werden. Dabei wird eine Plattform an einem von außen nicht zugänglichen Port („hinten“) aufgesetzt und eine zweite Plattform an einem zugänglichen Port („vorne“). Diese Plattformen können untereinander kommunizieren, da beide im selben Rechnernetz laufen. Die Agenten der hinteren Plattform melden ihre Dienste beim `DF` der vorderen Plattform an. Auf diese Weise sind die angebotenen Dienste sichtbar, wenn die vordere Plattform an Agentcities angebunden ist. Soll nun einer der registrierten Agenten angesprochen werden, muss die vordere Plattform die Nachrichten weiterleiten. Dabei kann eine beliebige Prüfung des Inhaltes vorgenommen werden.

Nach diesen Betrachtungen kann ein Blick zurück auf die Definitionen von Agent und Multiagentensystem in Kapitel 3 geworfen werden. Nach der strengen, an der künstlichen Intelligenz orientierten Definition 3.1 (Seite 26) bilden nur die Spieler-Agenten untereinander ein Multiagentensystem. Die administrativen Agenten passen nicht zu dieser Sichtweise. Nach der technisch orientierten Definition 3.2 (Seite 29) bilden alle Siedler-Agenten ein Multiagentensystem. Auch die gemäßigte Definition 3.3 (Seite 30) ist für die Siedler-Agenten zutreffend.

Insofern hat sich die Agententechnologie als brauchbare Abstraktionsweise für die administrativen Agenten erwiesen. Die agentenorientierte Software-Entwicklung zeigt ihre Stärken bei der parallelen Entwicklung von komplexer Software. Die Verbindungen zur künstlichen Intelligenz dagegen können eher bei den Spieler-Agenten genutzt werden.

7.4 Siedler im Rahmen dieser Diplomarbeit

Im Rahmen der vorliegenden Diplomarbeit wurde Siedler für die Teilnahme am Wettbewerb von Agentcities bearbeitet, wodurch ein großer Teil der Umsetzungszeit für Siedler verwendet wurde. Deshalb folgt hier ein Rückblick auf die Tätigkeiten bezüglich Siedler im Rahmen dieser Arbeit und danach ein Ausblick auf die Anbindung von Siedler an Agentcities.

7.4.1 Der Wettbewerb (Rückblick)

Durch die Fristen für den Wettbewerb war für die ersten 14 Wochen dieser Arbeit ein enger Zeitplan vorgegeben: Einarbeitung in Agententechnologie, in CAPA und in

die hier verwendeten Werkzeuge, Verfassen der Anmeldung und des Textes für die Hauptphase des Wettbewerbs, Entwurf und Implementierung der notwendigen Teile sowie Vorbereitung und Durchführung der Präsentation auf dem Wettbewerb.

Die Vorbereitungen bezüglich des Siedler-Spiels liefen zweigleisig, weil nicht klar war, welche Version auf dem Wettbewerb vorgestellt werden würde: Einerseits wurde im Siedler-Projekt auf einen frühen Prototypen mit minimaler Funktionalität gedrängt, andererseits wurde (außerhalb des Siedler-Projekts) das Ergebnis des vorigen Projekts an die Weiterentwicklungen in Capa angepasst, insbesondere wurde das alte Siedler-Spiel noch auf die Verwendung des `DF` zur dynamischen Adressenvergabe und auf SL0 als Inhaltssprache umgestellt.

Die Verwirrung, die durch die vorgegebene Codestruktur im Projekt entstand, wurde bereits erwähnt; an dieser Stelle trat sie besonders hervor: Der Code des alten Siedler-Spiels war bald eine nicht lauffähige Mischung aus altem Code, Anpassungen an Weiterentwicklungen in Capa und unbeabsichtigten und unkoordinierten Änderungen aus dem neuen Projekt. Durch diese ungünstige Entwicklung wurde relativ viel Zeit im Rahmen der Diplomarbeit in die Korrektur der alten Quellen (außerhalb des Siedler-Projekts) investiert. Zum Wettbewerb standen zuletzt zwei Versionen zur Verfügung, von denen die neue ausreichend weit entwickelt war und wegen der weitgehenden FIPA-Konformität für den Wettbewerb gewählt wurde.

Unter der Annahme, dass die Anbindung an Agentcities bis zur Testphase für den *special prize* des Wettbewerbs gelingen würde, wurden auch Vorbereitungen getroffen, Siedler dort zur Verfügung zu stellen. Renew, Capa und der Spieler-Agent mit GUI sollten zu einer `jar`-Datei zusammengefasst werden und samt Installations- und Startanleitung auf einer Webseite zum Herunterladen bereitgestellt werden. Der damit gestartete Spieler-Agent würde dann über Agentcities nach der Spielleitung suchen und sich beim Spiel anmelden. Bei dieser Arbeit zeigte sich ein weiteres Problem im Umgang mit dem Quellcode: Mit demselben Code sollte mit unterschiedlicher Parametrisierung nacheinander das Siedler-Spiel, ein entfernter Spieler-Agent und der Ping-Agent gestartet werden. Siedler war zeitweise nicht mit Einstellungen zur Anpassung der Agentennamen für die Anbindung an Agentcities lauffähig. Wenn ein Mechanismus zur Parametrisierung in Capa realisiert wird, könnte die Verwaltung verschiedener Startkonfigurationen dabei berücksichtigt werden.

Die Teilnahme an der Testphase für den *special prize* war nicht möglich, weil Siedler nicht lief (das alte aufgrund der beschriebenen Probleme, das neue wegen der bis dahin kurzen Entwicklungszeit). Die Anbindung mit dem Ping-Agenten lief ebenfalls nicht. So konnte in der Testphase kein System bereitgestellt werden.

Dass die Anbindung zum Zeitpunkt des Wettbewerbs noch nicht erfolgreich war, lag zum einen daran, dass die Administratoren von Agentcities selbst in höchstem Maße in die Vorbereitungen und Auswertungen zum Wettbewerb eingebunden waren und deshalb keine Unterstützung bei der Fehlersuche geben konnten. Zum anderen

wurde wie beschrieben ein großer Teil der Zeit auf die Entwicklung von Siedler verwendet.

Zur Ausstellung auf dem iD3 wurde schließlich die neue Version von Siedler vorgestellt. Die administrativen Agenten und ein Spieler-Agent mit GUI konnten verteilt gestartet und präsentiert werden. Die für den Spielverlauf grundlegenden Funktionen waren lauffähig: Initialisierung, Rundenablauf, Bauen und Ernten. So war das Spiel (ohne Tauschmöglichkeiten) eingeschränkt spielbar.

Die Wahl der Kategorie für den Wettbewerb lag beim Erstbetreuer dieser Arbeit. Angemeldet wurde Siedler in der Kategorie *Dienste und Anwendungen* in der Annahme, dass jede agentenorientierte Anwendung zuerst eine funktionierende Infrastruktur benötigt und es deshalb bereits von Vorteil sei, sich in der Kategorie *Dienste und Anwendungen* zu bewerben. Der zentrale Bewerbungstext sowie die zur Präsentation verwendeten Poster sind im Anhang gegeben. Eine gekürzte Version des Textes wurde im Tagungsband veröffentlicht (siehe [RDK+03]).

Im Laufe des Wettbewerbs zeigten sich folgende Aspekte im Hinblick vorgenommene Bewertung:

- Durch die Wahl der Kategorie *Dienste und Anwendungen* lag der Schwerpunkt der Bewerbung auf Siedler, wodurch eine sehr starke Kopplung mit und Abhängigkeit von dem (laufenden) Siedler-Projekt entstand. In der Kategorie *Infrastrukturen* wäre die Bewerbung nicht vom Fortschritt im Projekt abhängig gewesen.

- Siedler hatte zur Ausstellung noch keine planenden Spieler-Agenten. Intelligente Agenten sind für Agentcities von besonderem Interesse.

- Die Annahme, hinter jeder angemeldeten Anwendung stehe eine vollwertige Infrastruktur, ist nur zum Teil richtig, weil es möglich war, die frei verfügbare JADE-Plattform zu nutzen, so braucht die technische Infrastruktur nur konfiguriert, nicht aber erstellt zu werden. In der Kategorie *Dienste und Anwendungen* wird deshalb nur die Konzeption und Umsetzung von agentenorientierten Anwendungen honoriert.

- Capa ist als verwendete Infrastruktur komplex und neu, gleichzeitig FIPA-kompatibel und war zum Zeitpunk des Wettbewerbs beinahe in Agentcities eingebunden. Dies wird nur in der Kategorie *Infrastrukturen* bewertet.

- Capa und damit Siedler waren zum Zeitpunkt des Wettbewerbs noch nicht an Agentcities angebunden. Dadurch wurde die Arbeit, die hier auf das Transportmittel und den Ping-Agenten verwendet wurde, nicht honoriert. Das wäre nur in der Kategorie *Infrastrukturen* oder bei einer dann schon erfolgreichen Anbindung der Fall gewesen.

Rückblickend scheint es, als wäre eine Bewerbung in *beiden* Kategorien durchaus sinnvoll gewesen. Denn einerseits wurde das Potential, das Siedler für agentenorientierte Entwicklung bietet, mit Freude und Neugier aufgenommen (wobei der schlichte Entwicklungsstand nicht sehr hoch war), andererseits ist die verwendete Infrastruktur mit der auf Referenznetzen basierenden Plattform Capa etwas Besonderes. Für eine Bewerbung in beiden Kategorien fehlte jedoch die personelle Besetzung.

7.4.2 Anbindung an Agentcities (Ausblick)

Für die Anbindung von Siedler an Agentcities gibt es grundsätzlich zwei Möglichkeiten: Entweder bietet Siedler Dienste in Agentcities an oder Siedler nutzt Angebote von anderen Mitgliedern in Agentcities.

Sollen Dienste angeboten werden, kann entweder das Spiel als Ganzes angeboten werden oder die Dienste der einzelnen Agenten. Letzteres ist nicht sinnvoll, weil die Siedler-Agenten keine generischen Dienste anbieten; die Dienste sind nur im Verbund sinnvoll. Soll das ganze Spiel als Dienst angeboten werden, so kann durch Agentcities eine beliebige Verteilung des Spiels realisiert werden. Die verschiedenen administrativen Agenten, die menschlichen Spieler (mit GUI) und die Spieler-Agenten finden sich dann im Agentennetz zu einem kompletten Spiel zusammen.

Die Agenten in Siedler können Dienste von anderen Anbietern in Agentcities nutzen, um ihre Dienste im Spiel zu erfüllen. Das kann zum Beispiel eine bessere (persistente) Verwaltung des Spielstands sein, eine Auktion oder Börse für Tauschgeschäfte, oder es kann ein externer Bank-Dienst verwendet werden, welcher allerdings ein Depot für die Rohstoffe anbieten muss.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde die Agentenplattform CAPA erfolgreich an das Agentennetz Agentcities angebunden. Das Ergebnis ist eine dauerhaft laufende Plattform, die in allen zentralen Verzeichnissen von Agentcities eingetragen ist (vollständige Anbindung). Damit hat sich gezeigt, dass die Spezifikationen der FIPA funktionsfähig sind, also geeignet, um eine Verbindung verschiedener Implementationen zu einem Netz zu ermöglichen (allerdings war an einigen Stellen eine pragmatische Interpretation notwendig). MULAN wurde aus allgemeinen Erwägungen heraus standardisiert, ohne auf eine konkrete offene Umgebung ausgerichtet zu sein. Mit der Anbindung an Agentcities haben sich diese Bemühungen ebenfalls als erfolgreich erwiesen.

Zusammenfassung

Im Kapitel „*Referenznetze und* RENEW“ wurden die Programmierung mit Referenznetzen und das für diese Arbeit grundlegende Werkzeug RENEW eingeführt, das als Editor und Simulator für Referenznetze die Aufgaben einer Entwicklungsumgebung und einer Laufzeitumgebung für CAPA erfüllt.

Im Kapitel „*Agenten*“ wurden die Konzepte und Fachbegriffe aus der Agententechnologie eingeführt. Aus einer Diskussion des Begriffs Multiagentensystem geht ein Vorschlag für die Verwendung von spezifischen Begriffen für technische und anwendungsbezogene Zusammenhänge hervor, es folgt die Darstellung des umfassenden Agentenkonzepts MULAN. Schließlich wurden die Spezifikationen der FIPA vorgestellt, dabei wurde einerseits die Dokumentstruktur der Spezifikationen verdeutlicht, andererseits wurden die Komponenten einer FIPA-konformen Agentenplattform zusammengestellt.

Das Kapitel „*Agentcities*“ stellte Agentcities unter verschiedenen Aspekten vor. Die Darstellung der Organisationen wurde mit eigenen Bildern veranschaulicht. Der von Agentcities ausgeschriebene Wettbewerb wurde beschrieben und dessen Preisträger vorgestellt. Aus der Beschreibung des von Agentcities zur Verfügung gestellten Agentennetzes konnten die Anforderungen an CAPA abgeleitet werden. Einige konzeptuelle Betrachtungen rundeten die Betrachtung von Agentcities ab.

Im Kapitel „CAPA“ wurden die Konzepte und konkreten Erweiterungsmöglichkeiten von CAPA vorgestellt. Hier zeigte sich, welche der Anforderungen von Agentcities bereits erfüllt waren, wie z.B. die Verwaltung der Agenten und die Implementierung der Agentenkommunikationssprache ACL. Die im allgemeinen Abschnitt zu CAPA kurz vorgestellte Enwurfsmethode für Multiagentensysteme ist ein vielversprechen-

der Ansatz für eine parallele und schnelle Entwicklung komplexer Software. Der Abschluss des Kapitels wird durch einige Bemerkungen zu FIPA und zu Agentcities bezüglich CAPA gebildet.

Im zentralen Kapitel „*Anbindung von* CAPA *an Agentcities*“ wurden die Anforderungen aus dem Agentcities-Kapitel noch einmal zusammengestellt und einzeln bewertet. Es stellte sich heraus, dass ein Transportmittel für HTTP und ein Ping-Agent zu CAPA hinzugefügt werden mussten, jedoch kein XML-Transformer notwendig war. Weiter zeigte sich, dass zur erfolgreichen Anbindung die Kommunikation zwischen den menschlichen Beteiligten wesentlich ist und an dieser Stelle die Umsetzung langwieriger war als vorhergesehen. Das Ergebnis der hier dokumentierten Implementierungsarbeiten ist die vollständig an Agentcities angebundene Agentenplattform CAPA.

Im Kapitel „*Eine Anwendung: Siedler*“ wurden nach der Beschreibung des Brettspiels „Die Siedler von Catan“ in zwei weiteren Abschnitten das Projekt und das resultierende Multiagentensystem vorgestellt. Das Vorgehen im Projekt wurde in Bezug zum früheren Projektlauf gestellt und Verbesserungen und verbleibende Verbesserungsmöglichkeiten wurden aufgezeigt. Das resultierende Spiel wurde mit Bezug zu einer Auswahl von Eigenschaften für Agenten und Multiagentensysteme dargestellt. Im letzten Abschnitt wurden einige Aspekte der Verbindung zwischen Siedler und der vorliegenden Diplomarbeit dargestellt: ein Rückblick zum Wettbewerb und ein Ausblick zu Siedler in Agentcities.

Ausblick

Aufbauend auf dieser Arbeit ist es möglich, in CAPA realisierte Anwendungen an Agentcities anzubinden. Nach der technischen Anbindung der Agentenplattform ist nun die inhaltliche Anbindung von Diensten in Agentcities möglich. Die Varianten zur inhaltlichen Anbindung von Siedler an Agentcities sind bereits im Abschnitt *Anbindung an Agentcities (Ausblick)* auf Seite 124 vorgestellt.

Der Übersetzungsdienst in CAPA ist dem ACC zugeordnet, und nur dieser nutzt die Klassen, welche das Interface `Transformer` implementieren. Die FIPA hat in [FIPA 93] ein Konzept für einen Übersetzungs-Dienst spezifiziert. In einen solchen Übersetzungsdienst könnte auch die XML-Transformierung des HTTP-Transportmittels eingebunden werden. Zusätzlich zur Umsetzung von [FIPA 93] in CAPA ist eine Erweiterung der Schnittstelle zwischen Agenten und Nachrichtentransportdienst denkbar, bei der die Agenten auf die Darstellung von Nachrichten Einfluss nehmen können (oder sogar müssen).

Im weiteren Umfeld der Agententechnologie werden von verschiedenen Autoren Prognosen und Fernziele formuliert, mit denen Forschungsprojekte bezüglich ihrer Relevanz und Aktualität verglichen werden können. So stellen z.B. Luck, et al. in

einer *Roadmap* den Stand der Agententechnologie dar und geben einen umfassenden Ausblick auf die kommenden 15 Jahre [LMP03, Seite 36]:

- Die aktuelle Phase enthält die schon existierenden geschlossenen Multiagentensysteme, welche vordefinierte Interaktionsprotokolle verwenden und mit einem jeweils eigenen Vorgehen implementiert werden.

  Diese Beschreibung trifft auf das alte Siedler zu, bei dem noch ein proprietäres Datenformat verwendet wurde.

- Phase zwei bis zum Jahr 2005 benutzt aktuelle Technologien wie die FIPA-Spezifikationen und Agentcities.

  Dieses trifft auf die in dieser Arbeit beschriebene Siedler-Version zu.

- Es gibt einzelne Verbindungen zwischen im Voraus einander bekannten Multiagentensystemen, in denen mit Sprachen wie FIPA-ACL kommuniziert wird.

  Dies könnte der Fall sein, wenn Spieler-Agenten von „außerhalb“ am Siedlerspiel teilnehmen oder Siedler einen externen Bank-Dienst nutzt.

- Weiter entstehen in der zweiten Phase allgemeine Entwurfsmethoden und Werkzeuge für agentenbasierte Systeme.

  Dieses könnte auf eine deutlich ausgearbeitete Version der in dieser Arbeit vorgestellten Entwurfsmethode zutreffen.

- Größere Anzahlen von Agenten interagieren in dafür entworfenen Umgebungen, wie Agentcities eine bereitstellt.

  Mit der erfolgreichen Anbindung an Agentcities haben die Agenten von Capa *teil an dieser offenen Umgebung.*

- Erst in der Phase drei bis zum Jahre 2008 werden offene Systeme in spezifischen Anwendungsgebieten und begrenzte Übersetzungsmöglichkeiten zwischen verschiedenen Anwendungskontexten sowie allgemein akzeptierte Sprachen und Protokolle zur Verfügung stehen.

- In der noch weit entfernten Zukunft ab 2009 vermuten die Autoren wirklich offene Systeme, in denen die Agenten die passenden Protokolle erst lernen. Sprache, Protokoll und Verhaltensweise ergeben sich dann aus der Agenteninteraktion und es bilden sich Organisationsstrukturen zwischen Agenten heraus.

Moldt und Ortmann (siehe [MO04]) beziehen sich auf die speziellen Möglichkeiten der Verbindung von Petrinetzen mit den der Agententechnologie verwandten Bereichen WebServices, Ontologien und Workflows. Die hier formulierten Ziele – dynamische Auswahl von WebServices, dynamische Komposition von Webservices und dynamische Komposition von Workflows – bilden wichtige Schritte zu den vorher angedeuteten Entwicklungen in der vierten Phase. Die Autoren zeigen mögliche Wege zu diesen Zielen auf und wo jeweils die Lücken sind.

Die Agententechnologie soll dabei die Systemstruktur bereitstellen, die Petrinetze als wichtiges Modellierungswerkzeug für Workflows und Webservices mit der teilweise schon möglichen Verifizierbarkeit sollen eine formale Basis bilden. Speziell Capa eignet sich dazu, weil das Verhalten der Agenten mit Referenznetzen spezifiziert wird. Wenn WebServices und Workflows mit Referenznetzen ausgedrückt werden, kann die Verifizierbarkeit von Petrinetzen zusammen mit der dynamischen Erweiterbarkeit der Agenten in Capa um Protokolle bei der dynamischen Komposition von Komponenten verwendet werden. Die Forschungsarbeiten zur Verifizierbarkeit von Referenznetzen im Arbeitsbereich TGI (Dissertationsvorhaben von Roxana Melinte) sind ein wichtiger Beitrag zur formalen Betrachtung und letztlich zur Verifikation von offenen und dynamischen Multiagentensystemen.

Anhang A

Plakate und Bewerbungstexte

Im Rahmen dieser Diplomarbeit wurde das System aus CAPA, RENEW und Agentcities zweimal präsentiert: zuerst beim iD3 von Agentcities, später auf der „Expo“ (jährliche Ausstellung studentischer Projekte am Fachbereich Informatik der Universität Hamburg). Zu den drei Gebieten Anbindungszusammenhang, MULAN und Siedler wurde je ein Poster entworfen. Die Poster zu MULAN und Siedler wurden für die Präsentation bei Agentcities entworfen, das Poster zur Anbindung für die Expo. Das MULAN-Poster wurde auf beiden Präsentationen vorgestellt.

Die Verwendung der Begriffe wurde seit dem Posterentwurf noch klarer ausgearbeitet, so werden in den Poster-Texten die Begriffe MULAN und CAPA noch nicht genau abgegrenzt und der Begriff Multiagentensystem einmal für die Plattform und einmal für das Agentennetz verwendet.

Die Bilder aus den Postern wurden im Text der Arbeit verwendet:

- Das Protokollnetz des Ping-Agenten: Abbildung 6.2 auf Seite 96
- Die Elemente der Agentenplattform: Abbildung 5.1 auf Seite 69 und nach der Erweiterung in Abbildung 6.5 auf Seite 107
- Das Interaktionsdiagramm zur Runde in Siedler: Abbildung 7.1 auf Seite 118
- Die MULAN-Architektur: Abbildung 3.2 auf Seite 34

Nach den Postern ist der Hauptteil des Bewerbungstextes für den Wettbewerb vom 15. Dezember 2002 angegeben. Daneben waren noch tabellarische Angaben zu den Projektbeteiligten und zur Verfügbarkeit der realisierten Agenten zu machen, die hier nicht dargestellt sind.

Multiagentensysteme: Anbindung von Mulan an Agentcities

Agentcities

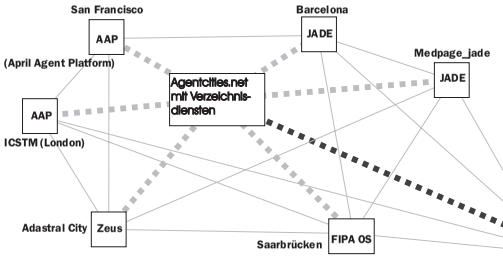
Ein offenes heterogenes Netzwerk von Agentenplattformen

Agentcities wird im fünften Rahmenprogramm der Europäischen Kommission finanziell gefördert. Die Teilprojekte sind: Agentcities.NET (Vernetzung von Forschern), Agentcities.RTD (Realisierung der technischen Infrastruktur) und die Agentcities Task Force (“ACTF”, Koordination der Aktionen und Veröffentlichungen) sowie weitere nationale Projekte.

Das von AC.RTD erstellte Netzwerk besteht aus folgenden Teilen:
- Eine Domäne (unter http://www.agentcities.net erreichbar)
- Eine Sammlung von Agentenplattformen in der Domäne, von denen jede einer Forschungsgruppe zugeordnet ist
- Eine Sammlung von Agenten, von denen jeder auf einer Plattform läuft
- Eine Sammlung von Diensten, von denen jeder von einem oder mehreren Agenten im Netz angeboten wird
- Ein zentraler Knoten, der Verzeichnisdienste für das gesamte Netz anbietet

Die Plattformen und Agenten folgen den Spezifikationen der FIPA. Nutzer der Dienste können Menschen oder Agenten sein. Ein Teil der Dienste ist zum *technischen* Aufbau des Netzwerkes notwendig, die Dienste im anderen Teil gehören zum *inhaltlichen* Aspekt des Netzwerkes.

Mögliche Zustände einer Plattform im Netz:
- Angemeldet: Dazu reicht ein Name
- Aktiv: Eine regelmäßige Kommunikation findet statt
- Vollständig angebunden: Agenten und deren Dienste werden automatisch in die zentralen Verzeichnisse aufgenommen



ACE

Das “Agentcities Environment” verbindet Mulan mit Agentcities

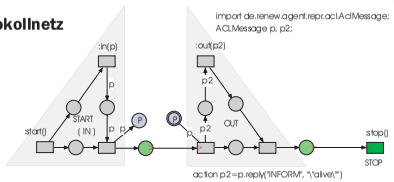
Das Ergebnis der vorgestellten Diplomarbeit ist eine dauerhaft laufende Agentenplattform im Informatikum, die im zentralen Plattformverzeichnis von Agentcities verzeichnet ist.
Differenzen zwischen den Anforderungen von Agentcities und dem, was in Capa bereits vorhanden war, bestanden in drei zentralen Punkten:
- Der Transportdienst von Capa muss HTTP benutzen können.
- Der Umschlag dieser Nachricht muss in XML dargestellt sein.
- Ein Ping-Agent muss die regelmäßige Anfrage von Agentcities beantworten.

Die ersten beiden Punkte werden mit der Benutzung des Transportmittels für HTTP adressiert.

Der Agent ist ein Mulan-Agent. Sein Verhalten ist durch eine Wissensbasis und ein Protokollnetz beschrieben.

Der Ping-Agent

Das Protokollnetz



Die Wissensbasis

```
Proaktiv=[generalAgentSetup]
%%%
serviceDesc=[(service-description :name ACLPing :type ping_acl_alpha_v1.0 :ownership "LIA EPFL")]
%%%
requiredServices=[]
%%%
protocol answer=(query-ref :content "ping" :language "PlainText")
```

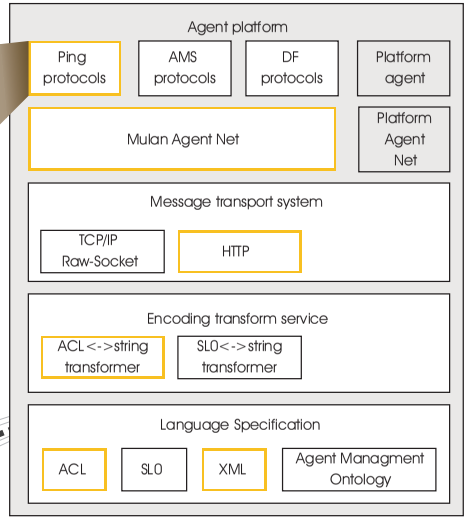
Mulan / Capa

Ein petrinetzbasiertes Multiagentensystem

Bei TGI wird die Architektur Mulan (MULtiAgentenNetze) zur Modellierung von Multiagentensystemen verwendet. Als Basis zur Entwicklung und Ausführung von Mulan dient das Petrinetzwerkzeug Renew (http://www.renew.de/).
Die Mulan-Architektur umfasst vier Schichten, die separat auf einem Poster dargestellt sind.

Mulan ist mit Capa (Concurrent Agent Platform Architecture) zur vollwertigen, FIPA-konformen Plattform gediehen. Capa ersetzt das Mulan-Plattformnetz durch eine FIPA-konforme Kommunikationsarchitektur und realisiert die von der FIPA spezifizierten Verwaltungsdienste für Agenten.

Das Duo Mulan/Capa stellt damit eine weitere Alternative zu den vielfältigen in Agentcities eingesetzten Plattformimplementierungen dar. Dank der Spezifikationen der FIPA ist eine Zusammenarbeit der Agenten auf diesen Plattformen möglich.



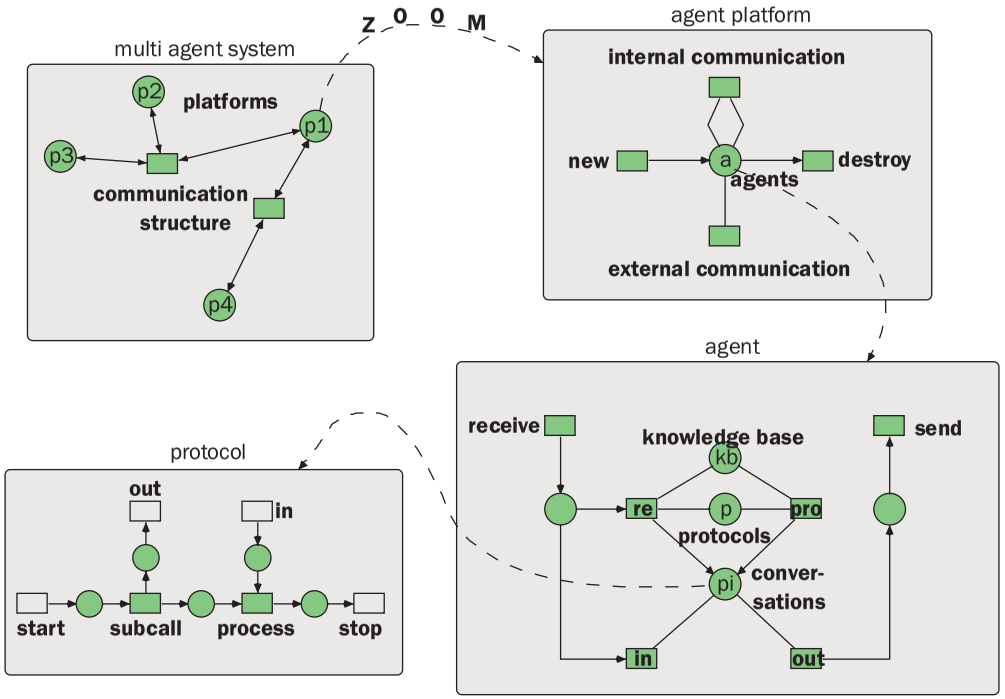
Diplomarbeit von Christine Reese (Erstbetreuer: Daniel Moldt)
in Zusammenarbeit mit Michael Duvigneau und Heiko Rölke

Abbildung A.1: Das Poster zur Expo

Agent Based Settler Game

Multi Agent Nets

Mulan Architecture Overview



- Modelled with Reference Nets using Renew (http://www.renew.de)
- Hierarchical structure: protocols *in* agents *in* platforms *in* a system net
- Vertical communication through synchronous channels

The Four Layers

System View
- Contains Platforms
- Describes communication structure
- According to a concrete MAS
- E.g. Agentcities

Platform
- Provides basic services
- Contains agents
- Fix structure
- The Mulan extension CAPA makes it FIPA-compliant

Agent
- Fix structure
- Contains Protocols
- Contains a knowledge base

Protocol
- Describes behaviour
- Application dependent
- Often corresponds to one column in an interaction protocol diagramme

University of Hamburg
Department of Computer Science
Theoretical Foundations Group
http://www.informatik.uni-hamburg.de/tgi

Abbildung A.2: Das Poster zu MULAN

Agent Based Settler Game

Game Design

Administrative Agents

Game Control
- Register player agents
- Phases: initial build, trade, build, game end (victory condition test)
- Count victory points
- Broadcast with update after every build action
- Roll the dice (robber on 7, else harvest)

Market Place
- Blackboard functionality to mediate trading players
- Transaction management: perform a trade between two players after they agreed upon it

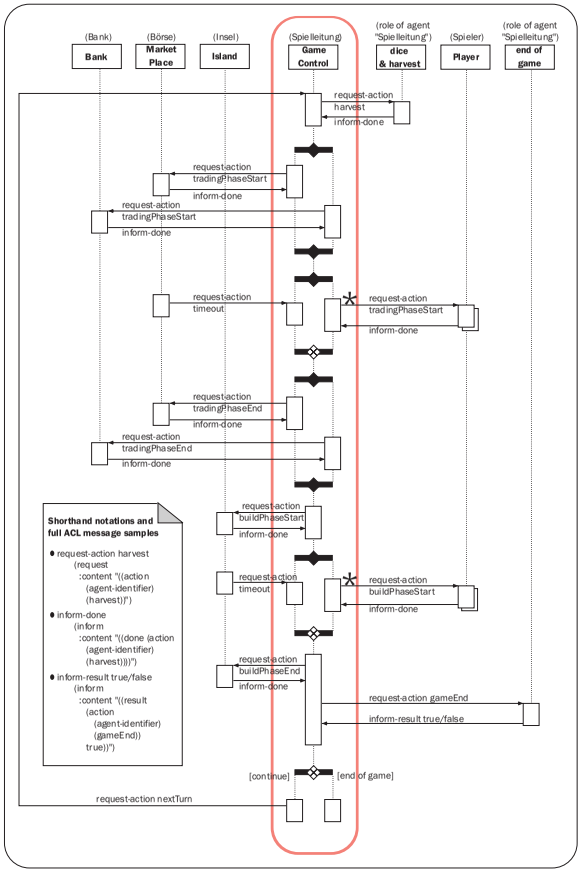
Island
- Topology: static landscape and dynamic state (locations of buildings and robber)
- Handle build requests: check validity depending on initial/normal round
- Calculate harvest depending on the roll of dice
- Robber placement
- Longest Trading Route

Bank
- Manages resource accounts for each player
- Three kinds of resources:
  - Building resources: Lumber, Wool, Grain, Ore, Brick
    - ⊗ In: harvest, trade, steal; Out: robber event, trade, payments
  - Building tokens: settlements, roads, cities
    - ⊗ Out: building; In: city building
  - Development cards
    - ⊗ In: buy; Out: play out
- Trusts only other administrative agents

Player Agent

- Obligatory:
  - Build initial settlements & roads
  - Place robber on request
- Possible:
  - Buy & play development cards
  - Trade
  - Build
  - Keep track of game situation

Turn Interaction Protocol



University of Hamburg, Department of Computer Science, Theoretical Foundations Group
http://www.informatik.uni-hamburg.de/tgi

Abbildung A.3: Das Poster zu Siedler

Abstract (10 lines max.)

Presented will be a settler game to illustrate our agent-oriented software development approach. The approach is aiming at open agent systems. To test and prove FIPA-compliance we will embed the application into Agentcities. The application is based on a distributed and truly concurrent framework illustrated below. Settler is a multiplayer game with a lot of possible interactions between agents.

We plan three steps for this competition: refactoring (application and toolset), embedding into Agentcities, and using other services deployed in Agentcities (earliest in February).

Keywords: Agents, Petri nets, game, settler, distribution, concurrency, Renew

Extended description (3 pages max.)

Instructions:

Include: a description of what the system does, particular features which make it stand out from other similar systems, why the system is innovative, who users are (or might be) and an example usage scenario.

What does it do?

"Die Siedler von Catan" is a game [10]. This game was the initial motivation for the presented implementation. Originally it is played with three to six players on a board. It is round-based, the players can interact freely by buying (event cards), building (settlements, roads, villages), trading (at ports, with the bank), negotiating (with each other about materials for building and buying), and by cooperating ("if you build over there, then I will stay here"). Two dices bring random to the play, and a robber can be placed tactically on the board to hinder some players and make friends with others. The basic way of getting resources (Lumber, Bricks, Ore, Grain, and Sheep) is harvesting after each round, depending on the roll of the dices. The expansion through building is limited by the coast of the Catan island.

By adapting a round-based table game to an agent based computer game, some changes needed to be done to the rules (to get full advantage of the possible concurrency, and to enable a lot of players at the same time).

Currently there are agents representing the bank, the board, the players, the game-control and a market place for trading and negotiating. The services they provide to each other enable a variety of conversations between agents, which make up the game on the whole.

Who are the Users?

The users are student players on the one hand and developers of agent based distributed concurrent platforms and applications on the other hand. The game gives the opportunity to learn about agent based development.

The Underlying Framework

The work takes place in several layers: Conceptually, we develop and evaluate a concurrent agent reference architecture. Practically, we use Java, Reference nets, Renew, Mulan and Capa for the settler application. The architecture is layered as seen in Fig. 1.1.

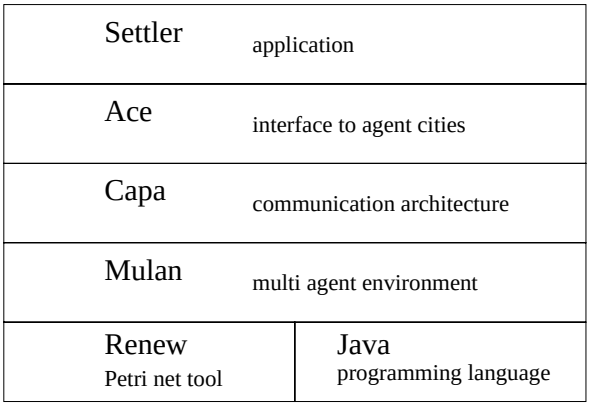


Fig. 1.1 Layered architecture of Settler. Each layer uses all layers below, e.g. each layer has direct access to Java

Lets have a glance at the multilayered architecture: Java, Renew, Mulan, Capa, and Ace.

Renew ("Reference Net Workshop") [7] is an efficient Petri net editor and simulator. Renew is a tool to develop software with reference nets as a powerful programming language. Reference nets [8] are a high-order Petri net formalism. Petri nets is the basic formalism for modelling of concurrent systems [5]. Due to their operational semantics it is possible to run them as a program. Very naturally they combine the concept of true concurrency with visual programming. There is no modelling or programming overhead dealing with management of concurrent processes; this is done by the tool. Within Renew, it is possible to link java code to a transition (the basic element of a Petri net), so that the visual appearance of a net (a program) is kept clear from long or technical sequential parts.

Mulan [9] is our conceptual agent framework. Mulan stands for "Multi Agent Nets". The architecture consists of four levels: (1) The system level with several platforms. (2) The platform level with several agents. (3) The agent level, where each agent is equipped with several protocols. (4) The protocol level. The interfaces

between the layers are simple: The system provides the communication structure between platforms, the platform provides "new", "destroy", "send" and "receive" to the agents, and each agent provides "start", "stop", "in" and "out" to its protocols.

An agent has a knowledge base, a set of protocols and the possibility to start a conversation.

The behaviour (and the services) of a special agent is determined by its knowledge base and by the protocols that are available, and by its environment (platform services and other agents). When developing a new agent based application or service, the knowledge base and the set of protocols (and interactions with other agents) are designed [4].

Capa [2] makes the Mulan-environment for agents capable of being distributed in unknown environments, through standardization (by making it FIPA-compliant [3]). This affects mainly the communication architecture. Included are the obligatory services defined by the FIPA specifications [3] like AMS, DF and a central transport service. The platform itself is implemented as an agent with special possibilities.

Ace will connect the framework to Agentcities. This includes a ping-agent which needs to be implemented. This is a small layer which should be exchangeable with layers for other agent networks.

Particular Features

The underlying toolset (Java, Renew) supports true concurrency. The architecture is aiming at distribution from the deepest layer (reference nets).

Innovative Points

Within this application, agents can use the whole range of possible capabilities, because the rules of the game allow a good deal of inspiration. Agents can act without receiving a message (being proactive), they can respond to messages, they can interact in many socially motivated ways (if there are a lot of players).

A restricted mobility is implemented (currently not used in the game): The code is transferred implicitly (only the names of the protocols), and a part of the dynamic state is transferred (the knowledge base), in a way that no information is lost (when the agent is idle).

So, the settler game is a challenge to agent developers to design agents that truly are distinct from simple services as provided in web services etc.

Overview Usage Scenario

Michael wants to play a game. He asks Julia, but they cannot find a third person to play with. So they start a settler session with three additional players played by computer agents. Each human player calls "settler" on the shell, and the initial game board opens in a window. One after the other in a random sequence, the five

players put their initial settlements and roads on the board (the human players use the mouse), on a place where they think they have a good start and where they are allowed to place them. Another frame shows all incoming messages to the human players.

Scenario of Negotiating

Agent A has the turn. It is in need of Grain and has one piece of Lumber and one piece of Sheep remaining according to its building plans. It sends a message to agents B and C saying "I need Grain. What do you ask for a Grain?" B answers "no offer", C gives two answers: "Sheep" and "Bricks". A sends a confirming message with reference to the "Sheep"-message. Both A and C send request messages to the bank-agent which sends a confirmation to A and C after the exchange is done.

Scenario of Building

Agent A wants to build a settlement. It sends an action request to the "island"-agent. This message contains what and where A wants to build. Agent "island" sends an action request to the "bank" agent to freeze the resources. "island" also blocks the place on the game board. If any of theese fails, blocked resources are freed (with a message to the bank), otherwise the resources are deleted and "island" sends a "done"-message to agent A.

Implementation / Use status (1/2 page max)

Instructions:

Is the system a prototype? A product? Has it been deployed, trialled or used?

A first proprietary but stable and flexible version of Settler has been built: version 1 of Settler is in use by students [1]. This is gradually extended, so there will always be a current running version. Version 2 will be the FIPA-compliant version of Settler. What we plan to present at the competition is version 3, running within Agentcities.

Settler is meant as a prototype, we don't plan to sell it or ship it yet, it is mainly for research on agent based application development. However, Settler is a game and runs stable, so students had hours of fun playing Settler.

The running version includes several agents. It is possible to play Settler distributed on several computers. Human and agent players can build and harvest. That includes specifying knowledge about rules and goals to an agent.

The currently developed version of Settler will include elaborated negotiating mechanisms (even performed by automatic players), and communicate FIPA-compliant.

The planned steps to version 3 are: refactor the infrastructure, embed the next version into Agentcities so that our agents communicate over Agentcities, and finally (version 4?) use other services within Agentcities where appropriate.

Possible services for natural integration are auction-, bank-, and deployment-services.

Relevance to open agent systems (1/2 page max)

Instructions:

Why is the system relevant to usage in open environments such as the public Internet, GRID, Web services or open Agent networks such as the DARPA GRID or Agentcities?

We deliver a reference architecture and a reference implementation (Mulan [9], Capa [2]) for an open agent system. A prototypical application (Settler) comes with it for illustration.

Concurrency is one of the most important attributes of open agent systems, and Renew [7] provides this.

The system is FIPA-compliant [3]. Developing according to a standard is a must to communicate with agents of other programmers.

The whole system will be integrated into a network (Agentcities) to test and prove compatibility and usefulness as an application and as a reference architecture.

Relevance to Agentcities (1/2 page max.)

Instructions:

Does the system have particular relevance to the Agentcities network?

We present an application for integration into Agentcities. We support Agentcities by adding diversity in FIPA-compliant [3] platform architecture (Mulan [9] , Capa [2]). We demonstrate the abilities of this architecture with an application (Settler). We provide a game-application as a service in a network. It is planned to use services provided by others for two reasons: First, we don't have to implement the more elaborated services provided by others, if we use those to substitute our simpler versions (of auction services etc.), and second, it is positive for other members of the Agentcities network if their services are used in a multi-agent environment.

Support /Acknowledgements

Instructions:

Please provide a short summary of the development/support behind the system entered for the competition.

All work is performed in the usual context of the Petri net group in Hamburg (Theoretical Foundation of Computer Science ("TGI")) and the Laboratory for agent-oriented systems (LAOS [6]) at the University of Hamburg, partially supported by a DFG-grant ("Deutsche Forschungsgesellschaft") for a Socionics project.

Renew [7] is a basic tool developed during the last years and used worldwide.

Mulan [9] was continuously developed and extended during the last three years in a PhD-project by Heiko Rölke.

Capa [2], based on Renew and Mulan, was developed during the last year in a diploma thesis by Michael Duvigneau.

Settler 1 and Settler 2 were student lecture projects, with currently 30 freshly worked-in students.

Ace is currently being developed during a diploma thesis by Christine Reese.

Bibliography

[1] Tobias Bosch, Oliver Gries, Heiko Kausch, Maxim Klenski, Kolja Lehmann, Michael Morales, Valentin Seegert and Anatolij Vilner: Agentenorientierte Implementierung des Spiels "Die Siedler von Catan". 2002. Available at FB Informatik, Universität Hamburg.

[2] Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In *Proceedings of the 2002 Workshop on Agent Oriented Software Engineering (AOSE'02)*, 2002.

[3] Foundation for Intelligent Physical Agents (FIPA). Specifications. 2001. Represented at http://www.fipa.org.

[4] M. Köhler, D. Moldt, and H. Rölke: Modeling the behaviour of Petri net agents. In J. M. Colom and M. Koutny, editors, Proceedings of the 22nd Conference on Application and Theory of Petri Nets, volume 2075 of LNCS, pp. 224–241, Springer 2001.

[5] O. Kummer: Introduction to Petri Nets and Reference Nets. Sozionik aktuell, No. 1, 2001. ISSN 1617-2477. Available at http://www.sozionik-aktuell.de.

[6] Laboratory for agent oriented systems (LAOS). Available at http://www.informatik.uni-hamburg.de/TGI/forschung/projekte/laos/laos-index.html

[7] O. Kummer, F. Wienberg and M. Duvigneau: Reference Net Workshop (Renew). Universität Hamburg 2001. Available at http://www.renew.de.

[8] O. Kummer: Referenznetze. Dissertation, Universität Hamburg, 2002.

[9] H. Rölke: Mulan: Modellierung und Simulation von Agenten und Multiagentensystemen mit Referenznetzen. Technical report. Universität Hamburg, Fachbereich Informatik 2002.

[10] Klaus Teuber: Homepage available at http://www.KlausTeuber.de Die Siedler von Catan, available at http://www.diesiedlervoncatan.de English version (Settler's of Catan) available at http://www.myfairgames.com

Anhang B

UML

Für die Modellierung komplexer Software ist UML ein inzwischen verbreiteter Standard. In der objektorientierten Programmierung sind die Klassendiagramme besonders verbreitet. In der agentenorientierten Programmierung hat die Kommunikation zwischen den Elementen eine herausragende Bedeutung und wird beim Entwurf von agentenorientierten Systemen hauptsächlich modelliert, deshalb sind hier Interaktionsdiagramme von größerer Bedeutung. Das Sequenzdiagramm ist ein Interaktionsdiagramm aus der UML, das eine Kommunikation *beispielhaft* oder *im normalen Ablauf* darstellen kann. Um eine allgemeine Kommunikation oder ein *Kommunikationsprotokoll* darzustellen, müssen Alternativen darstellbar sein. Vorschläge für eine solche Erweiterung werden mit AUML (siehe [OvDPB00]) angeboten.

Die Elemente eines AUML-Interaktionsdiagramms sind folgende: Die Agenten oder benannten Rollen von Agenten sind in Kästen oben im Diagramm angeordnet. Von jeder Rolle gibt es eine senkrechte Lebenslinie. Senkrechte Kästen auf einer Lebenslinie deuten eine Aktivierung, eine Aktivität oder eine Handlung an. Zwischen solchen Kästen können waagerechte beschriftete Pfeile für Nachrichten verlaufen, dabei können Antworten und Anfragen unterschiedlich dargestellt werden. Die neuen Elemente in AUML erlauben es, sowohl die Lebenslinien als auch die Nachrichtenpfeile aufzuspalten und zusammenzufassen, so dass Alternativen im Kommunikationsablauf darstellbar sind. In Abbildung B.1 sind zwei AUML-Interaktionsdiagramme dargestellt. Links ist eine Vorschrift für eine Anforderung dargestellt, rechts eine konkretere Umsetzung für ein spezielleres Beispiel. Neben der verwendeten Variante mit dem X für eine `exklusiv-Oder`-Situation gibt es noch die Varianten für `Und` und einfaches `Oder`, sie sind in B.2 in je einer horizontalen und vertikalen Variante dargestellt. Die jeweils umgekehrte Richtung ist ebenso möglich (Lebenslinien und Nachrichten zusammenfassen).

Bei der in dieser Arbeit vorgestellten Entwicklungsmethode mit Netzkomponenten und Renew werden von diesen neuen Elementen fast ausschließlich fünf der zwölf Varianten verwendet: für Lebenslinien das `AND` und das `XOR` in beiden Varianten und für Nachrichten das `XOR` in der vereinigenden Variante.

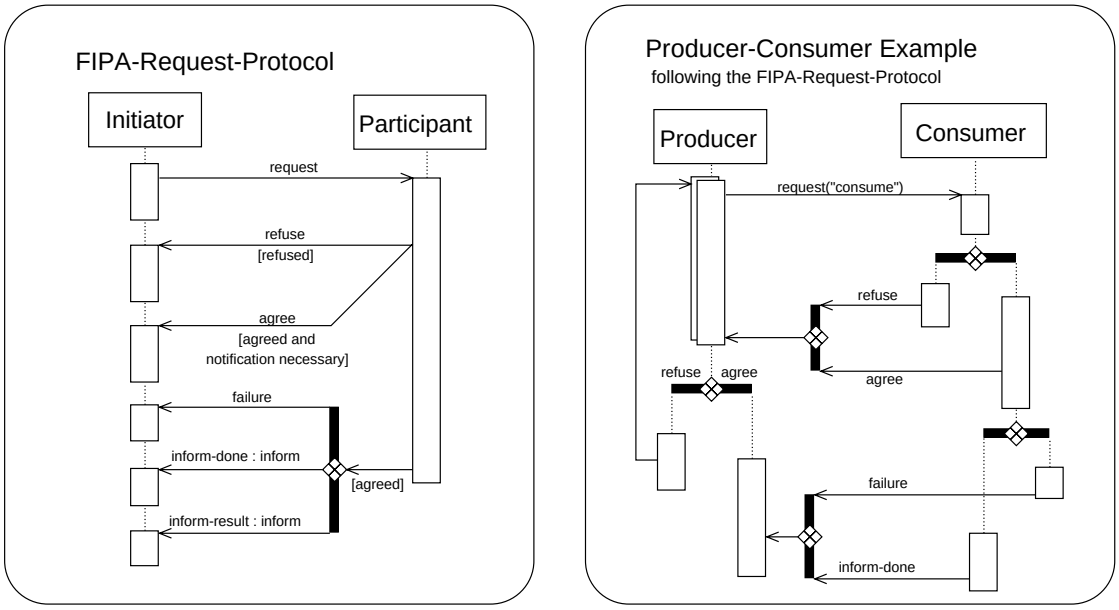


Abbildung B.1: Interaktionsdiagramme in der von der FIPA vorgeschlagenen und in der in Mulan verwendeten Form

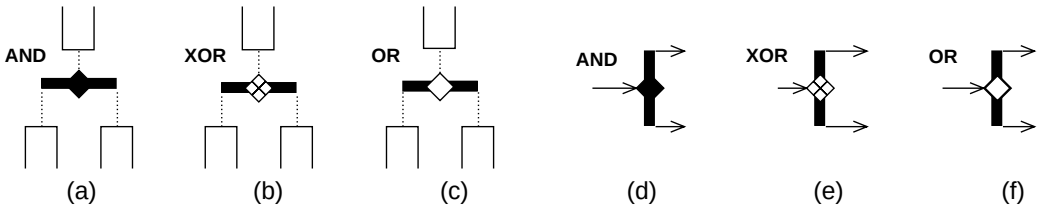


Abbildung B.2: Neue Protokolldiagramm-Elemente

Anhang C

Glossar

Der Grundstock für dieses Glossar stammt aus der Arbeit von Duvigneau [Duv02]. Zur Fortführung der Begriffsbildung wurde das Glossar erweitert.

Duvigneau differenziert in seiner Arbeit den Begriff Architektur. Zusammen mit der Differenzierung des Begriffs Multiagentensystem in der vorliegenden Arbeit entsteht eine gegenüber dem ursprünglichen Glossar veränderte Definition des Begriffs Multiagentensystem-Architektur. Der Begriff Agentennetz-Architektur kommt aus diesem Grund neu hinzu.

Duvigneau verfolgte mit dem Glossar die Absicht, möglichst allgemeingültige Begriffe darzustellen und so ein Begriffsgerüst zur FIPA darzustellen. Aus diesem Grund war das Stichwort CAPA z.B. nicht enthalten. In dieser Arbeit wird eher der Ansatz verfolgt, die tatsächlich verwendeten Begriffe zusammenfassend darzustellen, so wurde z.B. der Begriff Nachrichtentransportdienst als deutsche Bezeichnung für den von der FIPA definierten Begriff *Message Transport System* ins Glossar aufgenommen. Auf diese Weise ist das Glossar als Vorschlag für die weitere Verwendung der Begriffe zu verstehen.

Duvigneau betont den Unterschied zwischen der Abstrakten Architektur der FIPA und den FIPA2000-Spezifikationen. In dieser Arbeit sind jedoch die Unterschiede zwischen diesen beiden nicht Bestandteil der Betrachtung, deshalb wurde diese Unterscheidung auch im Glossar (außer bei der Definition selbst) fallen gelassen.

A

Abstrakte Architektur Gemeint ist im Kontext dieser Arbeit die „→*FIPA* Abstract Architecture Specification“ [FIPA 01], welche einen allgemeineren und flexibleren Rahmen für eine FIPA-konforme →*Kommunikationsarchitektur* definiert, als es die konkreten →*FIPA2000*-Spezifikationen leisten.

ACC Siehe →*Agent Communication Channel*.

ACE Siehe →*Agentcities Environment*.

ACL Siehe →*Agent Communication Language*.

Agentenplattform Unter einer Agentenplattform ist die Gesamtheit aus Hard- und Software zu verstehen, mittels derer Software-→*Agenten* ausgeführt werden. Plattformen können darüber hinaus verwendet werden, um Orte für Agenten zu modellieren.

Eine Agentenplattform kann auch als ein System aus mehreren Agenten mit einer gemeinsamen unmittelbaren Umgebung betrachtet werden (siehe →*Multiagentensystem*).

Agent Vielfältige Auffassungen machen eine Definition dieses Begriffs schwierig. In dieser Arbeit wird unter einem Agenten ein gekapselter Software-Agent verstanden, der auf einer →*Agentenplattform* angesiedelt ist und über Nachrichten mit seiner Umwelt, d.h. anderen Agenten, kommunizieren kann. Flexibles Verhalten und →*Autonomie* sind möglich, aber nicht zwingend für jeden Agenten erforderlich, denn auch ein Objekt kann als Spezialfall eines Agenten gesehen werden: Als rein reaktiver, nicht-autonomer Agent.

Agentcities Agentcities ist ein internationales Forschungsnetz. Es wird in zwei Projekten der Europäischen Kommission gefördert (Agentcities.NET und Agentcities.RTD). Agentcities.RTD ist für die Entwicklung und Bereitstellung eines →*Agentennetzes* verantwortlich, Agentcities.NET hat die Vernetzung der beteiligten Forschungsgruppen als Ziel, die z.T. auch finanziell unterstützt werden.

Das Agentennetz selbst besteht aus den beteiligten →*Agentenplattformen* (mit ihren →*Agenten* und deren →*Diensten*) und einem zentralen Knoten mit Verzeichnisdiensten. Dadurch wird eine →*Agentennetz-Architektur* vorgegeben. Um an das Agentennetz angebunden werden zu können, muss eine Agentenplattform einige Spezifikationen der →*FIPA* umsetzen.

Agent Communication Channel (ACC) Der ACC ist das zentrale Element des →*Nachrichtentransportdienstes* auf einer →*FIPA*-konformen →*Agentenplattform*. Mit seiner externen Nachrichtentransportschnittstelle ist der ACC für Versand, Empfang und Weiterleitung von Nachrichten zuständig, die plattformübergreifend versendet werden.

Agent Communication Language (ACL) Die ACL ist eine von der →*FIPA* definierte Sprache, die mittels ihrer Syntax, Semantik und Pragmatik die Kommunikation zwischen →*Agenten* aus unterschiedlichen Systemen erlaubt.

Agent Management System (AMS) Das in [FIPA 23] definierte AMS ist die zentrale Verwaltungskomponente einer →*FIPA*-konformen →*Agenten-*

plattform. Es hat die Oberhoheit über die →*Lebenszyklen* der auf der Plattform befindlichen →*Agenten* und bietet einen →*Verzeichnisdienst* zur Adressauflösung (→*weiße Seiten*) an. Das AMS kann (muss aber nicht) als Agent mit Sonderrechten implementiert werden. Nichtsdestotrotz muss das AMS unter der fixen Adresse „`ams@`*`hap`*“ wie ein Agent ansprechbar sein (wobei *`hap`* für den Namen der Plattform steht).

Agentcities Environment (ACE) Die Umgebung für die Anbindung von →*Capa* an →*Agentcities* umfasst in ihrer Grundausstattung einen Ping-Agent und einen Test-Agenten.

Agentennetz Dieser Begriff hat zwei Bedeutungen. In dieser Arbeit bezeichnet ein Agentennetz eine technisch abgegrenzte Vernetzung von Agenten mit einer gemeinsamen Infrastruktur, wie z.B. →*Agentcities*. Die Grenzen eines Agentennetzes laufen entlang der Plattformgrenzen, es kann also nicht ein Agent Teil des Netzes und ein anderer Agent derselben Plattform kein Teil des Netzes seins. Ein Agentennetz ist ein technisches System von mehreren Agenten im Gegensatz zu anwendungsbezogener Vernetzung (siehe →*Multiagentensystem*).

Innerhalb von →Mulan bezeichnet das Agentennetz das zentrale Referenznetz, mit dem ein Agent modelliert wird. In der Mehrzahl können auch die Wissensbasis und die Protokollfabrik eingeschlossen sein.

Agentennetz-Architektur Diese spezielle Ausprägung einer →*Architektur* umfasst das gesamte →*Agentennetz* in seinem grundlegenden Aufbau. Dabei werden nicht notwendigerweise alle detaillierten Architekturen wie die →*Entscheidungsarchitektur*, →*Plattformarchitektur* oder →*Kommunikationsarchitektur* gleich mit abgedeckt, sondern eher der grobe Rahmen für die Zusammenarbeit dieser Teilarchitekturen vorgegeben.

AMS Siehe →*Agent Management System*.

Architektur Eine Architektur beschreibt die grundlegende Konstruktion eines Systems. Dazu gehören die Elemente des Systems, deren Aufgaben und die Zusammenarbeit der Elemente, d.h. die Schnittstellen und Interaktionen zwischen den Elementen.

Unter einer Architektur wird je nach Anwendungsfeld eine andere Sichtweise auf ein System verstanden, daher wird hier zwischen den vier Ausprägungen →*Entscheidungsarchitektur*, →*Plattformarchitektur*, →*Kommunikationsarchitektur* und →*Multiagentensystem-Architektur* differenziert. Ein feststehender Begriff ist die →*Abstrakte Architektur*, welche eine Kommunikationsarchitektur der →*FIPA* ist.

AUML Diese Abkürzung steht für „**A**gent **UML**“; AUML ist also eine Erweiterung der →*Unified Modeling Language (UML)*. Ziel der AUML-Entwicklung ist es, eine allgemeine Semantik, Meta-Modell und abstrakte Syntax für agentenbasierte Methoden zu finden (siehe [OvDPB00]). Ein Schwerpunkt liegt dabei auf der Darstellung von →*Interaktionsprotokollen* für →*Agenten*. Die in AUML entwickelte Technik ist von der →*FIPA* in der „Interaction Protocol Library“ angewendet worden.

Autonomie Ein →*Agent* ist autonom, wenn er sein Verhalten und seinen Zustand selbst kontrollieren kann. Dies grenzt ihn vom Objekt ab, welches z.B. keine Kontrolle über den Aufruf seiner Methoden hat und nicht von sich aus – ohne äußeren Anreiz – seinen Zustand ändern kann. Allerdings hat die Autonomie immer ihre Grenzen, abhängig von den Beschränkungen, die die Umwelt dem Agenten auferlegt.

Die Voraussetzung für autonome Agenten ist die nebenläufige Ausführung von Agenten und ihre Kapselung, so dass nur der Agent selbst über seinen Zustand verfügen kann.

C

Capa Capa vereinigt das Konzept der petrinetzbasierten Agenten (→Mulan) und die von der →*FIPA* spezifizierte →*Kommunikationsarchitektur* zu einer standardkonformen →*Agentenplattform*. Zu den Besonderheiten von Capa gehört die Implementierung mit →*Referenznetzen* und Java, wodurch besonders gute Inspektionsmöglichkeiten der laufenden Plattform bestehen und wodurch eine besonders intuitive Modellierung und Spezifikation von nebenläufigen Prozessen möglich ist, ohne auf die Möglichkeiten von Java zu verzichten. Eine Besonderheit in der →*Plattformarchitektur* ist der Plattform-Agent, der Verwaltungsaufgaben der Plattform zusammen mit dem →*Nachrichtentransportdienst* und dem →*AMS* der Plattform erfüllt.

Capa verfügt über eine Implementierung von →*ACL*, →*SL0* und der Basisontologie. Als FIPA-konformes →*Transportmittel* steht eine Umsetzung des HTTP-MTPs (→*HTTP*) zur Verfügung. Dank der realisierten Anbindung an →*Agentcities* können →*Agenten* und ihre →*Dienste* im Internet bekannt gegeben werden.

CCL Die **C**onstraint **C**hoice **L**anguage ist eine der von der →*FIPA* vorgeschlagenen Sprachen, mittels derer der Inhalt von Nachrichten ausgedrückt werden kann.

D

DF Siehe →*Directory Facilitator*.

Directory Facilitator (DF) Diese Verwaltungskomponente einer →*FIPA*-konformen →*Agentenplattform* stellt einen →*Verzeichnisdienst* bereit. Der in [FIPA 23] definierte Dienst erlaubt die Registrierung von und Suche nach Dienste anbietenden →*Agenten* (→*gelbe Seiten*). Es kann mehrere Anbieter eines DF-Verzeichnisdienstes auf einer Plattform geben, auf jeder Plattform muss jedoch ein Standard-DF als Agent unter der Adresse „`df@`*`hap`*“ erreichbar sein (wobei *`hap`* für den Namen der Plattform steht).

Dienst In der Agententechnologie bezeichnet ein Dienst ein Angebot für →*Agenten*. Ein Dienst kann von einem Agenten, von einer →*Agentenplattform* oder von einer anderen Software angeboten werden. Jeder Dienst hat einen Namen und einen Anbieter. Dienstnamen werden in expliziten oder impliziten →*Ontologien* definiert.

Downlink Ein Downlink ist im Kontext der →*Referenznetze* eine der beiden Transitionsanschriften, die zum Aufbau eines →*synchronen Kanals* benötigt werden. Im Gegensatz zum →*Uplink* muss die mit dem Downlink versehene Transition über eine Referenz auf die Netzinstanz verfügen, zu der der Kanal aufgebaut werden soll.

E

Entscheidungsarchitektur Diese spezielle Ausprägung einer →*Architektur* in einem →*Multiagentensystem* legt fest, wie ein →*Agent* (re-)agiert und zu seinen Entscheidungen kommt. Entscheidungsarchitekturen werden vor allem in der (verteilten) Künstlichen Intelligenz betrachtet.

F

FIPA Siehe →*Foundation for Intelligent Physical Agents*.

FIPA2000 Dieser Begriff bezeichnet ein Paket von Spezifikationen der →*FIPA*, die eine →*Agentenplattform* aus der Sicht einer →*Kommunikationsarchitektur* beschreiben. Das Paket besteht zur Zeit aus rund 40 Dokumenten. →*Agenten* auf →*Plattformen*, die sich an diese Spezifikationen halten, können miteinander kommunizieren.

Trotz der suggestiven Jahreszahl im Namen dieses Pakets sind einige Spezifikationen jüngeren oder älteren Datums oder wurden in der Zwischenzeit überarbeitet. Leider bestehen an einigen Stellen Diskrepanzen

zur →*Abstrakten Architektur* der FIPA, so dass noch keine Interoperabilität mit anderen Instantiierungen der Abstrakten Architektur gegeben ist.

In dieser Arbeit bezieht sich der Begriff FIPA-kompatibel stets auf die FIPA2000-Spezifikationen.

FIPA-OS Unter dem Projektnamen „FIPA **O**pen **S**ource“ entstand die erste öffentlich verfügbare, kostenfrei nutzbare Implementierung einer →*Agentenplattform* nach →*FIPA*-Standards.

flexible Kante Flexible Kanten sind eine Besonderheit der →*Referenznetze*, damit kann eine Marke, die ein Java-Array enthält, in Marken für jedes Element des Arrays zerlegt werden. Andersherum können mehrere Marken zu einem Array bekannter Größe zusammengefasst werden.

Foundation for Intelligent Physical Agents (FIPA) Die FIPA ist eine internationale Organisation unter Beteiligung von Firmen und Universitäten, die es sich zur Aufgabe gemacht hat, durch die Entwicklung von Spezifikationen die Interoperabilität von →*Agenten* und agentenbasierten Anwendungen verschiedener Hersteller zu ermöglichen.

G

gelbe Seiten Dieses Schlagwort beschreibt einen →*Verzeichnisdienst* im Stile eines Branchenbuchs für Telefonnummern: Zu einer Dienstbezeichnung kann eine Menge von Anbietern dieses Dienstes ermittelt werden. Auf einer →*FIPA*-konformen →*Agentenplattform* wird dieser Dienst vom →*DF* geboten.

H

HTTP Das **H**yper**t**ext **T**ransfer **P**rotocol wird im Internet zur Übertragung von Webseiten verwendet. Die →*FIPA* hat in [FIPA 84] die Versendung von Agentennachrichten mittels HTTP spezifiziert.

I

IIOP Das **I**nternet **I**nter-**O**RB **P**rotocol wird im Rahmen der *Common Object Request Broker Architecture* (CORBA, [cor03b]) der Object Management Group (OMG) spezifiziert. Die →*FIPA* hat in [FIPA 75] die Versendung von Agentennachrichten mittels IIOP spezifiziert.

Interaktionsprotokoll Im Kontext der →*FIPA*-Spezifikationen beschreibt ein Interaktionsprotokoll den Ablauf einer →*Konversation* zwischen zwei oder mehr →*Agenten*, z.B. eine Auktion. Das Interaktionsprotokoll gibt die möglichen Antworten auf eine eingegangene Nachricht vor und engt so die unendlich große Menge möglicher Reaktionen auf ein sinnvolles Maß ein.

Interface Die englische Entsprechung des deutschen Wortes →*Schnittstelle* bezeichnet in der Regel Java-Schnittstellendefinitionen, welche durch das Schlüsselwort `Interface` gekennzeichnet werden.

J

JADE Das „**J**ava **A**gent **DE**velopment Framework“ stellt eine in Java implementierte →*Agentenplattform* nach →*FIPA*-Standards sowie einige Entwicklungswerkzeuge zur Verfügung.

JAS Das als „**J**ava **A**gent **S**ervices“ bezeichnete, in Entwicklung befindliche Bündel von Java-Paketen unter der `javax.agent`-Hierarchie definiert ein →*Interface*-Gerüst für Implementierungen der →*Abstrakten Architektur* der →*FIPA*.

K

Key-Value-Tuple (KVT) `KVT`s werden in in vielen Spezifikationen der →*FIPA* als flexible Datenstruktur eingesetzt. In einem `KVT` kann eine Menge von benannten Werten (*values*) abgelegt werden. Die Benennung durch Schlüssel (*keys*) erlaubt das gezielte Abfragen bestimmter Werte, wobei neben den offiziell spezifizierten Schlüsseln auch proprietäre Schlüssel verwendet werden dürfen. In der →*Abstrakten Architektur* wird darüber hinaus ein hierarchischer Namensraum für Schlüssel definiert (siehe [FIPA 01, S. 19]).

KIF Das **K**nowledge **I**nterchange **F**ormat ist eine der von der →*FIPA* vorgeschlagenen Sprachen, um den Inhalt von Nachrichten auszudrücken.

kommunikativer Akt Ein kommunikativer Akt ist das Grundelement der →*FIPA*-konformen Agentenkommunikationssprache (→*ACL*), angelehnt an die von Searle in [Sea69] ausgearbeitete →*Sprechakt*-Theorie. Ein →*Agent*, der eine Nachricht versendet, vollzieht einen kommunikativen Akt, durch den er den Empfänger der Nachricht zu etwas auffordert, etwas fragt, über etwas informiert, vor etwas warnt usw. Eine Reihe vorgefertigter Typen von kommunikativen Akten sind von der FIPA in Form von →*Performativen* vorgegeben.

Kommunikationsarchitektur Diese spezielle Ausprägung einer →*Architektur* besteht aus Vorgaben für die Kommunikationssprachen und die Interaktion zwischen →*Agenten*. Die in Kapitel 3.4 vorgestellten →*FIPA*-Spezifikationen beschreiben eine solche Architektur.

Konversation Im Allgemeinen versteht man unter einer Konversation ein Gespräch mit zwei oder mehr Beteiligten, so auch bei →*Multiagentensystemen*.

In der →*Entscheidungsarchitektur* des MULAN-Modells wird der Begriff auf technischer Ebene für ein instantiiertes →*Protokollnetz* verwendet. In den meisten Fällen entspricht dies dem intuitiven Begriff, weil das instantiierte Protokoll beschreibt, auf welche Art sich der →*Agent* an einer Konversation beteiligt. Protokollnetze können aber auch ohne Kommunikation arbeiten, wenn sie nur interne Überlegungen des Agenten beschreiben.

KVT Siehe →*Key-Value-Tuple*.

L

Lebenszyklus Die →*FIPA* sieht für einen →*Agenten* sechs mögliche Zustände in seinem Lebenszyklus vor: `unknown`, `initiated`, `active`, `waiting`, `suspended` und `transit`. Zwischen diesen Zuständen sind nur bestimmte Übergänge möglich. (siehe [FIPA 23])

M

MAS Siehe →*Multiagentensystem*.

Message Transport Service (MTS) Siehe →*Nachrichtentransportdienst*.

Message Transport Protocol (MTP) Siehe →*Transportmittel*.

Monitor Ein Monitor ist ein Konzept, welches in einer nebenläufigen Umgebung den wechselseitigen Ausschluss für kritische Abschnitte sicherstellen kann. Im geschützten Bereich des Monitors kann nur ein →*Thread* zur Zeit arbeiten, alle weiteren an den Daten interessierten Threads werden bis zur Freigabe des Monitors angehalten.

Mobilität Mobilität ist eine für Agenten erforschte Eigenschaft. Mit →*Agentenplattformen* ist die Modellierung eines Ortskonzeptes für Agenten möglich. Ein mobiler Agent überträgt seinen Zustand zu einer anderen Plattform, die z.B. andere Dienste anbietet.

MTP Kurz für →*Message Transport Protocol*, siehe →*Transportmittel*.

MTS Kurz für →*Message Transport Service*, siehe →*Nachrichtentransportdienst*.

Multiagentennetze (Mulan) Die in Kapitel 3.3 vorgestellten **Mul**ti**a**genten**n**etze beschreiben eine umfassende Modellierung von Agenten und ihrer Umgebung mittels geschachtelter →*Referenznetze*. Das System wird rekursiv in vier Ebenen modelliert: Von der äußersten, groben Systemsicht wird das Modell über die →*Agentenplattform* und den →*Agenten* bis zum →*Protokoll* immer detaillierter. Dank der operationalen Semantik der Referenznetze kann das MULAN-Modell ausgeführt werden. Mit MULAN können sowohl →*Agentennetze* als auch →*Multiagentensysteme* modelliert werden.

Multiagentensystem-Architektur Diese spezielle Ausprägung einer →*Architektur* beschreibt, wie eine agentenbasierte Anwendung (→*Multiagentensystem*) aus einzelnen Agenten zusammengefügt ist, d.h. wie Agenten entweder ein übergeordnetes Ziel erreichen oder ihre lokalen Ziele im Mittel gut erreichen können. Dabei können die detaillierteren Architekturen (→*Entscheidungsarchitektur*, →*Plattformarchitektur* und →*Kommunikationsarchitektur*) einbezogen sein.

Multiagentensystem (MAS) Dieser Begriff wird auf verschiedene Weise benutzt. Wo es sinnvoll scheint, sind in der folgenden Erläuterung alternative Begriffe angegeben, die in dieser Arbeit benutzt werden.

Zur Abgrenzung der Agententechnologie gegenüber verwandten Themengebieten wie der Künstlichen Intelligenz oder der Verteilten Systeme kann der Begriff Multiagentensysteme verwendet werden.

Ein Multiagentensystem als ein System von Agenten, die in einem Anwendungszusammenhang interagieren, kann grob folgendermaßen charakterisiert werden: Die Funktionalität des Systems wird durch mehrere →*Agenten* erbracht, die grundsätzlich nur über unvollständige Informationen bezüglich des Gesamtzustandes des Systems verfügen. Es gibt keine globale Kontrolle, Synchronisation oder Datenhaltung.

Ein →*Agentennetz* bezeichnet dagegen ein System von Agenten, die technisch untereinander vernetzt sind. Ob und inwiefern diese Agenten in einem oder mehreren Anwendungszusammenhängen interagieren, ist nicht Teil des Agentennetzes. Selbst eine →*Agentenplattform* wird zuweilen als Multiagentensystem beschrieben, da auch hier im Wortsinne mehrere Agenten ein System bilden.

Mulan Siehe →*Multiagentennetze*.

N

Nachrichtentransportdienst Der Nachrichtentransportdienst (englisch →*Message Transport Service (MTS)*) stellt sowohl in der →*Abstrakten Architektur* der →*FIPA* als auch in den konkreten →*FIPA2000*-Spezifikationen die grundlegenden Kommunikationsmöglichkeiten für die →*Agenten* bereit.

In der Abstrakten Architektur wird der MTS durch die Beschreibung seiner Dienstleistungen spezifiziert: Der Dienst erlaubt Agenten den Versand und Empfang von Nachrichten über die →*Transporte*, an die sich der Agent vorher gebunden hat. Zwecks Versand wird jede Nachricht in einem Umschlag von einer Transportbeschreibung begleitet, die die Adresse des Empfängers enthält und bei der Bindung des Empfängers an einen Transport erstellt wird.

In den Spezifikationen [FIPA 23] und [FIPA 67] der FIPA2000-Suite wird die Vielfalt der Transportmittel und die Entstehung der Transportbeschreibung nicht explizit modelliert. Stattdessen wird recht genau spezifiziert, wie der MTS bestimmte Informationen im Nachrichtenumschlag zu interpretieren hat. Die plattformübergreifende Schnittstelle des MTS wird auf einer FIPA2000-konformen →*Agentenplattform* vom →*Agent Communication Channel (ACC)* zur Verfügung gestellt, welcher aus den verfügbaren →*Transportmitteln* das passende auswählt. Die Schnittstelle zu den lokalen Agenten sowie die internen Kommunikationswege sind nicht Gegenstand der Spezifizikationen.

O

Ontologie Eine Ontologie definiert Begriffe, die zur Beschreibung und Repräsentierung eines Wissensgebiets verwendet werden können, sowie die Beziehungen und Eigenschaften der durch dieses Vokabular bezeichneten Objekte. In der →*FIPA*-Agentenkommunikation werden Ontologien herangezogen, um festzulegen, wie der Inhalt von Nachrichten interpretiert werden soll.

P

Performativ In dieser Arbeit bezieht sich der Begriff „Performativ“ auf den Bezeichner des Typs eines →*kommunikativen Akts*. Dabei handelt es sich meist um Verben, die die Intention des kommunikativen Akts verdeutlichen, wie zum Beispiel `inform`, `request` oder `propose`. In den →*FIPA*-Spezifikationen wird der Performativ-Begriff vermieden, lediglich das den

Typ einer →*ACL*-Nachricht festlegende Element trägt den Namen `performative`.

Plattform Siehe →*Agentenplattform*.

Plattformarchitektur Diese spezielle Ausprägung einer →*Architektur* beschreibt den technischen Aufbau einer →*Agentenplattform* mit allen Diensten, die den →*Agenten* zur Verfügung gestellt sind. Diese Sicht ist eher softwaretechnischen Ursprungs.

proaktiv Ein →*Agent* handelt proaktiv, wenn er von sich aus aktiv wird, also ohne äußeren Anreiz handelt. Dies kann z.B. in Folge der Ergebnisse seines eigenen Denkprozesses geschehen. Im Gegensatz dazu steht der →*reaktive* Agent, der nur auf externe Ereignisse reagiert. Mischformen aus reaktivem und proaktivem Verhalten sind üblich, die rein proaktive Form ist hingegen eher selten anzutreffen.

Protokoll Dieser Begriff ist mehrfach belegt, wobei in dieser Arbeit nach Möglichkeit die spezielleren Begriffe genutzt werden:

Im Kontext eines Standard-Agenten von →MULAN beschreiben Protokollnetze in Zusammenarbeit mit einer Wissensbasis das Verhalten des →*Agenten*.

Einige →*FIPA*-Spezifikationen definieren einzelne →*Interaktionsprotokolle*, die dem Ablauf einer Kommunikation zwischen zwei Agenten einen festen Rahmen geben.

Im Nachrichtentransportsystem einer →*FIPA*-konformen →*Agentenplattform* wird der Begriff Transportprotokoll (→*MTP*) alternativ zu →*Transport* verwendet. In dieser Arbeit wird dafür der Begriff →*Transportmittel* genutzt.

Zur Übertragung von Nachrichten benutzt jedes Transportmittel ein spezifisches Internet-Protokoll (siehe →*Transportmittel*).

R

RDF Das **R**esource **D**escription **F**ramework ist eine der von der →*FIPA* vorgeschlagenen Sprachen für den Nachrichteninhalt.

reaktiv Ein rein reaktiver →*Agent* handelt immer genau dann, wenn er Ereignisse aus der Umwelt wahrnimmt, also z.B. eine Nachricht empfängt. Im Gegensatz dazu steht der →*proaktive* Agent, der als Ergebnis seines Überlegungsprozesses von sich aus Handlungen anstößt. Mischformen aus reaktivem und proaktivem Verhalten sind üblich.

Referenznetze Referenznetze sind Petrinetze höherer Ordnung mit der Anschriftssprache Java, in denen Marken wiederum Netze sein können. Zur Kommunikation zwischen Netzinstanzen steht das Konzept der →*synchronen Kanäle* zur Verfügung.

Eine Einführung in die Semantik der Referenznetze bietet Kapitel 2.

Renew Der **Re**ference **Ne**t **W**orkshop ist eine grafische Entwicklungsumgebung, in der →*Referenznetze* gezeichnet und simuliert (→*Simulation*) werden können.

S

Schnittstelle Im Unterschied zu seinem englischen Pendant →*Interface* wird in dieser Arbeit der deutschen Begriff im allgemeinen Sinn verwendet. Er steht damit für die Beschreibung des Randes eines gekapselten Gebiets mit fest definierten Interaktions- und Kommunikationsmöglichkeiten.

Simulation In der Petrinetzgemeinde (und auch in dieser Arbeit) wird mit Simulation die Ausführung von Netzen bezeichnet. Dieses Verständnis sollte nicht mit dem weit verbreiteten Simulationsbegriff verwechselt werden, der die Nachbildung und Analyse realer Prozesse in einem Modell bezeichnet. Zwar ist die Nachbildung realer Prozesse durch die Ausführung von Petrinetzen möglich, aber der Petrinetz-Simulationsbegriff ist nicht spezifisch darauf festgelegt. Er ist eher wie die interpretierte Ausführung von Programmen zu verstehen.

Siedler Siedler ist ursprünglich ein Brettspiel mit variablem Spielfeldaufbau und vielfältigen Verhandlungsmöglichkeiten zwischen den Spielern. Nach dieser Vorlage wurde ein agentenorientiertes (→*Agent*) Spiel mit →*Capa* als Infrastruktur und →*Renew* als Entwicklungsumgebung umgesetzt. Dabei können verschiedene Aspekte beobachtet werden: Die Tauglichkeit und Verbesserung der Infrastruktur, das Vorgehen bei der agentenorientierten Softwareentwicklung, die Nutzung und Umsetzung von agentenspezifischen Eigenschaften wie →*Autonomie* und →*Mobilität*, das Anbieten und Nutzen von →*Diensten* in einem →*Agentennetz* sowie planende und verhandelnde Agenten. Für die Aspekte, die in den Regeln der Brettspielvorlage nicht abgedeckt sind, können erweiterte Regeln entworfen werden.

SL Die **S**emantic **L**anguage ist eine der von der →*FIPA* vorgeschlagenen Sprachen, mit denen der Inhalt von Nachrichten ausgedrückt werden kann (siehe [FIPA 08]). Für die Kommunikation mit den Verwaltungskomponenten einer →*FIPA*-konformen →*Agentenplattform* wird eine in

der Ausdrucksfähigkeit eingeschränkte Variante namens „SL0“ verwendet.

SL wird außerdem zur formalen Beschreibung der Semantik der in der „Communicative Act Library“ [FIPA 37] definierten →*Performative* eingesetzt.

SL0 →*SL*.

Sprechakt Siehe →*kommunikativer Akt*.

synchroner Kanal Im →*Referenznetzformalismus* können mittels eines synchronen Kanals zwei Transitionen, die auch zu verschiedenen Netzinstanzen gehören dürfen, für die Dauer eines Schaltvorgangs miteinander verschmolzen werden. Während dieser →*Synchronisation* ist ein bidirektionaler Informationsaustausch zwischen den beiden Netzinstanzen möglich. Ein Kanal wird durch zwei Transitionsanschriften erzeugt, durch →*Uplink* und →*Downlink*.

Synchronisation In einem nebenläufigen System müssen hin und wieder voneinander unabhängige Handlungsstränge miteinander synchronisiert werden, um z.B. Informationen auszutauschen oder die Erledigung unabhängiger Teilaufgaben abzuwarten. In einem Petrinetz kann Synchronisation sehr intuitiv modelliert werden. →*Referenznetze* erlauben darüber hinaus die Synchronisation von Transitionen aus unterschiedlichen Netzinstanzen mittels eines →*synchronen Kanals*.

T

TCP/IP Das auf dem ***I**nternet **P**rotocol* aufsetzende ***T**ransmission **C**ontrol **P**rotocol* dient als Grundlage für verbindungsorientierte Kommunikation im Internet. Viele Internet-Anwendungen bauen auf dieser kurz „TCP/IP“ genannten Protokollkombination auf.

Thread Ein Thread ist ein sequentieller Strang der Programmausführung, voneinander unabhängige Threads können nebenläufig ausgeführt werden. Eine alternative Bezeichnung für „Thread“ wäre „Prozess“, wenn nicht der Prozessbegriff bereits von vielen Betriebssystemen belegt wäre.

Die Programmiersprache Java unterstützt Threads zusammen mit dem →*Monitor*-Konzept zur →*Synchronisation* der nebenläufigen Stränge.

Transport Die →*Abstrakte Architektur* der →*FIPA* versteht unter einem „Transport“ sowohl ein Nachrichtentransportprotokoll als auch den Dienst des Nachrichtentransports über dieses Protokoll. Ein →*Agent* kann sich beim

Nachrichtentransportdienst für die Nutzung verschiedener Transporte registrieren und darüber Nachrichten versenden und empfangen.

Transportmittel Die auf einer →*Agentenplattform* verfügbaren Transportmittel erbringen zusammen mit dem lokalen →*ACC* den →*Nachrichtentransportdienst*. Jedes Transportmittel sorgt für den physikalischen Nachrichtenaustausch über ein spezifisches Internet-Protokoll. Von der →*FIPA* wurden Transportmittel für drei Internet-Protokolle spezifiziert: →*HTTP*, →*WAP* und →*IIOP*.

Der Begriff Transportmittel wird in dieser Arbeit durchgehend als deutsche Bezeichnung für →*Message Transport Protocol (MTP)* verwendet.

Nach →*FIPA2000* entspricht dem Transport in etwa ein →*Message Transport Protocol*, mit dem Unterschied, dass ein Agent sich nicht für ein Protokoll anmeldet, sondern der →*ACC* entscheidet, welches der verfügbaren Transportmittel geeignet ist.

U

UML Siehe →*Unified Modeling Language*.

Unified Modeling Language (UML) Die von der OMG (*Object Management Group*) zum Standard erhobene **U**nified **M**odeling **L**anguage definiert verschiedene Diagrammarten, die bei der objektorientierten Analyse, Spezifikation und Programmierung von Nutzen sind.

Uplink Ein Uplink ist im Kontext der →*Referenznetze* eine der beiden Transitionsanschriften, die zum Aufbau eines →*synchronen Kanals* benötigt werden. Im Gegensatz zum →*Downlink* hat die mit dem Uplink versehene Transition keine Referenz auf die andere am Kanal beteiligte Netzinstanz.

URL Ein **U**niform **R**esource **L**ocator adressiert eine im Internet abrufbare Ressource. Für URLs ist eine feste Syntax vorgegeben. Das mit der Syntax vorgegebene Schema umfasst die Internet-Adresse des ressourcenanbietenden Rechners, das zum Abruf zu verwendende Internet-Protokoll, den Pfad zur Ressource auf dem Rechner und ggf. Parameter für den Ressourcenabruf.

V

Verzeichnisdienst Dieser Dienst verwaltet für die ihn nutzenden →*Agenten* ein Verzeichnis, in dem Adressen von Agenten hinterlegt werden können, die – wahlweise nach dem Prinzip der →*weißen* oder →*gelben Seiten*

– gesucht werden können. Während die →*Abstrakte Architektur* nur die Existenz mindestens eines Verzeichnisdienstes vorschreibt und eine Grobstruktur der Verzeichniseinträge vorgibt, trennt die →*FIPA2000*-Spezifikation die Aufgabenbereiche in gelbe und weiße Seiten und sieht mindestens zwei Dienste vor: →*DF* und →*AMS*.

W

WAP Das **W**ireless **A**pplication **P**rotocol wird verwendet, um Internetkommunikation für drahtlose Geräte wie z.B. Handys zu ermöglichen. Die →*FIPA* hat die Versendung von Agentennachrichten mittels WAP spezifiziert.

weiße Seiten Dieses Schlagwort beschreibt einen →*Verzeichnisdienst* im Stile des klassischen Telefonbuchs: Zu einem Namen kann eine Adresse (im Telefonbuch z.B. die Telefonnummer) ermittelt werden. Auf einer →*FIPA-2000*-konformen →*Agentenplattform* wird dieser Dienst vom →*AMS* geboten; die erhältliche Information ist eine Agentenidentifikation mit Angaben für den →*Message Transport Service*.

X

XML Die E**x**tensible **M**arkup **L**anguage gibt ein allgemeines Schema vor, wie anwendungsspezifische Informationen so in hierarchisch strukturierter Textform abgelegt werden können, dass sie sowohl für Menschen als auch maschinell lesbar sind.

Anhang D

Literaturverzeichnis

[acn03] *Agentcities.NET, EU-Projekt.* Repräsentiert im Internet unter `http://www.agentcities.org/EUNET/`, 2003.

[acr03a] *Agentcities.RTD, EU-Projekt.* Repräsentiert im Internet unter `http://www.agentcities.org/EURTD/`, 2003.

[acr03b] Agentcities.RTD: *D22: (Network Integration) Coordination Services*, 2003. Verfügbar im Internet unter `http://www.agentcities.org/EURTD/Contents/Documents/Deliverables/WP2/ACITIES-D22-230102-APPROVED-1.0A.pdf`.

[acs03a] *Agentcities (Agentennetz).* Im Internet zugreifbar unter `http://www.agentcities.net/`, 2003.

[acs03b] *Agentcities (Organisation).* Repräsentiert im Internet unter `http://www.agentcities.org/`, 2003.

[acs03c] Agentcities: *Agentcities Interoperability Test Suite*, 2003. Verfügbar im Internet unter `http://www.agentcities.org/Testsuite/agentcities_testsuite_v1.11.pdf`. Die Testsuite kann im Internet genutzt werden unter `http://agentcities.crm-paris.com/services/Testsuite.jsp` (Letzter Zugriff: 24.3.2003).

[act02a] Agentcities Task Force: *Agentcities Network Architecture Recommendation*, 2002. Verfügbar im Internet unter `http://www.agentcities.org/rec/00001/actf-rec-00001a.pdf`.

[act02b] Agentcities Task Force: *Connection to the Agentcities Network Recommendation*, 2002. Verfügbar im Internet unter `http://www.agentcities.org/rec/00001/actf-rec-00002a.pdf`.

[agl03] *AgentLink, EU-Projekt.* Repräsentiert im Internet unter `http://www.agentlink.org/`, 2003.

[BG03] Brickley, Dan und R.V. Guha: *RDF Vocabulary Description Language 1.0: RDF Schema. Homepage.* `http://www.w3.org/TR/rdf-schema`, 5. September 2003.

[Bro02] Brown, David: *A Simple, Multithreaded WebServer*, 1997, zuletzt aktualisiert: 13. 8. 2002. Verfügbar im Internet unter `http://developer.java.sun.com/developer/technicalArticles/Networking/Webserver/`.

[Bro86] Brooks, R. A.: *A robust layered control system for a mobile robot.* IEEE Jounal of Robotics and Automation 2, (1):14–23, 1986.

[BRP+03] Bellifemine, Fabio, Giovanni Rimassa, Agostino Poggi, Tiziana Trucco, Giovanni Caire, Elisabetta Cortese, Filippo Quarta und Giosu Vitaglione: *Java Agent DEvelopment Framework. Homepage.* `http://sharon.cselt.it/projects/jade/`, 2003.

[BSS03] Bergenti, Federico, Onn Shehory und Arnon Sturm: *Agent-Oriented Software Engineering.* In: Luck, Michael, Wiebe van der Hoek und Carles Sierra (Herausgeber): *AgentLink* EASSS *2003*, Seiten 125–158. AgentLink, Februar 2003.

[Cab02] Cabac, Lawrence: *Entwicklung von geometrisch unterscheidbaren Komponenten zur Vereinheitlichung von Mulan-Protokollen.* Studienarbeit, Universität Hamburg, Fachbereich Informatik, 2002.

[Car03] Carl, Timo: *Evaluation und beispielhafte Erweiterung einer referenznetzbasierten Agentenumgebung.* Studienarbeit, Universität Hamburg, FB Informatik, August 2003.

[CFK+03] Crub, Monica, Ray Fergerson, Holger Knublauch, Mark Musen, Natasha Noy, Samson Tu und Jennifer Vendetti: *Protege-2000. Homepage.* `http://protege.stanford.edu/`, 2003.

[CMR03] Cabac, Lawrence, Daniel Moldt und Heiko Rölke: *A Proposal for Structuring Petri Net-Based Agent Interaction Protocols.* In: *Lecture Notes in Computer Science: 24th International Conference on Application and Theory of Petri Nets (ICATPN), Eindhoven, Netherlands*, Seiten 102–120, Berlin Heidelberg New York, Juni 2003. Springer-Verlag.

[cor03a] *Community Research and Development Information Service (Cordis). Projekt-Datenbank der IST.* Zugreifbar im Internet unter `http://www.cordis.lu`, 2003.

[cor03b] Object Management Group (OMG): *Common Object Request Broker Architecture*, 2003. Verfügbar im Internet unter `http://www.omg.org/technology/documents/corba_spec_catalog.htm`.

[DKW02] Duvigneau, Michael, Olaf Kummer und Frank Wienberg: *Renew - User Guide*. Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 1.6 Auflage, September 2002.

[DMR02] Duvigneau, Michael, Daniel Moldt und Heiko Rölke: *Concurrent Architecture for a Multi-agent Platform*. In: Giunchiglia, Fausto, James Odell und Gerhard Weiß (Herausgeber): *Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002, Revised Papers and Invited Contributions*, Band 2585 der Reihe *LNCS*. Springer-Verlag, 2002.

[Duv02] Duvigneau, Michael: *Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, Dezember 2002.

[EGVC98] Ezpeleta, J., F. García-Vallés und J.M. Colom: *A Class of Well Structured Petri Nets for Flexible Manufacturing Systems*. In: Desel, J. und M. Silva (Herausgeber): *Proceedings of 19nd International Conference on Application and Theory of Petri Nets 1998 (ICATPN 1998), Lissabon, Portugal*, Band 1420 der Reihe *LNCS*, Seiten 64–83, Berlin Heidelberg, 1998. Springer-Verlag.

[eur03] *Europäische Kommission*. Repräsentiert im Internet unter `http://europa.eu.int/comm/research`, 2003.

[fip03] FIPA: *Foundation for Intelligent Physical Agents. Homepage*. `http://www.fipa.org`, 2003.

[FIPA 01] Foundation for Intelligent Physical Agents: *FIPA Abstract Architecture Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00001/`.

[FIPA 08] Foundation for Intelligent Physical Agents: *FIPA SL Content Language Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00008/`.

[FIPA 23] Foundation for Intelligent Physical Agents: *FIPA Agent Management Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00023/`.

[FIPA 26] Foundation for Intelligent Physical Agents: *FIPA Request Interaction Protocol Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00026/`.

[FIPA 37] Foundation for Intelligent Physical Agents: *FIPA Communicative Act Library Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00037/`.

[FIPA 61] Foundation for Intelligent Physical Agents: *FIPA ACL Message Structure Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00061/`. .

[FIPA 67] Foundation for Intelligent Physical Agents: *FIPA Agent Message Transport Service Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00067/`.

[FIPA 70] Foundation for Intelligent Physical Agents: *FIPA ACL Representation in String Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00070/`. .

[FIPA 75] Foundation for Intelligent Physical Agents: *FIPA Agent Message Transport Protocol cor IIOP Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00075/`. .

[FIPA 84] Foundation for Intelligent Physical Agents: *FIPA Agent Message Transport Protocol for HTTP Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00084/`.

[FIPA 85] Foundation for Intelligent Physical Agents: *FIPA Agent Message Transport Envelope Representation in XML Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00085/`.

[FIPA 87] Foundation for Intelligent Physical Agents: *FIPA Agent Management Support for Mobility Specification*, 2003. Deprecated. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00087/`.

[FIPA 93] Foundation for Intelligent Physical Agents: *FIPA Messaging Interoperability Service Specification*, 2003. Verfügbar im Internet unter `http://www.fipa.org/specs/fipa00093/`.

[FZD+03] Finin, Youyong Zou, Li Ding, Harry Chen und Rong Pan: *A travel market Framework in Agentcities (Taga).* In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 30. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`.

[Gom03] Gomez, Mario: *Open, Reusable and Configurable Multi-Agent Systems (Orcas).* In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 36. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`.

[GV02] Gireault, Claude und Rüdiger Valk: *Petri nets for system engineering: a guide to modeling, verification and application.* Springer-Verlag, Berlin, 2002.

[ire03] *On the Integration of Restaurant Services (Ires).* In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 21. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`.

[Jac02] Jacob, Thomas: *Implementation einer sicheren und rollenbasierten Workflow-Managementkomponente für ein Petrinetzwerkzeug.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, 2002.

[jam02] *Java Agent Message Router (JAMR).* Im Internet verfügbar unter `http://liawww.epfl.ch/JAMR/` (letzter Zugriff: 12. 11. 2002), November 2002.

[Jen03] Jennings, Nick: *Applying Agent Technology.* In: Luck, Michael, Wiebe van der Hoek und Carles Sierra (Herausgeber): *AgentLink* EASSS *2003*, Seiten 263–280. AgentLink, Februar 2003.

[Klu03] Klusch, Matthias: *Intelligent Information Agents.* In: Luck, Michael, Wiebe van der Hoek und Carles Sierra (Herausgeber): *AgentLink* EASSS *2003*, Seiten 51–88. AgentLink, Februar 2003.

[KMR01] Köhler, Michael, Daniel Moldt und Heiko Rölke: *Modelling the Structure and Behaviour of Petri Net Agents.* In: Colom, J.M. und M. Koutny (Herausgeber): *Proceedings of the 22nd Conference on Application and Theory of Petri Nets*, Band 2075 der Reihe *LNCS*, Seiten 224–241. Springer-Verlag, Juni 2001.

[KMR03] Köhler, Michael, Daniel Moldt und Heiko Rölke: *Modelling Mobility and Mobile Agents Using Nets within Nets.* In: Aalst, Wil van der und Eike Best (Herausgeber): *Proceedings of 24nd International Conference on Application and Theory of Petri Nets 2003 (ICATPN 2003), Eindhoven, NL*, Band 2679 der Reihe *LNCS*, Seiten 121–139, Berlin Heidelberg New York, 2003. Springer-Verlag.

[KSI+03] Kurumatani, Koichi, Akio Sashima, Noriaki Izumi, Yutaka Matsuo und Shigeyoshi Hiratsuka: *Cognitive Resource Managing Agents in Ubiquitous Computing (Consorts).* In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 13. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`.

[Kum02] Kummer, Olaf: *Referenznetze.* Logos-Verlag, Berlin, 2002. Dissertation (Universität Hamburg, Fachbereich Informatik).

[KW99] Kummer, Olaf und Frank Wienberg: *Renew – The Reference Net Workshop.* Petri Net Newsletter, 56:12–16, 1999.

[LMP03] Luck, Michael, Peter McBurney und Chris Preist: *Agent Technology: Enabling Next Generation Computing. A Roadmap for Agent Based Computing.* AgentLink, 2003. Verfügbar im Internet unter `http://www.agentlink.org/roadmap`.

[MO04] Moldt, Daniel und Jan Ortmann: *A Conceptual and Practical Framework for Web-based Processes.* In: *Fundamental Approaches to Software Engineering (FASE)*, 2004. Eingereicht zur FASE'04.

[Mor03] Moreno, Antoni: *Deployment of agent-based healthware Services (Grusma).* In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 16. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`.

[Ort03] Ortmann, Jan: *Prozeß-Ontologien in Multiagentensystemen – Eine prototypische Umsetzung von DAML-S Beschreibungen in Petrinetzen und ihre Verwendung in Mulan.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, 2003.

[OvDPB00] Odell, James, H. van D. Parunak und B. Bauer: *Extending UML for Agents.* In: Wagner, Gerd, Yves Lesperance und Eric Yu (Herausgeber): *Agent-Oriented Information Systems. Workshop at the 17th National Conference on Artificial Intelligence (AAAI), AOIS 2000*, Seiten 3–17, 2000.

[PPW] Poutakidis, David, Lin Padgham und Michael Winikoff. In: *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS'02)*, Seiten 960–967. Cristiano Castelfranchi and Johnson, W. Lewis, Juli. Verfügbar im Internet unter `http://goanna.cs.rmit.edu.au/~winikoff/Papers/aamas02-debug.ps`.

[RDK+03] Reese, Christine, Michael Duvigneau, Michael Köhler, Daniel Moldt und Heiko Rölke: *Agent–based Settler Game.* In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 25. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`.

[ren03] *Renew – The Reference Net Workshop. Homepage.* `http://www.renew.de/`, 1998–2003.

[Rod03] Rodriguez, Juan: *iBundler: A decision support service for negotiating agents.* In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 18. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`.

[Röl04] Rölke, Heiko: *Agenten und Agentensysteme: Modellierung mit Referenznetzen (Arbeitstitel).* Dissertation, Universität Hamburg, Fachbereich Informatik, erscheint 2004.

[Röl99] Rölke, Heiko: *Multi-Agenten-Netze – Modellierung und Implementation eines Multi-Agenten-Systems auf der Basis von Referenznetzen.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 1999.

[Sea69] Searle, John R.: *Speech Acts.* Cambride University Press, 1969.

[Tan03] Tanenbaum, Andrew S.: *Computer Networks.* Pearson Studium, München u.a., 2003.

[Teu03] Teuber, Klaus: *Die Siedler von Catan. Homepage.* Weitere Informationen: `http://www.diesiedlervoncatan.de` und `http://www.KlausTeuber.de`, 2003.

[Wie03] Wiering, Marco: *Course on Reinforcement Learning.* In: Luck, Michael, Wiebe van der Hoek und Carles Sierra (Herausgeber): *AgentLink* EASSS *2003*, Seiten 333–352. AgentLink, Februar 2003.

[Wie96] Wienberg, Frank: *Multiagentensysteme auf der Basis Gefärbter Petri-Netze.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, November 1996.

[WJ95] Wooldridge, Michael und Nicholas R. Jennings: *Intelligent Agents: Theory and Practice.* Knowledge Engineering Review, 10(2):115–152, 1995.

[WL03] Wooldridge, Mike und Michael Luck: *Introduction to Agents*. In: Luck, Michael, Wiebe van der Hoek und Carles Sierra (Herausgeber): *AgentLink* EASSS *2003*, Seiten 3–34. AgentLink, Februar 2003.

[wsd03] *WSDL-Tool*. In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 44. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`. Das Tool ist benutzbar im Internet unter `http://sas.ilab.sztaki.hu:8080/wsdltool/`.

[xse03] *Communication Security in Multi-Agent Systems (X-Security*. In: *Agentcities Agent Technology Competition (ATC03), Barcelona, Spain*, Seite 47. Agentcities.NET, Februar 2003. Verfügbar im Internet unter `http://www.agentcities.org/EUNET/ID3/documents/exh_program.pdf`. Das Tool ist im Internet verfügbar unter `http://agents.felk.cvut.cz/security/`.

Erklärung

Ich erkläre hiermit, dass ich diese Arbeit selbstständig durchgeführt und nur die angegebenen Hilfsmittel und Quellen verwendet habe.

Ich bin mit der Einstellung in den Bestand der Bibliothek einverstanden.

Hamburg, 26. November 2003

Christine Reese