Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung

Peter Salchow

Verifikation von Multiagentensystemen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Peter Salchow

Verifikation von Multiagentensystemen

Ausarbeitung im Rahmen von Anwendungen II
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuer: Prof. Dr. Bettina Buth

Abgegeben am 25. Februar 2009

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Für eine Masterarbeit im Bereich Verifikation von Multiagentensystemen ist es wichtig, zu untersuchen, wie andere Projekte im gleichen Themenbereich die Problemstellung bearbeiten. Auf diesen Erkenntnissen können weitere Ideen aufgebaut werden. In folgender Ausarbeitung sollen deshalb die Arbeiten anderer Projekte vorgestellt werden. In diesem Abschnitt der Arbeit wird aber zunächst die zu Grunde liegende Motivation und die Zielsetzung dieser Ausarbeitung vorgestellt.

1.1 Motivation

Der Bereich der Multiagentensysteme ist sehr interdisziplinär. Einerseits lassen sich sehr viele Bereiche finden, in denen Multiagentensysteme zum Einsatz kommen (z. B. Simulationen und Optimierungen), andererseits gibt es verschiedene Sichten auf Agenten. Ein Beispiel hierfür ist ein Teilgebiet des Softwareengineerings. Agenten werden hierbei als Paradigma für die Entwicklung von Software verwendet. Besonders die Modellierung von Softwaresystemen mit komplexen internen Kommunikationsstrukturen ist schwierig, da viele dynamische Komponenten und deren Zusammenspiel entworfen werden müssen. Aus diesem Grund werden bei der agentenorientierten Softwareentwicklung unabhängige Einheiten unter dem Aspekt von Multiagentensystemen entwickelt [vgl. Wooldridge, 2006, S. 7].

Eine andere Sicht auf Multiagentensysteme bieten ubiquitäre vernetzte Systeme. Solche Systeme werden zum Teil weltweit genutzt und haben enorme Größen angenommen. Gemeint sind beispielsweise das Internet und Mobilfunknetze. In diesen Netzen existieren viele autonome (und teilweise auch mobile) Einheiten, die miteinander interagieren. Eine große Herausforderung wird es sein, das volle Potenzial dieser Systeme ausschöpfen zu können. Technologien auf Basis von Agenten scheinen in diesem Zusammenhang ein guter Ansatz zu sein, um sich dieser Herausforderung zu stellen [vgl. Wooldridge, 2006, S. 7].

Folgende Beispiele stellen eine kleine Auswahl von Projekten dar, die mit Hilfe von Agentensystemen realisiert wurden:

- Stundenplanerstellung und -optimierung durch ein Agentensystem
- Optimierung des Paketversands von DHL durch Einsatz von Agenten

- Autonome Raumschiffsteuerung durch Agenten für weite, unbemannte Missionen

Die Realisierung komplexer, sicherheitskritischer Systeme durch Agenten erfordert Mechanismen, um die Korrektheit dieser Systeme sicherzustellen. Die existierenden Mechanismen sind für herkömmliche Softwaresysteme stark entwickelt. Multiagentensysteme besitzen jedoch Besonderheiten, die andere Modelle zur Verifikation erfordern.

1.2 Zielsetzung

Es existieren bereits mehrere Projekte, die sich mit dem Thema der Verifikation von Multiagentensystemen auseinandergesetzt haben. In ihnen wurden neue Formalismen zur Verifikation von Multiagentensystemen entwickelt. Aus den Ergebnissen dieser Projekte können neue Ideen entwickelt werden. Indem die bekannten Formalismen aus anderen Projekten miteinander kombiniert und auf andere Art und Weise angewendet werden, können neue Methoden entstehen. Aus diesem Grund werden in dieser Ausarbeitung die Herangehensweisen anderer Projekte und die Ergebnisse von Studien vorgestellt und bewertet. Außerdem werden die daraus gewonnenen Erkenntnisse, hinsichtlich ihrer Eignung zur Bearbeitung der Masterarbeit, bewertet.

2 Grundlagen

In diesem Kapitel werden die Grundlagen zur Bearbeitung der Masterarbeit kurz vermittelt. Dabei wird zwischen den Teilbereichen formale Verifikation und Multiagentensysteme unterschieden.

2.1 Formale Verifikation

Die formale Verifikation bildet die Basis beim Bearbeiten einer Masterarbeit im vorgestellten Themenbereich. Um so ein komplexes Thema wissenschaftlich zu ergründen, müssen im Vorfeld wichtige Grundlagen erlernt und verstanden werden. Dieser Abschnitt gibt einen Überblick über die formale Verifikation.

Bei der formalen Verifikation soll die Korrektheit von Systemen festgestellt werden. Ein System wird dann als korrekt betrachtet, wenn es die Anforderungen erfüllt. Die einzelnen Anforderungen können in bestimmte Klassen unterteilt werden. Nachfolgend sollen einige genannt werden.

Funktionale Anforderungen Dazu zählen alle Anforderungen, die beschreiben, was das System funktional leisten muss. Sie beschreiben den Rahmen und den Leistungsumfang eines Softwaresystems. Funktionale Anforderungen legen zum Beispiel fest, welche Berechnungen vom System durchgeführt werden.

Safety Anforderungen Diese Anforderungen legen die Zuverlässigkeit eines Systems fest. Dazu zählt auch, ob ein System unerwünschte Zustände annehmen darf und wie in einem solchen Fall darauf reagiert wird. Dieser Bereich wird auch als Betriebssicherheit bezeichnet. Ein weiterer Punkt, der bei dieser Art von Anforderungen genannt werden muss, ist die Fehlertoleranz eines Systems. Hier wird festgelegt, welche Fehler in der Umgebung auftreten dürfen und wie das System darauf reagiert.

Performance Anforderungen Hier wird die Effizienz eines Systems vereinbart. Es wird festgelegt, welche Leistungsanforderungen erfüllt werden müssen. Dazu gehört nicht nur die Wirtschaftlichkeit, sondern auch die Schnelligkeit und der Ressourcenbedarf.

> Es wird unter anderem genau geregelt, wie hoch die maximale Antwortzeit für bestimmte Funktionen ist. Performance Anforderungen beschreiben oftmals die zeitlichen Aspekte eines Systems.

Um zu zeigen, dass ein Softwaresystem korrekt arbeitet, reicht es nicht aus zu zeigen, dass die vorgegebenen Anforderungen erfüllt werden können. Vielmehr muss es das Ziel sein zu zeigen, dass das System die Anforderungen ausnahmslos erfüllt. [vgl. Holzmann, 2004] Im Hinblick auf Multiagentensysteme, müssen sowohl die lokalen Eigenschaften des einzelnen Agenten verifiziert, als auch die globalen Eigenschaften des gesamten Systems verifiziert werden. Dabei ist es notwendig, Abstraktionsmechanismen anzuwenden, um den Zustandsraum eines Multiagentensystems einzugrenzen.

Zur Analyse und Verifikation von Softwaresystemen existieren diverse Werkzeuge. Eines davon ist SPIN. Dieses Tool ist open-source und hat bereits eine sehr lange Entwicklungszeit hinter sich. Es kann zur Verifikation von verteilten Systemen eingesetzt werden. Mit SPIN können Modelle, die in PROMELA beschrieben sind ausgeführt und verifiziert werden. Dazu können Spezifikationen erstellt werden, gegen die das Modell geprüft wird. Die spezifizierten Eigenschaften werden in Linear Temporal Logic beschrieben. SPIN selbst führt das model checking nicht selbst durch, sondern überführt das Modell in C-Code. Zur Überprüfung des Modells muss dann das generierte Programm ausgeführt werden. Die Generierung von C-Code hat zum Vorteil, dass über eigene Methoden in den Verifizierungsprozess eingegriffen werden kann [vgl. Holzmann, 2004].

2.2 Multiagentensysteme

Multiagentensysteme besitzen gegenüber anderen Systemen besondere Eigenschaften, die besondere Anforderungen an die Verifikation stellen. Aus diesem Grund werden an dieser Stelle einige Grundlagen zu Multiagentensystemen vermittelt.

> „Multiagent systems are systems composed of multiple interacting computation elements, known as agents.“ [Wooldridge, 2006]

Ein einzelner Agent ist eine autonome Einheit, die ihre Umwelt über Sensoren wahrnimmt und auf sie über Effektoren einwirkt, um bestimmte Ziele zu erreichen. Alle Aktionen werden unabhängig von jeglichen Benutzereingriffen ausgeführt. Das Zusammenspiel mehrerer Agenten in einer gemeinsamen Umgebung definiert ein Multiagentensystem. Es ist möglich, dass die Agenten in einem solchen System untereinander kommunizieren. Dabei können sie kooperieren, verhandeln oder sich um bestimmte Ressourcen streiten. Der einzelne Agent

in einem Multiagentensystem besitzt keine vollständige Information über das gesamte System. Jeder Agent verwaltet nur eigene Daten und führt Berechnungen aus. Ein Multiagentensystem ist daher ein verteiltes, nebenläufiges System mit dezentraler Datenhaltung. Die fehlende Gesamtsicht auf das System und die starke Parallelität sind die Eigenschaften, die eine Verifikation mit herkömmlichen Methoden erschweren.

Bei Agentenarchitekturen wird zwischen mehreren Typen unterschieden. Dazu gehören unter anderem

- Deliberative Agenten
- Reaktive Agenten
- Hybride Agenten

Diese Unterscheidung basiert auf Grundlagen des internen Verhaltens. Dabei ist es wichtig, wie ein Agent Entscheidungen über auszuführende Aktionen trifft.

Bei der Verifikation von Multiagentensystemen in der Masterarbeit sollen die deliberativen Architekturen verwendet werden. Diese zeichnen sich durch explizite symbolische Repräsentationen von Zuständen, Wissen und Zielen aus. Das eigene Wissen wird in einer internen Datenbank, der Beliefbase, verwaltet. Durch logische Schlussfolgerungen auf den bestehenden Daten können neue Fakten abgeleitet werden, die ebenfalls in der Beliefbase abgelegt werden. Ein deliberativer Agent verfolgt bestimmte Ziele, die ebenfalls explizit repräsentiert werden. Der Agent ist bestrebt, diese Ziele zu erreichen, indem die dort beschriebenen Zustände erreicht werden. Der Agent kann mehrere, auch widersprüchliche, Ziele verfolgen. Ziele können durch das Ausführen von Plänen erreicht werden. Welche Pläne ausgeführt werden, entscheidet der Agent anhand des Nutzen zur Erfüllung eines Ziels. Die weit verbreitete BDI-Architektur gehört zur Klasse der deliberativen Agenten. Das Agentensystem Jadex ist eine Implementation der BDI-Architektur.

3 Related Work

In diesem Kapitel der Ausarbeitung sollen Projekte und Technologien vorgestellt werden, die sich ebenfalls mit der Verifikation von Multiagentensystemen und verteilten Systemen beschäftigen. Ergebnisse dieser Projekte können als Grundlage für weiterführende Arbeiten an diesem Thema verwendet werden.

3.1 Agent PathFinder

Agent PathFinder ist ein Projekt, dass sich mit der automatischen Verifikation von Agentensystemen befasst. Bei den betrachteten Agenten handelt es sich um BDI-Agenten. Das Projekt entstand in Zusammenarbeit mit der NASA mit dem Ziel Agentensysteme, die zur Steuerung von Raumschiffen eingesetzt werden, zu verifizieren. Dabei soll bewiesen werden, dass das Softwaresystem genau die Aktionen ausführt, die erwartet werden. Speziell in kritischen Situationen, wie sie auf der Weltraummissionen vorkommen. Daraus entwickelte sich das Hauptziel des Forschungsprogramms, Verifikationstechniken, die auf deliberative Agenten angepasst sind, zu entwickeln und anzuwenden [vgl. Fisher u. Visser, 2002, S. 2]. Da die Agentensysteme der NASA in Java (mit Agentenerweiterungen) geschrieben sind, beruht Agent PathFinder auf dem Java PathFinder. Folgender Abschnitt beschreibt den Aufbau und die Arbeitsweise des Java PathFinders, bevor der Agent PathFinder näher beschrieben wird.

3.1.1 Java PathFinder

Mit dem Java PathFinder (JPF) können Eigenschaften von Java-Programmen verifiziert werden. Der Java PathFinder kann kurz mit den Worten „JPF is an explicit state software model checker for Java bytecode“ [JPF] beschrieben werden. Der Kern des JPF ist eine eigene Virtuelle Maschine (VM) die den Bytecode des gegebenen Programms ausführt und bestimmte Eigenschaften überprüft. Die VM von JPF ist selbst in Java geschrieben. JPF ist somit eine Ausführungsschicht zwischen der Java-VM und dem zu untersuchenden Java-Programm. Abbildung 3.1 veranschaulicht die Anordnung dieser Schichten in JPF.

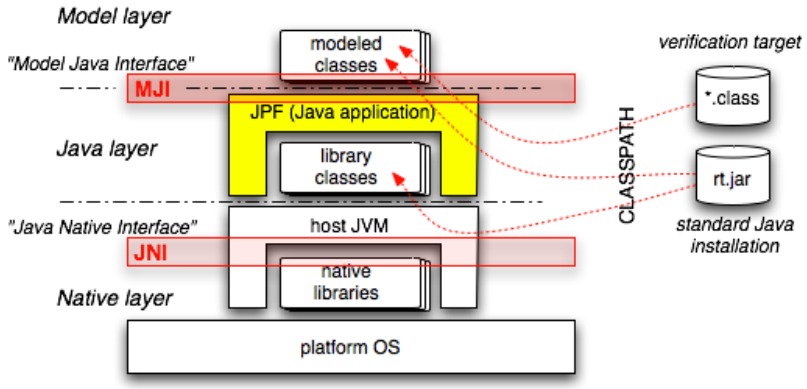

Abbildung 3.1: Java-Schichten in JPF [JPF]

Bei der Ausführung eines Bytecodes durchläuft JPF alle Ausführungspfade des zu untersuchenden Programms und überprüft die Korrektheit in jedem Zustand. Dabei untersucht JPF ein Programm standardmäßig nach nicht behandelten Exceptions und Deadlocks. Durch die Implementierung von Property-Klassen und die Erweiterung von Listenern können weitere Eigenschaften verifiziert werden. Wenn eine Eigenschaft in einem Zustand verletzt wird, wird der gesamte Ausführungspfad bis zum Auftreten des Fehlers ausgegeben [vgl. JPF]. Abbildung 3.2 zeigt den generellen Aufbau von JPF und den zugehörigen Elementen. Die Pfeile in der Abbildung verdeutlichen grob den Ablauf des Verifizierungsprozesses.

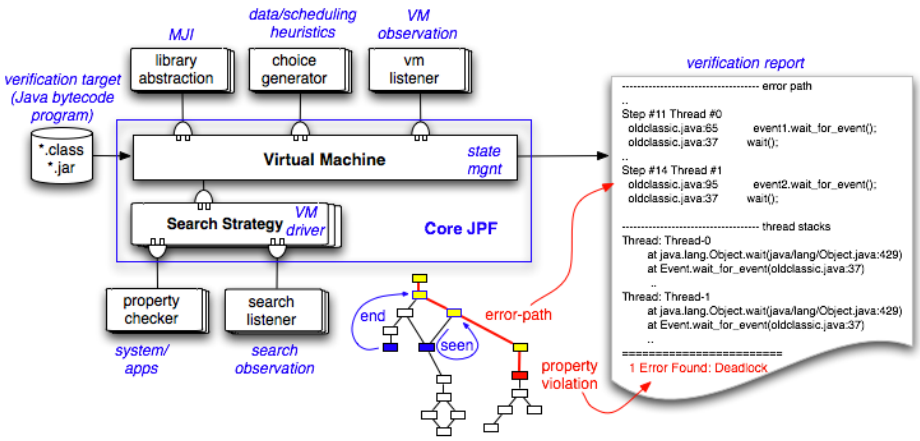

Abbildung 3.2: Ausführungsmodell von JPF [JPF]

Zum durchsuchen des Zustandsraumes verwendet JPF die beiden Mechanismen *Backtracking* und *State Matching*. Dadurch kann eine nichtdeterministische Ausführung effizient simuliert werden. Das Backtracking ermöglicht das rekursive Durchlaufen aller Systemzustände und mit Hilfe des State Matching wird untersucht, ob ein identischer Zustand bereits

durchlaufen wurde. Ist das der Fall, wird die Überprüfung des Teilbaums abgebrochen und beim nächsten Zustand fortgesetzt. Dennoch besteht das Problem, das der Zustandsraum sehr schnell sehr groß werden kann. Besonders bei parallelen Prozessen vergrößert sich der Zustandsraum sehr schnell, da die Einzelzustände der Prozesse in allen Kombinationen ausgeführt werden müssen. Dieses Problem wird auch als *State Space Explosion* bezeichnet. Abbildung 3.3 veranschaulicht diesen Sachverhalt. JPF versucht dieses Skalierungsproblem

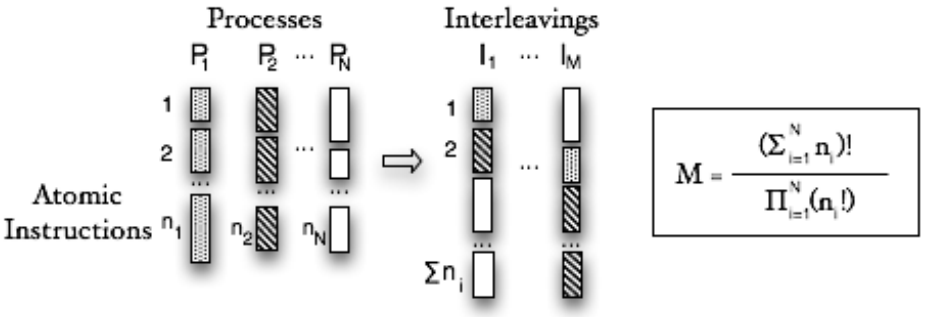

Abbildung 3.3: State Space Explosion durch interleaving der Prozesszustände [JPF]

mit drei Maßnahmen zu umgehen. Die Suchstrategien in JPF können konfiguriert werden. Heuristiken für das Durchlaufen der Suchpfade sind so anwendbar, dass Fehler schneller gefunden werden können. Die zweite Maßnahme ist die Reduzierung des Suchraums durch Abstraktionen und Heuristiken. Die dritte Maßnahme ist die Datenmenge der Zustände zu minimieren. Dies wird durch Indizes erreicht, welche die Eigenschaften eines einzelnen Zustands referenzieren [vgl. JPF].

3.1.2 Arbeitsweise des Agent PathFinder

Aufbauend auf dem JPF wurde der Agent PathFinder zur Verifikation von Agentensystemen entwickelt. Dieser berücksichtigt Java-Erweiterungen, die die BDI-Semantik in Java integrieren (z. B. JACK). Durch die Erweiterungen können Ziele und Beliefs, sowie logische Schlussfolgerungen, in Java repräsentiert werden. Das Ziel von Agent PathFinder ist es, auf BDI-Modelle angepasste Algorithmen zur Verifikation zu entwickeln und diese in JPF zu integrieren [vgl. Fisher u. Visser, 2002, S. 7]. Dazu muss eine logische Semantik für die verwendete Java-Agenten-Erweiterung entwickelt werden. Die zu verifizierenden Eigenschaften sollen in einer Kombination aus Temporal- und Modallogik angegeben werden. Um das State Space Explosion Problem zu umgehen werden im Agent PathFinder Projekt automatische Abstraktionstechniken entwickelt. Bestehende Techniken werden so erweitert, dass sie auf BDI-Erweiterungen angewendet werden können.

Die Vorteile von Agent PathFinder sind eindeutig. Zum Einen kann das Agentensystem in der Sprache verifiziert werden, in der es geschrieben worden ist. Das vermeidet Fehler bei der Transformation des Programmcodes in ein entsprechendes Modell. Zum Anderen wird durch JPF ein sehr mächtiges Tool zur Verfügung gestellt, dass die Traversierung des gesamten Zustandsraumes ermöglicht. Es muss allerdings beachtet werden, dass ab einer bestimmten Komplexität des Agentensystems nicht mehr alle Zustände durchlaufen werden können. Der Verifikationsprozess muss somit nach einer bestimmten Zeit abgebrochen werden. Die Korrektheit kann in diesem Fall nicht bewiesen werden. Die Entwicklung von Abstraktionen zur Reduzierung des Zustandsraumes ist sehr aufwändig. Darüber hinaus muss bei diesem Vorgehen eine BDI-Semantik Java definiert werden. Die darauf aufbauende Abstraktionen und Verifikationstechniken müssen für andere Semantiken und Java-Erweiterungen neu entwickelt werden. Dennoch stellt Agent PathFinder einen sehr interessanten und komplexen Mechanismus für die Verifikation von Multiagentensystemen zur Verfügung.

3.2 AgentSpeak

Weitere Ansätze zur Verifikation von Multiagentensystemen setzen auf der Sprache AgentSpeak auf, die ursprünglich auch als AgentSpeak(L) bezeichnet wird. AgentSpeak(L) stellt ein abstraktes Framework zur Programmierung von BDI-Agenten zur Verfügung. Folgender Abschnitt soll die grundlegenden Konzepte dieser Sprache kurz beschreiben.

3.2.1 Grundlagen von AgentSpeak(L)

Ein Agent wird in AgentSpeak(L) durch die Beschreibung von Beliefs (base beliefs) und Plänen erzeugt. Ein einzelnes Belief wird als Prädikat erster Stufe (in gewöhnlicher Notation) beschrieben. Die einzelnen Beliefs werden in einer Menge zusammengefasst. In AgentSpeak(L) wird zwischen zwei Arten von Zielen unterschieden. Zum Einen sind dies die *Achievement Goals*, die als Prädikate (mit einem vorangestelltem '!') beschrieben werden. Zum Anderen gibt es die *Test Goals*, die ebenfalls als Prädikate (mit vorangestelltem '?') definiert sind. *Achievement Goals* beschreiben Zustände, die der Agent erreichen will. Ein Ziel ist erreicht, wenn das zugehörige Prädikat wahr ist. Über die *Test Goals* kann der Agent abfragen, ob entsprechende Prädikate in der *Belief Base* enthalten sind. Des Weiteren werden in AgentSpeak(L) *Triggering Events* definiert. Diese Ereignisse lösen die Ausführung von Plänen aus. Die definierten Pläne beziehen sich wiederum auf atomare Aktionen, die der Agent auf seine Umwelt anwenden kann. Auch Aktionen werden als Prädikate erster Ordnung beschrieben. Zur Unterscheidung enthalten diese Prädikate spezielle *Action Symbols*.

Pläne werden in AgentSpeak(L) als $e : b_1 \,\&\, \ldots \,\&\, b_m$ <- $h_1 ; \ldots ; h_n$ definiert. Dabei ist e das *Triggering Event*, $b_1, \ldots, b_m$ sind Beliefs und $h_1, \ldots, h_n$ sind Ziele oder Aktionen. Ein Plan besitzt einen Kopf (Ausdruck link des Pfeils), der aus dem *Triggering Event* und einer Konjunktion von Beliefs (getrennt durch ':') besteht. Das Event bezeichnet den Zweck des Plans, die Beliefs bilden den Kontext. Diese Beliefs müssen erfüllt sein, damit der Plan ausgeführt werden kann. Rechts des Pfeils ist der *Body* des Plans, der aus (Unter-)Zielen und Aktionen besteht. Diese muss der Agent erreichen, wenn der Plan aufgerufen wird. Darüber hinaus legt sich der Agent auf Intentions fest. Das sind Abfolgen von Aktionen, die der Agent als nächstes ausführen wird, um ein bestimmtes Ziel zu erreichen. Jede Intention ist ein Stack von teilweise instanziierten Plänen. Das heißt, dass einige der Variablen im Plan mit Werten belegt wurden.

Ein AgentSpeak(L) Agent kann mit dem Tupel $\langle E, B, P, I, A, S_{\mathcal{E}}, S_{\mathcal{O}}, S_{\mathcal{I}} \rangle$ formal beschrieben werden. Dabei ist E die Menge der Ereignisse, B die Mende der Beliefs, P die Menge der Pläne, I die Menge der Intentions und A die Menge der Aktionen. Die Funktion $S_{\mathcal{E}}$ wählt ein Ereignis aus E aus; $S_{\mathcal{O}}$ wählt einen geeigneten Plan aus allen ausführbaren Plänen aus; und $S_{\mathcal{I}}$ wählt eine Intention aus I aus, die dann ausgeführt wird [vgl. Bordini u. a., 2003].

AgentSpeak(L) ist eine logikbasierte Sprache, die sehr formal beschrieben werden kann und daher zur Verifikation geeignet ist. Mit AgentSpeak(L) kann ein komplettes Multiagentensystem mit einer formalen Sprache beschrieben werden, das von den Plattformen direkt ausgeführt werden kann.

3.2.2 Verifikation von AgentSpeak

Es existieren mehrere Ansätze, Programme in AgentSpeak zu verifizieren. Einer davon beschäftigt sich mit der Übersetzung von AgentSpeak nach PROMELA [vgl. Bordini u. a., 2003]. Um diese Übersetzung durchführen zu können, wurde AgentSpeak(F) entwickelt. Es ist eine Abwandlung von AgentSpeak(L). Der Hauptunterschied zwischen beiden Sprachen ist, dass in AgentSpeak(F) der Zustandsraum von AgentSpeak(L) auf eine endliche Anzahl beschränkt wurde. Dies war nötig um das State Space Explosion Problem zu umgehen (s. Kapitel 3.1.1). Bei einem endlichen Zustandsraum ist es möglich, alle Ausführungspfade zu betrachten. Durch die Beschränkung wird eine maximale Anzahl an Typen, Datenstrukturen und Kommunikationskanälen vorgegeben. In diesem Projekt wurden diverse Tools unter dem Namen CAPS (Checking AgentSpeak Programs) entwickelt, die den AgentSpeak(F)-Code automatisch in PROMELA übersetzen. Der Vorteil dieser Überführung ist, dass kein neuer model checker entwickelt werden muss. Bestehende model checker haben eine sehr lange Entwicklungszeit hinter sich und es existieren zuverlässige Implementationen [vgl. Bordini u. a., 2004b, S. 46-47]. Zu diesen model checkern gehört auch SPIN, der PROMELA-Code

verifizieren kann. Für die Beschreibung der zu verifizierenden Eigenschaften wurde ein weiterer Übersetzer entwickelt, der, in einfacher BDI-Logik beschriebene Spezifikationen, in LTL-Formeln im SPIN-Format übersetzt.

Im gleichen Projekt wurden die CAPS-Tools so erweitert, dass sie aus AgentSpeak(F)-Code Java erzeugen können. Dieser wurde dann mit dem Java PathFinder (s. Kapitel 3.1.1) untersucht. Abbildung 3.4 veranschaulicht noch einmal den Verifikationsprozess von AgentSpeak.

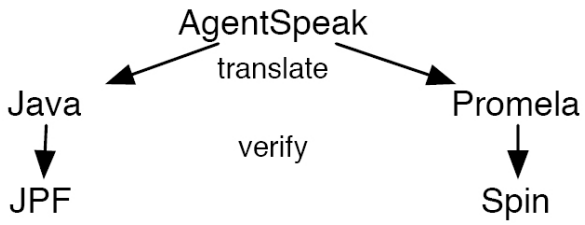

Abbildung 3.4: Verifikation von AgentSpeak [Dennis u. a., 2008]

In einer Weiterentwicklung dieses Projekts wurde die Beschränkung aus AgentSpeak aufgehoben. Es wurde ein komplexeres, auf Java basierendes Framework entworfen, welches flexibel auf sämtliche Agenten-Sprachen angepasst werden kann. Die grobe Architektur dieses Frameworks wird in Abbildung 3.5 dargestellt.

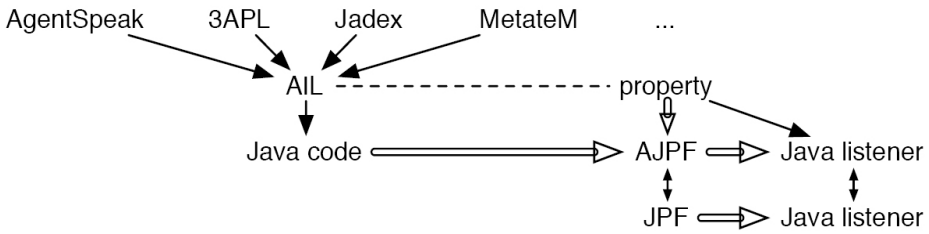

Abbildung 3.5: Framework zur Verifikation von Agentensystemen [Dennis u. a., 2008]

Eine zentrale Komponente in diesem Framework ist die Agent Infrastructure Layer (AIL). Die AIL ist eine Bibliothek von Java-Klassen die es ermöglicht, Interpreter für die verschiedenen Agenten-Programmiersprachen zu entwickeln. Außerdem enthält die AIL eine klare Semantik, die an die einzelnen Sprachen angepasst wird. In der AIL existieren Agenten verschiedener Sprachen nebeneinander. Darüber hinaus kann die AIL als Plattform angesehen werden, die über die einzelnen Interpreter den Agentencode ausführen und verifizieren kann. Dadurch entsteht ein lose gekoppeltes System, in dem die Implementierung der Agenten strikt von der Verifikation getrennt ist. Um ein Agentensystem zu verifizieren, wird der

Agent Java PathFinder (AJPF), eine Erweiterung für die AIL des JPF (s. Kapitel 3.1.1), benutzt. Der AJPF führt die AIL aus und verifiziert dabei die einzelnen Zustände. Der AJPF optimiert den JPF so, dass unnötige Zustände und Programmteile der AIL (z. B. die Interpreter) nicht durchlaufen und verifiziert werden. Dadurch werden nur die relevanten Teile des Agentensystems durchlaufen. Zu prüfende Eigenschaften werden in der Property Specification Language (PSL) beschrieben. PSL erlaubt temporallogische Ausdrücke über Beliefs, Ziele, Intentions und Aktionen [vgl. Dennis u. a., 2008].

3.3 AVACS

Automatic Verification and Analysis of Complex Systems (AVACS) ist ein gemeinsames Forschungsprojekt der Universität Oldenburg, der Universität Freiburg und der Universität des Saarlandes. In diesem Sonderforschungsbereich werden mathematische Methoden und Verfahren entwickelt, um komplexe sicherheitskritische Systeme zu analysieren und verifizieren. Dabei stehen eingebettete Systeme im Vordergrund. Von diesen Systemen werden Modelle erstellt, die anschließen verifiziert werden können. Der bisherige Stand der Methoden erlaubt nur die Behandlung einzelner Aspeke wie zum Beispiel Nebenläufigkeit und Zeitverhalten. Ziel von AVACS ist es daher, ein umfassendes Konzept, das alle Aspekte berücksichtigt, zur Verifikation zu entwickeln. Dieses Konzept beinhaltet die Automatisierung der Verifikationstechniken [vgl. AVACS].

Dieses Projekt beschäftigt sich nur indirekt mit Multiagentensystemen. Dennoch können Ergebnisse aus diesem Projekt auf Multiagentensysteme übertragen werden. Der Hintergrund dieser Überlegung ist der, dass sich die komplexen, eingebetteten Systeme, mit denen AVACS sich auseinandersetzt, als Multiagentensysteme modelliert werden können. Besonders die mathematischen Erkenntnisse aus AVACS können möglicherweise interessante Ansätze für Multiagentensysteme liefern.

4 Fazit und Ausblick

An dieser Stelle soll abschließend noch ein Fazit gezogen und ein Ausblick auf das weitere Vorgehen gegeben werden.

4.1 Fazit

Für die Verifikation von Multiagentensystemen existieren mehrere Ansätze, die eine vielversprechende Grundlage für eine Masterarbeit in diesem Thema geben. Besonders die in Kapitel 3.1 und 3.2.2 vorgestellten Technologien liefern interessante Ansätze, die weiter erarbeitet und ausgebaut werden können. Die Ausweitung des Anwendungsfeldes von Multiagentensystemen auf sicherheitskritische Systeme, erfordern die Anwendung von Verifikationsmechanismen. Nur so kann eine korrekte Ausführung der Systeme sichergestellt werden. Die Ergebnisse der vorgestellten Studien und Projekte haben gezeigt, dass die Verifikation von Multiagentensystemen prinzipiell durchführbar ist.

4.2 Ausblick

Das nächste Ziel ist es, eine genaue Eingrenzung und Beschreibung des zu bearbeitenden Themas zu erarbeiten. Dabei werden die in Kapitel 3.1 und 3.2.2 erwähnten Techniken einen erheblichen Einfluss haben. Mit einer genauen Beschreibung eines Themas kann der Kontakt zu führenden Persönlichkeiten auf dem Gebiet der Mutiagentenverifikation genutzt werden, um weitere Informationen und Anregungen zu erhalten.

Literaturverzeichnis

[Alechina 2006] ALECHINA, Natasha: Verifying space and time requirements for resource-bounded agents. In: *International Conference on Autonomous Agents*, ACM, 2006

[AVACS] *Automatic Verification and Analysis of Complex Systems*. `http://www.avacs.org/`, Abruf: 08. Februar 2009. – Einstiegsseite der AVACS Homepage

[Bordini u. a. 2003] BORDINI, Rafael H. ; FISHER, Michael ; PARDAVILA, Carmen ; WOOLDRIDGE, Michael: Model checking AgentSpeak. In: *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA : ACM Press, 2003. – ISBN 1–58113–683–8, S. 409–416

[Bordini u. a. 2004a] BORDINI, Rafael H. ; FISHER, Michael ; VISSER, Willem ; WOOLDRIDGE, Michael: State-space Reduction Techniques in Agent Verification. In: *AAMAS '04: Proceedings of the third international joint conference on Autonomous agents and multiagent systems*. Washington, DC, USA : ACM Press, 2004. – ISBN 1–58113–864–4, S. 896–903

[Bordini u. a. 2004b] BORDINI, Rafael H. ; FISHER, Michael ; WOOLDRIDGE, Michael ; VISSER, Willem: Model checking rational agents. In: *Intelligent Systems, IEEE* 19 (2004), Sept.-Oct., Nr. 5, S. 46–52. `http://dx.doi.org/10.1109/MIS.2004.47`. – DOI 10.1109/MIS.2004.47. – ISSN 1541–1672

[Dennis u. a. 2008] DENNIS, Louise A. ; FARWER, Berndt ; BORDINI, Rafael H. ; FISHER, Michael: A flexible framework for verifying agent programs. In: *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC : International Foundation for Autonomous Agents and Multiagent Systems, 2008. – ISBN 978–0–9817381–2–X, S. 1303–1306

[Fisher u. Visser 2002] FISHER, Michael ; VISSER, Willem: Verification of Autonomous Spacecraft Control - A logical vision of the future. In: *Proc. Workshop on AI Planning and Scheduling For Autonomy in Space Applications* (2002)

[Henesey u. a. 2008] HENESEY, Lawrence ; DAVIDSSON, Paul ; PERSSON, Jan A.: Agent based simulation architecture for evaluating operational policies in transshipping containers. In: *Autonomous Agents and Multi-Agent Systems* 18 (2008), April, Nr. 2, S. 220–238

[Holzmann 2004] HOLZMANN, Gerard J.: *The Spin Model Checker : Primer and Reference Manual*. Boston, Mass. [u.a.] : Addison-Wesley, 2004. – ISBN 0–321–22862–6

[JPF] *Java PathFinder*. http://javapathfinder.sourceforge.net/, Abruf: 07. Februar 2009. – Einstiegsseite der Java PathFinder Homepage

[Kramer 2007] KRAMER, Jeff: Is abstraction the key to computing? In: *Commun. ACM* 50 (2007), April, Nr. 4, S. 36–42. http://dx.doi.org/10.1145/1232743.1232745. – DOI 10.1145/1232743.1232745. – ISSN 0001–0782

[Lam u. Barber 2004] LAM, Dung N. ; BARBER, K. S.: Verifying and Explaining Agent Behavior in an Implemented Agent System. In: *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. Washington, DC, USA : IEEE Computer Society, 2004. – ISBN 1–58113–864–4, S. 1226–1227

[Roos u. Witteveen 2007] ROOS, Nico ; WITTEVEEN, Cees: Models and methods for plan diagnosis. In: *Autonomous Agents and Multi-Agent Systems* (2007). http://dx.doi.org/10.1007/s10458-007-9017-6. – DOI 10.1007/s10458–007–9017–6

[Sudeikat u. Renz 2008] SUDEIKAT, Jan ; RENZ, Wolfgang: A Systemic Approach to the Validation of Self-Organizing Dynamics within MAS. In: *9th International Workshop on Agent Oriented Software Engineering*, Springer, 2008

[Wooldridge 2006] WOOLDRIDGE, Michael: *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2006. – ISBN 0–471–49691–X

[Wooldridge u. a. 2002] WOOLDRIDGE, Michael ; FISHER, Michael ; HUGET, Marc-Philippe ; PARSONS, Simon: Model Checking Multi-Agent Systems with MABLE. In: *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multi-agent systems*. New York, NY, USA : ACM Press, 2002. – ISBN 1–58113–480–0, S. 952–959