

Lars Ehrler

# Formale Semantik für AgentUML Sequenzdiagramme

Diplomarbeit für den akademischen Grad „Diplom Medieninformatiker“

Technische Universität Dresden, Fakultät Informatik, Institut für Künstliche Intelligenz  
betreuender Hochschullehrer: **Prof. Dr. rer.nat. Michael Thielscher**

eingereicht am 04.11.2005

---

---

# Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur, Verweise und Hilfsmittel erstellt zu haben. Verwendete Quellen wurden als solche gekennzeichnet. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

---

Dresden, 04.11.2005

---

---

# Danksagung

Intelligente Systeme, speziell Multi-Agenten-Systeme habe ich durch mein Studium an der University of Otago in Dunedin, Neuseeland kennen- und auch lieben gelernt. Speziell Dr. Stephen Cranefield hat mein Interesse für diesen hochspannenden Bereich der Informatik geweckt, mir jede Menge Fragen beantwortet und steht mir auch von Deutschland aus bei Fragen und Problemen bei.

Der von mir verwendete CPN-Simulator „JFern“ sowie das zu Testzwecken benutzte Agentensystem „Opal“ wurde von Mariusz Nowostawski entwickelt, der mir auch während meiner Diplomarbeit eine Menge E-mails beantworten mußte.

Vielen Dank an Prof. Dr. Michael Thielscher in Dresden, der die Betreuung dieser Arbeit übernommen hat, bei unseren Treffen immer gut vorbereitet war und mir bei schwierigen Entscheidungen mit Rat zur Seite stand.

Danke auch an den Lehrstuhl für Internettechnologie an der BTU Cottbus mit Prof. Dr. Gerd Wagner, Frau Karla Kersten und Herrn Ekkehard Schwaar, die mir für meine Arbeit eine sehr gute Arbeitsumgebung schufen und mich in ihr Team aufnahmen. Danke auch an Bastian Schenke für die vielen guten Diskussionen über UML und Petri-Netze.

Vielen Dank an Diana Walter, Thomas Raak und Silvana Jung für das Korrekturlesen dieser Arbeit und für die vielen nützlichen Hinweise, die ich dabei bekommen habe.

Ich danke meiner Frau, Katharina Ehrler, für ihre Liebe, ihre Unterstützung und auch Geduld, wenn ich mich nicht so um sie kümmern konnte, wie es notwendig gewesen wäre. Danke auch für die vielen Hinweise und Erfahrungen aus der eigenen Diplomarbeitszeit.

Der größte Dank geht an meinen himmlischen Vater, der mich erschaffen, mir meine Fähigkeiten gegeben und es mir ermöglicht hat, an diesem Thema an vorderster Front mitzuarbeiten.

---

---

# Aufgabenstellung

Für die Sprache AgentUML soll eine formale Semantik und, falls notwendig, auch eine Syntax definiert werden. Dazu werden die verschiedenen Arten von formalen Semantiken evaluiert sowie die bisherige Anwendung von Formalismen auf UML untersucht. Nach der Entscheidung für einen Ansatz zur Semantikdefinition soll dieser auf AgentUML angewendet werden.

Darauf aufbauend wird ein Werkzeug entwickelt, das es einem Agenten erlaubt, in AgentUML definierte Interaktionsprotokolle automatisch zu interpretieren. Dieses Werkzeug bekommt Interaktionsdiagramme (in serialisierter Form) als Eingabe. Ein Agent erhält dann von diesem Werkzeug durch Angabe seines derzeitigen Zustandes und der bisher ausgeführten Aktionen die Menge der nach dem Protokoll möglichen Aktionen.

---

---

# Zusammenfassung

Interaktionsprotokolle in Multi-Agenten-Systemen sind Hilfsmittel für Agenten-Entwickler. Sie spezifizieren die Art und den Ablauf der Kommunikationen zwischen Agenten und innerhalb von Agentengruppen.

AgentUML wurde entwickelt, um Erfahrungen aus dem Bereich der objekt-orientierten Softwareentwicklung mit UML im Bereich der Spezifizierung von Interaktionsprotokollen wiederzuverwenden. Es enthält UML-Sequenzdiagramme, die für die Benutzung in Agentensystemen angepaßt und erweitert wurden. Die Spezifizierung beinhaltet jedoch bisher nur eine informelle Beschreibung der Sprachelemente (konkrete Syntax).

In der vorliegenden Arbeit wurde, aufbauend auf Vorarbeiten, ein Metamodell (abstrakte Syntax) für AgentUML entwickelt. Weiterhin wurden verschiedene Formalismen zur Semantikdefinition evaluiert, um dann mit Hilfe von Farbigen Petri-Netzen (Coloured Petri Nets (CPN)) eine formale Semantik von AgentUML zu entwerfen.

Unter Verwendung dieser formalen Semantik wurde dann ein Werkzeug geschaffen, das Interaktionsprotokolle, die in AgentUML spezifiziert sind, analysiert und von einem Agenten benutzt werden kann, um solche Interaktionsprotokolle zu interpretieren und zu benutzen.

---

---

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>xi</b>
<b>Tabellenverzeichnis</b>	<b>xiii</b>
<b>Abkürzungsverzeichnis</b>	<b>xv</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Aufbau . . . . .	2
<b>2 Hintergrund</b>	<b>3</b>
2.1 Intelligente Agenten . . . . .	3
2.1.1 Was ist ein Agent? . . . . .	4
2.1.2 Multi-Agenten-Systeme . . . . .	5
2.1.3 Agenten-orientierte Programmierung (AOP) . . . . .	7
2.1.4 Agenten Kommunikationssprachen (Agent Communication Language - ACLs) . . . . .	8
2.1.5 Ontologien . . . . .	10
2.1.6 Interaktionsprotokolle . . . . .	12
2.2 UML - the Unified Modeling Language . . . . .	14
2.2.1 Model Driven Architecture . . . . .	14
2.2.2 Metamodellierung . . . . .	15
2.2.3 Unified Modeling Language (UML) . . . . .	16
2.2.4 Object Constraint Language OCL . . . . .	16
2.2.5 Interaktionsprotokolle und Sequenzdiagramme in UML . . . . .	17
2.3 Sprachwissenschaften / Formale Methoden . . . . .	17
2.3.1 Überblick über die Linguistik . . . . .	18
2.3.2 Was ist Semantik? . . . . .	19
2.3.3 Formale Methoden . . . . .	20
2.3.4 Geschichte und Klassifizierung formaler Semantiken . . . . .	20
2.3.5 Eigenschaften und Anwendungsgebiete einer semantischen Definition	22
2.3.6 Formalismen zur Semantikdefinition . . . . .	23
2.4 Petri-Netze . . . . .	26
2.4.1 Petri-Netze . . . . .	26
2.4.2 Farbige Petri Netze (Coloured Petri Nets - CPN) . . . . .	27
2.5 Modellierung von Interaktionsprotokollen mit AgentUML und CPN . . . . .	28

2.5.1	Interaktionsprotokolle modelliert mit AgentUML . . . . .	29
2.5.2	Interaktionsprotokolle modelliert mit CPN . . . . .	30
2.5.3	Vergleich von Farbigen Petri-Netzen und AgentUML . . . . .	31
<b>3</b>	<b>Formale Semantik von AgentUML</b>	<b>33</b>
3.1	Formalismen auf AgentUML angewendet . . . . .	33
3.1.1	AgentUML durch Modallogik definiert . . . . .	33
3.1.2	AgentUML in Petri-Netze übersetzen . . . . .	35
3.2	AgentUML-Metamodell . . . . .	35
3.2.1	Struktur des Metamodells . . . . .	37
3.2.2	Klassenbeschreibung des Metamodells . . . . .	39
3.3	AgentUML-Semantik . . . . .	41
<b>4</b>	<b>Design von PAUL</b>	<b>45</b>
4.1	Zielstellung . . . . .	45
4.2	Vorgehensweise . . . . .	46
4.2.1	Spezifikation von AgentUML-Sequenzdiagrammen . . . . .	46
4.2.2	Übersetzung von AgentUML nach CPN . . . . .	48
4.2.3	Erstellung des entsprechenden Teilnetzes . . . . .	48
4.2.4	Agentenspezifisches Verhalten . . . . .	48
4.2.5	Ausführen einer Konversation . . . . .	49
4.3	Aufbau . . . . .	49
4.3.1	Benutzte Werkzeuge . . . . .	50
4.4	Funktionsweise . . . . .	50
4.4.1	Initialisierung eines Protokolls . . . . .	51
4.4.2	Initialisierung einer Konversation . . . . .	51
4.4.3	Ausführung einer Konversation . . . . .	52
<b>5</b>	<b>Fazit und Ausblick</b>	<b>53</b>
5.1	Evaluation möglicher Formalismen . . . . .	53
5.2	Evaluation der AgentUML-Semantik . . . . .	54
5.3	Evaluation von PAUL . . . . .	55
5.4	Ausblick . . . . .	56
	<b>Literaturverzeichnis</b>	<b>57</b>
<b>A</b>	<b>CD-Rom</b>	<b>73</b>
A.1	Inhalt . . . . .	73
A.2	Anleitung . . . . .	73
<b>B</b>	<b>AgentUML-Standardisierungsvorschlag für die FIPA</b>	<b>75</b>

# Abbildungsverzeichnis

1.1	Nutzung von Interaktionsprotokollen . . . . .	2
2.1	Historische Entwicklung von Programmier-Paradigmen . . . . .	7
2.2	Beispielkonversation in KQML & FIPA-ACL . . . . .	10
2.3	Ontologien für Kommunikation . . . . .	11
2.4	Überblick über MDA (Born u. a. 2004) . . . . .	14
2.5	Überblick über MDA (Kempa u. Mann 2005) . . . . .	14
2.6	Metamodellierungsbeispiel anhand eines Sequenzdiagramm-Fragmentes . .	15
2.7	Beispiel-Sequenzdiagramm . . . . .	17
2.8	Zwei-Ebenenstruktur der Semantik . . . . .	19
2.9	Petri-Netz Beispiel . . . . .	26
2.10	Ein AgentUML-Beispiel-Diagramm . . . . .	29
2.11	Ein Interaktionsprotokoll mit nur einem CPN modelliert . . . . .	30
2.12	Interaktionsprotokoll mit je einem CPN für jede beteiligte Rolle . . . . .	31
3.1	Metamodell-Pakete von AgentUML . . . . .	36
3.2	Vererbungshierarchie des AgentUML-Metamodells . . . . .	36
3.3	UML-Klassendiagramm der wichtigsten Klassen des AgentUML-Metamodells	38
3.4	Metamodell für ein Agenten-Strukturdiagramm . . . . .	39
3.5	verschiedene CPN-Muster . . . . .	42
3.6	Beispiel: FIPA Request Interaction Protocol . . . . .	43
4.1	Vorgehensweise bei einer IP-Ausführung durch PAUL 2 . . . . .	46
4.2	Editor-Plug-In für AgentUML in Eclipse . . . . .	47
4.3	Beispiel AgentUML-Sequenzdiagramm . . . . .	48
4.4	Umwandlung des Diagramms aus Abbildung 4.3 in ein CPN-Teilnetz . . . .	49
4.5	Struktur von PAUL . . . . .	50
4.6	Initialisierung eines Interaktionsprotokolls . . . . .	51
4.7	Initialisierung einer Konversation . . . . .	51
4.8	Vollzug der Konversation . . . . .	52

---

---

# Tabellenverzeichnis

2.1 Historische Entwicklung von Programmier-Paradigmen . . . . .	8
--	---

---

---

# Abkürzungsverzeichnis

ACL .....	<b>A</b> gent- <b>C</b> ommunication- <b>L</b> anguage, Seite 8
AgentUML .....	<b>A</b> gent <b>U</b> nified <b>M</b> odeling <b>L</b> anguage, Seite 28
AOP .....	<b>A</b> genten- <b>o</b> rientierte <b>P</b> rogrammierung, Seite 7
ASM .....	<b>A</b> bstract <b>S</b> tate <b>M</b> achines, Seite 21
CPN .....	<b>C</b> oloured <b>P</b> etri <b>N</b> ets, Seite 27
CSP .....	<b>C</b> ommunicating <b>S</b> equential <b>P</b> rocesses, Seite 26
FIPA .....	<b>F</b> oundation for <b>I</b> ntelligent <b>P</b> hysical <b>A</b> gents, Seite 9
IP .....	<b>I</b> nteraktions <b>p</b> rotokolle, Seite 12
KIF .....	<b>K</b> nowledge <b>I</b> nterchange <b>F</b> ormat, Seite 9
KQML .....	<b>K</b> nowledge <b>Q</b> uery and <b>M</b> anipulation <b>L</b> anguage, Seite 9
MAS .....	<b>M</b> ulti- <b>A</b> genten- <b>S</b> ystem, Seite 5
MDA .....	<b>M</b> odel <b>D</b> riven <b>A</b> rchitecture, Seite 14
OCL .....	<b>O</b> bject <b>C</b> onstraint <b>L</b> anguage, Seite 16
OMG .....	<b>O</b> bject <b>M</b> anagement <b>G</b> roup, Seite 14
OWL .....	Web Ontology Language, Seite 25
UML .....	<b>U</b> nified <b>M</b> odeling <b>L</b> anguage, Seite 14

---

---

# 1

## Einleitung

„Since the beginning of recorded history, people have been fascinated with the idea of non-human agencies. Popular notions about androids, humanoids, robots, cyborgs, and science fiction creatures permeate our culture, forming the unconscious backdrop against which software agents are perceived.“<sup>1</sup>(Bradshaw 1997a, Seite 3)

### 1.1 Motivation

Die Idee von nicht-menschlichen Agenten ist fast so alt wie die Menschheit selbst. Doch erst mit der Entwicklung der Computertechnik rückte ihre Realisierung Schritt für Schritt näher. Mittlerweile fahren fern- und selbstgesteuerte Fahrzeuge auf dem Mars spazieren und an Fließbändern werden Menschen durch Roboter ersetzt. Von echter Intelligenz und Autonomie kann man dabei jedoch noch nicht sprechen, da jede Aktion und Reaktion von Menschen vorgeplant und durchdacht wurde. Bei der Entwicklung von intelligenten Systemen und Agenten fehlen immer noch grundlegende Werkzeuge, die es diesen Systemen ermöglichen, sich in unbekanntem Umgebungen autonom zu rechtezufinden.

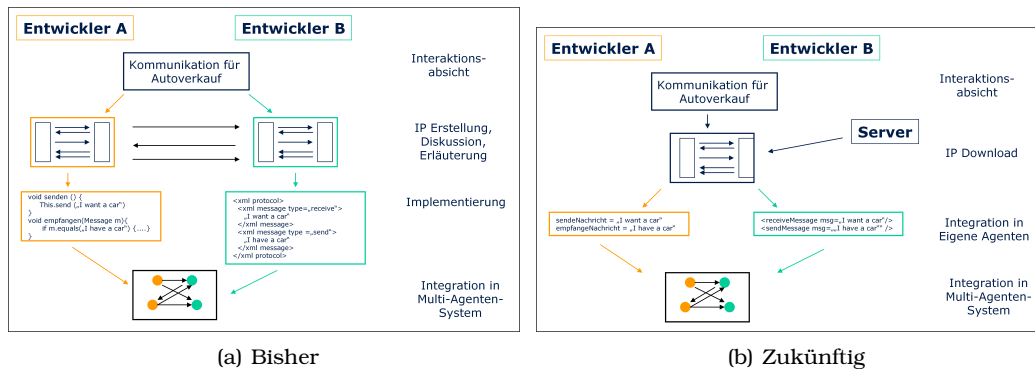
Eines dieser Werkzeuge sind Interaktionsprotokolle. Diese ermöglichen es, daß sich intelligente Systeme miteinander unterhalten können. Sie bestimmen, welche Nachrichten für die Erreichung eines (Kommunikations-)Zieles erforderlich sind.

Für die Implementation einer Kommunikation war es bisher notwendig, sich ein Interaktionsprotokoll zu überlegen. Dieses mußte dann, wenn mehrere Agenten miteinander kommunizieren sollten, den verschiedenen Agenten-Entwicklern zugänglich ge-

---

<sup>1</sup>Deutsche Übersetzung: Seit der Zeit, als geschichtliche Aufzeichnungen anfangen, waren Menschen von der Idee nicht-menschlicher Wesen fasziniert. Populäre Bezeichnungen wie Androiden, Humanoiden, Roboter, Cyborgs und andere Science-Fiction-Kreaturen haben unsere Kultur durchdrungen und damit den unbewußten Hintergrund für die Wahrnehmung von Software-Agenten geschaffen.

macht und erklärt werden. Jeder einzelne hatte das Protokoll dann in seinen Agenten zu implementieren, bevor die Agenten miteinander kommunizieren konnten. Schematisch ist dieser Ablauf in Abbildung 1.1(a) abgebildet.



**Abbildung 1.1:** Nutzung von Interaktionsprotokollen

Einfacher und effizienter wäre es, wenn Interaktionsprotokolle wiederverwendet werden könnten. So könnte es zum Beispiel ein Archiv von verschiedenen Protokollen im Internet geben. Wenn nun Agenten miteinander kommunizieren sollen, einigen sich die Entwickler auf ein zu nutzendes Protokoll, laden dies herunter und integrieren es in ihre Agenten (siehe auch Abbildung 1.1(b)).

Um diese Art von Wiederverwendbarkeit zu ermöglichen, benötigen Interaktionsprotokolle eine formale Semantik (Huguet 2004). Diese verhindert, daß Diagramme unterschiedlich oder falsch interpretiert werden können. Sie legen eine eindeutige, berechenbare Bedeutung fest und ermöglichen so die automatische Interpretation von Interaktionsprotokollen.

Die vorliegende Arbeit beinhaltet für die Interaktionsprotokoll-Modellierungssprache AgentUML (FIPA AUML 2003) eine formale Syntax sowie Semantik und beschreibt ein darauf basierendes Werkzeug zur automatischen Interpretation von AgentUML-Diagrammen.

## 1.2 Aufbau

Die vorliegende Arbeit ist folgendermaßen aufgebaut:

Kapitel 2 auf der nächsten Seite enthält eine Zusammenstellung der bisherigen Arbeiten in den verschiedenen Themenbereichen, die diese Arbeit berührt: Intelligente Agenten, Unified Modeling Language, Formale Methoden, Petri-Netze und die Modellierung von Interaktionsprotokollen.

Die entwickelte Formale Semantik von AgentUML wird dann in Kapitel 3 auf Seite 33 beschrieben. Eine vollständige, englischsprachige Spezifikation kann in Anhang B auf Seite 75 nachgelesen werden.

Ein prototypisches Werkzeug zur Interpretation von AgentUML-Sequenzdiagrammen wird in Kapitel 4 auf Seite 45 vorgestellt, worauf abschließend Bemerkungen in Kapitel 5 auf Seite 53 folgen.

# 2

## Hintergrund

Diese Diplomarbeit hat ihren Hintergrund in vier Forschungsbereichen der Informatik. Zum Ersten der Bereich der Multi-Agenten-Systeme, einem Teilgebiet der Künstlichen Intelligenz. Zum Zweiten der Bereich der UML (Unified Modeling Language), die eine aus der Praxis der Software-Entwicklung kommende Modellierungssprache ist. Des Weiteren der Bereich der formalen Sprachen, die ihre Wurzeln in der Mathematik und der Sprachwissenschaft hat, jedoch mittlerweile viele Gebiete der Informatik beeinflusst. Und als letztes das Gebiet der Petri-Netze, einer formalen Sprache zur Modellierung von nebenläufigen Prozessen.

Im Folgenden wird in Abschnitt 2.1 die bisherige Forschung an intelligenten Agenten näher beleuchtet. Anschließend kann in Abschnitt 2.2 die Geschichte der Sprache UML nachgelesen werden. Ausgehend von den Sprachwissenschaften wird in Abschnitt 2.3 eine Einführung in die Theorie der formalen Sprachen, insbesondere in den Bereich der formalen Semantik gegeben. Darauf aufbauend wird die Sprache der Farbigen Petri-Netze in Sektion 2.4 vorgestellt. Eine kurze Einführung in die Modellierung von Interaktionsprotokollen bildet in Abschnitt 2.5 den Schluss dieses Kapitels.

### 2.1 Intelligente Agenten

Die in der Künstlichen Intelligenz schon länger erforschten und entwickelten intelligenten Agenten haben in den letzten 15 Jahren größeren Einfluß und Beachtung in der praktischen Softwaretechnik gefunden. Wie Wooldridge (2002, Seite 312f) ausführt: „Interest in agent systems grew very rapidly in the first half of the 1990s. There were several reasons for this. The first, and probably most important reason was the spread of the Internet, which through the 1990s changed from being a tool unknown outside

academica to something in everyday use for commercial and leisure across the globe.“<sup>1</sup>

Im Folgenden wird nach dem Versuch einer Definition der Begriffe „Agent“ und „Multi-Agenten-System“ die agenten-orientierte Programmierung vorgestellt, woraufhin die Konzepte von Agentenkommunikationssprachen, Ontologien und Interaktionsprotokolle erläutert werden.

Komprimierte Einführungen in das Gebiet der Multi-Agenten-Systeme sind von Lämmel u. Cleve (2001, Abschnitt 1.3) und Vlacic u. a. (2000) erschienen, ausführlichere Werke wurden unter anderem von Bradshaw (1997b), Weiß (1999) oder Wooldridge (2002) geschrieben.

### 2.1.1 Was ist ein Agent?

Es gibt keine einheitliche Definition des Begriffes „Agenten“. Bei Schumacher (2001) werden verschiedene aufgeführt, so auch die von Franklin u. Graesser (1997). Diese definieren einen autonomen Agenten als ein System, das seine Umgebung wahrnimmt und in dieser Umgebung agiert, um eigene Ziele zu verfolgen. Diese allgemeine Definition ist nicht auf Computer oder gar Software-Agenten beschränkt, sondern kann auch auf humane Agenten angewandt werden. Wooldridge (1999) argumentiert ähnlich; stellt jedoch gleichzeitig fest, daß die Definition eines Agenten stark davon abhängt, in welchem Kontext er verwendet wird.

Lämmel u. Cleve (2001) erläutern den Agentenbegriff im Kontext der Informatik/Software-technik durch Angabe einer Reihe von Eigenschaften:

- **autonom:** Ein Agent handelt selbständig bei der Erfüllung der übertragenen Aufgabe und trifft dabei eigenständige Entscheidungen.
- **reaktiv:** Ein Agent reagiert auf seine Umwelt, nimmt Daten aus der Umwelt auf und reagiert mit seinen Aktionen darauf.
- **zielorientiert:** Die Aktivitäten eines Agenten sind an der Erfüllung seines Zieles orientiert.
- **kontinuierlich:** Ein Agent ist für eine gewisse oder unendliche Zeit ein kontinuierlich aktiver Prozess.
- **kommunikativ:** Ein Agent besitzt die Fähigkeit, mit seinem „Auftraggeber“, aber auch mit anderen Agenten zu kommunizieren und zu kooperieren - er hat also soziales Verhalten.
- **lernend:** Ein Agent ist lernfähig.
- **mobil:** Ein Agent kann mobil sein. Bei einem Roboter bedeutet dies, daß er sich örtlich bewegen kann; bei einem Software-Agenten, daß er sich mittels Computer-Netzen von einem Rechner zu einem anderen Rechner bewegen kann. Es ist umstritten, ob dies eine notwendige Eigenschaft eines Agenten ist.

---

<sup>1</sup>Deutsche Übersetzung: Das Interesse an Agentensystemen stieg sehr schnell in der ersten Hälfte der 90er Jahre. [...] Der erste, und wahrscheinlich wichtigste Grund dafür war die Ausbreitung des Internets, das sich in den 90er Jahren von einem nur in akademischen Kreisen bekannten Werkzeug zu etwas wandelte, das jeder überall auf der Welt für Handel und Freizeit nutzt.

---

Analoge Charakteristika werden auch von Bradshaw (1997a) zur Definition eines Agenten genutzt, der dabei betont, daß die Eigenschaften je nach Aufgabe und Verwendung des Agenten unterschiedlich stark ausgeprägt sind.

Eine ausführliche Charakterisierung und Klassifikation von Agenten wurde von Brenner u. a. (1998) zusammengestellt. Auch dort wird ausgeführt, daß verschiedene Fachdisziplinen unterschiedliche Anforderungen an Agenten haben. Als einen intelligenten Software-Agenten wird dort ein Softwareprogramm bezeichnet, „das für einen Benutzer bestimmte Aufgaben erledigen kann und dabei einen Grad an Intelligenz besitzt, der es befähigt, seine Aufgaben in Teilen autonom durchzuführen und mit seiner Umwelt auf sinnvolle Art und Weise zu interagieren.“ (Brenner u. a. 1998)

Die wohl wichtigste Eigenschaft eines Agenten ist die Autonomie. Wooldridge (1999) und ähnlich auch Shoham (1997) oder Silva u. de Lucena (2004) haben die Unterschiede zwischen Agenten und Objekten diskutiert. Wooldridge (1999) betont dabei insbesondere drei Unterschiede, die alle im Zusammenhang mit der Autonomie eines Agenten stehen:

- Agenten haben Kontrolle über ihr Verhalten. Wenn eine Methode eines Objektes aufgerufen wird, dann hat dieses Objekt keine Kontrolle darüber. Ein Agent hingegen kann nur gefragt werden, ob er etwas tut - er kann nicht dazu gezwungen werden.
- Agenten verhalten sich flexibel. Sie können agieren oder reagieren - selbst Prozesse initiieren oder auf die Initiative anderer Agenten warten.
- Agenten besitzen (mindestens) einen eigenen Kontrollbereich. So wird beim Programmieren von Software-Agenten jedem Agenten normalerweise mindestens ein eigener „thread“ zugewiesen.

### 2.1.2 Multi-Agenten-Systeme

Wenn mehrere Agenten zusammenarbeiten, dann spricht man von „Multi-Agenten-Systemen“ (MAS). Nach Wooldridge (2002, Anhang A) war die Forschung in Multi-Agenten-Systemen lange Zeit unabhängig von der Forschung von einzelnen Agenten. Erst in den 80er Jahren wurde der Begriff eines Multi-Agenten-Systems geprägt.

Ein Multi-Agenten-System hat (außer den in Abschnitt 2.1.1 genannten Eigenschaften eines einzelnen Agenten) nach Singh u. Huhns (2005, Seite 342) folgende Eigenschaften:

- "**Dynamism**": eine dynamische Änderung der Beteiligung/Mitgliedschaft der Akteure/Agenten.
  - "**Scale**": Multi-Agenten-Systeme sind mittels der beteiligten Anzahl von Agenten skalierbar.
  - "**Control Structure**": Die Kontrollstrukturen können zwischen Hierarchie zu Demokratie frei definiert werden.
  - "**Coordination**": Die Koordination der Komponenten (Agenten) wird vom Selbstinteresse (und zwar dem Interesse, das eigene Ziel zu erreichen) getrieben.
  - "**Uniqueness**": Ein Multi-Agenten-System kann homogen als auch heterogen sein.
-

- **"Interface Autonomy"**: Die Schnittstelle zwischen den Agenten kann mittels Vokabular, Sprachen und Protokollen frei definiert werden, genauso wie durch die Spezifikation von Zielen, Ontologien und Fähigkeiten ein einzelner Agent von dem jeweiligen „Besitzer“ definiert wird.

Multi-Agenten-Systeme sind laut Wooldridge (2002) interdisziplinär: Sie berühren Bereiche von Ökonomie, Philosophie, Logik, Ökologie und Sozialwissenschaften. Er sieht daher zwei unterschiedliche Sichtweisen auf das Verständnis von MAS:

- **Software-Paradigma**: Software wird stetig komplexer und nähert sich den Strukturen der realen Welt immer mehr an. Dabei stellt sich vor allem die Interaktion zwischen den Komponenten als die größte Herausforderung dar. Multi-Agenten-Systeme stellen ein Paradigma zur Verfügung, mit der eben diese Komplexität modelliert und entsprechende Software erstellt werden kann.
- **Simulation**: Die Simulation von Gesellschaften und anderen komplexen Vorgängen ist für die Sozialwissenschaften unumgänglich. Multi-Agenten-Systeme können dazu genutzt werden, da es mit ihnen sehr einfach ist, große Mengen an Individuen zu modellieren und ihr Verhalten zu simulieren.

In dieser Arbeit wird die Sicht von Multi-Agenten-Systemen als Softwaretechnik-Paradigma genutzt (siehe auch Abschnitt 2.1.3). In der Softwaretechnik sind die Anwendungsgebiete von Multi-Agenten-Systemen vielfältig. Wichtige Beispiele sind (nach Murch u. Johnson (2000), Wooldridge (2002) und Zhang u. Zhang (2004)):

- **Informationszugriff und -suche**: („*Semantic Web*“) Sowohl auf dem heimischen Desktop als auch im Internet können Agenten eingesetzt werden, um bestimmte, vom Benutzer verlangte Informationen zu suchen, auszuwerten, aufzubereiten und zu verwalten.
  - **Elektronischer Handel**: („*E-Business*“) Die Umstellung des Handels auf elektronische Plattformen nimmt ständig zu. Um dies zu automatisieren, können zur Informationssammlung und -aufbereitung, Produktauswahl, zum Aufstellen von Spezifikationen, zu Vertragsverhandlungen, zur Entscheidungsvorbereitung und zu weiteren Aufgaben Agenten eingesetzt werden.
  - **Mensch-Computer-Schnittstellen**: („*Ambient Intelligence*“ oder auch „*ubiquitous computing*“) Die Bereitstellung von benutzerfreundlichen Schnittstellen ist eine der wichtigsten Forschungsbereiche der Informatik. Intelligente, lernfähige Agenten können dazu beitragen, indem sie die Nutzer-Vorlieben lernen und deuten, Vereinfachungen vorschlagen und aufgrund ihrer Kenntnisse den Nutzer bei seiner Arbeit unterstützen. Sie sind dabei in vielen kleinen und großen technischen Geräten enthalten, die uns umgeben, und ermöglichen einen einfachen Zugriff auf diese Ressourcen.
  - **Entwicklung verteilter Systeme**: („*Grid-Computing*“) Wie oben schon angedeutet, bieten Agenten durch ihre hohe Abstraktion und ihre Kapselung große Vorteile in der Entwicklung von verteilten und komplexen Software-Systemen und helfen, eine effiziente Nutzung von Ressourcen zu realisieren.
-

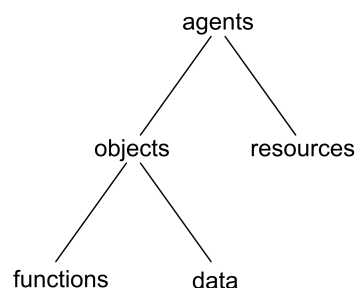
- **Simulation:** Wie schon angedeutet werden Agenten als Werkzeug benutzt, um komplexe Vorgänge und Verhaltensweisen in großen Mengen von Individuen (zum Beispiel Menschen in menschlichen Gesellschaften oder Atome in Gasen) zu simulieren und zu erforschen. Entsprechende Tools, wie zum Beispiel „Netlogo“ (Wilensky 1999), sind im Internet frei verfügbar.
- **Geschäftsprozeßorganisation:** („*Workflow and Buisness Process Management*“) Arbeitsabläufe und Geschäftsprozesse werden immer stärker automatisiert. Die Organisation erfordert adaptive und intelligente Systeme, die auf Veränderungen reagieren und Sonderanforderungen berücksichtigen können. Agentenbasierte Systeme werden hierzu verstärkt eingesetzt (siehe zum Beispiel das System „JBees“ (Ehrler u. a. 2003)).

### 2.1.3 Agenten-orientierte Programmierung (AOP)

Die in Abschnitt 2.1.1 auf Seite 4 genannte Autonomie veranlaßte Shoham (1997), von einem neuen Softwareentwicklungs-Paradigma zu sprechen: der sogenannten „Agenten-orientierten Programmierung“ (AOP). Diese Einschätzung wird von immer mehr Forschern aufgegriffen (zum Beispiel von Zambonelli u. Omicini (2004)) und dabei als Weiterentwicklung der bisherigen Programmier-Paradigmen angesehen (Lind 2001a). Abbildung 2.1 und Tabelle 2.1 verdeutlichen den Zusammenhang der Programmier-Paradigmen. Agenten werden dabei von Lind (2001a) als eine Weiterentwicklung der objekt-orientierten Programmierung gesehen, wobei ein Agent ein Objekt mit Unabhängigkeit, Initiative und eigenen Ressourcen ist.

Die aktuelle Forschung arbeitet an originären Methodiken für AOP. Einige wichtige sind:

- **MaSE:** (Multiagent Systems Engineering) (Wood u. DeLoach 2001) enthält Richtlinien für die Entwicklung von MAS auf der Grundlage von mehreren Schritten durch die Analyse- und Entwicklungsphase. Sie basiert stark auf objekt-orientierten Methoden, reichert diese jedoch mit neuen, agenten-spezifischen Konzepten an.
- **Tropos:** (Bresciani u. a. 2004) erweitert die bekannten objekt-orientierten Methoden mit den agenten-orientierten Konzepten „Überzeugungen“, „Wünsche“ und „Absichten“.



**Abbildung 2.1:** Historische Entwicklung von Programmier-Paradigmen (Lind 2001a, Abbildung 2.9)

	Structural Unit	Relation to previous Level
Machine Language	Program	
Structured Programming	Subroutine	Bounded unit of Program
Object Oriented Programming	Object	Subroutine + persistent local state
Agent Oriented Programming	Agent	Object + independent thread of execution + initiative

**Tabelle 2.1:** Historische Entwicklung von Programmier-Paradigmen (Lind 2001a, Tabelle 2.2)

- **Gaia:** (Wooldridge u. a. 2000; Zambonelli u. a. 2003) baut auf der Metapher einer Organisation auf. Hier werden die Analyse und das Design eines MAS als die Konstruktion einer rechnerbasierten Organisation aufgefaßt.

Eine ausführliche Abhandlung von agenten-orientierten Methoden und Methodologien wurde von Henderson-Sellers u. Giorgini (2005) erarbeitet; kürzere Einführungen können unter anderem bei Wooldridge (2002, Seite 225ff) oder bei Arazy u. Woo (2002) nachgelesen werden.

Es gibt verschiedene Vorteile, die für den Einsatz von MAS in der Software-Entwicklung sprechen:

- MAS ermöglichen einen hohen Grad an Kapselung und Abstraktion. Da Agenten unabhängig sind, kann jeder Agent selbst die Strategie aussuchen, die er zur Lösung des gestellten Problems nutzen möchte. Die Agenten in einem MAS können von verschiedenen (Software-)Entwicklern entwickelt werden und trotzdem problemlos zusammenarbeiten, solange sie die Fähigkeit haben, miteinander zu kommunizieren. Dies ist ein sehr hoher Grad an Abstraktion und Kapselung (Sycara 1998).
- MAS stellen eine Plattform für offene, verteilte Systeme bereit. Sie erlauben ein System, das sich zur Laufzeit ändert, ohne daß alle Komponenten des Systems schon im Vorfeld spezifiziert wurden. Dies ermöglicht dem System, sich während des Betriebes an die sich verändernden Gegebenheiten anzupassen. (Dafür ist jedoch eine entsprechende Infrastruktur notwendig) (Ehrler u. a. 2005).
- Im Gegensatz zu herkömmlichen Verfahren, bei denen eine zentrale Instanz die Verwaltung von kritischen oder beschränkten Ressourcen vornimmt, erfolgt die Konfliktlösung bei MAS auf Basis von Verhandlungen zwischen den betroffenen Agenten. Die konkurrierenden Interessen der im System modellierten Beteiligten spiegeln sich auf natürliche Weise in den einzelnen Agenten des Systems wider (Hutter u. a. 2003).

#### 2.1.4 Agenten Kommunikationssprachen (Agent Communication Languages - ACLs)

Wie in der menschlichen Gesellschaft ist auch in Multi-Agenten-Systemen das grundlegende Medium für Informationsaustausch Sprache. Es gibt verschiedene Ansätze, eine

passende Sprache zu definieren. Finin u. a. (1997) nennt einige Anforderungen an eine Kommunikationssprache für Agenten:

- Die Sprache sollte deklarativ, syntaktisch einfach und für Menschen leicht lesbar sein. („Eine Nachricht in einer deklarativen Sprache beschreibt das Ziel einer möglichen Kette von Aktionen [...]. Anschaulich gesprochen wird beschrieben, was der Sender vom Empfänger erwartet“ (Geihs u. a. 2001, Seite 129)).
- Es sollte möglich sein, zwischen der Kommunikationssprache und der Sprache, die den Inhalt beschreibt, zu unterscheiden. Dabei beschreibt die Kommunikationssprache den Kommunikationsprozess und die Inhaltssprache die zu übermittelnde Information.
- Die Semantik der Sprache sollte formal beschrieben sein.
- Es sollte eine effiziente Implementierung der Sprache vorhanden oder zumindest möglich sein.

In der Forschung wurden verschiedene Sprachen entwickelt. Chaib-draa u. Dignum (2002), und ähnlich auch Dignum u. Greaves (2000a), sehen vor allem zwei deklarative, sprechakt-basierte Sprachen, die am wichtigsten für Multi-Agenten-Systeme sind:

- **KQML**: („*Knowledge Query and Manipulation Language*“ (KQML 1993)), wurde für den Austausch von Informationen zwischen wissensbasierten Systemen entwickelt. Finin u. a. (1997) nutzten KQML als Agentenkommunikationssprache in Verbindung mit KIF („*Knowledge Interchange Format*“ (KIF 1992)) als Inhalts-Sprache.
- **FIPA-ACL**: („*FIPA Agent Communication Language*“ (FIPA-ACL 2002)) wurde von der „*Foundation for Intelligent Physical Agents*“ (FIPA) als „verbessertes KQML“ speziell für die Anwendung in Multi-Agenten-Systemen entwickelt.

FIPA-ACL war laut Wooldridge (2002, Seite 175ff) aufgrund einiger Kritiken an KQML entstanden. Hauptkritikpunkte waren dabei die fehlende formale Semantik von KQML sowie die Menge der Performative<sup>2</sup>, die nach Ansicht vieler zu groß, ungenau festgelegt und unvollständig war. Aus diesem Grunde kann man FIPA-ACL als eine Art weiterentwickeltes KQML ansehen. Geihs u. a. (2001) stellten in einem Vergleich fest:

- Beide Sprachen sind deklarative Sprachen, deren Syntax sehr ähnlich ist.
- Beide Sprachen trennen Inhalt und Nachricht und eignen sich selbst nicht zur Wissensrepräsentation, sondern nutzen dafür andere Sprachen (wie etwa KIF oder Prolog).
- Beide Sprachen nutzen Sprechakte.

Semantisch dagegen gibt es große Unterschiede, vor allem im Bereich der Performative. Die Performative in FIPA-ACL wurden im Vergleich zu KQML vereinfacht, um Verpflichtungen erweitert sowie mit einer formalen Semantik versehen. Auch im Bereich der

---

<pre>(ask-one :sender Joe :content "price(ibm,X) " :receiver Ticker :reply-with ibm-stock :language Prolog :ontology NYSE-TICKS)</pre>	<pre>(query-ref :sender Joe :content "price(ibm,X) " :receiver Ticker :reply-with ibm-stock :language Prolog :ontology NYSE-TICKS)</pre>
<pre>(tell :sender Ticker :content "price(ibm,14.32) " :receiver Joe :in-reply-to ibm-stock :language Prolog :ontology NYSE-TICKS)</pre>	<pre>(inform :sender Ticker :content "price(ibm,14.32) " :receiver Joe :in-reply-to ibm-stock :language Prolog :ontology NYSE-TICKS)</pre>

**Abbildung 2.2:** Beispielkonversation in KQML & FIPA-ACL nach Geihs u. a. (2001, Abbildung 2)

Mediationsverfahren - also der Nachrichtenübermittlung über einen Mittler-Agenten - sind große semantische Unterschiede zu sehen.

Abbildung 2.2 zeigt eine einfache Konversation, links in KQML, rechts in FIPA-ACL. Wie man erkennen kann, sind die Nachrichten fast gleich - der einzige Unterschied besteht in den verwendeten Performativen „ask-one“ vs. „query-ref“ bzw. „tell“ vs. „inform“. Genau darin liegt auch der Hauptunterschied der beiden Sprachen: Die möglichen Performative der Sprachen sowie deren semantische Unterlegung sind komplett unterschiedlich.

Chaib-draa u. Dignum (2002) führen aus, daß die Schwäche beider Sprachen ihre semantische Definition ist, die sich auf die mentalen Zustände, Überzeugungen und Absichten der Agenten stützt. Bis heute werden Agenten jedoch selten mit direkter Unterstützung der mentalen Zustände programmiert und daher existiert hier eine große Lücke zwischen Theorie und Praxis. Dieses Problem wird von Singh (2000) angegangen, der eine auf Verpflichtungen basierende Kommunikationstheorie entwickelte, abgeleitet von philosophischen Kommunikationstheorien nach Habermas (1981). Der Vorteil dieser „sozialen Semantik“ von Singh (2000) liegt darin, daß ein Agent nicht über den internen mentalen Zustand eines anderen Agenten nachdenken muß, sondern wählen kann, ob er den öffentlichen Zusicherungen seines Kommunikationspartners vertraut. Dem Autor dieser Diplomarbeit ist jedoch noch keine ACL bekannt, die auf dieser Theorie basiert.

### 2.1.5 Ontologien

Eine Ontologie ist eine explizite Repräsentation der Bedeutung von Begriffen in einem Vokabular (Geroimenko 2004). Ontologien definieren explizit Konzepte und die Beziehungen zwischen Konzepten. Damit stellen sie sicher, daß Begriffe mit einer einheitlichen Bedeutung versehen werden.

<sup>2</sup>Performative können als Nachrichtentyp verstanden werden. Der Performative „ask-one“ in Abbildung 2.2 bezeichnet zum Beispiel den Nachrichtentyp „Frage an einen Einzelnen“. Je nach semantischer Definition der Performative spezifiziert ein Performativ die Struktur des Nachrichteninhaltes.

Maedche (2002) hat einen Schichtenansatz für eine ontologie-basierte Kommunikation vorgestellt, die in Abbildung 2.3 gezeigt wird. Die oberste Schicht beschreibt die Kommunikation zwischen Menschen oder auch zwischen Maschinen. Diese Kommunikation benutzt syntaktische Konstrukte und Symbole (zweite Schicht), die eine bestimmte semantische Bedeutung haben (Schicht 3). Diese Bedeutung bezieht sich auf Objekte in der realen Welt (Schicht 4). Eine Ontologie kann nun über ein explizites formales Modell sicherstellen, daß beide Maschinenagenten im Beispiel über das Tier „Jaguar“, und nicht über das gleichnamige Automobil kommunizieren.

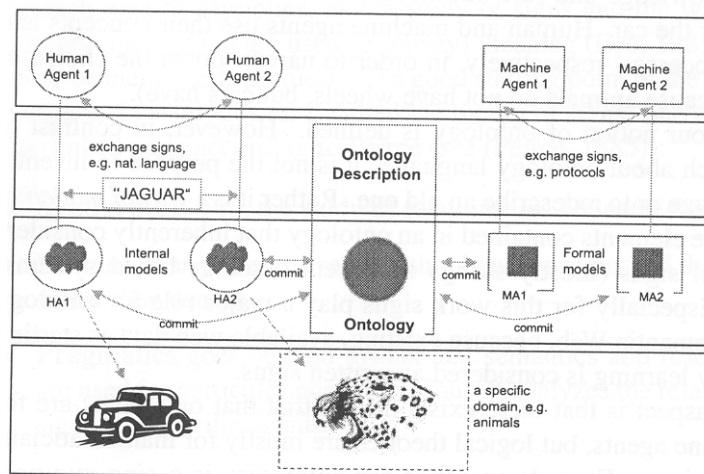


Abbildung 2.3: Ontologien für Kommunikation (Maedche 2002, Abbildung 2.2)

Ontologien im Bereich von Multi-Agenten-Systemen lassen sich laut Nwana u. Wooldridge (1997) folgendermaßen klassifizieren:

- Ad Hoc Ontologien:** Die meisten Agentensysteme haben implizite Ontologien. So basiert die Entwicklung eines Geschäftsprozeß-Werkzeuges auf Fachbegriffen der Geschäftsprozeßmodellierung. Die Agenten-Entwickler nutzen diese Begriffe in der Entwicklung und bauen sie implizit in die Agenten mit ein. Nachteil dieser impliziten Ontologien ist die geringe oder nicht vorhandene Möglichkeit, daß sich mehrere Agentensysteme auf die gleiche Ontologie stützen. Außerdem ist dieser Ansatz nicht skalierbar.
- Standard-Ontologien:** Der Gegensatz zu den oben genannten impliziten „Ad Hoc Ontologien“ sind explizite Ontologien. Im Standardfall sind diese in einer bestimmten Sprache (z.B. KIF (KIF 1992) oder OWL (OWL 2004)) für eine bestimmte Anwendung geschrieben. Da sie explizit sind, können sie zwischen Entwicklern, aber auch zwischen Agenten ausgetauscht, erweitert und geändert werden.
- globale Ontologien:** Eine Erweiterung der „Standard Ontologien“ sind die globalen Ontologien, die versuchen, eine allgemeine, nicht anwendungsspezifische Begriffsdatenbank und -repräsentation anzubieten. Dies hat den Vorteil, daß jeder Agent auf die gleiche Ontologie zurückgreifen und damit alle Agenten miteinander kommunizieren könnten. Die Menge der zu verarbeitenden Konzepte stellt jedoch die Machbarkeit solcher globaler Ontologien in Frage.

Standard- und globale Ontologien müssen, wie auch Interaktionsprotokolle, verteilt und ausgetauscht werden, so daß unterschiedliche Agenten und Systeme auf die gleichen Ontologien zugreifen können. Pan u. a. (2003) entwickelten ein Verteilungssystem für Ontologien für offene Multi-Agenten-Systeme. Das „*ontology-repository*“ (Ontologien-Depot) ist ein Speicherort von Ontologien, der mit Hilfe der offenen Standards HTTP (Hypertext Transfer Protocol) (HTTP 1999), RDF (Resource Description Framework) (RDF 2004) und URN (Uniform Resource Name) (URI 2005) verwaltet wird. Dadurch wird ein Speicher- und Verteilungssystem für offene Multi-Agenten-Systeme bereitgestellt, das auch für Interaktionsprotokolle genutzt werden kann.

Ontologien werden nicht nur bei Multi-Agenten-Systemen verwendet. Sie sind ebenfalls in Gebieten des Semantic Web (vgl. Abschnitt 2.3.6 auf Seite 25) oder der Bioinformatik (Lambrix u. a. 2003) wichtig. Eine ausführlichere Zusammenstellung zum Thema Wissensrepräsentation, -integration und Ontologien wurde von Visser (2004) erarbeitet.

### 2.1.6 Interaktionsprotokolle

Interaktionsprotokolle (IP) sind ein Werkzeug für Agenten-Entwickler, um die Kommunikation zwischen Agenten zu modellieren. Wenn IP öffentlich und von Agenten interpretierbar sind, können sie dabei helfen, dynamisch-offene Agenten-Systeme zu entwerfen (Ehrler u. Cranefield 2004).

Greaves u. a. (2000) beschreiben ein Problem von Software-Agenten, das sie das „Basisproblem“ nennen: „Modern ACLs, especially those based on logic, are frequently powerful enough to encompass several different semantically coherent ways to achieve the same communicative goal, and inversely, also powerful enough to achieve several different communicative goals with the same ACL message.“<sup>3</sup> (Greaves u. a. 2000, Seite 118) Deshalb müssen Software-Agenten bei einer Kommunikation mit hohem Aufwand alle Nachrichtenmöglichkeiten zur Erreichung eines bestimmten Kommunikationszieles validieren und bewerten, um die bestmögliche Nachricht zu wählen. Um den Aufwand zu verringern und die Interaktion zu vereinfachen, schlagen Greaves u. a. (2000) vor, daß die miteinander kommunizierenden Agenten sich auf eine mögliche Abfolge der Nachrichten für ein bestimmtes Kommunikationsziel einigen sollen. Diese Abfolge kann in Interaktionsprotokollen spezifiziert werden.

#### Was sind Interaktionsprotokolle?

Cranefield u. a. (2002, Seite 1) beschreiben, warum Interaktionsprotokolle notwendig sind: „Interaction protocols are descriptions of standard patterns of interaction between two or more agents. They constrain the possible sequences of messages that can be sent amongst a set of agents to form a conversation of a particular type.“<sup>4</sup> Interaktionsprotokolle begrenzen die Anzahl der möglichen Nachrichten zwischen Kommunikationspartnern an einem bestimmten Punkt in der Kommunikation und vereinfachen dadurch die

<sup>3</sup>Deutsche Übersetzung: Moderne ACL's, speziell die logik-basierten, sind normalerweise genügend ausdrucksstark, um über unterschiedliche, semantisch schlüssige Wege das gleiche Kommunikationsziel zu verfolgen; und gleichzeitig verschiedene Kommunikationsziele mit der gleichen Nachricht zu verfolgen.

<sup>4</sup>Deutsche Übersetzung: Interaktionsprotokolle beschreiben Standard-Muster für Interaktionsprotokolle zwischen zwei oder mehr Agenten. Sie begrenzen die möglichen Sequenzen von Nachrichten, die zwischen einer Menge von Agenten gesandt werden können, um eine bestimmte Konversationsart zu erreichen.

Schlußfolgerung eines Agenten, welche Nachricht nun gesandt werden muß.

Zum besseren Verständnis ein Beispiel aus dem menschlichen Leben (nach Purvis u. a. (2002b)): Ein Gast besucht ein Restaurant und setzt sich an den Tisch. Nun muß der Gast nicht über alle möglichen Aussagen, die über Essen getroffen werden können, nachdenken. Er (oder sie) erwartet stattdessen, daß er eine Speisekarte ausgehändigt bekommt und eine Bestellung aufgeben kann. Einige Zeit später wird das Essen gebracht und bevor er geht, bestellt er die Rechnung und es wird von ihm erwartet, daß er diese Rechnung begleicht.

Dies kann als „Restaurant Interaktionsprotokoll“ angesehen werden. Es ist allgemein anerkannt, und es würde jeden sehr verwundern, wenn der Gast, nachdem er die Speisekarte bekommen hat, anfangen würde, seinen eigenen Topf und Spirituskocher auszuwickeln und selbst zu kochen. Dies ist im Interaktionsprotokoll nicht vorgesehen - und die Nicht-Existenz dieser Möglichkeit vereinfacht sowohl für das Restaurant als auch für den Gast die Interaktion. Jeder Kommunikationsteilnehmer (Koch, Kellner, Gast) merkt sich den jeweiligen Stand des Protokoll und weiß daher genau, was als nächstes zu tun ist.

In der Literatur wird häufig der Begriff „Konversationsrichtlinie“ (conversation policy) verwendet. Aber es gibt Forscher, die zwischen Protokollen und Richtlinien unterscheiden. Nowostawski u. a. (2001b) schlagen zum Beispiel eine Schichten-Architektur von Interaktions-Modellen vor. Dabei werden die drei Ebenen

- **Protokoll:** Ein Muster für eine Sequenz von Nachrichten,
- **Konversation:** Eine Instanz eines Protokolls - eine einzelne Nachrichtenabfolge,
- **Richtlinie:** Eine Strategie, Richtlinien und Einschränkungen, die eine Konversation anleiten.

Diese Terminologie ist sinnvoll, da sie eine Unterscheidung zwischen einem Protokoll (eine Vereinbarung zwischen den kommunizierenden Agenten) und einer Richtlinie (die Strategie eines Agenten) erlaubt. Deshalb wird in dieser Arbeit der Begriff „Interaktionsprotokoll“ genutzt, obwohl in einem Teil der zitierten Literatur der Begriff „Konversationsrichtlinie“ verwendet wird.

### Anforderungen an Interaktionsprotokolle

Greaves u. a. (2000) heben vier Anforderungen an Interaktionsprotokolle hervor:

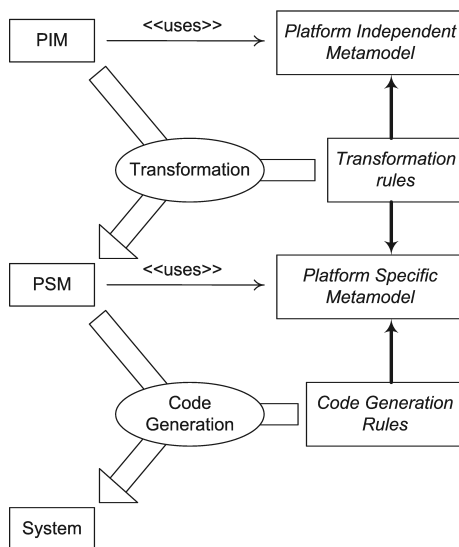
- Die Trennung von Protokoll und Implementation. Das (Kommunikations-)Verhalten eines Agenten kann unabhängig von der Implementierung spezifiziert werden.
  - Mit Hilfe der IP-Spezifikation können Äquivalenzklassen festgestellt werden.
  - Es ist möglich, Interaktionsprotokolle zu komponieren. Verschiedene Teile können dann zusammengesetzt werden, um anwendungsspezifische Konversationen zu regeln.
  - Protokolle sollten flexibel genug sein, um unterschiedlich komplexen Agenten die Interaktion miteinander zu ermöglichen.
-

## 2.2 UML - the Unified Modeling Language

Die „Unified Modeling Language“ (UML) (UML 2.0 2004) ist Teil der „Model Driven Architecture“ (MDA) (MDA 2001) der „Object Management Group“ (OMG).

Deshalb wird am Anfang dieses Abschnitts die MDA vorgestellt. Danach folgt ein Überblick über UML sowie eine Darstellung der Interaktionsprotokolle und Sequenzdiagramme in UML. Die Nutzung von UML in Agentensystemen und eine Vorstellung bisheriger Versuche, eine formale Semantik für UML-Diagramme zu entwickeln, beschließen den Abschnitt.

### 2.2.1 Model Driven Architecture

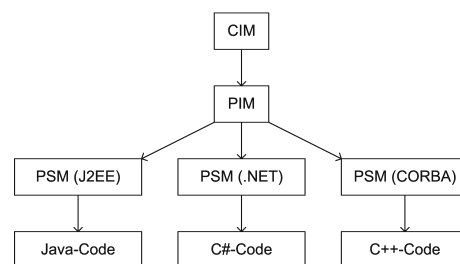


**Abbildung 2.4:** Überblick über MDA nach Born u. a. (2004, Seite 286)

Die Entwicklung in der Informationstechnologie ist rasant. Deshalb ist es schwierig, Systeme zu entwickeln, die für eine lange Zeit effizient genutzt werden können. Die OMG versucht diesem Problem mit dem Konzept der modelgesteuerte Architektur (MDA) (MDA 2001) zu begegnen. Abbildung 2.4 verdeutlicht das Konzept der MDA. Es sieht vor, ein System (oder ein Teilsystem) in einem abstrakten, fachlichen Modell („*Computation Independent Model*“ (CIM)) zu beschreiben. Kempa u. Mann (2005) führen aus: „Es liegt in einer Sprache vor, die für die fachlichen Anwender des Systems verständlich ist, und dient zum Diskurs zwischen Softwarearchitekten und Anwendern über Leistungsumfang und Anforderungen. Das CIM legt fest, was ein Softwaresystem leistet. Es

definiert nicht, wie es dies leistet oder wie das System strukturiert ist.“ (Kempa u. Mann 2005, Seite 299) Der Softwareentwickler spezifiziert dann die Funktionalität des Systems im plattformunabhängigen Modell („*Platform Independent Model*“ (PIM)). Dieses Modell ist unabhängig von jeglichen Implementierungsdetails. Deshalb kann es auch dann weiter genutzt werden, wenn sich die genutzte Technologie ändert.

Die Modellierungstechnik des abstrakten Modells besitzt nun Abbildungsspezifikationen („*Transformation Rules*“) für die einzelnen Implementierungsmöglichkeiten. Kommt eine neue Implementierungstechnik hinzu, so muß auch eine neue Abbildungsspezifikation entwickelt werden. Das PIM des System wird nun mit Hilfe dieser Abbildungsregeln möglichst automatisch in ein implementationsspezifisches Modell



**Abbildung 2.5:** Überblick über MDA nach Kempa u. Mann (2005)

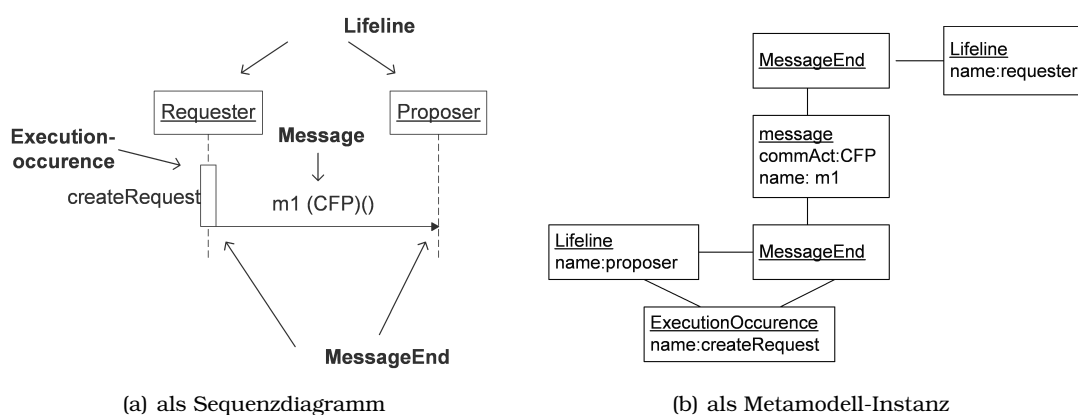
(“Platform Specific Model“ (PSM)) übersetzt.

Ändert sich die unterliegende Technologie, so wird die Abbildungsspezifikation gewechselt, um ein der neuen Technologie angepaßtes Modell zu erhalten. Aus dem PSM kann dann mittels Code-Erzeugungsregeln Quelltext erzeugt werden.

Das MDA-Konzept ist umstritten (Vor- und Nachteile werden bei Kempa u. Mann (2005) diskutiert) und wird bisher auch von keinem Werkzeug vollständig umgesetzt. Dies liegt vor allem an der aufwendigen Erstellung entsprechender Abbildungsregeln sowie an der zweideutigen und ungenauen Definition von Modellierungssprachen. Dies bedeutet, daß momentan auch bei der Anwendung von MDA ein Großteil der Arbeit vom Software-Entwickler erledigt werden muß.

### 2.2.2 Metamodellierung

Die MDA basiert auf dem Konzept der Metamodellierung. Born u. a. (2004) erklären dazu: „Die deutsche Sprache ist mit der deutschen Sprache selbst erklärt. Der Duden regelt, welche Wörter es gibt, was sie bedeuten und wie sie zu Sätzen zusammengestellt werden. [...] Dieses Vorgehen [...] wurde auf die Definition von Modellierungssprachen übertragen: Die Sprache UML 2 ist mit der Sprache UML 2 erklärt. [...] Das Resultat dieser Sprachdefinition selbst ist natürlich wieder ein Modell.“ (Born u. a. 2004, Seite 61)



**Abbildung 2.6:** Metamodellierungsbeispiel anhand eines Sequenzdiagramm-Fragmentes

Ein Metamodell beschreibt formal die Modellelemente sowie die Syntax und Semantik ihrer Notation und Verwendung (Muller 1997). Abbildung 2.6(a) verdeutlicht dies anhand eines Sequenzdiagrammfragmentes. Das Sequenzdiagramm besteht aus verschiedenen Elementen, so zum Beispiel „Lifelines“<sup>5</sup>, Nachrichten und Nachrichtenden. Diese Elemente, und ihre möglichen und nötigen Beziehungen untereinander, werden von ihrem Metamodell beschrieben. Ein Nachrichtenende muß zum Beispiel immer eine Verbindung zu einer Nachricht und zu einer Lifeline haben.

Abbildung 2.6(b) zeigt dasselbe Sequenzdiagrammfragment wie Abbildung 2.6(a), nur diesmal als Instanz des Metamodells. Mit Hilfe dieser Instanz ist es nun möglich, das

<sup>5</sup>Die Namen der einzelnen Elemente eines Diagrammes werden in dieser Arbeit nur dann in deutsche Begriffe übersetzt, wenn es gute deutsche Äquivalente gibt. So wird message in Nachricht übersetzt, für Lifeline gibt es hingegen keine passende Übersetzung.

Sequenzdiagramm aus Abbildung 2.6(a) zu analysieren und zu verarbeiten.

### 2.2.3 Unified Modeling Language (UML)

Die UML (UML 1.3 2000) wurde ursprünglich als Vereinigung der drei Sprachen Booch (Booch 1993), Object Modeling Technique (OMT) (Rumbaugh u. a. 1990) und Object Oriented Software Engineering (OOSE) (Jacobson u. a. 1992) entwickelt. Laut den Hauptentwicklern, James Rumbaugh, Ivar Jacobson und Grady Booch, ist UML eine visuelle Modellierungssprache für breite Anwendungsgebiete, die genutzt werden kann, um Artefakte eines Software Systems zu spezifizieren, visualisieren, konstruieren und dokumentieren (Rumbaugh u. a. 2004). UML wurde schnell der inoffizielle Standard für objekt-orientiertes Design sowie objekt-orientierte Analyse und daher 1997 durch die OMG offiziell standardisiert. Im September 2000 startete die OMG den Entwicklungsprozeß für UML 2.0, der 2004 beendet wurde.

Laut Rumbaugh u. a. (2004) modelliert UML folgende Merkmale eines Systems:

- **statische Strukturen:** Die wichtigsten Konzepte einer Anwendung, deren interne Eigenschaften sowie die Beziehungen zwischen diesen Konzepten.
- **dynamisches Verhalten:** Die Interaktion eines Objektes mit seiner Umgebung, die Kommunikation zwischen Objekten, um ein Verhalten zu implementieren, die Zustände eines Objektes.
- **Implementierungskonstrukte:** Modelle für die Implementierung.

Außerdem bietet es folgende Werkzeuge für die Modellerstellung an (Rumbaugh u. a. 2004):

- **Modellorganisation:** Einteilung eines großen Modells in Teilmodelle, die von Menschen besser zu lesen und zu verstehen sind.
- **Erweiterungsmechanismen:** Eine (begrenzte) Erweiterungsfähigkeit, um den ändernden Anforderungen gerecht zu werden.

### 2.2.4 Object Constraint Language OCL

Die Object Constraint Language OCL (OCL 2003) wurde laut Warmer u. Kleppe (1999) 1995 von IBM für die Modellierung von Geschäftsprozeßmodellierung entwickelt. Als UML sich als Modellierungssprache durchsetzte, wurde erkannt, daß zur Beschreibung von Bedingungen eine textuelle Sprache nötig ist. Bis dahin wurden Bedingungen in UML nur natürlichsprachig beschrieben. Dies führte einerseits zu sehr großen Dokumenten und andererseits zu mehrdeutigen Beschreibungen. Die Übernahme von OCL als Zusatz sollte dieses Manko beheben.

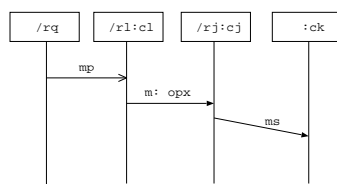
OCL ist eine deklarative Sprache, mit der Abfragen, Bedingungen, Referenzen von Werten oder das Bestimmen von Geschäftsregeln spezifiziert werden können. Sie ist typisiert und nutzt dabei als Typen die in UML spezifizierten Konzepte. Sie besitzt mit einem Metamodell (Richters u. Gogolla 1999) eine formale Syntax-Beschreibung, und außerdem verschiedene formale Semantiken, zum Beispiel die von Richters (2002) oder

---

Baar (2003). Beide Semantiken basieren auf mengentheoretischen Überlegungen, bei Baar (2003) über den Umweg der Metasprache CINV. Flake (2004) hat verschiedene Lücken in der Definition der OCL-Semantik gefunden und versucht, diese zu schließen. Kiesner u. a. (2002) haben eine visuelle Notation für OCL erstellt, die den Nachteil der rein textuellen Notation von OCL beheben soll. Diese Notation macht jedoch Diagramme, die in UML und OCL erstellt wurden, noch komplexer und unübersichtlicher.

Mit der UML 2.0-Version wurde von der OMG ebenfalls eine grundlegende Überarbeitung von OCL vorgenommen. Warmer u. Kleppe (2004) bieten einen umfassenden Überblick in diese neue Version.

### 2.2.5 Interaktionsprotokolle und Sequenzdiagramme in UML



**Abbildung 2.7:** Beispiel-Sequenzdiagramm  
(Bernardi u. a. 2002)

Sequenzdiagramme sind in UML 2.0 (neben Aktivitätsdiagrammen, Zustandsautomaten und Anwendungsfalldiagrammen) eine Möglichkeit der Beschreibung dynamischer Aspekte eines Systems. „Eine Verhaltensbeschreibung in UML spezifiziert vollständig oder partiell das gewünschte oder auch das unerwünschte Verhalten eines Classifiers“ (Born u. a. 2004). Ein Verhaltensdiagramm bezieht sich also immer auf Elemente, die in einer statischen Strukturbeschreibung definiert wurden - zum Beispiel durch ein Klassendiagramm. Bei Aktivitäts- und Zustandsdiagrammen liegt dabei der Schwerpunkt auf die innerhalb eines Objektes auftretenden Verhaltensweisen. Interaktionsdiagramme hingegen verdeutlichen das Verhalten von Objekten untereinander.

Abbildung 2.7 zeigt ein einfaches Sequenzdiagramm. Sie beschreibt, wie eine Nachricht in einer Objekthierarchie „nach unten“ gegeben wird. Ein Sequenzdiagramm wird von oben nach unten gelesen - das heißt, die oberste Nachricht ist die erste, die gesandt wird, dann folgt die zweite von oben usw. Eine Nachricht wird durch einen Pfeil dargestellt. Dabei repräsentieren in UML offene Pfeile (wie in Abbildung 2.7 die erste Nachricht) asynchrone Nachrichten, während geschlossene Pfeile (wie die weiteren Nachrichten) synchrone Nachrichten darstellen. Die Richtung der Pfeile - also ob sie waagrecht oder schräg gezeichnet wurden, hat für die Semantik keinerlei Bedeutung. Die Reihenfolge wird nur durch die Anordnung der Verbindungen an den Lifelines bestimmt. Da also Nachricht „mp“ am Objekt „rl“ über der Nachricht „m:opx“ angeordnet ist, muß sie erst empfangen werden, bevor die „m:opx“ gesandt werden kann.

Abbildung 2.7 zeigt ein einfaches Sequenzdiagramm. Sie beschreibt, wie eine Nachricht in einer Objekthierarchie „nach unten“ gegeben wird. Ein Sequenzdiagramm wird von oben nach unten gelesen - das heißt, die oberste Nachricht ist die erste, die gesandt wird, dann folgt die zweite von oben usw. Eine Nachricht wird durch einen Pfeil dargestellt. Dabei repräsentieren in UML offene Pfeile (wie in Abbildung 2.7 die erste Nachricht) asynchrone Nachrichten, während geschlossene Pfeile (wie die weiteren Nachrichten) synchrone Nachrichten darstellen. Die Richtung der Pfeile - also ob sie waagrecht oder schräg gezeichnet wurden, hat für die Semantik keinerlei Bedeutung. Die Reihenfolge wird nur durch die Anordnung der Verbindungen an den Lifelines bestimmt. Da also Nachricht „mp“ am Objekt „rl“ über der Nachricht „m:opx“ angeordnet ist, muß sie erst empfangen werden, bevor die „m:opx“ gesandt werden kann.

## 2.3 Sprachwissenschaften / Formale Methoden

„**Sprache** die, das den Menschen eigentümliche Mittel, sich auszudrücken und miteinander zu verständigen; die S. befähigt den Menschen, auf seine Umwelt nicht nur zu reagieren, sondern sich mit ihr bewußt auseinanderzusetzen und Gedanken unabhängig von Zeit und Ort mitzuteilen und zu überliefern“ (Liebold u. Werner 1991, Seite 663).

Die Sprache als wichtige Basis der menschlichen Kommunikationsfähigkeit wurde im Laufe der Zeit gründlich untersucht. Die sich daraus entwickelnde Wissenschaft, die „Linguistik“, wurde von Ferdinand de Saussure (1857-1913) begründet. Die formale Sprachtheorie, um die es in diesem Abschnitt gehen soll, ist ein Gebiet der Linguistik, das auf Grundlagen der Mathematik, Philosophie und Informatik basiert.

Im ersten Abschnitt wird ein Überblick über die Linguistik gegeben, um darauf aufbauend eine Annäherung an den Begriff „Semantik“ zu versuchen. Abschnitt 2.3.3 führt in den Bereich der formalen Methoden ein, wonach dann in 2.3.4 eine Klassifizierung der semantischen Beschreibungsformalismen erarbeitet wird. Verschiedene Eigenschaften und Anwendungsgebiete einer formalen Semantik (Abschnitt 2.3.5) sowie Beispielformalismen (Abschnitt 2.3.6) schließen den Abschnitt ab.

### 2.3.1 Überblick über die Linguistik

Ferdinand de Saussure, der Begründer der modernen Linguistik, unterscheidet drei Bedeutungen des Begriffes Sprache (Adamzik 2001):

- **"langage"**: ist die spezifische menschliche Fähigkeit zur Spracherlernung und -entwicklung
- **"langues"**: sind die verschiedenen Sprachen, wie zum Beispiel Deutsch, Englisch oder Sorbisch.
- **"parole"**: bezeichnet den konkreten Gebrauch einer Sprache - das Bedürfnis und die Notwendigkeit nach Kommunikation, Gesprächen und Texten.

Die Sprachwissenschaft ist die Wissenschaft von der Sprache. Dabei werden alle drei oben genannten Bedeutungen untersucht. Nach Vater (1996) unterteilt sich dabei die Sprachwissenschaft in die Bereiche:

- **Phonologie**: Die Phonologie untersucht die Lautstruktur von Sprache. Fragestellungen sind unter anderem: Welche Lauteinheiten kommen universell, welche in einer bestimmten Sprache vor? Nach welchen Regularitäten werden sie verknüpft. Die Phonologie ist natürlich nur für Sprachen interessant, die auch gesprochen werden können. Daher ist sie für Computersprachen wenig interessant. Genutzt wird sie in der Informatik in den Bereichen Spracherkennung und Sprachsynthese.
  - **Morphologie**: Die Morphologie untersucht die Wortstruktur. Wie werden morphologische Einheiten zu Wörtern zusammengesetzt. Dies ist in der Informatik ebenfalls hauptsächlich im Bereich der Spracherkennung und -synthese interessant.
  - **Syntax**: Die Syntax analysiert die Regularitäten einer Sprache, so zum Beispiel mit der Frage, wie Wörter zu größeren Einheiten zusammengesetzt werden.
  - **Semantik**: Die Semantik beschäftigt sich mit der Bedeutung von Wörtern und daraus zusammengesetzten Einheiten.
  - **Pragmatik**: Den Einfluß sozialer und situativer Faktoren auf die Bedeutung der Sprache untersucht die Pragmatik.
-

Für die Definition von Programmiersprachen sind vor allem die Syntax, die Semantik und die Pragmatik von Interesse. Olderog u. Steffen (1999) führen aus: „Zur Definition einer Programmiersprache gehören drei Aspekte: Syntax, Semantik und Pragmatik. Die Syntax klärt die Frage, was ein wohlgeformtes Programm ist. [...] Die Semantik klärt die Frage, was ein wohlgeformtes Programm bedeutet. Die Pragmatik klärt die Frage, für welchen Anwendungsbereich die Programmiersprache am Besten geeignet ist.“

Im weiteren werden sich die Ausführungen zum großen Teil auf die Semantik beschränken, da diese für die vorliegende Arbeit von ausschlaggebender Bedeutung ist.

### 2.3.2 Was ist Semantik?

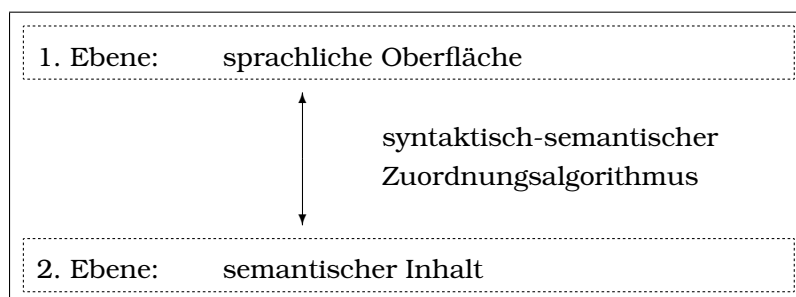
Was bedeutet das Wort „Semantik“? Im Internet-Lexikon kann man dazu die folgende Definition finden: „Die Semantik (Bedeutungslehre) ist das Teilgebiet der Sprachwissenschaft (Linguistik), das sich mit Sinn und Bedeutung von Sprache beziehungsweise sprachlichen Zeichen befasst.“ (Wikipedia 2001, Stichwort „Semantik“). Daraus leitet sich die Frage ab, was denn die Bedeutung eines Begriffes ausmacht.

De Mauro (1982) versucht dieser Frage auf den Grund zu gehen. Dabei erkennt er verschiedene Möglichkeiten einer Antwort. Eine einfache Antwort wäre nach de Mauro (1982) zum Beispiel, „daß die Bedeutung das ist, was durch ein Zeichen mitgeteilt wird“. Er verweist aber in seiner Argumentation darauf, daß diese Definition nur Sinn macht, wenn die Bedeutung des Zeichens klar ist - also die Semantik des Zeichens. Es folgt also aus der Frage „Was ist die Bedeutung?“ die Frage „Was ist das Zeichen?“. Ein Zeichen ist etwas, was mit irgendetwas verbunden ist, was es repräsentiert. Anders ausgedrückt: „[...] Zeichen ist das, was eine Bedeutung hat.“ (de Mauro 1982). Damit ist die Bestimmung des Zeichens nur möglich, wenn der Begriff „Bedeutung“ geklärt ist.

So wird deutlich, daß die Frage „Was ist Bedeutung?“ nicht wirklich geklärt werden kann. De Mauro (1982) schließt daraus, daß Sprache „an sich“ keinerlei Bedeutung hat. „Allein die Menschen schaffen Bedeutung, indem sie die Sätze und Wörter als Zeichen gebrauchen“. Dadurch wird die Semantik eine „Theorie des Zeichengebrauchs in seinen historisch bestimmten Formen“.

Dies paßt zu der bei Hausser (2000) angegebener Semantikdefinition. Danach bezieht sich die Semantik eines Begriffes immer auf etwas anderes. Es gibt einen Zuordnungsalgorithmus von sprachlicher Oberfläche und semantischem Inhalt (siehe Abbildung 2.8).

Hausser (2000) unterscheidet dabei drei verschiedene Systemtypen von Sprache:



**Abbildung 2.8:** Zwei-Ebenenstruktur der Semantik (Hausser 2000, Abbildung 19.1.1)

1. Logiksprachen: Sie sollen ausdrücken, ob eine bestimmte Aussage in einem bestimmten Modell der Wahrheit entspricht.
2. Programmiersprachen: Die Semantik von Aussagen in Programmiersprachen beruht auf der Ausführung dieser Aussagen (hier: Befehle) auf einer abstrakten oder realen Maschine.
3. Natürliche Sprachen: Die Semantik von natürlichen Sprachen wird durch die Einigung der Sprachgemeinschaft auf die Bedeutung eines bestimmten Wortes festgelegt (Konvention).

### 2.3.3 Formale Methoden

Formale Methoden sind mathematisch basierte Sprachen, Techniken und Werkzeuge zur Spezifikation und Verifikation von Systemen. (Clarke u. Wing 1996) Sie helfen, Konzepten eine eindeutige und unmißverständliche Bedeutung zu geben (Meyer u. Schobens 1999). Dies geschieht, indem eine formale Spezifikation einem System eine Sammlung von Eigenschaften zuweist. Diese Eigenschaften werden in einer formalen Sprache auf einem (zumindest teilweise) abstraktem Niveau beschrieben (van Lamsweerde 2000). Das heißt, wenn das Modell aus Abbildung 2.8 zugrunde gelegt wird, dann ist das zu spezifizierende System die sprachliche Oberfläche; die formale Sprache gibt den semantischen Inhalt vor, und die Spezifikation entspricht dem Zuordnungsalgorithmus. Die formale Spezifikation von Sprachen wird nicht nur in der Informatik verwendet. So werden zum Beispiel von Link (1998) Ansätze gezeigt, für natürliche Sprache eine mathematische Semantik zu finden.

Die Forschung an formaler Spezifikation begann etwa gleichzeitig mit der Erfindung des Computers. Van Lamsweerde (2000) hat eine Übersicht über die Entwicklung der formalen Spezifikation zusammengestellt. Mit der Entwicklung programmierbarer Computer wurde die Entwicklung von Programmiersprachen notwendig. Für diese mußten Regeln zur Bildung von Ausdrücken erstellt werden. Für den Bereich der Syntax von Programmiersprachen wurden zwischen 1950 und 1970 weitreichende Forschungsarbeiten geleistet. Als theoretisches Fundament gelten dabei insbesondere die Arbeiten von Chomsky (1956, 1959) und Backus (1959), aber auch von Knuth (1965), Greibach (1965), Korenjak (1969) oder de Remer (1969). So wurden damals Spezifikationsformalismen für die Syntax von Programmiersprachen entwickelt, die ausdrucksstark genug sind, um diese genau und präzise zu definieren. Als Beispiel sei die auch heute genutzte Backus-Naur-Form (Naur u. a. 1960, 1963) genannt.

Demgegenüber gibt es laut Zhang u. Xu (2004) immer noch keine allgemein akzeptierte formale Notation für eine semantische Beschreibung einer Programmiersprache. Dies liegt nach Wansbrough (1997, zitiert in Zhang u. Xu (2004)) vor allem daran, daß das Verhalten eines Programmes eine viel höhere Komplexität aufweist als seine Struktur.

### 2.3.4 Geschichte und Klassifizierung formaler Semantiken

Die Entwicklung formaler Semantiken kann in drei Abschnitte eingeteilt werden. Dabei wurden verschiedene Ansätze entwickelt - der *denotationelle*, *axiomatische*, *operationelle*

---

und der *hybride* Ansatz. Zur Vertiefung des Gebietes sei auf die Bücher von Best (1995), Winskel (1994), Berghammer (2001) oder Watt (1991) verwiesen.

### 1960-1980

Zhang u. Xu (2004) erarbeitete eine detaillierte Übersicht über die Geschichte von Semantik-Formalismen. Danach entstanden zwischen 1960 und 1980 zwei der drei Klassen semantischer Beschreibung: der *denotationelle* und der *axiomatische* Ansatz.

Der *denotationelle* Ansatz ordnet einem Programm als Semantik ein mathematisches Objekt zu. Damit kann genau berechnet werden, was ein Programm der definierten Sprache bewirkt (Berghammer 2001). Die klassische Variante denotationeller Semantik ist die Semantik nach Scott und Strachey (Tennent 1976). Sie definiert Datentypen durch Gleichungen und darauf aufbauende semantische „Funktionen“ - die Denotationen. Die Elemente der Datentypen werden durch die typisierte  $\lambda$ -Notation beschrieben. Weitere bekannte denotationellen Beschreibungen sind die „Vienna Development Method“ (Bjørner u. Jones 1978) oder die „predicate transformation semantics“ (Dijkstra 1975).

Im *axiomatischen* Ansatz wird die Bedeutung eines Programmes durch Regeln definiert, die es erlauben, von Eigenschaften der Programm-Eingabe auf Eigenschaften der Programm-Ausgabe zu schließen (oder umgekehrt). Dies wird insbesondere genutzt, um zu testen, ob ein Programm das tut, was der Programmierer beabsichtigt hat (die Verifikation des Programmes) (Berghammer 2001). Die wichtigste axiomatische Semantikdefinition ist die „Hoare-Logik“ (Hoare 1969). Sie basiert auf Prädikatenlogik und gibt Regeln für die Variablenbelegung vor und nach Ausführung eines Sprachkonstruktes an.

### 1980-1990

In diesen Jahren wurde der zeitliche Einfluß der Programmausführung in die Beschreibungs-Formalismen mit eingearbeitet. Dadurch entstand der *operationelle* Ansatz. Hier wird eine abstrakte Maschine genutzt und das Programm als Operationen auf dieser abstrakten Maschine definiert. Dies kann man sich auch als Ausführungsprotokoll von Zustandsübergängen vorstellen. Die Anwendung dieser Semantik besteht vor allem in der Sicherstellung einer korrekten Implementation einer Sprache. Beispiele dieses Ansatzes sind „Structural Operational Semantics“ (SOS) (Plotkin 1983) oder „Abstract State Machines“ (ASM) (Gurevich 1991)

### seit 1990

In der weiteren Forschung wurden verschiedene Formalismen entwickelt, die die oben genannten grundsätzlichen Ansätze weiter verfeinerten und verbesserten. Dazu gehört zum Beispiel die Modularisierung von semantischen Beschreibungen, die Wiederverwendbarkeit oder die Erweiterbarkeit. Beispiele dafür sind die „modular monadic action semantics“ (MMAS) (Wansbrough 1997, zitiert von Zhang u. Xu (2004)) oder die „game semantics“ (Blass 1992, zitiert von Zhang u. Xu (2004)).

---

In den einzelnen Formalismen können verschiedene Ansätze noch kombiniert sein - ein sogenannter *hybrider* Ansatz (Mosses 2001, zitiert von Zhang u. Xu (2004)).

### 2.3.5 Eigenschaften und Anwendungsgebiete einer semantischen Definition

Laut Zhang u. Xu (2004) sollte eine Programmiersprachen-Spezifikation folgende Eigenschaften aufweisen:

- "**Lesbarkeit**": Die Spezifikation muß für Sprachentwickler, Sprachimplementierer und Programmierer gleichermaßen lesbar sein.
- "**Modularität**": Durch einen modularen Aufbau großer Semantikbeschreibungen aus kleineren Komponenten kann die Wiederverwendbarkeit und Modifizierbarkeit erhöht werden.
- "**Abstraktheit**": Die Konzentration auf wichtige Sprachgestaltungsprobleme anstelle von Implementationsdetails wird durch eine abstrakte Beschreibung erleichtert.
- "**Vergleichbarkeit**": Eine Semantik sollte den Vergleich von Programmiersprachen erlauben.
- "**Auswertbarkeit**": Die Möglichkeit, über in der Sprache geschriebene Programme zu schlußfolgern, sollte gegeben sein.
- "**Anwendbarkeit**": Ein Spezifikationsformalismus sollte auf möglichst viele Sprachkonzepte wie Zustände, Ein- und Ausgabe, oder Ausnahmefehler angewandt werden können.
- "**Werkzeugunterstützung**": Die Unterstützung durch Werkzeuge zur Erstellung, Überprüfung und Implementierung von semantischen Beschreibungen ist nötig für eine weite Verbreitung des Formalismusses.

Marcotty u. a. (1976) sieht als wichtige Anforderungen an eine Semantikdefinition:

- "**Vollständigkeit**": - eine lückenlose Definition.
- "**Übersichtlichkeit**": - der Nutzer erhält die Antworten auf seine Fragen auf einfachem Wege.
- "**Natürlichkeit**": - die Verständlichkeit einer Definition wird durch eine größere Natürlichkeit erhöht.
- "**Realität**": - die Grenzen der Sprache müssen klar und eindeutig aufgezeigt sein.

Viele Sprachen besitzen keine formale Beschreibung der Semantik. Die Bedeutung der Sprache wird dann meist nur in natürlicher Sprache definiert. Eine formale Beschreibung kann jedoch laut Zhang u. Xu (2004) in folgenden Anwendungsgebieten hilfreich sein:

---

- "**design**": Die Semantik ist das Ergebnis des Sprachentwicklungsprozesses und sollte den Entwickler während des Prozesses auf Auslassungen oder Unregelmäßigkeiten hinweisen.
- "**implementation**": Während der Implementation der Sprache sollte die Semantikspezifikation die Absicht der Entwickler widerspiegeln und dadurch die korrekte Implementation sicherstellen.
- "**standardization**": Durch eine eindeutige Semantik wird es erleichtert, zwischen verschiedenen Implementationen zu wechseln.
- "**understanding**": Ein Softwareentwickler, der die Sprache nutzt, sollte durch die Semantik die Sprache erlernen können. Das geschieht, indem die Semantik die Beziehung zwischen der Sprache und bekannten Konzepten herstellt.
- "**reasoning**": Eine Semantik sollte es erlauben, Programme zu verifizieren, ob sie das tun, was sie sollen.
- "**insight**": Theoretiker können neue Erkenntnisse über Programmierkonzepte erlangen.
- "**generate**": Semantische Spezifikationen werden genutzt, um Compiler und Interpreter zu entwickeln.

### 2.3.6 Formalismen zur Semantikdefinition

Es gibt mittlerweile viele Formalismen zur Definition von Semantiken. Als Zusammenfassungen seien hier nur die Arbeiten von Clarke u. Wing (1996); van Lamsweerde (2000) und Zhang u. Xu (2004) genannt. Aber, wie Rushby (1996) ausführt, sind die meisten der Techniken und Werkzeuge sehr eng auf ihre Anwendung zugeschnitten - dies bedeutet, daß für jedes Problem ein neues Werkzeug oder eine neue Methode bereitgestellt werden muß.

Im Folgenden werden ausgewählte Formalismen vorgestellt:

**Prädikatenlogik** „Die Prädikatenlogik oder Quantorenlogik ist ein Teilgebiet der Logik. Sie stellt eine Erweiterung der Aussagenlogik dar, in der die 'einfachen Aussagen' der Aussagenlogik auf ihre Struktur untersucht und komplexere Aussagen daraus gebildet werden. Zusätzlich zur Verknüpfung von Aussagen (beispielsweise durch „und“ oder „oder“) werden auch die Eigenschaften von Objekten und ihre Geltungsbereiche betrachtet“ (Wikipedia 2001, Stichwort „Prädikatenlogik“).

Prädikatenlogik ist eine grundlegende Möglichkeit, um Semantiken zu definieren. Sie wurden unter anderem zur Definition von Agentenkommunikationssprachen benutzt. Labrou u. Finin (1997) erarbeiteten zum Beispiel ein Rahmenwerk zur Semantikbeschreibung von KQLM (vgl. Abschnitt 2.1.4). Dabei wurde die Sprechakte durch formale Angabe der Vorbedingungen (Was kann über den Zustand des Senders angenommen werden, wenn er die Nachricht schickt? Welchen Zustand muß der Empfänger haben, um es annehmen und bearbeiten zu können?) und der Nachbedingungen (Welche

---

Annahmen können über den Zustand des Senders nach dem Senden der Nachricht getroffen werden können? Welche über den Zustand des Empfängers nach Empfang und Bearbeitung der Nachricht?) spezifiziert. Aufgrund der Definition der Semantik über Vor- und Nachbedingungen kann diese Semantikdefinition als axiomatische Definition angesehen werden.

Ähnlich wurde laut Wooldridge (2002, Seite 175ff) auch die Semantik von FIPA-ACL definiert FIPA-ACL (2002).

**Beschreibungslogik** Beschreibungslogik ist eine formale Sprache für die Wissensrepräsentation und Wissensverarbeitung. Sie wird in vielen wissensbasierten Systemen genutzt, aber sie findet auch in Softwaretechnik, medizinischer Informatik und Web-basierten System Anwendungen. Eine Einführung in die Benutzung von Beschreibungslogiken kann bei Donini u. a. (1996) oder bei Baader u. a. (2003) gefunden werden. Beschreibungslogiken wurden auch schon auf Ontologien (siehe Abschnitt 2.1.5 ) angewendet (Horrocks u. Sattler 2001).

**Abstract State Machines** Zhang u. Xu (2004) haben in ihrer Untersuchung Abstract State Machines (ASM) als Kandidaten für ein „ideales Rahmenwerk“ zur Semantikdefinition genannt. ASM zählen zu den operationalen Semantik-Formalismen.

ASM definieren Zustandsänderungen über den Abstrakten Syntaxbaum der zu definierenden Sprache. Deshalb werden sie auch „Entwickelnde Algebren“ genannt. Zustände in ASM enthalten Kontrollflußgraphen, die das gesamte Programm enthalten. Laut Zhang u. Xu (2004) wurden ASM als Formalismen für die Spezifikationen für Prolog, VHDL und SDL genutzt.

**Modular Monadic Action Semantics (MMAS)** Modular Monadic Action Semantics (MMAS) wurden, neben Abstract State Machines (ASM), von Zhang u. Xu (2004) als guter Kandidat für einen idealen Semantikformalismus benannt. Sie sind eine hybride Semantik, da sie verschiedene Ansätze vereinen. Vereinfachend beschrieben sind MMAS eine operationelle Semantik, deren Aktionen eine modulare, denotationelle Definition erhalten haben. Das gibt ihnen die Lesbarkeit, die operationelle Ansätze haben, und zusätzlich wurde die eine denotationelle Modularität und Erweiterbarkeit hinzugefügt.

**Transitionssysteme** Hindriks u. a. (1997) benutzt Transitionssysteme, um eine abstrakte Agentenprogrammiersprache semantisch zu definieren. Ein Transitionssystem ist ein deduktives System, das es erlaubt, die Übergänge (Transitionen) eines Programmes abzuleiten (operationelle Semantik). Es besteht aus einer Menge von Transitionregeln, die die Bedeutung eines jeden Sprachelements definiert. Außerdem gibt es in Transitionssystemen Konfigurationen, die einen Zustand beschreiben. So zum Beispiel die inneren Zustände eines Agenten, oder in der imperativen Programmierung ein Paar  $\langle \pi, \sigma \rangle$  ( $\pi$  repräsentiert das Programm,  $\sigma$  eine Variablenbelegung). Eine Transition überführt nun eine Konfiguration in eine andere durch das Ausführen eines eines Programmelements. Eine ausführliche Beschreibung von Transitionssysteme hat Plotkin (1981) erstellt.

---

Hindriks u. a. (1997) arbeiten mit sogenannten BDI-Agenten<sup>6</sup> und verstehen die Überzeugungen, Absichten und Wünsche der Agenten als Konfiguration. Die Elemente der Agentensprachen ändern nun diese Konfiguration in einer bestimmten Art und Weise, definiert in Transitionssystemen.

**Web Ontology Language (OWL)** die „Web Ontology Language“ (OWL) (OWL 2004) ist ein Standard zur Spezifikation, zum Austausch und zur Verarbeitung von Ontologien (siehe Abschnitt 2.1.5). Er entstand aufgrund der Weiterentwicklung des World Wide Web (WWW) ins „Semantic Web“. Diese Vision, erstmals formuliert von Berners-Lee u. a. (2001), versteht Webseiten nicht nur als syntaktisch verbunden, sondern enthält semantische Verknüpfungen, durch die der Inhalt von Webseiten computerverarbeitbar werden soll.

Aufgrund dieses Hintergrundes ist OWL vor allem zur Beschreibung von statischen Strukturen und Konzepten geeignet. Mehr Informationen zu OWL kann bei Antoniou u. van Harmelen (2004) gefunden werden.

**UML-Semantikbeschreibung mit Hilfe von OCL** „Die in Abschnitt 2.2.4 beschriebene Sprache OCL kann auch zur Definition einer Semantik von UML benutzt werden. Dies liegt daran, daß das UML-Metamodell (siehe Abschnitt 2.2.2) in der verwandten Sprache Meta Object Facility (MOF) spezifiziert wurde, auf die ebenfalls OCL angewandt werden kann. So schreiben Warmer u. Kleppe (2004): „Tatsächlich wurde der UML-Standard selbst unter Verwendung der OCL definiert. [...] Im UML-Standard werden diese Definitionen *Wohlgeformtheitsregeln* genannt.“

Natürlich kann eine Sprache, die mit Hilfe von OCL definiert wurde, nur den Formalitätsgrad erhalten, den OCL selbst auch hat.

Eine Erweiterung von OCL ist die Executable Object Constraint Language (xOCL). (Clark u. a. 2004). Die Absicht dahinter ist, ausführbare Modelle und Metamodelle erstellen zu können. Dafür wurden OCL einige Programmierkonstrukte wie `while`-Schleifen, Datenstrukturen und Zuweisungsoperatoren. Wichtig ist ebenfalls, daß neue Objekte mittels xOCL erstellt werden können.

**Graph-Transformationen** Engels u. Heckel (2000) schlagen vor, die Erfahrungen aus der Definition von Programmiersprachen mittels Graphtransformationen auf die Definition von visuellen Diagrammsprachen zu übertragen. Dies wurde mittlerweile von verschiedenen Forschern versucht:

Hausmann u. a. (2004) geben UML-Multimediadiagrammen eine formale, operationelle Semantik mit Hilfe von Graphtransformationen. Sie nennen ihren Ansatz *Dynamic Meta Modelling (DMM)*, beziehungsweise durch die Modellierungsmöglichkeit von zeitkritischen Systemen *Dynamic Meta Modelling with time(DMM+t)*. Die Graphtransformation ist algebraisch über eine mengentheoretische Definition beschrieben.

Ein für die Agenten-Technik herausstechendes Merkmal ist die Möglichkeit der Zeitmodellierung durch „lokale Uhren“ - das heißt, jedes Element mißt seine eigenen Zeitab-

---

<sup>6</sup>BDI steht für **B**eliefes - **D**esires - **I**ntentions; Ein BDI-Agent ist ein Agent, der durch seine Überzeugungen, Wünsche und Absichten charakterisiert wird.

stände und es gibt keine globale Synchronisation. Dieses Merkmal könnte bei Software-Agenten von Vorteil sein, da diese ebenfalls jeweils ihre eigene Zeitmessung haben.

De Lara (2005) verallgemeinert diesen Ansatz. Er verwendet die Metamodellierung und die Graphtransformation allgemein zur Definition von visuellen Sprachen. Er zeigt dies anhand der Definition von *distributed event graphs (DEGs)*. Die Syntax der Sprache wird durch eine Metamodellierung erreicht, die Semantik durch Nutzung von Graphtransformationen. Das Tool AToM (de Lara u. Vangheluwe 2002) wird zur Implementation genutzt.

Auch Varró (2004) hat Graphtransformationen auf visuelle Modellierungssprachen angewandt. Dabei wendet er sie zusammen mit ASMs (siehe Abschnitt 2.3.6) auf der Modell- und auf der Metamodellebene an.

**Communicating Sequential Processes (CSP)** Communicating Sequential Processes (CSP) wurden von Hoare (1985) entwickelt und sollten angewandt werden zur „Spezifikation, Entwicklung und Implementation von Computersystemen, die fortwährend laufen und mit ihrer Umgebung agieren.“ Davies u. Crichton (2003) wenden CSP an, um eine formale Semantik für UML zu entwerfen.

## 2.4 Petri-Netze

Petri-Netze (Petri 1962) sind ein in der Informatik weitverbreitetes formales Hilfsmittel zur Spezifikation von Systemen. Es hat viele positive Eigenschaften, wozu laut Rosenstengel u. Winand (1991) „[...] die Einfachheit, Modularität und Flexibilität und ihre Anschaulichkeit sowie die Mächtigkeit hinsichtlich der kompletten oder gezielten Analyse dynamischer Verhaltensweisen von Systemen [...]“ zählen.

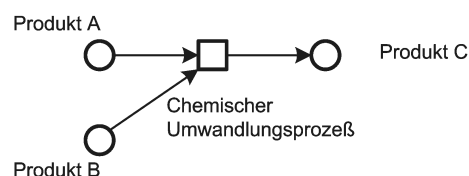
Der Abschnitt führt im ersten Teil in die Theorie der Petri-Netze ein, um darauf aufbauend Farbige Petri-Netze vorzustellen.

### 2.4.1 Petri-Netze

Folgende Kurzeinführung wurde mit Hilfe von Rosenstengel u. Winand (1991) erarbeitet:

Petri-Netze sind gerichtete Graphen, denen eine axiomatisierte mathematische Theorie zugrunde liegt. Die Graphen bestehen aus zwei Arten von Knoten: Ereignisse („*Transitionen*“), durch Quadrate dargestellt, und Zustände („*Plätze*“), durch Kreise repräsentiert. Zustände stehen für einen bestimmten Stand eines Systems, während Ereignisse für Zustandsübergänge stehen. Der zwischen Zuständen und Ereignissen bestehende formale Zusammenhang wird von Rosenstengel u. Winand (1991) folgendermaßen formuliert:

„Es wird unterstellt, daß jedes Ereignis eine exakt definierbare Menge realisierter Zustände (Vorbedingungen) voraussetzt und/oder eine exakt definierte Menge von Zuständen (Nachbedingungen) realisiert.“ (Rosenstengel u. Winand 1991, Seite 8). Dieser Zusammenhang wird durch gerichtete Kanten (Pfeile) visualisiert, wobei Kanten stets Ereignisse



**Abbildung 2.9:** Petri-Netz Beispiel (Rosenstengel u. Winand 1991, Abbildung 4, Seite 8)

mit Zuständen (oder umgedreht) verbinden - jedoch nie Ereignisse miteinander oder Zustände miteinander. Das dynamische Verhalten der Systeme („sprich das Realisieren von Zuständen in Abhängigkeit von der Realisation anderer Zustände“ (Rosenstengel u. Winand 1991)) wird durch die „Markierung“ von Zuständen dargestellt. Die Markierung beschreibt die Anzahl der Marken in diesem Platz. Zustände und Ereignisse in der Petri-Netz-Theorie sind durch ihre unmittelbare „lokale“ Umgebung beschreibbar. Sie haben keine Nebenwirkungen an anderen Stellen im Modell und ermöglichen daher eine modulare Modellierung von Systemen.

Die Vorteile der Modellierung mit Petri-Netzen wurde von Reisig (1998) zusammengefaßt:

- Petri-Netze nutzen die elementarsten Konzepte von Zuständen und Zustandsübergängen.
- Sie betonen die Lokalität von Ursache und Wirkung.
- Es werden explizit fundamentale Fragen von verteilten Systemen repräsentiert, wie zum Beispiel Atomarität, Synchronisation, Unabhängigkeit von Aktionen, Nachrichten und ein gemeinsames Gedächtnis
- Zustände und Aktionen sind gleichberechtigte Konzepte.
- Petri-Netze sind implementierbar und unabhängig von konkreten Implementationsprachen.

Abbildung 2.9 zeigt ein Beispiel eines Petri-Netzes. Der chemische Umformungsprozeß, der durch eine Transition repräsentiert wird, benötigt Produkt A und B als Ausgangsstoffe, um Produkt C herstellen zu können. Die Produkte werden durch Plätze dargestellt. Nur wenn sowohl im Platz „Produkt A“ als auch im Platz „Produkt B“ ein Token vorhanden ist, kann die Transition „Chemischer Umwandlungsprozeß“ feuern.

### 2.4.2 Farbige Petri Netze (Coloured Petri Nets - CPN)

Es gibt hinreichend viele Erweiterungen von Petri-Netzen, sogenannte „*High-Level Petri Nets*“ (Jensen u. Rozenberg 1991). Alle Erweiterungen ermöglichen es im Gegensatz zu normalen Petri-Netzen, daß die Marken („*Token*“) Informationen beinhalten können. Jensen u. Rozenberg (1991) vergleichen den Schritt von normalen zu höheren Petri-Netzen mit dem Schritt von Assembler-Sprachen zu getypten Programmiersprachen. Höhere Petri-Netze können in „normale“ Petri-Netze überführt werden, ermöglichen jedoch durch ihre kompaktere Darstellung knappere und lesbarere Figuren und damit die Modellierung von größeren und komplexeren Systemen.

Beispiele für höhere Petri-Netze sind Prädikat/Transitions-Netze (Genrich u. Lautenbach 1981), Farbige Petri-Netze (Jensen 1992), Referenznetze (Kummer 2002) und der Petri-Box-Kalkül (Best u. a. 2001). Die vorliegende Arbeit nutzt Farbige Petri-Netze, die aufgrund der üblichen englischen Bezeichnung „*Coloured Petri Nets*“ mit CPN abgekürzt werden. In dieser Arbeit wird dieser Konvention gefolgt.

---

Die folgende Einführung in Farbige Petri-Netze wurde zum überwiegenden Teil von Ehrler u. a. (2005) übernommen. Für eine detaillierte Einführung in die Grundlagen, Anwendungen und Analysen sei auf Jensen (1992, 1997, 1999) und Kristensen u. a. (1998) verwiesen.

Farbige Petri-Netze bestehen aus:

- **Marken:** („tokens“), die typisiert sind und Informationen besitzen können. Sie repräsentieren Prozesse oder Ressourcen.
- **Transitionen:** („transitions“), die Aktionen repräsentieren. Die Ausführung dieser Aktionen („firing“) kann die Eigenschaften, Anzahl und den Ort der Marken in den angeschlossenen Plätzen verändern. Typischerweise entnehmen sie einigen Plätzen Marken und fügen anderen Plätzen Marken hinzu. Transitionen können „guards“ besitzen, die aus booleschen Ausdrücken bestehen und deren Ergebnis zu wahr evaluieren muß, bevor die Aktion ausgeführt werden kann. Transitionen werden üblicherweise mit einem Rechteck dargestellt.
- **Plätzen:** („places“), die ebenfalls typisiert sind und Zustände darstellen. Sie können Marken beinhalten - das bedeutet, daß der entsprechende Prozeß oder die Ressource sich in dem durch den Platz repräsentierten Zustand befindet. Es können sich nur Marken mit demselben Typ in dem Platz befinden. Plätze werden üblicherweise durch einen Kreis dargestellt.
- **Kanten:** („arcs“), die Plätze und Transitionen verbinden. An ihnen können Beschriftungen angebracht sein, die das Ausführen der jeweiligen Transition beeinflussen. Kanten werden üblicherweise durch Pfeile dargestellt.

## 2.5 Modellierung von Interaktionsprotokollen mit AgentUML und CPN

Die Forschung hat verschiedene Formalismen zur Beschreibung von Interaktionsprotokollen in Multi-Agenten-Systemen entwickelt. Zwei der wichtigsten Sprachen sind zum einen die „Agent Unified Modeling Language“ (AgentUML), zum anderen Farbige Petri-Netze (vorgestellt in Abschnitt 2.4.2 auf der vorherigen Seite). Vorgeschlagen als Modellierungssprachen wurden zum Beispiel auch erweiterte Zustandsgraphen (kombiniert mit dynamischer Propositionslogik) (Paurobally u. Cunningham 2003) oder Z als formale Beschreibung von Interaktionsprotokollen (Huget 2005). Eine Kombination verschiedener Formalismen schlägt Peña u. a. (2003) vor.

Der Nachteil der meisten dieser alternativen Ansätze ist, daß die verwendeten Notationen einen geringen Bekanntheitsgrad haben und wenig verbreitet sind. Sie müssen meist erst den Agenten-Entwicklern erklärt werden und ist daher für kommerzielle Software-Agenten-Entwicklungen ungeeignet. Demgegenüber sind sowohl Farbige Petri-Netze als auch UML allgemein anerkannt und werden von den meisten Informatikern verstanden.

---

### 2.5.1 Interaktionsprotokolle modelliert mit AgentUML

UML ist unter Softwareentwicklern bekannt. Deshalb haben viele Forscher UML als Spezifikationformalismus für Multi-Agenten-Systeme vorgeschlagen. Cranefield u. Purvis (1999) schlugen UML zum Beispiel als Modellierungssprache für Ontologien vor (siehe Abschnitt 2.1.5). Ihrer Meinung nach sind die Vorteile von UML die größer werdende Benutzerzahl, die grafische Repräsentation von UML (im Vergleich zu den üblicherweise für die Ontologiespezifikation genutzten logischen Formalismen) und die Möglichkeit, Bedingungen mit Hilfe der Object Constraint Language (OCL) zu formulieren.

Parunak u. Odell (2001) modellierten soziale Strukturen in UML. Dies ist wichtig, da Multi-Agenten-Systeme mit einer menschlichen Gesellschaft verglichen werden können. Bauer u. a. (2001a) stellten fest, daß bereits bekannte Standards und Technologien die Nutzung der Multi-Agenten-Technologie in kommerziellen Bereichen erhöhen würde. Da viele MAS mithilfe von objekt-orientierter Technologie implementiert werden können und viele Software-Entwickler UML kennen, ist die Nutzung von UML für MAS vorteilhaft.

Die Anpassung von UML zur Nutzung in Multi-Agenten-Systemen wurde 2000 von Odell u. a. (2001) und Bauer u. a. (2001b) begonnen. Sie schlugen UML für die Modellierung von Interaktionsprotokollen vor. FIPA griff dies auf und schlug darauf aufbauend eine Standardbibliothek von Interaktionsprotokollen vor (FIPA IP Library 2000). Weitere Arbeiten in der Anpassung von UML für Agenten wurden zum Beispiel von Koning u. a. (2001), Lind (2001b) oder Wagner (2003) durchgeführt.

Im Jahre 2003 wurde von der FIPA entschieden, ein Technical Committee zu gründen, um AgentUML auf der Basis von UML 2.0 zu standardisieren (FIPA AUML workplan 2003). Der Standardisierungsprozeß dauert an und erste Ergebnisse wurden veröffentlicht (FIPA AUML 2003; Huget 2004; Huget u. Odell 2005; Huget u. a. 2004).

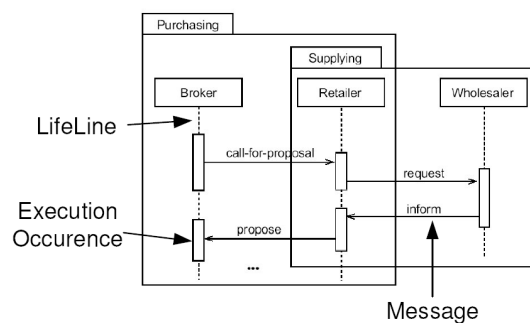


Abbildung 2.10: Ein AgentUML-Beispiel-Diagramm (Odell u. a. 2001)

**Wie sind AgentUML-Diagramme zu lesen** Abbildung 2.10 zeigt ein AgentUML-Beispieldiagramm. Ein Sequenzdiagramm besteht aus Lifelines, die durch gestrichelte, vertikale Linien und einem Rechteck am oberen Ende dargestellt werden. Das Rechteck enthält den Namen der Lifeline. Eine Lifeline repräsentiert eine oder mehrere Akteure, die im Falle von Multi-Agenten-Systemen eine oder mehrere Agenten sind (In Abbildung 2.10 könnte zum Beispiel ein Einzelhändler (Retailer) von mehreren Großhändlern (Wholesaler) versorgt werden). Diese Lifelines werden durch Nachrichten verbunden, die durch horizontale Linien mit einem Pfeil am Ende dargestellt werden. Die Lifeline, zu der

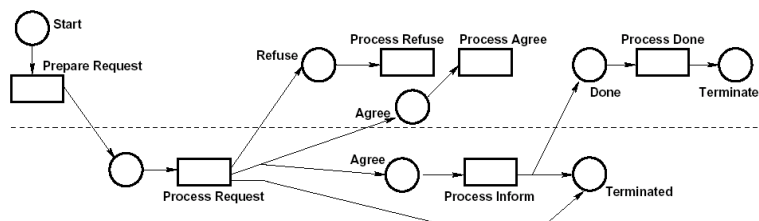
der Pfeil zeigt, repräsentiert den Empfänger der Nachricht, die andere mit der Nachricht verbundene Lifeline den Sender der Nachricht. Einer Lifeline können Aktionen („ExecutionOccurrences“) zugeordnet sein. Diese werden durch Rechtecke dargestellt, die an die Lifeline angefügt sind.

Ein Sequenzdiagramm wird von oben nach unten gelesen. Das heißt, die oberste Nachricht ist die erste Nachricht, die gesandt wird, die unterste Nachricht ist die letzte Nachricht, die gesandt wird. Eine genaue Beschreibung der einzelnen Teile und ein Metamodell für AgentUML-Diagramme wurde vorgeschlagen. (Ehrler 2003).

## 2.5.2 Interaktionsprotokolle modelliert mit CPN

Es gibt zwei verschiedene Ansätze, Interaktionsprotokolle mit CPN zu modellieren. Cost u. a. (2000) beschreiben den ersten Ansatz. Dabei wird ein CPN für die gesamte Interaktion genutzt. Ein Beispiel ist in Abbildung 2.11 gezeigt. Im angezeigten CPN stehen die Plätze für einzelne Nachrichten (oder für Start und Beendet) und die Transitionen für die Verarbeitung/Erstellung von Nachrichten. Die gestrichelte Linie zeigt an, welcher Teil des Protokolls zu welcher Rolle gehört.

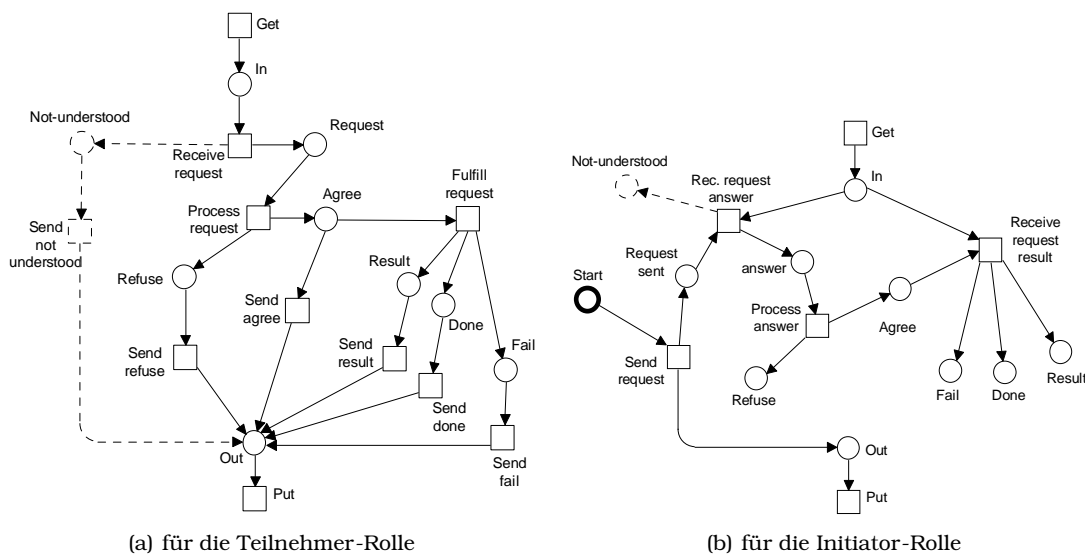
Dieser Ansatz wurde von vielen Forschern aufgegriffen und erweitert, so zum Beispiel von Nowostawski u. a. (2001a). Sie betonen, daß Farbige Petri-Netze eine nützliche Abstraktion für die Modellierung von nebenläufigen Interaktionen bieten.



**Abbildung 2.11:** Ein Interaktionsprotokoll mit nur einem CPN modelliert (Nowostawski u. a. 2001b)

Der andere Ansatz zur Modellierung von IP mit Hilfe von Farbigen Petri-Netzen wurde von (Purvis u. a. 2002b) entwickelt. Sie nutzen für jede Rolle ein separates CPN, wodurch die Zustandsänderung der einzelnen Agenten besonders hervorgehoben wird. Abbildung 2.12 zeigt ein solches Protokoll. Darin wurde das Request-Protokoll der FIPA mit den beiden beteiligten Rollen „Initiator“ und „Teilnehmer“ dargestellt. Beide Rollen haben ihr eigenes CPN - dabei gibt es in jedem Netz eine „Get“-Transition, die die eingehenden Nachrichten empfängt, und eine „Put“-Transition, die die ausgehenden Nachrichten sendet. Die initiiierende Rolle hat immer auch einen „Start“-Platz, der ein Token erhält, wenn der betreffende Agent die Kommunikation beginnen möchte.

Für diese Modellierungsmöglichkeit wurde auch eine Werkzeug-Unterstützung entwickelt: Das Tool JFern (Nowostawski 2000) kann Farbige Petri-Netze für einen Agenten der Agentenplattform Opal (Purvis u. a. 2002a) ausführen.



**Abbildung 2.12:** Request-Interaktionsprotokoll mit je einem CPN für jede beteiligte Rolle (Purvis u. a. 2002b, Abbildungen 1+2)

### 2.5.3 Vergleich von Farbigen Petri-Netzen und AgentUML

Unterschiedliche Anforderungen benötigen unterschiedliche Modellierungssprachen für Interaktionsprotokolle. Im Vergleich von AgentUML und Farbigen Petri-Netzen plädieren Craneffeld u. a. (2002) für CPN anstelle von AgentUML, da

- AgentUML zu dem Zeitpunkt ungenau spezifiziert war,
- Keine Werkzeuge für AgentUML verfügbar waren / sind,
- Es zu diesem Zeitpunkt keine Möglichkeit gab, den internen Zustand eines Agenten explizit zu modellieren.

Aber es gibt auch Argumente, die für die Nutzung von UML sprechen. Bauer u. a. (2001b) machen geltend, daß die Nutzung von bekannten Standards (zu denen er UML zählt) die Verbreitung von Software-Agenten in der kommerziellen Software-Entwicklung fördern kann.

Außerdem betonen Farbige Petri-Netze besonders den internen Zustand eines Agenten. Das ist zwar wichtig, aber es kann für Anwendungen auch unwichtig sein. In AgentUML wird der Kommunikationsverlauf selbst klarer dargestellt und kann daher einfacher überblickt werden. Die Probleme, die von Craneffeld u. a. (2002) angesprochen wurden, werden angegangen:

- Das Technical Comitee der FIPA arbeitet momentan an einer genaueren Spezifikation der Notation von AgentUML. Die vorliegende Arbeit will daran anknüpfend eine genaue semantische Definition erarbeiten.
- FIPA untersucht verschiedene UML-Editoren. Das Werkzeug PAUL (Ehrler u. Craneffeld 2004) stellte einen Ansatz zur Verfügung, wie AgentUML-Diagramme von Agenten interpretiert werden können. Außerdem arbeiten verschiedene Arbeitsgruppen an weiteren Werkzeugen zur AgentUML-Unterstützung (El-Kady 2004; Guru 2005; Ingenias 2004).

AgentUML hat demzufolge Vorteile, wenn in einem Kommunikationsmodell nicht der interne Zustand eines Agenten das Wichtigste ist, sondern die Modellierung der Interaktion selbst.

---

# 3

## Formale Semantik von AgentUML

Dieses Kapitel enthält die formale Semantik von AgentUML. Nach einem Überblick über die bisherigen Ansätze, eine formale Semantik zu entwickeln, wird in Abschnitt 3.2 das erstellte Metamodell von AgentUML vorgestellt, um dann im letzten Abschnitt eine Übersicht über die Semantik zu zeigen. Die vollständige Definition der AgentUML-Syntax und -Semantik ist englischsprachig in Anhang B zu finden.

### 3.1 Formalismen auf AgentUML angewendet

Die UML-Definition besitzt keine formale Beschreibung der Semantik. In der UML-Spezifikation wird die Semantik „nur“ englischsprachig beschrieben. Jeckle u. a. (2004) sehen darin einen Nachteil von UML: „Als Kritikpunkt verbleibt auch bei UML 2.0 daher, dass immer noch sehr viele Sprachkonzepte ausschließlich durch natürlichsprachliche Beschreibungen charakterisiert sind, welche teilweise nicht alle Facetten der möglichen Verwendung eindeutig klären oder an einigen Stellen in offenen Widerspruch zu anderen Aussagen der Spezifikation treten.“ Deshalb können aus der UML-Spezifikation keine Semantiken „übernommen“ werden.

Es gab nun in den letzten Jahren verschiedene Ansätze, AgentUML auf formale Notationen abzubilden: Die Beschreibung mit modaler Logik (Abschnitt 3.1.1) und die Abbildung auf eine Art von Petri-Netzen (Abschnitt 3.1.2). Diese beiden bisherigen Ansätze sollen im Folgenden beschrieben werden.

#### 3.1.1 AgentUML durch Modallogik definiert

Modallogik ist eine Weiterentwicklung klassischer mathematischer Logik. In klassischer Aussagen- oder Prädikatenlogik definiert sich die Semantik einer Aussage nur aus den

Bedeutungen der Teilaussagen. Sie ist also kompositionell. Wenn man jedoch über Bewußtseinszustände wie Überzeugungen und Absichten nachdenkt, dann können diese mit jener herkömmlichen Logik nicht dargestellt werden, da hier nicht nur nach den Bedeutungen aufgrund des Bezuges auf die Teilaussagen, sondern nach dem Sinn der Gesamtaussage gefragt wird. Wooldridge (2002, Kapitel 12) führt den Begriff der „referential opacity“ ein. Dieser Begriff (der sich schwer ins Deutsche übersetzen läßt) sagt aus, daß sich die Semantik von bestimmten Dingen nicht durch Wahrheitstabellen definieren läßt (wie in herkömmlicher, formaler Logik üblich).

Zum besseren Verständnis ein von Lyons (1977) zitiertes Beispiel des deutschen Philosophen und Mathematikers Friedrich Ludwig Gottlob Frege (1848–1925):

*Der Morgenstern ist der Abendstern.*

Morgen- und Abendstern beziehen sich beide auf die Venus, haben also beide denselben Bezug - die selbe Bedeutung. Und doch haben sie nicht den selben Sinn. Der Morgenstern ist ein heller Punkt am Morgenhimmel, der Abendstern am Abendhimmel. Die Venus ist nur dann der Morgenstern, wenn sie am Morgenhimmel steht - und nur dann der Abendstern, wenn sie am Abendhimmel steht. Ähnlich ist es bei anderen Überzeugungen und Absichten. Häufig weichen Bedeutung (die über einen Bezug definiert ist) und Sinn (der meist über einen Kontext definiert ist) voneinander ab (ausführlicher wird der Unterschied von Lyons (1977, Kapitel 7) betrachtet).

Genau für diese Dinge wurde die Modallogik erschaffen. „Unter Modallogiken versteht man solche Logiken, die zusätzlich zu den aus der Aussagenlogik und Prädikatenlogik bekannten Konstrukten Operatoren enthalten, mit denen man über Modalitäten wie *möglich* und *notwendig* (alethische Logik), *immer* (*in der Zukunft*) und *manchmal/irgendwann* (*in der Zukunft*) (temporale Logik) etc. sprechen kann.“ (Wikipedia 2001, Stichwort: „Modallogik“).

Da Agenten häufig über Absichten und Ziele definiert werden (BDI-Agenten), wird Modallogik oft in Multi-Agenten-Systemen benutzt. Laut Wooldridge (2002, Kapitel 12) ist es die am meisten genutzte Logikform in Multi-Agenten-Systemen.

Baldoni u. a. (2005) verwenden die Sprache *DyLOG* (Baldoni u. a. 2004), einer auf Modallogik basierten Agentenprogrammierungssprache, um AgentUML zu definieren. Sie übersetzen dabei die AgentUML-Diagramme in eine Grammatik, bestehend aus Terminalsymbolen (die die Nachrichtenoperatoren repräsentieren) und Produktionsregeln (die das Interaktionsprotokoll definieren). Dabei muß beachtet werden, daß sowohl die Semantik der Sprechakte nicht implementiert wurde, als auch die möglichen Kommentare und Bedingungen im Diagramm. Diese müssen per Hand hinzugefügt werden.

Hauptkritikpunkt an dieser Arbeit ist, daß das hauptsächliche Ziel war, AgentUML in die prototypische Agentenplattform „DCaseLP“ einzubinden, um über Interaktionsprotokolle schlußfolgern zu können. Darunter litt vor allem die menschliche Lesbarkeit - was diesen Ansatz nach den Anforderungen an eine Semantikspezifikation in Abschnitt 2.3.5 auf Seite 22 für ein Spezifikationsinstrument unbrauchbar macht.

---

### 3.1.2 AgentUML in Petri-Netze übersetzen

Die Übertragung der ersten Version von AgentUML in Petri-Netze wurde von einigen Forschern versucht, die dabei unterschiedliche Erweiterungen von Petri-Netzen nutzten.

Mazouzi u. a. (2002) übersetzte AgentUML in CPN, um eine formale Verifikation und Validation zu ermöglichen. Dabei repräsentierte er die Lifelines eines Diagrammes mit einer implizit durch Plätze und Transitionen. Der Nachrichtenaustausch wurde dargestellt durch einen Platz, verbunden mit je einer Transition für das Senden und Empfangen. Cabac u. Moldt (2005) nutzte sogenannte Mulan-Netze für AgentUML. Mulan-Netze sind eine Art von Referenznetzen (Kummer 2002), die grafisch besonders angeordnet sind. Diese grafische Anordnung soll eine erhöhte Lesbarkeit gewährleisten. Die erhaltenen Netze können durch das Tool Renew (Kummer u. a. 2004) ausgeführt werden. Gutnik u. Kaminka (2005) entwickelte eine Petri-Net-Repräsentation von Interaktionsprotokollen speziell zur Überwachung von Agentenkommunikationen. Er zeigt dann, daß sich auch AgentUML mit dieser Repräsentation darstellen läßt. Die Besonderheit dieses Ansatzes liegt darin, daß nicht der Zustand des Agenten im Vordergrund steht, sondern der Zustand des Protokolles.

Der nach Kenntnis des Autors bisher erste und einzige Versuch, die aktuelle Version von AgentUML in Petri-Netze zu übersetzen, wurde von Bagic (2004) präsentiert. Sie zeigt vor allem, daß es realisierbar ist, die verschiedenen Arten von möglichen AgentUML-Diagrammen (Agenten-Klassendiagramme, -Sequenzdiagramme, -Aktivitätendiagramme usw.) mit CPN zu definieren.

Alle Ansätze haben gemeinsam, daß sie nur mit wichtigen Sequenz-Diagramm-Fragmenten arbeiten, und keine formale Syntaxdefinition als Grundlage haben. Dies macht es schwierig, eine allgemeingültige und vollständige Spezifikation von AgentUML abzuleiten.

## 3.2 AgentUML-Metamodell

Die Sprache UML (und darauf basierende Erweiterungen) sind, wie in Abschnitt 2.2.2 auf Seite 15 beschrieben, syntaktisch durch ein Metamodell definiert. Da die AgentUML-Sequenzdiagramme auf den UML 2.0 Sequenzdiagramme basieren, wurde das entsprechende Metamodell als Grundlage für ein UML-Profil benutzt.

Ein UML-Profil ist ein Erweiterungsmechanismus von UML 2.0, der es erlaubt, UML an spezielle Einsatzgebiete anzupassen. Born u. a. (2004, Abschnitt 5.1) führt aus: „In einem UML-Profil werden Adaptionen von Konstruktionen der Sprache mit einer bestimmten Zielstellung zusammengefasst“.

Eine grundlegende Einschränkung von UML-Profilen ist, daß es keine Generalisierungsmöglichkeiten gibt. Ein UML-Profil verhält sich zum Original wie eine abgeleitete Klasse zur Oberklasse - ein Profil ist eine Konkretisierung des allgemeineren Originals. Dies bereitet dann Schwierigkeiten, wenn das Original Einschränkungen vornimmt, die im speziellen Anwendungsgebiet nicht gelten.

Aus diesem Grunde wurden für AgentUML das UML 2.0-Sequenzdiagramm-Metamodell als Vorbild genutzt, das Profil jedoch nicht von diesen Klassen abgeleitet, sondern neue Klassen entwickelt (mit einem „A“ vor jedem Namen für „Agent“). Dies hat

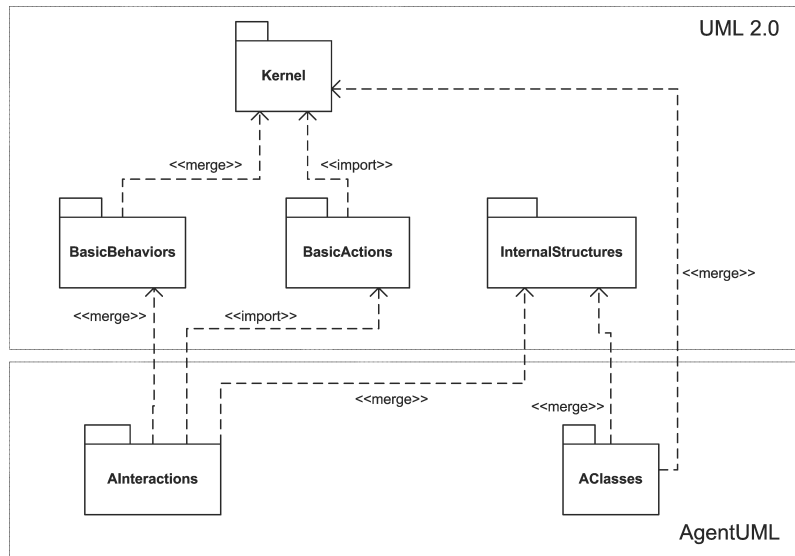


Abbildung 3.1: Metamodell-Pakete von AgentUML

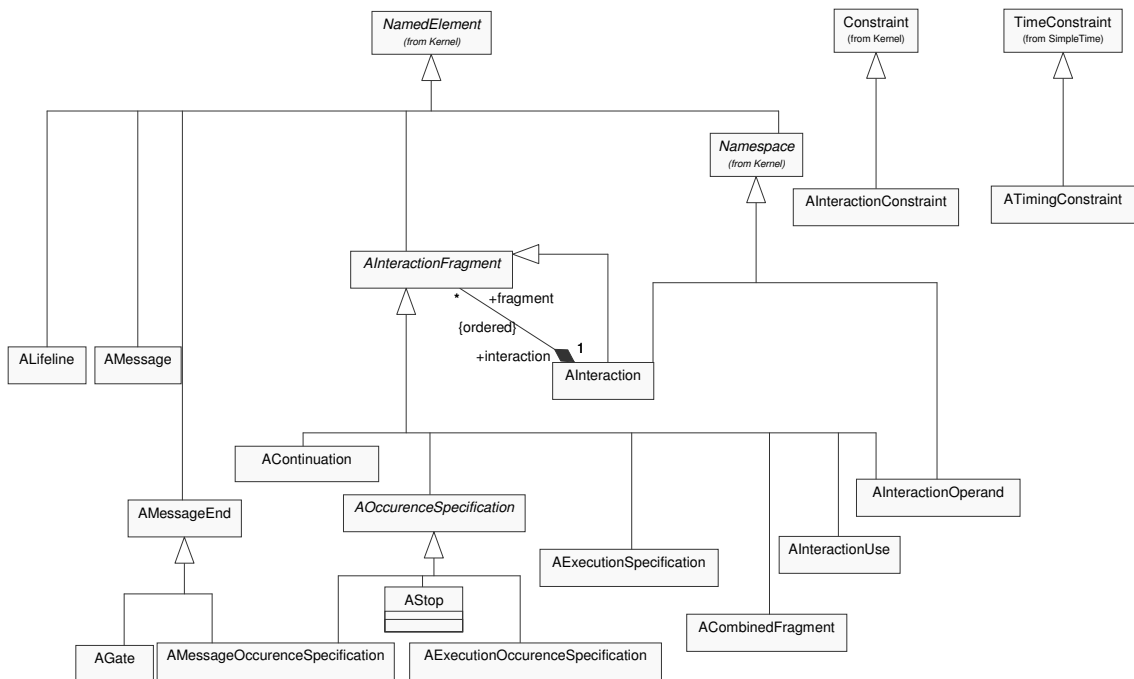


Abbildung 3.2: Vererbungshierarchie des AgentUML-Metamodells

den Nachteil, daß die Wiederbenutzung sinkt, jedoch ist man nicht an Festlegungen der UML-Sequenzdiagramme gebunden. Dies betrifft insbesondere die Angabe, welche Objekte durch Lifelines repräsentiert werden können, aber auch die Angabe der Operatoren von *CombinedFragments*.

Abbildung 3.1 verdeutlicht die Einbindung der AgentUML-Metamodell-Pakete in die Pakete des UML2.0-Rahmenwerks.

Es gibt zwei Pakete: das Paket „AInteractions“, das alle Elemente der Interaktionsprotokolle umfaßt (siehe Abschnitt 3.2.1), sowie das Paket „AClasses“, das alle Element der AgentUML-Klassendiagramme enthält. (Da diese nicht im Fokus der vorliegenden Arbeit standen, wurden sie im Abschnitt 3.2.2 im Eintrag „Paket: AClasses“ nur kurz beschrieben.) Abbildung 3.2 enthält die Vererbungshierarchie des AgentUML-Metamodells als UML-Klassendiagramm.

Im Folgenden wird ein Überblick über die Struktur des Metamodells gegeben sowie eine Kurzbeschreibung der einzelnen Klassen. Eine englischsprachige Übersicht kann im Anhang B nachgelesen werden.

### 3.2.1 Struktur des Metamodells

Das Metamodell ist eng an das UML 2.0 Metamodell angelehnt (UML 2.0 2004, Kapitel 14), außerdem nimmt es Ideen von Odell u. a. (2003) auf. Ziel des Metamodells war es, möglichst genau die AgentUML-Spezifikation der FIPA (FIPA AUML 2003) abzubilden. Das AgentUML-Metamodell ist eng an das UML 2.0 Metamodell angelehnt (UML 2.0 2004, Kapitel 14), baut auf das von Ehrler u. Cranefield (2004) vorgeschlagene Metamodell auf und übernimmt Ideen von Odell u. a. (2003). Im Gegensatz zum Metamodell von Ehrler u. Cranefield (2004), das nur als vorläufige Definition einer AgentUML-Teilmenge vorgeschlagen wurde, war es Ziel dieser Arbeit, möglichst genau die AgentUML-Spezifikation der FIPA (FIPA AUML 2003) abzubilden.

Abbildung 3.3 stellt die wichtigsten Klassen des Metamodells sowie ihre Beziehungen untereinander in Form eines UML-Klassendiagrammes dar.

Die wichtigsten Eigenschaften des Metamodells:

- *AInteractionFragment* ist eine abstrakte Generalisierung aller in einer Interaktion vorkommenden Elemente (außer Nachrichten). Da *AInteractionFragment* in *AInteraction* als Komponente enthalten ist, sind alle Unterklassen ebenfalls als Komponenten enthalten.
  - Die Teilnehmer an der Interaktion werden durch *ALifeline* repräsentiert. Teilnehmen können einzelne Agenten, Gruppen von Agenten, Agenten-Rollen und Arten von Agenten. Sie werden unter dem Begriff *AAddressableEntity* zusammengefasst.
  - Das Verhalten eines Interaktionspartners wird durch die mit der entsprechenden *ALifeline* assoziierten *AOccurrenceSpecifications* bestimmt. Diese spezifizieren entweder das Senden oder Empfangen einer Nachricht (*AMessageOccurrenceSpecification*) oder den Start/das Ende einer Aktion (*AExecutionOccurrenceSpecification*).
  - Auswahlmöglichkeiten werden durch kombinierte Fragmente (*ACombinedFragments*) modelliert. Ihr Verhalten wird durch einen Operator (*AInteractionOperator*)
-



bestimmt, der die enthaltenen Operanden (*AInteractionOperands*) miteinander verknüpft.

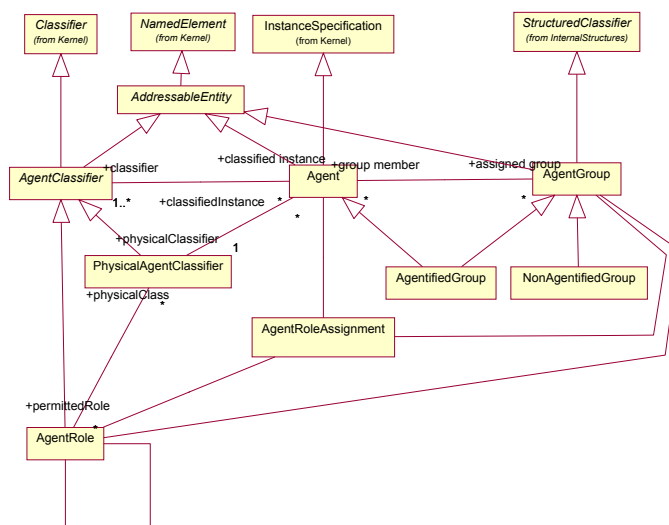
### 3.2.2 Klassenbeschreibung des Metamodells

Das AgentUML-Metamodell umfaßt folgende Klassen:

#### AClasses-Paket

- **AAddressableEntity:** **AAddressableEntity** ist eine Superklasse für alle Entitäten, die durch eine *ALifeline* repräsentiert werden können. Odell u. a. (2003) erarbeiteten ein Metamodell für die Beziehungen von Agenten und ihren Rollen untereinander. Dieses Modell wurde durch die vorliegende Arbeit um die Klasse *AAddressableEntity* erweitert. Sie faßt zusammen:

- einzelne Agenten
- Agentengruppen
- Agenten-Klassifizierer (Rollen sowie Arten von Agenten)



**Abbildung 3.4:** Metamodell für ein Agenten-Strukturdiagramm (als UM-Klassendiagramm) (veränderte Version von FIPA AUML Class diagrams (2004, Abbildung 1))

Abbildung 3.4 verdeutlicht dies.

- **Agent:** ein Agent.
- **AgentGroup:** eine Gruppe von Agenten.
- **AgentifiedGroup:** eine *AgentGroup*, die von anderen Agenten als einzelner Agent angesehen werden kann.
- **NonAgentifiedGroup:** eine *AgentGroup*, die keine *AgentifiedGroup* ist.
- **AgentRoleAssignment:** das „Spielen“ einer *AgentRole* durch einen Agenten innerhalb einer *AgentGroup*.

- **AgentClassifier:** eine Agenten-Einordnung (entweder *AgentRole* oder *PhysicalAgentClassifier*).
- **AgentRole:** die Rolle eines Agenten innerhalb einer *AgentGroup*.
- **PhysicalAgentClassifier:** ein Agenten-Typ (gemeint ist hier zum Beispiel, ob ein Agent ein JADE-Agent (Bellifemine u. a. 1999) oder ein Opal-Agent (Purvis u. a. 2002a) ist).

### **AInteraction-Paket**

- **ACombinedFragment:** *ACombinedFragment* verallgemeinert verschiedene Auswahlmöglichkeiten wie Alternativen, Schleifen oder parallele Ausführungen. Ein kombiniertes Fragment besteht aus einem oder mehreren *AInteractionOperands*, die durch einen *AInteractionOperator* miteinander verknüpft sind.
  - **AContinuation:** *AContinuation* repräsentiert eine Sprungmarke. Sie repräsentiert entweder das Ziel eines Sprunges (der Pfeil ist vor dem Namen) oder den Start (der Pfeil ist hinter dem Namen). Sie soll Protokolle vereinfachen, indem keine aufwendigen *ACombinedFragments* nötig sind, um eine bestimmte Art der Kommunikation zu erreichen.
  - **AExecutionSpecification:** *AExecutionSpecification* stellt eine Aktion dar. Sie gehört zu einer *ALifeline*, die diese Aktion ausführt. Das Start- und das Endereignis werden durch *AExecutionOccurenceSpecification* repräsentiert
  - **AExecutionOccurenceSpecification:** *AExecutionOccurenceSpecification* repräsentiert das Ereignis des Beginns oder des Endes einer Aktion und ist mit einer *AExecutionSpecification* verbunden.
  - **AGate:** *AGate* modelliert die Möglichkeit, daß eine Nachricht in eine *AInteraction* oder einen *AInteractionOperand* hinein oder heraus gesandt wird. Es stellt den Übergang an der Begrenzungslinie dar und „teilt“ die Nachricht in zwei Teilnachrichten.
  - **AInteraction:** *AInteraction* repräsentiert ein Interaktionsdiagramm. Es umfaßt alle Objekte, die in diesem Diagramm enthalten sind.
  - **AInteractionConstraint:** Ein *AInteractionConstraint* ist ein boolescher Ausdruck, der bestimmt, ob der zugehörige *AInteractionOperand* ausgeführt wird oder nicht. Wenn der Ausdruck wahr ist, dann wird der *AInteractionOperand* ausgeführt, ansonsten nicht.
  - **AInteractionFragment:** *AInteractionFragment* ist eine abstrakte Klasse, die als Generalisierung der Fragmente einer Interaktion dient.
  - **AInteractionOperand:** Ein *AInteractionOperand* ist ein Teil eines *ACombinedFragments* und stellt einen möglichen Pfad dieses Fragmentes dar. Es enthält alle Elemente dieses Pfades.
-

- **AInteractionOperator:** *AInteractionOperator* bestimmt das Verhalten eines *ACombinedFragments*. Es ist eine Aufzählung mit den möglichen verschiedenen Verhaltensweisen von *ACombinedFragment*: Alternative, Option, Break, Parallel, Weak Sequencing, Strict Sequencing, Critical Region, Loop.
- **AInteractionUse:** *AInteractionUse* ist ein Platzhalter-Objekt für eine innerhalb dieser Interaktion vorkommende andere *AInteraction*.
- **ALifeline:** *ALifeline* spezifiziert die in Interaktionsdiagrammen vorkommenden Lifelines. Sie repräsentieren die Teilnahme eines/mehrerer Interaktionspartner in der Interaktion.
- **AMessage:** *AMessage* repräsentiert die Nachrichten, die in der Interaktion vorkommen.
- **AMessageEnd:** *AMessageEnd* modelliert das Ende einer Nachricht. Es ist eine abstrakte Generalisierung der Klassen *AGate* und *AMessageOccurrenceSpecification*.
- **AMessageOccurrenceSpecification:** *AMessageOccurrenceSpecification* stellt das Ereignis des Sendens und Empfangens einer Nachricht dar. Eine *AMessageOccurrenceSpecification* ist immer mit einer *AMessage* und einer *ALifeline* verbunden.
- **AMessageType:** *AMessageType* ist eine Aufzählung der möglichen Typen einer Nachricht. Sie wird durch die verwendete Agentenkommunikationssprache (Abschnitt 2.1.4 auf Seite 8) festgelegt.
- **AOccurrenceSpecification:** Eine *AOccurrenceSpecification* ist eine abstrakte Generalisierung von Ereignissen, die an einer *ALifeline* stattfinden können. Sie sind entweder der Start/das Ende einer Aktion (*AExecutionOccurrenceSpecification*) oder das Senden/Empfangen einer Nachricht (*AMessageOccurrenceSpecification*).
- **AStop:** *AStop* bezeichnet das Ende einer *ALifeline*. Das bedeutet, daß die von dieser *ALifeline* repräsentierten *AAddressableEntities* nicht mehr an der Interaktion teilnehmen.
- **ATimingConstraint:** *ATimingConstraint* erlauben es, bestimmte Zeitbeschränkungen festzulegen. Sie werden einer Nachricht zugeordnet, der diese Beschränkung auferlegt wird (zum Beispiel, innerhalb eines Zeitintervalls einzutreffen).

### 3.3 AgentUML-Semantik

Nach UML 2.0 (2004, Abschnitt 14.3.13, Seite 526) ist die Semantik von Interaktionen in UML kompositionell aufgebaut. Auch Farbige Petri-Netze können hierarchisch, und damit kompositionell aufgebaut werden. Deshalb wurde für jede Klasse des in Abschnitt 3.2 beschriebenen Metamodells eine CPN-Repräsentation entwickelt. Die Semantik eines Diagrammes kann nun folgendermaßen gewonnen werden:

1. Analyse des Diagrammes
-

2. Identifikation der Metamodell-Objekte (Instanzen der Metaklassen wie beispielhaft in Abbildung 2.6 gezeigt)
3. Abbildung der einzelnen Objekte auf ihre CPN-Repräsentation
4. Zusammenfügen der einzelnen Elemente

Im folgenden wird die Gesamt-Semantik eines Interaktionsprotokolls überblickshaft vorgestellt. Die vollständige, detaillierte Spezifikation ist in englischer Sprache in Anhang B aufgeführt.

Die Semantik von AgentUML, die in dieser Diplomarbeit präsentiert wird, basiert auf dem Ansatz, der von Mazouzi u. a. (2002) präsentiert wurde. Die Struktur des CPN besteht danach aus folgenden Elementen (siehe auch Abbildung 3.5):

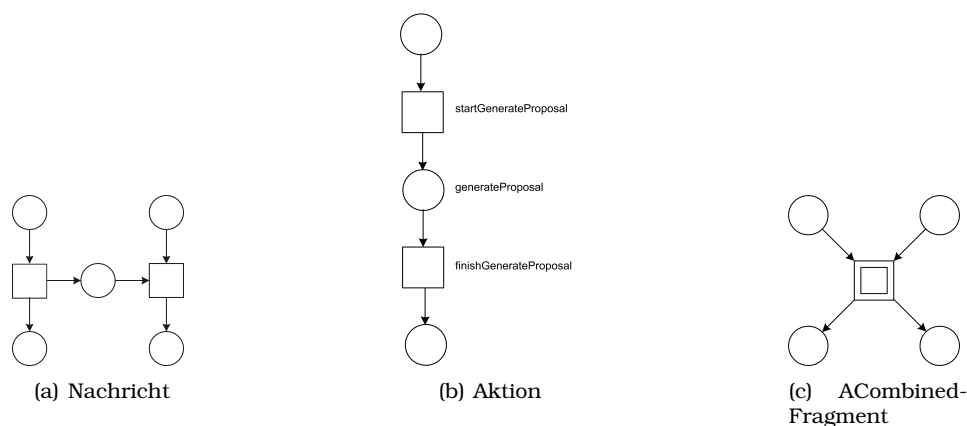
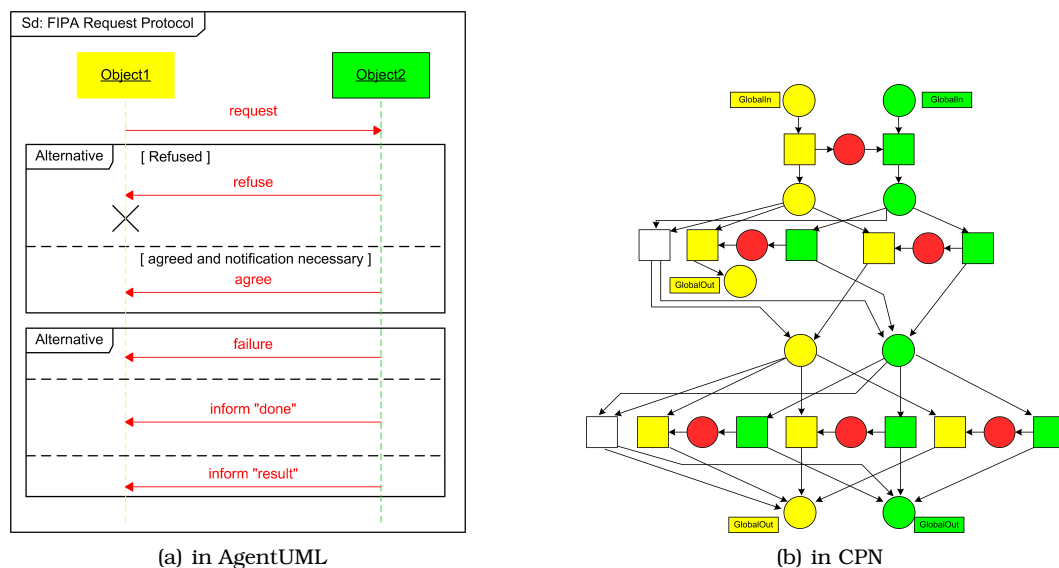


Abbildung 3.5: verschiedene CPN-Muster

- **Lifeline:** Die Lifelines eines Diagrammes sind durch eine Sequenz von Plätzen und Transitionen repräsentiert. Sie formen ein Teilnetz des Gesamtnetzes. Jedes Teilnetz steht bei Mazouzi u. a. (2002) für eine Agentenrolle. In der vorliegenden Arbeit wurde das erweitert. Ein Teilnetz kann nun - genauso wie die entsprechende Lifeline - außer einer Agentenrolle auch einen einzelnen Agenten oder eine Agentengruppe darstellen.
- **Nachrichten:** Der Nachrichtenaustausch ist repräsentiert durch einen Platz, der für die Nachrichtenübermittlung steht, und je einer Kante von dem Nachrichtensender und zu dem Nachrichteneempfänger. In den beteiligten Teilnetzen sind die Kanten mit entsprechenden „Sende-“ bzw. „Empfangs“-Transitionen verbunden (Abbildung 3.5(a)).
- **Aktionen:** Mazouzi u. a. (2002) definiert etwas unklar die Repräsentation von internen Aktionen. In der vorliegenden Arbeit werden Aktionen eines Agenten durch zwei Transitionen („starteAktion“ und „beendeAktion“) sowie durch einen Platz („inAktion“) repräsentiert, die miteinander verbunden sind (Abbildung 3.5(b)).
- **Verzweigungen:** Verzweigungen, die in AgentUML durch ein *ACombinedFragment* dargestellt werden können, werden im entsprechenden farbigen Petri-Netz durch eine *substitution transition* repräsentiert. Die entsprechende *subpage* beinhaltet dann die Fragmente, die im *ACombinedFragment* enthalten sind.

- **Start:** Jedes Teilnetz besitzt einen „Start“-Platz. Dieser Platz kann als Interface für die Einbindung des Interaktionsprotokolls in ein größeres Gesamtnetz des Agenten dienen. Außerdem gibt es einen „End“-Platz jedes Teilnetzes.
- **Farben:** Es gibt zwei Farbmengen („Colour Sets“) in diesem CPN:
  - Die Menge aller Instanzen der Klasse *AAddressableEntities*, die in dem Protokoll vorkommen können.
  - Alle Elemente der Enumeration *AMessageType*.
- **Token:** Entsprechend der zwei Farbmengen gibt es zwei Typen von Token:
  - Jeder Agent, der das Interaktionsprotokoll ausführt, wird durch je einen Token für jedes *AAddressableEntity*, das ihn repräsentiert, dargestellt. Sollte er also in Form von zwei verschiedenen Rollen an der Interaktion teilnehmen (zum Beispiel in einer Auktion ist er sowohl Bieter als auch Auktionator), dann besitzt er in diesem CPN zwei Token (im Beispiel eines mit der Farbe „Bieter“ und eines mit der Farbe „Auktionator“).
  - Jede gesandte Nachricht wird durch ein Token repräsentiert, das die Farbe des benutzen Performatives besitzt.



**Abbildung 3.6:** Beispiel: FIPA Request Interaction Protocol (FIPA AUML 2003, Abbildung 26)

Abbildung 3.6 verdeutlicht dies anhand des „FIPA Request Interaction Protocol“ (FIPA AUML 2003). Dabei zeigt Abbildung 3.6(a) die AgentUML-Darstellung, und Abbildung 3.6(b) die entsprechende CPN-Semantik.

1. Die Lifelines sind zum besseren Verständnis farbig (gelb und grün) gekennzeichnet. Im CPN repräsentieren die ebenfalls gelb/grün gekennzeichneten Plätze und Transitionen die jeweiligen Teilnetze.
2. Die rot markierten Nachrichten werden durch die ebenfalls rot gekennzeichneten Plätze dargestellt.

3. Es gibt „GlobalIn“- und „GlobalOut“-Plätze für jede Lifeline. Da dies logische Plätze sind, können sie zur besseren Lesbarkeit auch mehrfach vorkommen - sie bezeichnen dennoch immer denselben Platz (im Beispiel wird das Stop-Element der gelben Lifeline durch einen „GlobalOut“-Platz gekennzeichnet).
-

# 4

## Design von PAUL

Die Definition der Semantik für AgentUML in dieser Arbeit erfolgte unter der Zielstellung, ein automatisches Interpretieren von Interaktionsdiagrammen zu ermöglichen. Um zu überprüfen, ob die spezifizierte Semantik diese Zielstellung erfüllt, wurde PAUL 2 (Plug-In for AgentUML-Linking Version 2), ein Werkzeug zur Interpretation von AgentUML-Sequenzdiagrammen, programmiert.

PAUL 2 ist eine Weiterentwicklung des Prototyp-Werkzeuges PAUL (Ehrler 2003). Das Ziel von PAUL 1 war es, auf der Basis einer Syntax-Definition von AgentUML, entsprechende Diagramme für die Agentenplattform Opal (Purvis u. a. 2002a) zu interpretieren. Dabei wurde jedoch festgestellt, daß eine eindeutige Semantikdefinition für diese Art von Interpretation nötig ist.

### 4.1 Zielstellung

Der Sinn eines Interaktionsprotokolles nach Cranefield u. a. (2002) und Greaves u. a. (2000) ist es, dem Agenten zu helfen, die für eine Kommunikation notwendigen Kommunikationsschritte herauszufinden. Dies bedeutet, der Agent soll anhand eines Interaktionsprotokolls herausfinden können, welches die nächsten Möglichkeiten innerhalb der entsprechenden Kommunikation ist.

Ziel der Entwicklung von PAUL war ein Werkzeug mit den folgenden Parametern:

- **Initialisierung:** Initialisierung durch Angabe eines Interaktionsprotokolls in Form eines AgentUML-Sequenzdiagrammes und durch Angabe der Lifeline, die der Agent wahrnehmen möchte.
- **Eingabe:** die Eingabe einer Anfrage beinhaltet die ausgeführten Aktionen bzw. die gesendeten/empfangenen Nachrichten.

- **Ausgabe:** auf Grundlage der Eingabe werden die nächsten möglichen Aktionen berechnet und dem Agenten als Auswahl zur Verfügung gestellt.

Es ist zu beachten, daß PAUL insbesondere nicht die folgenden Funktionen beinhaltet:

- die Interpretation von Bedingungen und Anmerkungen,
- das Treffen von Entscheidungen über den Kommunikationsverlauf, unabhängig davon; ob sie im Protokoll vordefiniert/angedeutet sind.

Diese Aufgaben sind dem Agenten vorbehalten, um eine möglichst hohe Verwendbarkeit von PAUL zu ermöglichen. Andernfalls hätte PAUL für verschiedene Sprachen (wie z.B. OCL (OCL 2003) oder auch natürliche Sprachen) entsprechende Interpreter bereitstellen müssen.

## 4.2 Vorgehensweise

Die Ausführung eines AgentUML-Interaktionsprotokolles mit Hilfe von PAUL läuft nach dem in Abbildung 4.1 dargestellten Schema ab. Die dort aufgeführten Punkte werden im folgenden näher beschrieben

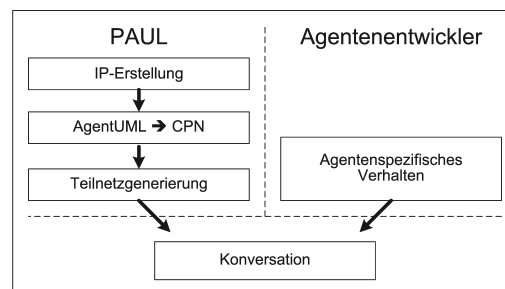


Abbildung 4.1: Vorgehensweise bei einer IP-Ausführung durch PAUL 2

### 4.2.1 Spezifikation von AgentUML-Sequenzdiagrammen

UML, und damit auch AgentUML, ist eine grafische Sprache. Um Vorgänge in dieser Sprache zu beschreiben, werden grafische Editoren benötigt. Für AgentUML existieren jedoch noch keine ausgereiften Werkzeuge, die existierenden Prototypen (Ingenias 2004) waren für die Zwecke dieses Projektes nicht nutzbar.

Um die Möglichkeit zu geben, trotz eines fehlenden grafischen Editors Diagramme spezifizieren zu können, wurde aufbauend auf dem in Abschnitt 3.2 auf Seite 35 entwickelten Metamodell mit Hilfe des Code-Generierungs-Werkzeuges EMF (Eclipse Modeling Framework) (EMF 2001) eine Java-Implementation des Metamodells geschaffen. Mit Hilfe von EMF wurde auch das in Abbildung 4.2 abgebildete Editor-Plug-In für Eclipse (Eclipse 2001) erstellt, mit dem Instanzen des Metamodells eingegeben werden können. So wurde ein Weg geschaffen, ein Interaktionsprotokoll für die computerbasierte Analyse und Verarbeitung zu spezifizieren.

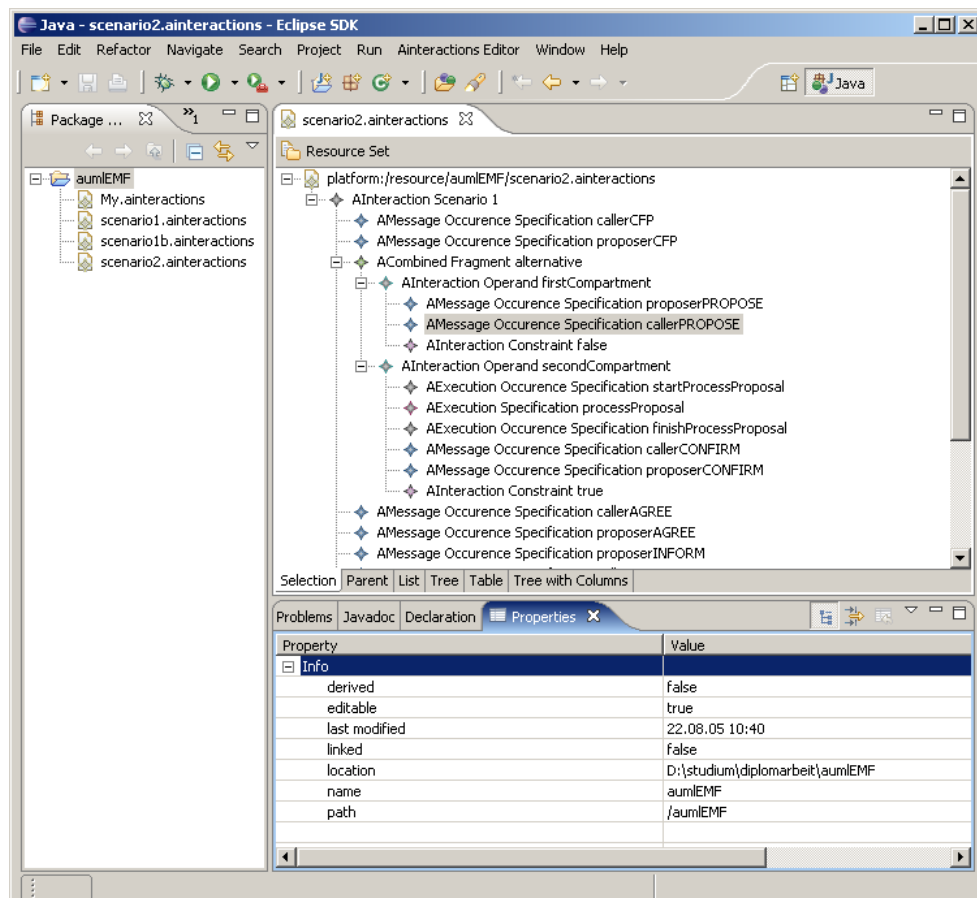


Abbildung 4.2: Editor-Plug-In für AgentUML in Eclipse

## 4.2.2 Übersetzung von AgentUML nach CPN

Um ein Interaktionsdiagramm auf der Grundlage der vorliegenden Semantik zu interpretieren, muß es in ein Farbiges Petri-Netz übersetzt werden. Dies geschieht, indem die Gesamtsemantik aus den verschiedenen Semantiken der Interaktionselemente zusammengesetzt wird. Die OMG schreibt dazu „The semantics of Interactions are compositional in the sense that the semantics of an Interaction is mechanisch aus den Semantiken der built from the semantics of its constituent InteractionFragments.“<sup>1</sup> (UML 2.0 2004, Seite 526).

Daher wird durch die Menge der *InteractionFragment*-Objekte in der festgelegten Reihenfolge <sup>2</sup> iteriert und für jedes Element die Semantik nach der in Anhang B definierten Abbildung bestimmt, die zu einer großen Semantik zusammengesetzt werden. Ein Beispiel solch einer Übertragung ist in Abbildung 3.6 auf Seite 43 zu sehen.

## 4.2.3 Erstellung des entsprechenden Teilnetzes

Da das im vorhergehenden Schritt entstandene CPN alle Aktionen aller Akteure in der Interaktion spezifiziert, muß als nächster Schritt das Teilnetz der *ALifeline* erstellt werden, die der Agent „spielen“ möchte. Dies geschieht, indem alle vom „GlobalIn“-Platz der *ALifeline* erreichbaren Plätze überprüft werden, ob es „Nachrichten-Plätze“ sind. Nicht-Nachrichten-Plätze und alle dazwischen liegenden Transitionen werden dem Teilnetz hinzugefügt; Nachrichten-Plätze werden als Schnittstellen-Plätze aufgefaßt und als solche markiert.

**Beispiel** Abbildung 4.3 enthält ein AgentUML-Sequenzdiagramm. Das entsprechende CPN davon wird in Abbildung 4.4(a) gezeigt. Dieses Netz setzt sich zusammen aus den Teilnetzen in den Abbildungen 4.4(b) und 4.4(c). Dabei sind die Nachrichten-Plätze als Interface gekennzeichnet. Diese Teilnetze sind Grundlage für die Ausführung des Interaktionsprotokolles.

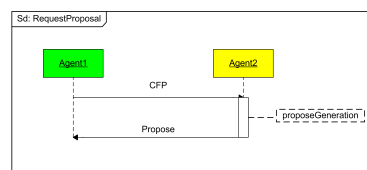


Abbildung 4.3: Beispiel AgentUML-Sequenzdiagramm

## 4.2.4 Agentenspezifisches Verhalten

Jeder Agent verhält sich je nach Bestimmung, Programmierer und „Auftraggeber“ unterschiedlich. Dieses Verhalten wird durch den Agentenprogrammierer implementiert. Es ist nicht Teil des Interaktionsprotokolles. PAUL 2 ist so angelegt. So werden zum Beispiel vom Protokoll nur die Nachrichtentypen festgelegt - der Inhalt der Nachricht wird

<sup>1</sup>Deutsche Übersetzung: Die Semantik von Interaktionen ist kombinatorisch in dem Sinne, daß die Semantik einer Interaktion mechanisch aus den Semantiken ihrer Teil-Interaktionsfragmente zusammengesetzt ist.

<sup>2</sup>Die Reihenfolge wird im Eclipse-Plug-In durch die horizontale Lage der Elemente festgelegt: Je weiter oben ein Element plaziert wird, umso weiter vorn liegt es in der Reihenfolge.

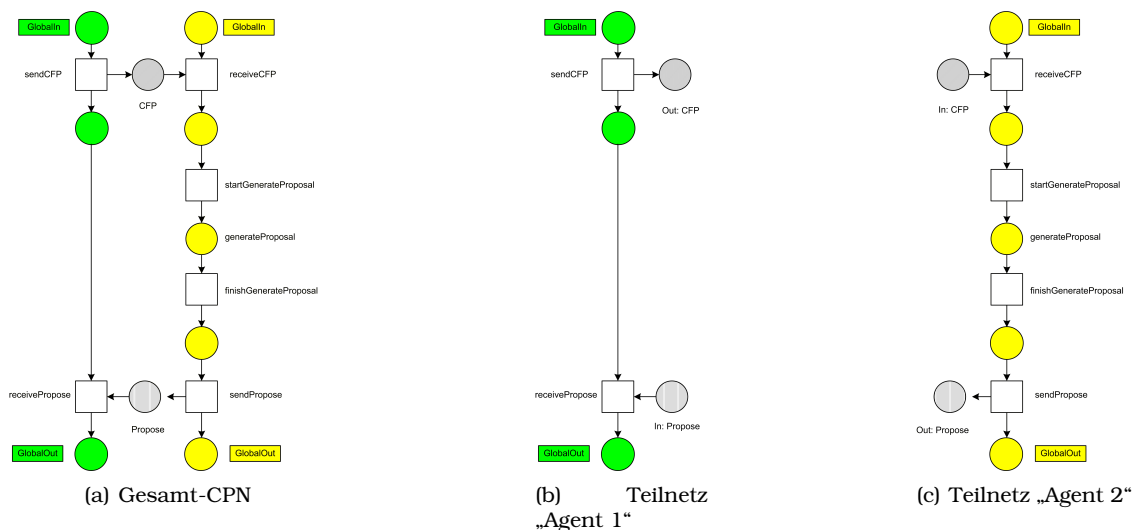


Abbildung 4.4: Umwandlung des Diagramms aus Abbildung 4.3 in ein CPN-Teilnetz

außerhalb des Protokoll von Agenten selbst bestimmt. Auch die Aktivitäten des Agenten müssen außerhalb spezifiziert werden. Bevor ein Agent ein Interaktionsprotokoll mit PAUL 2 ausführen kann, muß der Entwickler sicherstellen, daß jedes im Protokoll vorausgesetzte Verhalten vom Agenten „beherrscht“ wird.

#### 4.2.5 Ausführen einer Konversation

Ein Agent kann mehrere Konversationen desselben Protokolls führen. Die Verwaltung dieser Konversationen muß der Agent selbst übernehmen. Für jede Konversation werden anhand des derzeitigen Standes dieser Konversation die nächsten Möglichkeiten ermittelt. Dazu werden in die dem Kommunikationsstand entsprechenden Plätze Token eingefügt, um dann zu berechnen, welche Transitionen aktiviert sind. Die diesen Transitionen entsprechenden Aktionen werden dem Agenten zur Auswahl gestellt. Dieser entscheidet dann, welche Aktion ausgeführt wird.

### 4.3 Aufbau

PAUL 2 besteht aus den folgenden vier Komponenten (siehe auch Abbildung 4.5):

- **AumlInterpreter:** Das „Hauptinterface“ des Interpreters. Der *AumlInterpreter* liest das AgentUML-Diagramm ein, läßt es vom *SubNetGenerator* in ein Farbiges Petri-Netz übersetzen und das für die angegebene Lifeline nötige Teilnetz erstellen und instanziiert für diese Interaktion mit dieser Lifeline einen *ConversationManager*
- **ConversationManager:** verwaltet die Konversationen einer bestimmten Interaktion mit einer bestimmten Lifeline und erstellt für jede neue Konversation ein neues *Conversation*-Objekt. Sollte ein Agent in einem Interaktionsprotokoll unterschiedliche Lifelines „spielen“, so gibt es für jede Lifeline einen eigenen *ConversationManager*.
- **Conversation:** ist zuständig für die Ausführung einer einzelnen Konversation.

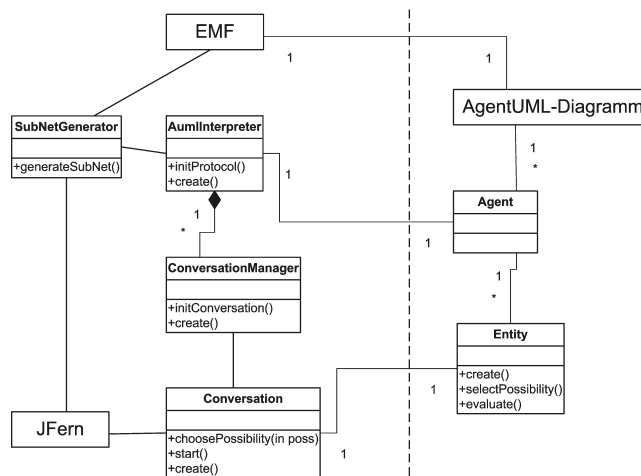


Abbildung 4.5: Struktur von PAUL

- **SubNetGenerator:** Diese Komponente erstellt aus einem AgentUML-Sequenzdiagramm ein Farbiges Petri-Netz und „schneidet“ dann aus diesem Petri-Netz das der gegebenen Lifeline entsprechende Teilnetz aus.

Ein Agenten-Entwickler muß das Interface *Entity* implementieren und an PAUL übergeben. Über diese Klasse wird die Interaktion zwischen PAUL und dem entsprechenden Agenten „abgewickelt“. Sie initiiert eine Konversation und stellt die notwendige Funktionalität zur Ausführung des Protokolls zur Verfügung.

#### 4.3.1 Benutzte Werkzeuge

Für die Entwicklung von PAUL wurden die folgenden Werkzeuge genutzt:

- **Eclipse Modeling Framework:** Das Eclipse Modeling Framework (EMF) (EMF 2001) ist ein java-basiertes Code-Generierungs-Werkzeug für die modellgestützte Programmierung. Aus einem strukturierten Modell (zum Beispiel ein UML-Klassendiagramm) werden entsprechende Klassen und Beziehungen zwischen den Klassen implementiert. Außerdem wird ein Editierungs-Werkzeug sowie Datei-Ein- & Ausgabe-Mechanismen zur Verfügung gestellt. EMF wurde bei PAUL genutzt, um automatisch Klassen für das AgentUML-Metamodell zu entwerfen und darauf zugreifen zu können.
- **Rational Rose:** Das UML-Modellierungswerkzeug „Rational Rose“ (Rational Rose 2005) wurde für die Entwicklung des AgentUML-Metamodell genutzt, um dieses dann in EMF weiterverwenden zu können.
- **JFern:** JFern (Nowostawski 2000) ist ein java-basiertes CPN-Werkzeug. Es wird von PAUL zur Erstellung und zum Ausführen der Farbigen Petri-Netze genutzt.

## 4.4 Funktionsweise

In diesem Abschnitt wird die Funktions- und Verwendungsweise von PAUL beschrieben.

### 4.4.1 Initialisierung eines Protokolls

Abbildung 4.6 zeigt in einem UML-Sequenzdiagramm die Initialisierung eines Interaktionsprotokolls. Der *Agent* (hier „testAgent“ genannt) übergibt der vorhandenen oder

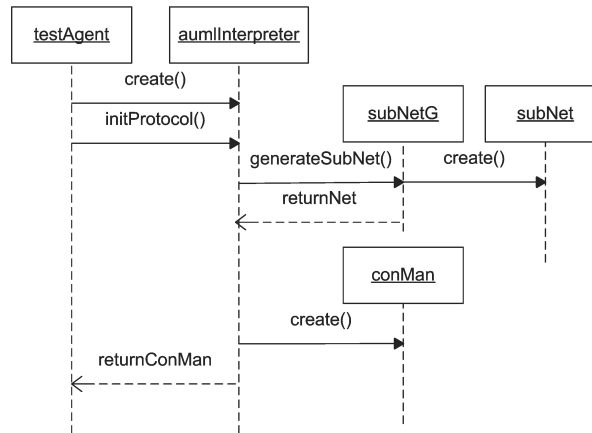


Abbildung 4.6: Initialisierung eines Interaktionsprotokolls

neu geschaffenen Instanz des *AumlInterpreters* das Protokoll. Mit Hilfe des *SubNetGenerators* wird dieses Protokoll erst in ein CPN übersetzt, um dann das entsprechende Teilnetz „auszuschneiden“ (siehe Abschnitt 4.2.3 auf Seite 48). Für das so gewonnene Teilnetz wird eine Instanz des *ConversationManagers* erstellt und dem *Agenten* übergeben.

### 4.4.2 Initialisierung einer Konversation

Nachdem das Protokoll initialisiert wurde, muß eine Konversation mit diesem Protokoll gestartet werden. Dazu wird, wie in Abbildung 4.7 zu sehen ist, vom Agenten eine

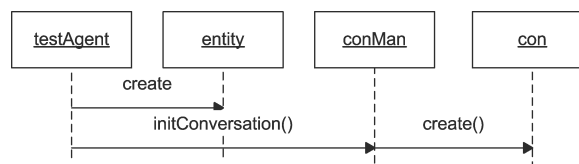


Abbildung 4.7: Initialisierung einer Konversation

Instanz von *Entity* erschaffen werden. Ob diese Instanz nur eine oder auch mehrere Konversationen gleichzeitig vollziehen kann und ob die entsprechende Klasse noch andere Funktionalität besitzt (oder der Agent selbst ist), wird von PAUL nicht festgelegt - jedes Objekt, daß das Interface *Entity* implementiert, kann übergeben werden. Mit diesem *Entity*-Objekt als Übergabeparameter wird am *ConversationManager* eine neue Konversation gestartet. Der *ConversationManager* erstellt für jede Konversation eine neue Instanz der Klasse *Conversation* und übergibt das *Entity*-Objekt diesem *Conversation*-Objekt. Die gesamte Kommunikation zwischen PAUL und dem Agenten für diese Konversation wird ab nun zwischen diesen beiden Objekten vollführt.

### 4.4.3 Ausführung einer Konversation

Durch die Instanziierung der Klasse *Conversation* (im vorigen Schritt geschehen) wird die Konversation gestartet. Abbildung 4.8 zeigt, daß das *Conversation*-Objekt das Teilnetz (im ersten Schritt erschaffen) auf die aktivierten Transitionen untersucht und diese dann der *Entity*-Instanz als nächste Möglichkeiten anbietet. Das *Entity*-Objekt muß

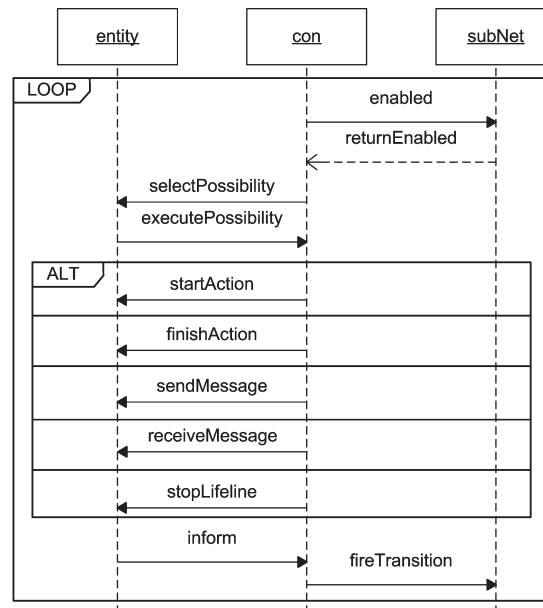


Abbildung 4.8: Vollzug der Konversation

davon mindestens eins wählen. Es kann dann mehr als eins wählen, wenn im Interaktionsprotokoll verschiedene Aktionsmöglichkeiten des Kommunikationspartners angegeben sind. So kann es zum Beispiel sein, daß es mehrere unterschiedliche Nachrichten gibt, die empfangen werden können.

Nach der Wahl der Möglichkeiten wird nun von dem *Conversation*-Objekt an die *Entity*-Instanz durch Aufruf der entsprechenden Methoden angezeigt, welche Aktivitäten der Agent durchzuführen hat. Damit wird gleichzeitig auch angezeigt, das auf den Vollzug einer dieser Aktivitäten gewartet wird. Die Ausführung einer der Möglichkeiten wird dann durch Aufruf der „*inform*“-Methode auf dem *Conversation*-Objekt angezeigt (als Parameter wird die Identifizierung der gewählten Möglichkeit angegeben). Die der ausgeführten Möglichkeit entsprechende Transition wird dann vom *Conversation*-Objekt im Teilnetz „gefeuert“, um daraufhin wieder zu überprüfen, welche Transitionen nun aktiviert sind. Dies geschieht solange, bis die Kommunikation beendet wird.

# 5

## Fazit und Ausblick

Aufgabe und Ziel der vorliegenden Diplomarbeit war es, für die grafische Sprache Agent-UML (FIPA AUML 2003) eine formale Semantik zu spezifizieren, sowie mit Hilfe dieser Semantik eine automatische Interpretation von Interaktionsprotokollen prototypisch zu implementieren. Um dieses Ziel zu erreichen, wurde eine Evaluation verschiedener Formalismen vorgenommen; eine formale Semantik mit Hilfe von Farbigen Petri-Netzen (Jensen 1992) erstellt sowie das Werkzeug PAUL geschaffen.

### 5.1 Evaluation möglicher Formalismen

Es gibt sehr viele Sprachen und Formalismen, mit denen eine formale Semantik erstellt werden kann, so daß es kaum realisierbar ist, alle Möglichkeiten zur Semantikspezifikation im Rahmen einer Diplomarbeit zu evaluieren. Für eine ausführlichere Evaluation sei unter anderem auf die Arbeiten von Clarke u. Wing (1996) und Zhang u. Xu (2004) verwiesen.

In die engere Auswahl für diese Arbeit fielen:

- **Communicating Sequential Processes (CSP):** Communicating Sequential Processes wurde von Hoare (1985) entwickelt, um konkurrierende, miteinander kommunizierende Systeme zu modellieren. Es gibt für diesen Formalismus sowohl eine Java-Implementierung (University of Kent 1997), als auch Arbeiten, die UML in CSP übersetzen (Davies u. Crichton 2003). Leider konnte keine Arbeit gefunden werden, in der CSP im Agentenbereich genutzt wurde.
- **Object Constraint Language (OCL):** Die zur UML gehörende Sprache OCL (OCL 2003) besitzt ein Metamodell und ist schon vielfach im UML-Bereich eingesetzt worden. Mit xOCL (Clark u. a. 2004) gibt es auch eine ausführbare Erweiterung. Auf-

grund seiner Geschichte ist OCL nicht so mächtig wie die meisten anderen hier vorgestellten Formalismen, so daß nicht klar ist, ob sich die komplette Semantik mit OCL spezifizieren läßt.

- **Modal- und Beschreibungslogiken:** Logik-basierte Formalismen werden in der Künstlichen Intelligenz sehr häufig benutzt, und speziell Modallogik ist in Agentensystemen weit verbreitet. Natürlicherweise haben sie auch eine starke formale Grundlage und sind sehr mächtig. Leider sind diese Formalismen meist nur einer kleinen Gruppe von Spezialisten bekannt und für Softwareentwickler zu komplex und abstrakt.
- **Web Ontology Language (OWL):** Die Web Ontology Language ist formale Sprache des Semantic Web. Anders als viele formale Sprachen ist sie mittlerweile bekannt und zu einem anerkannten Standard geworden. Sie bietet jedoch vor allem Sprachkonstrukte zur Beschreibung von statischen Strukturen und ist daher zur Spezifikation von dynamischem Verhalten ungeeignet.
- **(Coloured) Petri Netze (CPN):** Petri Netze (Petri 1962) sind ein weit verbreiteter und gut untersuchter Formalismus. Sie sind aufgrund ihrer grafischen Repräsentation leicht verständlich, und in Form ihrer Erweiterung *Coloured Petri Nets* auch modular und hierarchisch. Sie wurden auch schon im Agentenbereich genutzt.

Für die Semantik von AgentUML wurde eine Entscheidung zugunsten der Benutzung von Farbigen Petri-Netzen getroffen, da:

- Es eine breite Anzahl von Arbeiten im Bereich der Interaktionsprotokollmodellierung mit CPN gibt.
- Es gut verfügbare Werkzeuge (z.B. JFern (Nowostawski 2000)) für CPN gibt.
- Die Notation intuitiv und leicht verständlich ist.
- CPN ausreichend mächtig sind.

## 5.2 Evaluation der AgentUML-Semantik

Aufbauend auf den Vorarbeiten von Mazouzi u. a. (2002) und Ehrler (2003) wurde ein Metamodell und eine formale Semantik für AgentUML entwickelt. Diese Spezifikation umfaßt einen großen Teil der von der FIPA definierten Sprachelemente (FIPA AUML 2003). Nicht inbegriffen sind folgende Elemente:

- **Blockierende Bedingungen:** In der Spezifikation von AgentUML (FIPA AUML 2003, Abschnitt 2.4) werden blockierende Bedingungen folgendermaßen definiert: „Blocking constraints imply that the interaction is blocked as long as the constraints are not satisfied“<sup>1</sup>.(FIPA AUML 2003, Abschnitt 2.4). Blockierende Bedingungen wurden in die Spezifikation aufgrund ihrer sehr komplexen Umsetzung nicht mit aufgenommen. Sie beziehen sich häufig auf Sachverhalte, die außerhalb des jeweiligen Protokolls

<sup>1</sup>Deutsche Übersetzung: Blockierende Bedingungen implizieren, daß die Interaktion solange blockiert ist wie die Bedingungen nicht erfüllt sind.

bzw. Agenten sind (z.B. die Anzahl der Agenten in der Kommunikation) und deshalb ist die Erfüllung schwer automatisch zu überprüfen.

- **Rollenkardinalität:** Laut FIPA AUML (2003) kann man an Lifelines eine Kardinalität der Rollen angeben. Dies wurde aufgrund der damit verbundenen Komplexität innerhalb dieser Arbeit nicht übernommen.
- **Nachrichten:** Die Spezifikation von Nachrichten wurde geändert, da diese in FIPA AUML (2003) teilweise einfach aus der UML 2.0-Spezifikation übernommen wurde und teilweise zu kompliziert ist, um sie im Rahmen dieser Arbeit umzusetzen.
- **AInteractionOperator:** Die Operatoren wurden nach Absprache mit Marc-Phillipe Huguet, dem Hauptautor der FIPA-Spezifikation, reduziert, da einige Operatoren im Agentenkontext keine Verwendung finden.

Es wurde keine Semantik für die Elemente

- AInteractionOperand:WeakSequencing,
- AInteractionOperand:StrictSequencing,
- AInteractionoperand:CriticalRegion,
- ATimingConstraint,

erstellt. Dies hätte jeweils eine Ausweitung der Gültigkeit der Semantik auf alle Interaktionen des Agenten bedeutet - oder aber die Benutzung von CPN mit Zeit.

Die vorliegende Syntax und Semantikspezifikation umfaßt den weitaus größten Teil des Sprachumfangs für AgentUML-Sequenzdiagramme. Ein Großteil der in FIPA AUML (2003, Abschnitt 2.14) genannten FIPA-Protokolle können mit dieser Semantik beschrieben werden. Sie ermöglicht es außerdem, Interaktionsprotokolle zu verifizieren und auf Korrektheit zu überprüfen.

## 5.3 Evaluation von PAUL

Das „Plug-In for AgentUML-Linking“ (PAUL) stellt ein Werkzeug zur automatischen Interpretation von AgentUML-Sequenzdiagrammen dar. Es übersetzt ein AgentUML-Diagramm in ein Farbiges Petri-Netz und nutzt dieses Netz sowie den CPN-Simulator JFern (Nowostawski 2000), um das Interaktionsprotokoll auszuführen. Dabei untersucht es das CPN nach Eingabe des derzeitigen Standes einer Konversation auf die nächstmöglichen Handlungsmöglichkeiten des Agenten. Der Agent muß danach selbst wählen, welche dieser Möglichkeiten er ausführen möchte.

PAUL ist unabhängig von einem speziellen Agentensystem und kann von jedem java-basierten Agentensystem genutzt werden. Die Sprache, in der Bedingungen in einem Diagramm geschrieben werden, ist ebenfalls frei wählbar, da die Auswertung der Bedingungen von dem Agenten durchgeführt werden muß. Durch die Nutzung von EMF (EMF 2001) kann PAUL mit relativ wenig Aufwand an ein verändertes AgentUML-Metamodell angepaßt werden.

---

## 5.4 Ausblick

Die vorliegende Arbeit beinhaltet einen Vorschlag an die FIPA zur Spezifikation von AgentUML. Die FIPA muß diesen Vorschlag nun überarbeiten und weiter standardisieren. Dabei ist es insbesondere wichtig, die Kardinalität von Lifelines, blockierende sowie Zeit-Bedingungen zu spezifizieren.

Desweiteren ist die Entwicklung eines grafischen Editors für Interaktionsprotokolle notwendig, um die Vorteile einer grafischen Sprache wirklich vollständig auszunutzen.

Aufbauend auf die vorgestellte formale Semantik können nun Werkzeuge entwickelt werden, die ein gegebenes Interaktionsprotokoll analysieren und vor allem verifizieren. Dies ermöglicht, ein Interaktionsprotokoll auf Korrektheit zu überprüfen und könnte Protokollentwickler bei ihrer Arbeit unterstützen.

Das Ziel der weiteren Forschung ist, eine Interpretation von Interaktionsprotokollen ohne die vorherige Kenntnis dieser Protokolle zu ermöglichen. Dazu ist es notwendig, Standards für die semantische Beschreibung der Aktionen innerhalb der Protokolle zu entwickeln, so daß ein Agent ein Protokoll untersuchen und herausfinden kann, ob er die notwendigen Fähigkeiten besitzt, dieses Protokoll auszuführen.

---

# Literaturverzeichnis

## **Adamzik 2001**

ADAMZIK, Kirsten: *Sprache: Wege zum Verstehen*. Tübingen, Basel : A.-Francke-Verlag, 2001. – ISBN 3-7720-2279-0

## **Antoniou u. van Harmelen 2004**

ANTONIOU, Grigoris ; HARMELEN, Frank van: *A Semantic Web Primer*. Cambridge, MA, USA : MIT Press, 2004. – ISBN 0-262-01210-3

## **Arazy u. Woo 2002**

ARAZY, Ofer ; WOO, Carson C.: Analysis and design of agent-oriented information systems. In: *Knowledge Engineering Review* 17 (2002), Nr. 3, S. 215–260

## **Baader u. a. 2003**

BAADER, Franz ; CALVANESE, Diego ; MCGUINNESS, Deborah ; NARDI, Daniele ; PATEL-SCHNEIDER, Peter: *The Description Logic Handbook*. Cambridge, UK : Cambridge University Press, 2003. – ISBN 0521781760

## **Baar 2003**

BAAR, Thomas: *Über die Semantikbeschreibung OCL-artiger Sprachen*. Berlin, Germany : Logos-Verlag, 2003. – ISBN 3-8325-0433-8

## **Backus 1959**

BACKUS, John W.: The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference. In: *Proceedings of the International Conference on Information Processing*, UNESCO, 1959, S. 125–132

## **Bagic 2004**

BAGIC, Marina: Formal Infrastructure for Modelling Intelligent Agents with Agent UML and Petri Nets. In: MEERSMAN, Robert (Hrsg.) ; TARI, Zahir (Hrsg.) ; CORSARO, Angelo (Hrsg.): *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops: OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, October 25-29, 2004. Proceedings*. Heidelberg, Germany : Springer-Verlag, 2004 (Lecture Notes in Computer Science 3292), S. 842–853. – ISBN 3-540-23664-3

## **Baldoni u. a. 2005**

BALDONI, M. ; BAROGLIO, C. ; GUNGUI, I. ; MARTELLI, A. ; MARTELLI, M. ; MASCARDI, V. ; PATTI, V. ; SCHIFANELLA, C.: Reasoning About Agents' Interaction Protocols Inside DCaseLP. In: LEITE, J. (Hrsg.) ; OMICINI, A. (Hrsg.) ; TORRONI, P. (Hrsg.) ; YOLUM, P. (Hrsg.): *Post-Proceedings of the International Workshop on Declarative Agent Languages and Technologies, DALT'04*. Heidelberg, Germany : Springer-Verlag, 2005 (Lecture Notes in Artificial Intelligence 3476). – ISBN 3-540-00713-X, S. 112–131

## **Baldoni u. a. 2004**

BALDONI, Matteo ; MARTELLI, Alberto ; PATTI, Viviana ; GIORDANO, Laura: Programming Rational Agents in a Modal Action Logic. In: *Annals of Mathematics and Artificial Intelligence* 41 (2004), Nr. 2–4, S. 207–257

## **Bauer u. a. 2001a**

BAUER, Bernhard ; BERGENTI, Federico ; MASSONET, Philippe ; ODELL, James J.:

Agents and the UML: A Unified Notation for Agents and Multi-agent Systems? In: (Wooldridge u. a. 2001), S. 148–150

**Bauer u. a. 2001b**

BAUER, Bernhard ; MÜLLER, Joerg P. ; ODELL, James J.: Agent UML: A Formalism for Specifying Multiagent Interaction. In: (Ciancarini u. Wooldridge 2001), S. 91–103. – ISBN 3–540–41594–7

**Bellifemine u. a. 1999**

BELLIFEMINE, Fabio ; POGGI, Agostino ; RIMASSA, Giovanni: JADE - A FIPA-compliant agent framework. In: *Proceedings of the 4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*. London, UK : The Practical Application Company Ltd, 1999, S. 97–108

**Berghammer 2001**

BERGHAMMER, Rudolf: *Semantik von Programmiersprachen*. Berlin, Germany : Logos-Verlag, 2001. – ISBN 3–89722–489–5

**Bernardi u. a. 2002**

BERNARDI, Simona ; DONATELLI, Susanna ; MERSEGUER, José: From UML sequence diagrams and statecharts to analysable petri net models. In: *WOSP '02: Proceedings of the 3rd international workshop on Software and performance*. New York, NY, USA : ACM Press, 2002. – ISBN 1–58113–563–7, S. 35–45

**Berners-Lee u. a. 2001**

BERNERS-LEE, Tim ; HENDLER, James ; LASSILA, Ora: *The Semantic Web*. Version: May 2001. <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>. – Online-Ressource, Abruf: 03.11.2005

**Best 1995**

BEST, Eike: *Semantik - Theorie sequentieller und paralleler Programmierung*. Braunschweig, Germany : vieweg-Verlag, 1995. – ISBN 3–528–05431–X

**Best u. a. 2001**

BEST, Eike ; DEVILLERS, Raymond ; KOUTNY, Maciej: *Petri Net Algebra*. Heidelberg, Germany : Springer-Verlag, 2001 (Monographs in Theoretical Computer Science). – ISBN 3–540–67398–9

**Bjørner u. Jones 1978**

BJØRNER, D. ; JONES, C.B.: *The Vienna Development Method: the meta-language*. Heidelberg, Germany : Springer-Verlag, 1978 (Lecture Notes in Computer Science 61)

**Blass 1992**

BLASS, A.: A game semantics for linear logic. In: *Annals of Pure and Applied Logic* 56 (1992), Nr. 1–3, S. 183–220

**Booch 1993**

BOOCH, Grady: *Object-oriented Analysis and Design with Applications*. 2. Redwood City, CA, USA : Benjamin Cummings, 1993. – ISBN 0–8053–5340–2

**Born u. a. 2004**

BORN, Marc ; HOLZ, Eckhardt ; KATH, Olaf: *Softwareentwicklung mit UML 2*. München, Germany : Addison Wesley Verlag, 2004. – ISBN 3–8273–2086–0

**Bradshaw 1997a**

BRADSHAW, Jeffrey M.: An Introduction to Software Agents. In: *Software Agents* (Bradshaw 1997b), S. 3–46. – ISBN 0–262–52234–9

**Bradshaw 1997b**

BRADSHAW, Jeffrey M. (Hrsg.): *Software Agents*. Cambridge, MA, USA : MIT Press, 1997. – ISBN 0-262-52234-9

**Brenner u. a. 1998**

BRENNER, Walter ; ZARNEKOV, Rüdiger ; WITTIG, Hartmut: *Intelligente Softwareagenten - Grundlagen und Anwendungen*. Heidelberg, Germany : Springer-Verlag, 1998. – ISBN 3-540-63431-2

**Bresciani u. a. 2004**

BRESCIANI, Paolo ; PERINI, Anna ; GIORGINI, Paolo ; GIUNCHIGLIA, Fausto ; MYLOPOULOS, John: Tropos: An Agent-Oriented Software Development Methodology. In: *Autonomous Agents and Multi-Agent Systems* 8 (2004), Nr. 3, S. 203-236

**Cabac u. Moldt 2005**

CABAC, Lawrence ; MOLDT, Daniel: Formal Semantics for AUML Agent Interaction Protocol Diagrams. In: (**Odell u. a. 2005**), S. 47-61. – ISBN 3-540-24286-4

**Chaib-draa u. Dignum 2002**

CHAIB-DRAA, Brahim ; DIGNUM, Frank: Trends in Agent Communication Language. In: *Computational Intelligence* 18 (2002), Nr. 2, S. 89-101

**Chomsky 1956**

CHOMSKY, Noam: Three models for the description of languages. In: *IRE Transactions on Information Theory* 2 (1956), Nr. 3, S. 113-124

**Chomsky 1959**

CHOMSKY, Noam: On certain formal properties of grammars. In: *Information and Control* 2 (1959), Nr. 2, S. 137-167

**Ciancarini u. Wooldridge 2001**

CIANCARINI, Paolo (Hrsg.) ; WOOLDRIDGE, Michael (Hrsg.): *Agent-Oriented Software Engineering, First International Workshop, AOSE 2000, Limerick, Ireland, June 10, 2000, Revised Papers*. Heidelberg, Germany : Springer-Verlag, 2001 (Lecture Notes in Computer Science 1957). – ISBN 3-540-41594-7

**Clark u. a. 2004**

CLARK, Tony ; EVANS, Andy ; SAMMUT, Paul ; WILLANS, James: *Applied Metamodelling - A Foundation for Language Driven Development*. Version: 2004. <http://alбини.xactium.com>. – Online-Ressource, Abruf: 28.04.05

**Clarke u. Wing 1996**

CLARKE, Edmund M. ; WING, Jeannette M.: Formal Methods: State of the Art and Future Directions. In: *ACM Computing Surveys* 28 (1996), Nr. 4, S. 626-643

**Cost u. a. 2000**

COST, R. ; CHEN, Y. ; FININ, T. ; LABROU, Y. ; PENG, Y.: Using Colored Petri Nets for Conversation modeling. In: (**Dignum u. Greaves 2000b**), S. 178-192

**Cranefield u. Purvis 1999**

CRANEFIELD, Stephen ; PURVIS, Martin: UML as an ontology modelling language. In: *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*. Tilburg : CEUR Publications, 1999

**Cranefield u. a. 2002**

CRANEFIELD, Stephen ; PURVIS, Martin K. ; NOWOSTAWSKI, Mariusz ; HWANG, Peter: Ontologies for Interaction Protocols. Version: 2002. <http://CEUR-WS.org/Vol-66/oas02-16.pdf>. In: *Proceedings of the Workshop on Ontologies in Agent Systems, 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy*. CEUR Publications. – Online-Ressource, Abruf: 01.08.05

**Davies u. Crichton 2003**

DAVIES, Jim ; CRICHTON, Charles: Concurrency and Refinement in the Unified Modeling Language. In: *Formal Aspects of Computing* 15 (2003), Nr. 2-3, S. 118–145

**Dignum u. Greaves 2000a**

DIGNUM, Frank ; GREAVES, Mark: Introduction. In: *Issues in Agent Communication*(**Dignum u. Greaves 2000b**), S. 1–16

**Dignum u. Greaves 2000b**

DIGNUM, Frank (Hrsg.) ; GREAVES, Mark (Hrsg.): *Issues in Agent Communication*. Heidelberg, Germany : Springer-Verlag, 2000 (Lecture Notes in Computer Science 1916)

**Dijkstra 1975**

DIJKSTRA, Edsger W.: Guarded commands, nondeterminacy and formal derivation of programs. In: *Communications of the ACM* 18 (1975), Nr. 8, S. 453–457

**Donini u. a. 1996**

DONINI, Francesco M. ; LENZERINI, Maurizio ; NARDI, Daniele ; SCHAEFER, Andrea: Reasoning in Description Logics. In: BREWKA, Gerhard (Hrsg.): *Foundation of Knowledge Representation*. Stanford, CA, USA : CSLI Publications, 1996

**Eclipse 2001**

*Eclipse*. Version: 2001–2005. <http://www.eclipse.org/>. – Online-Ressource, Abruf: 10.10.05

**Ehrler 2003**

EHRLER, Lars: *Declarative graphical descriptions of Interaction Protocols for Multi-agent Systems*, Department of Information Science, University of Otago, Dissertation for a Postgraduate Diploma of Arts, 2003. <http://www.larsehrler.de/veroeff/2003Paul.pdf>. – Online-Ressource, Abruf: 29.07.2005

**Ehrler u. Cranefield 2004**

EHRLER, Lars ; CRANEFIELD, Stephen: Executing Agent UML diagrams. Version: 2004. [http://www.aamas2004.org/proceedings/110\\_Ehrler1\\_executingAuml.pdf](http://www.aamas2004.org/proceedings/110_Ehrler1_executingAuml.pdf). In: JENNINGS, Nicholas R. (Hrsg.) ; SIERRA, Carles (Hrsg.) ; SONNENBERG, Liz (Hrsg.) ; TAMBE, Milind (Hrsg.): *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. IEEE Computer Society Press, 906–913. – Online-Ressource, Abruf: 29.07.2005. – ISBN 1-58113-864-4

**Ehrler u. a. 2005**

EHRLER, Lars ; FLEURKE, Martin ; PURVIS, Maryam ; SAVARIMUTHU, Bastin Tony R.: Agent-based Workflow Management Systems (WfMSs) - JBees – a distributed and adaptive WfMS with monitoring and controlling capabilities. In: *Journal of Information Systems and E-Business Management* to appear (2005)

**Ehrler u. a. 2003**

EHRLER, Lars ; FLEURKE, Martin ; SAVARIMUTHU, Bastin Tony R.: *JBees - an adaptive workflow management system*. Version: 2003–2005. [http://waitaki.otago.ac.nz/~tonyr/jbees\\_download/](http://waitaki.otago.ac.nz/~tonyr/jbees_download/). – Online-Ressource, Abruf: 29.07.2005

**El-Kady 2004**

EL-KADY, Manar: *Personal Communication*. 2004

**EMF 2001**

*Eclipse Modeling Framework*. Version: 2001–2005. <http://www.eclipse.org/emf>. – Online-Ressource, Abruf: 10.10.05

---

**Engels u. Heckel 2000**

ENGELS, Gregor ; HECKEL, Reiko: From Trees to Graphs: Defining the Semantics of Diagram Languages with Graph Transformation. In: ROLIM, José D. P. (Hrsg.) ; BRODER, Andrei Z. (Hrsg.) ; CORRADINI, Andrea (Hrsg.) ; GORRIERI, Roberto (Hrsg.) ; HECKEL, Reiko (Hrsg.) ; HROMKOVIC, Juraj (Hrsg.) ; VACCARO, Ugo (Hrsg.) ; WELLS, J. B. (Hrsg.): *Proc. ICALP'2000 Satellite Workshops*. Waterloo, Ontario, Canada : Carleton Scientific, 2000. – ISBN 1-894145-07-0, S. 373-382

**Finin u. a. 1997**

FININ, Tim ; LABROU, Yanis ; MAYFIELD, James: KQML as an agent communication language. In: (**Bradshaw 1997b**), S. 291-316. – ISBN 0-262-52234-9

**FIPA-ACL 2002**

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (Hrsg.): *FIPA Communicative Act Library Specification*. Version: 2002. <http://www.fipa.org/specs/fipa00037/>. – Online-Ressource, Abruf: 13.04.2005

**FIPA AUML 2003**

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (Hrsg.): *FIPA Modeling: Interaction Diagrams, Version 2003-07-02 – Working Draft*. Version: 2003. <http://www.auml.org/auml/documents/ID-03-07-02.pdf>. – Online-Ressource, Abruf: 16.04.2005

**FIPA AUML Class diagrams 2004**

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (Hrsg.): *FIPA Modeling TC: Agent Class Superstructure Metamodel, Version 2004-01-21 – Working Draft*. Version: 2004. <http://www.auml.org/auml/documents/CD2-04-01-21.doc>. – Online-Ressource, Abruf: 05.07.2005

**FIPA AUML workplan 2003**

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (Hrsg.): *AUML workplan*. Version: 2003. <http://www.fipa.org/docs/input/f-in-00085/f-in-00085.htm>. – Online-Ressource, Abruf: 16.04.2005

**FIPA IP Library 2000**

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS (Hrsg.): *FIPA Interaction Protocol Library Specification - deprecated*. Version: 2000. <http://www.fipa.org/specs/fipa00025/>. – Online-Ressource, Abruf: 16.04.2005

**Flake 2004**

FLAKE, Stephan: Towards the completion of the formal semantics of OCL 2.0. In: *CRPIT '26: Proceedings of the 27th conference on Australasian computer science*. Darlinghurst, Australia : Australian Computer Society, Inc., 2004, S. 73-82

**Franklin u. Graesser 1997**

FRANKLIN, Stan ; GRAESSER, Art: Is it an Agent or just a Program ? A Taxonomy for Autonomous Agents. In: MÜLLER, Jörg P. (Hrsg.) ; WOOLDRIDGE, Michael J. (Hrsg.) ; JENNINGS, Nicholas R. (Hrsg.): *Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Budapest, Hungary, 1996*. Heidelberg, Germany : Springer-Verlag, 1997 (Lecture Notes in Artificial Intelligence 1193). – ISBN 3-540-62507-0, S. 21-36

**Geihs u. a. 2001**

GEIHS, Kurt ; JAHR, Ulf ; ZAPF, Michael: Kommunikation zwischen autonomen Agenten. In: *Praxis der Informationsverarbeitung und Kommunikation* 24 (2001), Nr. 3, S. 127-135

**Genrich u. Lautenbach 1981**

GENRICH, Hartmann J. ; LAUTENBACH, Kurt: System Modelling with High-Level Petri Nets. In: *Theoretical Computer Science* 13 (1981), S. 109-136

---

**Geroimenko 2004**

GEROIMENKO, Vladimir: *Dictionary of XML Technologies and the Semantic Web*. Heidelberg, Germany : Springer-Verlag, 2004. – ISBN 1-85233-768-0

**Greaves u. a. 2000**

GREAVES, M. ; HOLMBACK, H. ; BRADSHAW, J.: What is a conversation policy? In: **(Dignum u. Greaves 2000b)**, S. 118–131

**Greibach 1965**

GREIBACH, S.A.: A new normal form theorem for context-free phrase structure grammars. In: *Journal of the ACM* 12 (1965), Nr. 1, S. 42–52

**Gurevich 1991**

GUREVICH, Yuri: Evolving Algebras: An Attempt to Discover Semantics. In: *Bulletin of the European Association for Theoretical Computer Science (EATCS)* 43 (1991), S. 264–284

**Guru 2005**

GURU: *Personal Communication*. 2005

**Gutnik u. Kaminka 2005**

GUTNIK, Gery ; KAMINKA, Gal A.: A Scalable Petri Net Representation of Interaction Protocols for Overhearing. In: EIJK, Rogier M. (Hrsg.) ; HUGET, Marc-Philippe (Hrsg.) ; DIGNUM, Frank (Hrsg.): *Agent Communication: International Workshop on Agent Communication, AC 2004, New York, NY, USA, July 19, 2004, Revised Selected and Invited Papers*. Heidelberg, Germany : Springer-Verlag, 2005 (Lecture Notes in Computer Science 3396). – ISBN 3-540-25015-8, S. 50–64

**Habermas 1981**

HABERMAS, Jürgen: *Theorie des kommunikativen Handelns*. Bd. 1+2. Frankfurt am Main, Germany : Suhrkamp Verlag, 1981. – ISBN 3-518-28775-3

**Hausmann u. a. 2004**

HAUSMANN, Jan H. ; HECKEL, Reiko ; SAUER, Stefan: Dynamic Meta Modeling with time: Specifying the semantics of multimedia sequence diagrams. In: *Software and Systems Modeling* 3 (2004), Nr. 3, S. 181 – 193

**Hausser 2000**

HAUSSER, Roland: *Grundlagen der Computerlinguistik – Mensch-Maschine-Kommunikation in natürlicher Sprache*. Heidelberg, Germany : Springer-Verlag, 2000. – ISBN 3-540-67187-0

**Henderson-Sellers u. Giorgini 2005**

HENDERSON-SELLERS, Brian ; GIORGINI, Paolo: *Agent-Oriented Methodologies*. Hershey, PA, USA : Idea Group Publishing, 2005. – ISBN 1591405866

**Hindriks u. a. 1997**

HINDRIKS, Koen V. ; BOER, Frank S. ; HOEK, Wiebe van d. ; MEYER, John-Jules C.: Formal Semantics for an Abstract Agent Programming Language. In: **(Singh u. a. 1997)**, S. 215–229. – ISBN 3-540-64162-9

**Hoare 1969**

HOARE, Charles Antony R.: A Sampler of Formal Definitions. In: *Communications of the ACM* 12 (1969), Nr. 10, S. 576–580,583

**Hoare 1985**

HOARE, Charles Antony R.: *Communicating Sequential Processes*. Version: 1985. <http://www.usingcsp.com/cspbook.pdf>. – Online-Ressource, Abruf: 28.04.05. ISBN 0131532715

---

**Horrocks u. Sattler 2001**

HORROCKS, Ian ; SATTLER, Ulrike: Ontology Reasoning in the Description Logic. In: NEBEL, B. (Hrsg.): *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*. San Francisco, CA, USA : Morgan Kaufmann Publishers, 2001. – ISBN 1-55860-777-3, S. 199-204

**HTTP 1999**

WORLD WIDE WEB CONSORTIUM (W3C) (Hrsg.): *Hypertext Transfer Protocol 1.1*. Version: 1999. <http://www.w3.org/Protocols/>. – Online-Ressource, Abruf: 03.11.2005

**Huget 2004**

HUGET, Marc-Philippe: Agent UML Notation for Multiagent System Design. In: *IEEE Internet Computing* 8 (2004), Nr. 4, S. 63-71

**Huget 2005**

HUGET, Marc-Philippe: Modeling Languages for Multiagent Systems. Version: 2005. <http://www.agentgroup.unimore.it/aose05/papers/49.pdf>. In: MÜLLER, Jörg P. (Hrsg.) ; ZAMBONELLI, Franco (Hrsg.): *Proceedings of the Workshop on Agent-Oriented Software Engineering, 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems, Utrecht, The Netherlands*. Springer-Verlag. – Online-Ressource, Abruf: 01.08.05. – to appear as volume of Lecture Notes in Computer Science

**Huget u. Odell 2005**

HUGET, Marc-Philippe ; ODELL, James: Representing Agent Interaction Protocols with AgentUML. In: (**Odell u. a. 2005**), S. 16-31. – ISBN 3-540-24286-4

**Huget u. a. 2004**

HUGET, Marc-Philippe ; ODELL, James ; BAUER, Bernhard: The AUML Approach. In: BERGENTI, Federico (Hrsg.) ; GLEIZES, Marie-Pierre (Hrsg.) ; ZAMBONELLI, Franco (Hrsg.): *Methodologies and Software Engineering for Agent Systems*. Dordrecht, The Netherlands : Kluwer Academic publishers, 2004 (Multiagent Systems, Artificial Societies, and Simulated Organizations 11), S. 1-26

**Hutter u. a. 2003**

HUTTER, Dieter ; MANTEL, Heiko ; SCHAIRER, Axel: Informationsflußkontrolle als Grundlage für die Sicherheit von Multi-Agenten-Systemen. In: *Praxis der Informationsverarbeitung und Kommunikation* 26 (2003), Nr. 1, S. 40-48

**Ingenias 2004**

*Ingenias Development Kit (IDK)*. Version: 2004. <http://ingenias.sourceforge.net/>. – Online-Ressource, Abruf: 03.08.05

**Jacobson u. a. 1992**

JACOBSON, Ivar ; CHRISTERSON, M. ; JONSSON, P.: *Object-Oriented Software Engineering: A Use Case Driven Approach*. Reading, MA, USA : Addison Wesley Longman Publishing, 1992. – ISBN ISBN 0-2015-4435-0

**Jeckle u. a. 2004**

JECKLE, Mario ; RUPP, Chris ; ZENGLER, Barbara ; QUEINS, Stefan ; HAHN, Jürgen: UML 2.0 - Neue Möglichkeiten und alte Probleme. In: *Informatik Spektrum* 27 (2004), Nr. 4, S. 323-331

**Jensen 1992**

JENSEN, Kurt: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Heidelberg, Germany : Springer-Verlag, 1992 (Monographs in Theoretical Computer Science). – ISBN 3-540-55597-8

---

**Jensen 1997**

JENSEN, Kurt: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods.* Heidelberg, Germany : Springer-Verlag, 1997 (Monographs in Theoretical Computer Science). – ISBN 3-540-58276-2

**Jensen 1999**

JENSEN, Kurt: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use.* Heidelberg, Germany : Springer-Verlag, 1999 (Monographs in Theoretical Computer Science). – ISBN 3-540-62867-3

**Jensen u. Rozenberg 1991**

JENSEN, Kurt ; ROZENBERG, Grzegorz: Preface. In: JENSEN, Kurt (Hrsg.) ; ROZENBERG, Grzegorz (Hrsg.): *High-Level Petri Nets, Theory and Application.* Heidelberg, Germany : Springer-Verlag, 1991, S. V-VI. – ISBN 3-540-54125-X

**Kempa u. Mann 2005**

KEMPA, Martin ; MANN, Zoltán Á.: Model Driven Architecture. In: *Informatik Spektrum* 28 (2005), Nr. 4, S. 298-302

**Kiesner u. a. 2002**

KIESNER, Christiane ; TAENTZER, Gabriele ; WINKELMANN, Jessica: Visual OCL: Eine visuelle Notation der Object Constraint Language / Technische Universität Berlin, Fakultät IV - Elektrotechnik und Informatik. 2002 (2002/23). – Forschungsbericht. Link überprüft am 2. Mai 2005. – ISSN 1436-9915

**KIF 1992**

DARPA KNOWLEDGE SHARING INITIATIVE (Hrsg.): *Knowledge Interchange Format, Version 3.0, Reference Manual.* Version: 1992. <http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps>. – Online-Ressource, Abruf: 13.04.2005

**Knuth 1965**

KNUTH, D.E.: On the translation of languages from left to right. In: *Information and Control* 8 (1965), Nr. 6, S. 607-639

**Koning u. a. 2001**

KONING, Jean-Luc ; HUGET, Marc-Philippe ; WEI, Jun ; WANG, Xu: Extended modeling languages for interaction protocol design. In: **(Wooldridge u. a. 2001)**, S. 93-100

**Korenjak 1969**

KORENJAK, A. J.: A practical method for constructing LR(k) processors. In: *Communications of the ACM* 12 (1969), Nr. 11, S. 613-623

**KQML 1993**

DARPA KNOWLEDGE SHARING INITIATIVE (Hrsg.): *Specification of the KQML Agent-Communication Language – plus example agent policies and architectures.* Version: 1993. <http://www.cs.umbc.edu/kqml/papers/kqmlspec.ps>. – Online-Ressource, Abruf: 12.04.2005

**Kristensen u. a. 1998**

KRISTENSEN, Lars M. ; CHRISTENSEN, Søren ; JENSEN, Kurt: The Practitioner's Guide to Coloured Petri Nets. In: *International Journal on Software Tools for Technology Transfer* 2 (1998), Nr. 2, S. 98-132

**Kummer 2002**

KUMMER, Olaf: *Referenznetze.* Berlin, Germany : Logos-Verlag, 2002. – ISBN 3-8325-0035-9

**Kummer u. a. 2004**

KUMMER, Olaf ; WIENBERG, Frank ; DUVIGNEAU, Michael: *Renew – The Reference Net Workshop.* <http://www.renew.de/>. Version: Juni 2004. – Release 2.0

---

**Labrou u. Finin 1997**

LABROU, Yannis ; FININ, Tim: Semantics for an Agent Communication Language. In: **(Singh u. a. 1997)**, S. 209–214. – ISBN 3–540–64162–9

**Lambrix u. a. 2003**

LAMBRIX, Patrick ; HABBOUCHE, Manal ; PÉREZ, Marta: Evaluation of ontology development tools for bioinformatics. In: *Bioinformatics* 19 (2003), Nr. 12, S. 1564–1571

**van Lamsweerde 2000**

LAMSWEERDE, Axel van: Formal specification: a roadmap. In: FINKELSTEIN, Anthony (Hrsg.): *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA : ACM Press, 2000, S. 147–159. – ISBN 1–58113–253–0

**de Lara 2005**

LARA, Juan de: Distributed Event Graphs: Formalizing Component-based Modelling and Simulation. In: *Electronical Notes in Theoretical Computer Science* 127 (2005), S. 145–162

**de Lara u. Vangheluwe 2002**

LARA, Juan de ; VANGHELUWE, Hans: AToM<sup>3</sup>: A Tool for Multi-formalism and Meta-modelling. In: KUTSCHE, Ralf-Detlef (Hrsg.) ; WEBER, Herbert (Hrsg.): *Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*. Heidelberg, Germany : Springer-Verlag, 2002 (Lecture Notes in Computer Science 2306). – ISBN 3–540–43353–8, S. 174–188

**Liebold u. Werner 1991**

LIEBOLD, Christine (Hrsg.) ; WERNER, Martin (Hrsg.): *Das neue große farbige Lexikon*. 5. Niedernhausen, Germany : Bassermann'sche Verlagsbuchhandlung, 1991. – ISBN 3–8094–0002–5

**Lind 2001a**

LIND, Jürgen: *Iterative Software Engineering for Multiagent Systems*. Heidelberg, Germany : Springer-Verlag, 2001 (Lecture Notes in Artificial Intelligence 1994). – ISBN 3–540–42166–1

**Lind 2001b**

LIND, Jürgen: Specifying Agent Interaction Protocols with Standard UML. In: **(Wooldridge u. a. 2001)**, S. 136–147

**Link 1998**

LINK, Godehard: *Algebraic Semantics in Language and Philosophy*. Stanford, CA, USA : CSLI Publications, 1998 (CSLI lecture notes 74). – ISBN 1–57586–091–0

**Lämmel u. Cleve 2001**

LÄMMELE, Uwe ; CLEVE, Jürgen: *Lehr- und Übungsbuch Künstliche Intelligenz*. München, Wien : Fachbuchverlag Leipzig im Carl Hanser Verlag, 2001. – ISBN 3–446–21421–6

**Lyons 1977**

LYONS, John: *Semantics*. Cambridge, UK : Cambridge University Press, 1977. – ISBN 0–521–29165–8

**Maedche 2002**

MAEDCHE, Alexander: *Ontology Learning for the Semantic Web*. Dordrecht, The Netherlands : Kluwer Academic publishers, 2002. – ISBN 0–7923–7656–0

**Marcotty u. a. 1976**

MARCOTTY, Michael ; LEDGARD, Henry F. ; BOCHMANN, Gregor V.: A Sampler of Formal Definitions. In: *Computing Surveys* 8 (1976), Nr. 2

**de Mauro 1982**

MAURO, Tullio de: *Einführung in die Semantik*. Tübingen : Niemeyer-Verlag, 1982. – deutsche Übersetzung des italienischen Titels „Introduzione alle semantica“, 1966. – ISBN 3-484-22027-9

**Mazouzi u. a. 2002**

MAZOUZI, Hamza ; FALLAH-SEGHROUCHNI, Amal E. ; HADDAD, Serge: Open protocol design for complex interactions in multi-agent systems. In: GINI, Maria (Hrsg.) ; ISHIDA, Toru (Hrsg.) ; CASTELFRANCHI, Cristiano (Hrsg.) ; JOHNSON, W. L. (Hrsg.): *AA-MAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA : ACM Press, 2002, S. 517-526. – ISBN 1-58113-480-0

**MDA 2001**

OBJECT MANAGEMENT GROUP (Hrsg.): *Model Driven Architecture*. Version: 2001. <http://www.omg.org/docs/ormsc/01-07-01.pdf>. – Online-Ressource, Abruf: 15.04.2005

**Meyer u. Schobbens 1999**

MEYER, John-Jules C. ; SCHOBENS, Pierre-Yves: Formal Models of Agents: An Introduction. In: MEYER, John-Jules C. (Hrsg.) ; SCHOBENS, Pierre-Yves (Hrsg.): *Formal Models of Agents - ESPRIT Project ModelAge Final Workshop; Selected Papers*. Heidelberg, Germany : Springer-Verlag, 1999 (Lecture Notes in Artificial Intelligence 1760), S. 1-7. – ISBN 3-540-67027-0

**Mosses 2001**

MOSSES, P.D.: The varieties of programming language semantics and their uses. In: BJØRNER, D. (Hrsg.) ; BROY, M. (Hrsg.) ; ZAMULIN, A.V. (Hrsg.): *Proceedings of the 4th International Andrei Ershov Memorial Conference on Perspective of System Informatics, PSI 01*. Heidelberg, Germany : Springer-Verlag, 2001 (Lecture Notes in Computer Science 2244), S. 165-190

**Muller 1997**

MULLER, Pierre-Alain: *Instant UML*. Birmingham, UK : Wrox-Press, 1997. – ISBN 1861000871

**Murch u. Johnson 2000**

MURCH, Richard ; JOHNSON, Tony: *Agententechnologie: Die Einführung*. München, Germany : Addison Wesley Verlag, 2000. – deutsche Übersetzung des englischen Originals „Intelligent Software Agents“ (1999). – ISBN 3-8273-1652-9

**Naur u. a. 1960**

NAUR, Peter u. a.: Report of the algorithmic language ALGOL 60. In: *Communications of the ACM* 3 (1960), Nr. 5, S. 299-314

**Naur u. a. 1963**

NAUR, Peter u. a.: Revised Report of the algorithmic language ALGOL 60. In: *Communications of the ACM* 6 (1963), Nr. 1, S. 1-17

**Nowostawski 2000**

NOWOSTAWSKI, Mariusz: *JFern: Java-based Petri Net framework*. <http://sourceforge.net/projects/jfern/>. Version: 2000-2005

**Nowostawski u. a. 2001a**

NOWOSTAWSKI, Mariusz ; PURVIS, Martin ; CRANFIELD, Stephen: KEA - Multi-level Agent Infrastructure. In: *Proceedings of the 2nd International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001)*, Department of Computer Science, University of Mining and Metallurgy, Krakow, Poland, 2001, S. 355-362

---

**Nowostawski u. a. 2001b**

NOWOSTAWSKI, Mariusz ; PURVIS, Martin K. ; CRANEFIELD, Stephen: A layered approach for modelling agent conversations. In: *Proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, 5th International Conference on Autonomous Agents, Montreal*. New York, NY, USA : ACM Press, 2001, S. 163–170

**Nwana u. Wooldridge 1997**

NWANA, Hyacinth S. ; WOOLDRIDGE, Michael: Software Agent Technologies. In: NWANA, Hyacinth S. (Hrsg.) ; AZARMI, Nader (Hrsg.): *Software Agents and Soft Computing: Towards Enhancing Machine Intelligence, Concepts and Applications*. Heidelberg, Germany : Springer-Verlag, 1997 (Lecture Notes in Computer Science 1198). – ISBN 3-540-62560-7, S. 59–78

**OCL 2003**

OBJECT MANAGEMENT GROUP (Hrsg.): *UML 2.0 OCL Final Adopted specification*. Version: 2003. <http://www.omg.org/cgi-bin/doc?ptc/2003-10-14>. – Online-Ressource, Abruf: 28.04.2005

**Odell u. a. 2005**

ODELL, James (Hrsg.) ; GIORGINI, Paolo (Hrsg.) ; MÜLLER, Jörg P. (Hrsg.): *Agent-Oriented Software Engineering V - 5th International Workshop, AOSE 2004, New York, NY, USA, July 2004, Revised Selected Papers*. Heidelberg, Germany : Springer-Verlag, 2005 (Lecture Notes in Computer Science 3382). – ISBN 3-540-24286-4

**Odell u. a. 2001**

ODELL, James ; PARUNAK, H. Van D. ; BAUER, Bernhard: Representing Agent Interaction Protocols in UML. In: **(Ciancarini u. Wooldridge 2001)**, S. 121–140. – ISBN 3-540-41594-7

**Odell u. a. 2003**

ODELL, James ; PARUNAK, H. Van D. ; FLEISCHER, Mitchell: The Role of Roles in Designing Effective Agent Organizations. In: GARCIA, Alessandro F. (Hrsg.) ; LUCENA, Carlos José P. (Hrsg.) ; ZAMBONELLI, Franco (Hrsg.) ; OMICINI, Andrea (Hrsg.) ; CASTRO, Jaelson (Hrsg.): *Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications [the book is a result of SELMAS 2002]*. Heidelberg, Germany : Springer-Verlag, 2003 (Lecture Notes in Computer Science 2603). – ISBN 3-540-08772-9, S. 27–38

**Olderog u. Steffen 1999**

OLDEROG, Ernst-Rüdiger ; STEFFEN, Bernhard: Formale Semantik und Programmverifikation. In: RECHENBERG, Peter (Hrsg.) ; POMBERGER, Gustav (Hrsg.): *Informatik-Handbuch*. 2. München, Wien : Carl-Hanser-Verlag, 1999, S. 143–163. – ISBN 3-446-19601-3

**OWL 2004**

MCGUINNESS, D. (Hrsg.) ; HARMELEN, F. van (Hrsg.): *OWL - Web Ontology Language Overview*. Version: 2004. <http://www.w3.org/TR/owl-features>. – Online-Ressource, Abruf: 29.04.2005

**Pan u. a. 2003**

PAN, Jin ; CRANEFIELD, Stephen ; CARTER, Daniel: A lightweight ontology repository. In: ROSENSCHEIN, Jeffrey S. (Hrsg.) ; WOOLDRIDGE, Michael (Hrsg.) ; SANDHOLM, Tuomas (Hrsg.) ; YOKOO, Makoto (Hrsg.): *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*. New York, NY, USA : ACM Press, 2003. – ISBN 1-58113-683-8, S. 632–638

---

**Parunak u. Odell 2001**

PARUNAK, H. Van D. ; ODELL, James: Representing Social Structures in UML. In: (Wooldridge u. a. 2001), S. 1–16

**Paurobally u. Cunningham 2003**

PAUROBALLY, Shamimabi ; CUNNINGHAM, Jim: Achieving Common Interaction Protocols in Open Agent Environments. In: *Proceedings of the 2nd international workshop on Challenges in Open Agent Environments, Melbourne, Australia*

**Peña u. a. 2003**

PEÑA, Joaquin ; CORCHUELO, Rafael ; ARJONA, Jose L.: A Top Down Approach for MAS Protocol Descriptions. Version:2003. [http://www.lsi.us.es/~thicksim\\$joaquin/res/A\\_Top\\_Down\\_Approach\\_for\\_MAS\\_Protocol\\_Descriptions.pdf](http://www.lsi.us.es/~thicksim$joaquin/res/A_Top_Down_Approach_for_MAS_Protocol_Descriptions.pdf). In: *Proceedings of the ACM Symposium on Applied Computing SAC'03*. ACM Press, 49–54. – Online-Ressource, Abruf: 01.08.05

**Petri 1962**

PETRI, Carl A.: *Kommunikation mit Automaten*. Bonn, Institut für Instrumentelle Mathematik, Diss., 1962

**Plotkin 1981**

PLOTKIN, G.D.: A Structural Approach to Operational Semantics / Computer Science Department, University of Aarhus. Version: 1981. <http://research.nii.ac.jp/~ichiro/lecture/model2002/SOS.pdf> (DAIMI FN-19). – Forschungsbericht. – Online-Ressource, Abruf: 27.04.05

**Plotkin 1983**

PLOTKIN, Gordon: An operational semantics for CSP. In: BJØRNER, D. (Hrsg.): *Proceedings IFIP TC2 Working Conference on Formal Description of Programming Concepts*. Amsterdam, North-Holland : IFIP, 1983, S. 199–223

**Purvis u. a. 2002a**

PURVIS, Martin ; CRANFIELD, Stephen ; NOWOSTAWSKI, Mariusz ; CARTER, Dan: Opal: A multi-level infrastructure for agent-oriented software development / Department of Information Science, University of Otago. Version:2002. <http://www.business.otago.ac.nz/infosci/pubs/papers/papers/dp2002-01.pdf> (2002/01). – Discussion Paper. – Online-Ressource, Abruf: 01.08.05. PO Box 56, Dunedin, New Zealand

**Purvis u. a. 2002b**

PURVIS, Martin K. ; HWANG, Peter ; PURVIS, Maryam A. ; CRANFIELD, Stephen ; SCHIEVINK, Martin: Interaction Protocols for a Network of Environmental Problem Solvers. In: RIZZOLI, Andrea E. (Hrsg.) ; JAKEMAN, Anthony J. (Hrsg.): *Proceedings of the 2002 iEMSs International Meeting: Integrated Assessment and Decision Support (iEMSs 2002)* Bd. 3. Lugano : The International Environmental Modelling and Software Society, 2002, S. 318–323

**Rational Rose 2005**

INTERNATIONAL BUSINESS MACHINES CORPORATION (Hrsg.): *Rational Rose*. Version:2005. <http://www-306.ibm.com/software/rational/>. – Online-Ressource, Abruf: 10.10.05

**RDF 2004**

WORLD WIDE WEB CONSORTIUM (W3C) (Hrsg.): *Resource Description Framework*. Version:2004. <http://www.w3.org/RDF/>. – Online-Ressource, Abruf: 03.11.2005

**Reisig 1998**

REISIG, Wolfgang: *Elements of Distributed Algorithms - Modeling and Analysis with Petri Nets*. Heidelberg, Germany : Springer-Verlag, 1998. – ISBN 3-540-62752-9

---

**de Remer 1969**

REMER, F.L. de: Generating parsers for BNF grammars. In: *Proceedings of the 1969 Spring Joint Computer Conference*. Montvale, New Jersey : AFIPS Press, 1969, S. 793–799

**Richters 2002**

RICHTERS, Mark: *A precise Approach to Validating UML Models and OCL Constraints*. Berlin, Germany : Logos-Verlag, 2002 (BISS Monographs). – ISBN 3–89722–842–4

**Richters u. Gogolla 1999**

RICHTERS, Mark ; GOGOLLA, Martin: A Metamodel for OCL. In: FRANCE, Robert (Hrsg.) ; RUMPE, Bernhard (Hrsg.): *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30, 1999, Proceedings*, Springer-Verlag, 1999 (Lecture Notes in Computer Science 1723), S. 156–171

**Rosenstengel u. Winand 1991**

ROSENSTENGEL, Bernd ; WINAND, Udo: *Petri-Netze - eine anwendungsorientierte Einführung*. 4. Braunschweig, Germany : vieweg-Verlag, 1991. – ISBN 3–528–33582–3

**Rumbaugh u. a. 1990**

RUMBAUGH, James R. ; BLAHA, Michael R. ; LORENSEN, William ; EDDY, Frederick ; PREMERLANI, William: *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ, USA : Prentice Hall, 1990. – ISBN 0136298419

**Rumbaugh u. a. 2004**

RUMBAUGH, James R. ; JACOBSON, Ivar ; BOOCH, Grady: *The Unified Modeling Language Reference Manual*. 2. Reading, MA, USA : Addison Wesley Longman Publishing, 2004. – ISBN 0321245628

**Rushby 1996**

RUSHBY, John: Enhancing the Utility of Formal Methods. In: *ACM Computing Surveys* 28 (1996), Nr. 4es, S. 123

**Schumacher 2001**

SCHUMACHER, Michael: *Objective Coordination in Multi-Agent-Systems - Engineering, Design and Implementation*. Heidelberg, Germany : Springer-Verlag, 2001 (Lecture Notes in Artificial Intelligence 2039). – ISBN 3–540–41982–9

**Shoham 1997**

SHOHAM, Y.: An Overview of Agent-Oriented Programming. In: (**Bradshaw 1997b**), S. 271–290. – ISBN 0–262–52234–9

**Silva u. de Lucena 2004**

SILVA, Viviane Torres D. ; LUCENA, Carlos José P.: From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language. In: *Autonomous Agents and Multi-Agent Systems* 9 (2004), Nr. 1–2, S. 145–189

**Singh 2000**

SINGH, Munindar P.: A Social Semantics for Agent Communication Languages. In: (**Dignum u. Greaves 2000b**), S. 31–45

**Singh u. Huhns 2005**

SINGH, Munindar P. ; HUHNS, Michael N.: *Service-Oriented Computing. Semantics, Processes, Agents*. Chichester, UK : John Wiley & Sons Ltd, 2005. – ISBN 0–470–09148–7

**Singh u. a. 1997**

SINGH, Munindar P. (Hrsg.) ; RAO, Anand (Hrsg.) ; WOOLDRIDGE, Michael J. (Hrsg.):

---

*Intelligent Agents IV - Agent Theories, Architectures and Languages*. Heidelberg, Germany : Springer-Verlag, 1997 (Lecture Notes in Computer Science 1365). – ISBN 3-540-64162-9

**Sycara 1998**

SYCARA, Katia P.: Multiagent Systems. In: *AI Magazine* 19 (1998), Nr. 2, S. 79–92

**Tennent 1976**

TENNENT, R. D.: The denotational semantics of programming languages. In: *Communications of the ACM* 19 (1976), Nr. 8, S. 437–453

**UML 1.3 2000**

OBJECT MANAGEMENT GROUP (Hrsg.): *Complete UML 1.3 specification (UML 1.3 Superstructure)*. Version: 2000. <http://www.omg.org/cgi-bin/apps/doc?formal/00-03-01.pdf>. – Online-Ressource, Abruf: 11.10.2005

**UML 2.0 2004**

OBJECT MANAGEMENT GROUP (Hrsg.): *UML 2.0 Superstructure FTF convenience document*. Version: 2004. <http://www.omg.org/cgi-bin/doc?ptc/2004-10-02>. – Online-Ressource, Abruf: 15.04.2005

**University of Kent 1997**

UNIVERSITY OF KENT: *JCSP: Communicating Sequential Processes for Java*. <http://www.cs.kent.ac.uk/projects/ofa/jcsp/>. Version: 1997–2004

**URI 2005**

THE INTERNET SOCIETY (Hrsg.): *Uniform Resource Identifier (URI): Generic Syntax*. Version: 2005. <http://www.gbiv.com/protocols/uri/rfc/rfc3986.html>. – Online-Ressource, Abruf: 03.11.2005

**Varró 2004**

VARRÓ, Dániel: Automated Formal Verification of Visual Modeling Languages by Model Checking. In: *Journal of Software and Systems Modeling* 3 (2004), May, Nr. 2, 85–113. [http://www.inf.mit.bme.hu/FTSRG/Publications/varro/2004/sosym2004\\_varro.pdf](http://www.inf.mit.bme.hu/FTSRG/Publications/varro/2004/sosym2004_varro.pdf)

**Vater 1996**

VATER, Heinz: *Einführung in die Sprachwissenschaft*. 2. München, Germany : Wilhelm Fink Verlag, 1996. – ISBN 3-8252-1799-X

**Visser 2004**

VISSER, Ubbo: *Intelligent Information Integration for the Semantic Web*. Heidelberg, Germany : Springer-Verlag, 2004 (Lecture Notes in Artificial Intelligence 3159). – ISBN 3-540-22993-0

**Vlacic u. a. 2000**

VLACIC, Ljubo ; ENGWIRDA, Anthony ; KAJITANI, Makoto: Cooperative Behavior of Intelligent Agents: Theory and Practice. In: SINHA, Naresh K. (Hrsg.) ; GUPTA, Madan M. (Hrsg.) ; ZADEH, Lotfi A. (Hrsg.): *Soft Computing & Intelligent Systems - Theory & Applications*. San Diego, CA, USA : Academic Press, 2000, S. 279–307. – ISBN 0-12-646490-1

**Wagner 2003**

WAGNER, Gerd: A UML Profile for External Agent-Object-Relationship (AOR) Models. In: GIUNCHIGLIA, Fausto (Hrsg.) ; ODELL, James (Hrsg.) ; WEISS, Gerhard (Hrsg.): *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002, Revised Papers and Invited Contributions*. Heidelberg, Germany : Springer-Verlag, 2003 (Lecture Notes in Computer Science 2585). – ISBN 3-540-00713-X, S. 138–149

**Wansbrough 1997**

WANSBROUGH, K.: *A modular monadic semantic action semantics*. Auckland, New Zealand, Department of Computer Science, University of Auckland, M.Sc.Thesis, 1997

**Warmer u. Kleppe 1999**

WARMER, JOS ; KLEPPE, ANNEKE: *The Object Constraint Language - Precise Modeling with UML*. Reading, MA, USA : Addison Wesley Longman Publishing, 1999. – ISBN 0-201-37940-6

**Warmer u. Kleppe 2004**

WARMER, JOS ; KLEPPE, ANNEKE: *Object Constraint Language 2.0*. Bonn, Germany : mitp-Verlag, 2004. – ISBN 3-8266-1445-3

**Watt 1991**

WATT, DAVID A.: *Programming Language Syntax and Semantics*. 2. Englewood Cliffs, NJ, USA : Prentice Hall, 1991. – ISBN 0-13-726266-3

**Weiß 1999**

WEISS, GERHARD (Hrsg.): *Multi-Agent Systems*. Cambridge, MA, USA : MIT Press, 1999. – ISBN 0-262-23203-0

**Wikipedia 2001**

*Wikipedia - die freie Enzyklopädie*. Version: 2001–2005. <http://de.wikipedia.org/>. – Webpage, Abruf: 03.08.05.

**Wilensky 1999**

WILENSKY, URI: *NetLogo*. Version: 1999–2005. <http://ccl.northwestern.edu/netlogo/>. – Online-Ressource, Abruf: 29.07.05

**Winkel 1994**

WINKEL, GLYNN: *The formal semantics of programming languages: An Introduction*. 2. Cambridge, MA, USA : MIT Press, 1994. – ISBN 0-262-23169-7

**Wood u. DeLoach 2001**

WOOD, MARK F. ; DELOACH, SCOTT A.: An overview of the multiagent systems engineering methodology. In: **(Ciancarini u. Wooldridge 2001)**, S. 207–221. – ISBN 3-540-41594-7

**Wooldridge 1999**

WOOLDRIDGE, MICHAEL: Intelligent Agents. In: **(Weiß 1999)**, S. 27–77. – ISBN 0-262-23203-0

**Wooldridge 2002**

WOOLDRIDGE, MICHAEL: *An Introduction to Multi-Agent Systems*. Chichester, UK : John Wiley & Sons Ltd, 2002. – ISBN 0-471-49691-X

**Wooldridge u. a. 2000**

WOOLDRIDGE, MICHAEL ; JENNINGS, NICHOLAS R. ; KINNY, DAVID: The Gaia Methodology for Agent-Oriented Analysis and Design. In: *Autonomous Agents and Multi-Agent Systems* 3 (2000), Nr. 3, S. 285–312

**Wooldridge u. a. 2001**

WOOLDRIDGE, MICHAEL (Hrsg.) ; WEISS, GERHARD (Hrsg.) ; ODELL, JAMES (Hrsg.) ; CIANCARINI, PAOLO (Hrsg.): *Agent-Oriented Software Engineering (AOSE) II: revised papers and invited contributions*. Heidelberg, Germany : Springer-Verlag, 2001 (Lecture Notes in Computer Science 2222)

**Zambonelli u. a. 2003**

ZAMBONELLI, FRANCO ; JENNINGS, NICHOLAS R. ; WOOLDRIDGE, MICHAEL: Developing multiagent systems: The Gaia methodology. In: *ACM Transactions on Software Engineering and Methodology* 12 (2003), Nr. 3, S. 317–370

**Zambonelli u. Omicini 2004**

ZAMBONELLI, Franco ; OMICINI, Andrea: Challenges and Research Directions in Agent-Oriented Software Engineering. In: *Autonomous Agents and Multi-Agent Systems* 9 (2004), Nr. 3, S. 253–283

**Zhang u. Xu 2004**

ZHANG, Yingzhou ; XU, Baowen: A survey of semantic description frameworks for programming languages. In: *ACM SIGPLAN Notices* 39 (2004), Nr. 3, S. 14–30

**Zhang u. Zhang 2004**

ZHANG, Zili ; ZHANG, Chengqi: *Agent-Based Hybrid Intelligent Agents - An Agent-Based Framework for Complex Problem Solving*. Springer-Verlag, 2004 (Lecture Notes in Artificial Intelligence 2938). – ISBN 3-540-20908-5

---



## A.1 Inhalt

Die beigefügte CD enthält folgende Dateien:

- /arbeit      Diplomarbeit als pdf-Dateien
- /nzdis      Quelltexte des CPN-Simulators „JFern“ und der Agentenplattform „Opal“.
- /agentuml    Quelltexte für „PAUL 2“ sowie für die AgentUML-EMF-PlugIn's.

## A.2 Anleitung

1. Zur Nutzung von PAUL wird Eclipse inklusive EMF (Eclipse Modeling Framework) benötigt. Dies muß von [www.eclipse.org](http://www.eclipse.org) heruntergeladen werden, und installiert werden.
2. „JFern“ und „Opal“ nach Anleitung (/nzdis/InstallationManual.pdf) installieren.
3. „PAUL2“ und die AgentUML-PlugIn's nach Anleitung (agentuml/installation.txt) installieren.
4. Ein TestszENARIO kann nach Anweisung (agentuml/testscenario.txt) gestartet werden.



# B

## **AgentUML-Standardisierungsvorschlag für die FIPA**

Die folgende Spezifikation wurde als Vorschlag an die FIPA gesandt. Sie soll als Grundlage für eine Standardisierung der AgentUML-Semantik dienen.