Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminararbeit

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

Seminarausarbeitung im Rahmen der Ringvorlesung
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

In der folgenden Ausarbeitung soll die Vision für eine Masterarbeit im Bereich der Verifikation von Multiagentensystemen vorgestellt werden. Der Schwerpunkt liegt dabei auf der Beschreibung der Zielsetzung und möglicher Risiken. Konkrete Ansätze und einen Überblick über themenverwandte Projekte werden in der Ausarbeitung im Fach *Anwendungen 2* behandelt. In diesem Abschnitt der Arbeit wird aber zunächst die zu Grunde liegende Motivation und die darauf aufbauende Zielsetzung 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, 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. So kann den Problemen solch komplexer Systeme begegnet werden [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].

Darüber hinaus sind bereits viele Systeme mit Hilfe von Agenten realisiert worden. Einige Beispiele hierfür sind:

Stundenplanerstellung mit Hilfe von Agenten: In diesem Projekt an der FH Gelsenkirchen wird ein Agentensystem entwickelt, das den Stunden- und Raumplan nach bestimmten Vorgaben optimiert. Dabei wird jede Veranstaltung durch einen Agenten repräsentiert. Die Agenten versuchen beim „Buchen“ von Räumen und Zeiten eine hohe Zufriedenheit zu erzielen (z. B. späte Zeiten vermeiden, zusammenhängender Stundenplan). Um die Gesamtzufriedenheit zu steigern, verhandeln die Agenten untereinander um die zur Verfügung stehenden Ressourcen.

Optimierung des Paketversands von DHL durch Agenten: Ziel dieses Projekts war es, Wege und Ressourcen beim Versand zu minimieren und Kosten zu sparen. In dem Simulationssystem wurden die Pakete und die LKWs als Agenten modelliert. Den Paketen wurde ein Betrag (das Porto) zur Verfügung gestellt, mit dem sie an ihr Ziel „reisen“ müssen. Dazu verhandeln sie mit den LKWs über die Fahrkosten. Um diese Simulation realitätsnah durchführen zu können, wurden von DHL statistische Daten über Ziele und Menge von Paketen zur Verfügung gestellt. Dieses Projekt fand in Zusammenarbeit mit der HAW Hamburg statt.

Autonome Raumschiffsteuerung durch Agenten: In der Raumfahrt können auch unbemannte Missionen nicht beliebig weit in den Weltraum vordringen. Die hohen Laufzeiten der Signale verhindern bei großen Entfernungen ein schnelles Eingreifen in die Steuerung des Raumschiffs. Aus diesem Grund soll ein autonomes Softwaresystem die Steuerung des Raumschiffs übernehmen. Die Steuerung eines Raumschiffs ist sehr komplex und erfordert „intelligente“ Entscheidungen vom Softwaresystem. In diesem System werden Agenten eingesetzt, die gemeinsam über mögliche Entscheidungen abstimmen und so das Raumschiff autonom steuern [vgl. Fisher u. Visser, 2002].

Die Beispiele zeigen, dass komplexe, aber auch sicherheitskritische, Systeme durch Agenten realisiert werden. Um sicherzustellen, dass diese Systeme alle Anforderungen erfüllen und sicher arbeiten, müssen Tests durchlaufen und Verifikationsmechanismen angewendet werden. Für Softwaresysteme ohne Agenten existieren bereits lang bewährte Methoden und Ansätze zur Verifikation. Aber bereits in diesem Umfeld hat sich gezeigt, dass eine umfassende Verifikation von Softwaresystemen nicht trivial ist. Dennoch ist es unerlässlich, Softwaresysteme unter dem Aspekt von Multiagentensystemen zur verifizieren. Multiagentensysteme besitzen besondere Eigenschaften (z. B. emergentes Verhalten, logisches Schließen), die die Anwendung bisheriger Methoden erschweren oder verhindern.

1.2 Zielsetzung

Aus diesem Grund soll in der Masterarbeit die Frage, wie Multiagentensysteme verifiziert werden können, im Mittelpunkt der Betrachtung stehen. Ein wichtiger Punkt wird es dabei

sein, die Grenzen bei diesem Vorgehen aufzuzeigen und zu untersuchen, bis zu welchem Grad sich Multiagentensysteme verifizieren lassen. Dabei ist es wichtig, bestehende Ansätze auf diesem Gebiet näher zu betrachten und zu bewerten. Außerdem muss untersucht werden, welche Aspekte eines Multiagentensystems sich bei der Verifikation anbieten und wie auf diese Aspekte bekannte Formalismen angewandt werden können. Ein großes Ziel wird es sein, einen Aspekt von Multiagentensystemen zu verifizieren. Dafür soll dieser Aspekt zunächst für ein bestimmtes Multiagentensystem verifiziert werden. Der zu untersuchende Aspekt wird erst durch die Erkenntnisse in der Masterarbeit konkretisiert. Mögliche Aspekte könnten der Nachweis der Korrektheit erzielter Ergebnisse oder der Ausschluss ungültiger Systemzustände sein.

2 Vorgehen

In diesem Kapitel werden die Grundlagen zu den beiden Teilbereichen, Multiagentensysteme und formale Verifikation, kurz vermittelt. Dabei wird auch erläutert, wie diese Bereiche in der Masterarbeit bearbeitet und verwendet werden sollen, damit das gesetzte Ziel erreicht wird.

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. Daher wird im Folgenden der Bereich der formalen Verifikation näher beleuchtet. Dazu wird zunächst auf die Grundlagen eingegangen. Anschließend werden einige gängige Werkzeuge vorgestellt, mit denen Modelle von Systemen verifiziert werden können.

2.1.1 Grundlagen der formalen Verifikation

Die formale Verifikation beschäftigt sich mit der Feststellung der Korrektheit von Systemen. Ein System wird als korrekt angesehen, wenn es die gestellten Anforderungen erfüllt. Bei dieser Betrachtung können die Anforderungen in bestimmte Klassen unterteilt werden. Dazu gehören zum Beispiel:

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.

Die oben beschriebene Auswahl an Anforderungen zeigt die wichtigsten Aspekte, die beim Testen und Verifizieren von Softwaresystemen berücksichtigt werden müssen. Die Funktionalen Anforderungen können weitestgehend durch Tests sichergestellt werden und sollen deshalb beim Bearbeiten des Themas nicht weiter betrachtet werden.

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] Die zeitlichen Aspekte der Performance Anforderungen erschweren eine formale Spezifikation und Verifikation. Es ist sehr aufwändig das Erfüllen dieser Anforderungen nachzuweisen. Aus diesem Grund soll der Nachweis von Performance Anforderungen in der Masterarbeit nicht weiter behandelt werden. Im Mittelpunkt werden die Safety Anforderungen stehen. Besonders der Ausschluss unzulässiger Zustände in Multiagentensystemen, soll einen Kernpunkt der Arbeit darstellen. Dabei sollen sowohl die einzelnen Agenten, als auch das Zusammenspiel mehrerer Agenten unter diesem Gesichtspunkt verifiziert werden. Aber auch die Fehlertoleranz von Multiagentensystemen wird ein Bestandteil der Betrachtung sein. Hierbei wird es sehr wichtig sein zu beschreiben, welche Zustände die Umgebung eines Agenten annehmen kann. Aufgrund dieser Erkenntnisse muss geprüft werden, ob ein Agent sich in jedem möglichen Zustand korrekt verhält.

Eine große Herausforderung wir es sein, die Komplexität, die durch die möglichen Zustände von Agenten und Umwelt entsteht, zu beherrschen. Aus diesem Grund müssen Wege zur Abstraktion entwickelt werden. Ein mögliches Vorgehen könnte es sein, den Zustandsraum über Abhängigkeiten zu strukturieren und in einer formalen Sprache zu beschreiben.

2.1.2 Werkzeuge

Zur Untersuchung und Verarbeitung von formalen Spezifikationen existieren Werkzeuge. Diese Werkzeuge können Modelle von Softwaresystemen ausführen und gegen Spezifikationen prüfen. Diese Methode wird *Model Checking* genannt.

Die folgende Übersicht gibt eine kurze Beschreibung der wichtigsten Eigenschaften für eine Auswahl an Werkzeugen.

FDR Der Name FDR steht für „Failures Divergences Refinement“ und ist ein Model Checker des Herstellers FormalSystems. Mit diesem Tool können insbesondere Sicherheitsprotokolle analysiert und verifiziert werden. Modelle von Softwaresystemen werden in der Sprache (Prozessalgebra) CSP (Communicating Sequential Processes) erstellt und mit FDR verarbeitet werden. Mit Hilfe von CSP können von komplexen Systemen abstrakte Modelle erstellt werden, die die wichtigsten Eigenschaften des Systems beschreiben. In CSP lassen sich einzelne, miteinander kommunizierende, Prozesse modellieren. Diese Prozesse können dann in beliebiger Anzahl und Reihenfolge miteinander verbunden werden. Die Ausführung von Prozessen kann sowohl sequenziell als auch parallel stattfinden. Bei der Analyse von Programmen in CSP steht die Kommunikation der Prozesse im Vordergrund. Es werden die Kommunikationskanäle zwischen den Prozessen überwacht und ausgewertet. Auf diesem Wege können die von Prozessen erzeugten Ausgaben (Traces) analysiert und gegen eine Spezifikation geprüft werden.

SPIN Dieses Tool ist ein open-source Model Checker, der zur Verifikation von verteilten Systemen eingesetzt werden kann. Der Name SPIN steht für „Simple Promela INterpreter“. Wie der Name schon sagt, können mit SPIN Modelle, die in PROMELA (Process Meta Language) geschrieben sind, verifiziert werden. Ähnlich zu CSP werden auch in PROMELA einzelne Prozesse mit einem bestimmten Verhalten modelliert. Auch diese Prozesse werden ausgeführt und können dabei miteinander kommunizieren. Im Gegensatz zu CSP können in PROMELA Kontrollstrukturen leichter umgesetzt werden. Außerdem haben die Prozesse Zustände mit expliziten Eigenschaften. Intern werden die PROMELA-Prozesse als nichtdeterministische Automaten repräsentiert. Die zu verifizierenden Eigenschaften werden in SPIN in Linear Temporal Logic (LTL) ausgedrückt. Das System negiert die Ausdrücke und überführt diese in Büchi-Automaten. Mit Hilfe dieser Strukturen können die Eigenschaften des Modells (und somit auch des zugrunde liegenden Systems) verifiziert werden. Im Gegensatz zu anderen Analyse-Tools, wird das *Model Checking* nicht von SPIN selbst durchgeführt. SPIN generiert aus den Eingaben C-Code. Dieser kann dann compiliert und ausgeführt werden. Während der Ausführung wird das *Model Checking* durchgeführt und Rückmeldungen über die Ausführung an SPIN übermittelt. Einen Überblick über diese Struktur zeigt Abbildung 2.1. Die Generierung von C-Code hat zum Vorteil, dass über eigene Methoden in die Verifizierung eingegriffen werden kann [vgl. Holzmann, 2004].

Netlab Mit dem Tool Netlab, das an der RWTH Aachen entwickelt wird, können Petrinetze erstellt, simuliert und analysiert werden. Das Tool ist auf die Petrinetze der Klasse Stellen/Transitions-Netze (S/T-Netze) beschränkt. Netlab ist in der Lage, die Eigenschaften (z. B. Invarianten, Überdeckungsgraph) der eingegebenen Netze zu analy-

sieren. Darüber hinaus kann Netlab aus den berechneten Eigenschaften weitere Ergebnisse schlussfolgern. Petrinetze haben den Vorteil, dass sie sich sowohl formal beschreiben, als auch sehr gut graphisch darstellen lassen. Die analytischen Berechnungen lassen sich sehr effizient implementieren. Dadurch können auch große Netze sehr schnell verarbeitet werden. Allerdings lassen sich nur kleinere Sachverhalte gut mit Petrinetzen modellieren. Bei größeren Systemen mit komplexen Zusammenhängen werden die Netze schnell sehr groß und unübersichtlich, so dass sie nur noch schwer verstanden werden können.

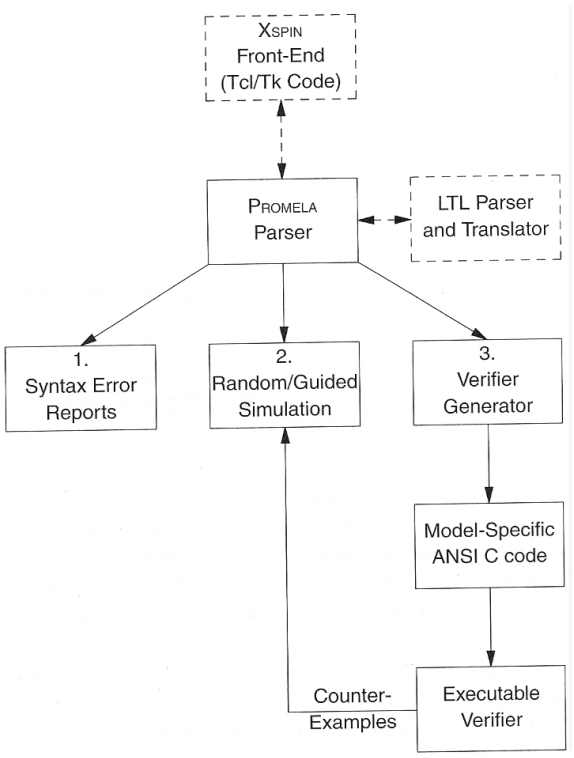

Abbildung 2.1: Struktur von SPIN [vgl. Holzmann, 2004, S. 246]

Die zuvor vorgestellten Werkzeuge können für die Verifikation von Multiagentensystemen benutzt werden. Es müssen jedoch die Vor- und Nachteile der einzelnen Tools bewertet werden, da nicht alle gleich gut für diesen Aufgabenbereich geeignet sind. Petrinetze haben einerseits den Vorteil, dass auch große Modelle sehr schnell analysiert werden können. Andererseits ist die Modellierung eines kompletten Multiagentensystems mit Perinetzen zu komplex. Selbst bei der Modellierung nur eines Agenten, können möglicherweise nicht alle Eigenschaften umgesetzt werden. Aus diesem Grund werden Petrinetze im Rahmen der Masterarbeit keine Anwendung finden. In CSP können Multiagentensysteme wesentlich besser modelliert werden. Die einzelnen Abläufe innerhalb eines Agenten können als Prozess modelliert werden. Die Nebenläufige Ausführung aller Prozesse würde dann das Verhalten

des Multiagentensystems repräsentieren. Da sich in FDR die Kommunikationskanäle sehr gut überwachen lassen, können die Kommunikationsprotokolle und die kommunizierten Daten in CSP modellierten Agentensystemen verifiziert werden. Die internen Abläufe innerhalb eines Agenten lassen sich nur schwer über Traces verifizieren. Diese Semantik wird nicht ausreichen um alle wichtigen Abläufe in einem Multiagentensystem zu verifizieren. Ein Ansatz mit PROMELA wäre, so wie in CSP, die Agenten als PROMELA-Prozesse zu modellieren. Zur Überwachung der Kommunikationskanäle bietet PROMELA die selben Möglichkeiten wie auch CSP. Den Vorteil, den PROMELA mit sich bringt, ist das die Spezifikationen der zu prüfenden Eigenschaften in LTL beschrieben werden können. Dies ermöglicht die Verifikation sehr komplexer Eigenschaften. Aus diesem Grund wird PROMELA einen zentralen Ansatzpunkt für die Verifikation von Multiagentensystemen darstellen.

Ein Problem, dass bei allen drei, der vorgestellten Tools besteht ist, dass das zu verifizierende System zuerst in ein entsprechendes Modell überführt werden muss. Bei diesem Prozess können Fehler in der Modellierung auftauchen, so dass das System und sein Modell, in möglicherweise entscheidenden Aspekten, nicht übereinstimmen. Um dieser Gefahr entgegenzuwirken, sind zwei Wege vorstellbar. Entweder könnte der Code des Systems direkt verifiziert werden oder der Übersetzungsprozess wird automatisiert. Damit eine direkte Codeverifikation in der Masterarbeit in Betracht gezogen werden kann, sollte das zu verifizierende System in einer weitestgehend formalisierten Sprache implementiert sein. Die Herausforderung wird darin liegen, eine formale Sprache zu finden, mit der ein Multiagentensystem sowohl implementiert, als auch verifiziert werden kann. Da ein solches Vorgehen aber zu komplex sein wird, wird die direkte Codeverifikation wahrscheinlich nur ansatzweise betrachtet werden können. Eine wesentlich größere Bedeutung wird die Automatisierung des Übersetzungsprozesses haben.

2.2 Multiagentensysteme

Das Thema der Multiagentensysteme ist der zweite große Bereich, der in der Masterarbeit aufgegriffen wird. Sie besitzen gegenüber anderen Systemen besondere Eigenschaften, die besondere Anforderungen an die Verifikation stellen. Aus diesem Grund werden in den folgenden Abschnitten einige Grundlagen zu Multiagentensystemen vermittelt.

2.2.1 Grundlagen der Multiagentensysteme

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

Ein einzelner Agent kann als autonome Einheit beschrieben werden, die ihre Umwelt über Sensoren wahrnimmt und auf sie über Effektoren einwirkt. Sie führen ihre Aktionen unabhängig von jeglichen Benutzereingriffen aus. Durch die Ausführung der Aktionen wird ein bestimmtest Ziel angestrebt. Erst das Zusammenspiel mehrerer Agenten in einer Umgebung macht ein Multiagentensystem aus. In der Regel besitzen Agenten in einem Multiagentensystem ein soziales Verhalten. Das heißt, sie kommunizieren mit anderen Agenten. Dabei ist es möglich, dass sie kooperieren, verhandeln oder sich um bestimmte Ressourcen streiten. Ein einzelner Agent in einem Multiagentensystem besitzt keine vollständige Information über das gesamte System. Da jeder einzelne Agent seine eigenen Daten verwaltet und Berechnungen ausführt, ist ein Multiagentensystem ein verteiltes, nebenläufiges System mit dezentraler Datenhaltung. Es existiert keine zentrale Steuerung. Die fehlende Gesamtsicht auf das System und die starke Parallelität sind die Eigenschaften, die eine Verifikation mit herkömmlichen Methoden erschweren.

2.2.2 Agententypen

Bei Agentenarchitekturen wird zwischen mehreren Typen unterschieden. Diese Unterscheidung basiert auf Grundlagen des internen Verhaltens. Dabei ist es wichtig, wie ein Agent Entscheidungen über auszuführende Aktionen trifft. Folgende Übersicht zeigt bekannte Agententypen und deren wichtigste Merkmale.

Deliberative Agenten Diese Art von Agenten basiert auf symbolischen Strukturen. Die über Sensoren wahrgenommene Umwelt wird in symbolisches Wissen transformiert und vom Agenten in einer internen Datenbank (Beliefbase) gespeichert. Hier werden auch Informationen über andere Agenten (durch Kommunikation) gespeichert. Durch logische Schlussfolgerungen auf den bestehenden Daten können neue Fakten abgeleitet werden, die ebenfalls in der Beliefbase gespeichert werden. Somit besitzt der Agent eine symbolische Repräsentation seines eigenen Zustandes und seiner Umwelt. Darüber hinaus verfolgt ein deliberativer Agent bestimmte Ziele, die ebenfalls explizit repräsentiert werden. Der Agent ist bestrebt diese Ziele zu erfüllen, indem der dort beschriebene Zustand erreicht wird. Dabei kann es vorkommen, dass gegensätzliche Ziele existieren. Um Ziele erfüllen zu können, stehen dem Agenten Aktionen oder ganze Pläne zur Verfügung, die er ausführen kann. Welche Aktionen ausgeführt werden, entscheidet der Agent indem er die Aktionen hinsichtlich ihres Nutzen für ein bestimmtest Ziel bewertet. Zu der Klasse der deliberativen Agenten zählen beispielsweise die stark verbreitete BDI-Architektur (BDI steht für Beliefs Desires Intentions). Zur Veranschaulichung der BDI-Architektur wird oftmals die Architektur des *Procedural Reasongin Systems* (siehe Abbildung 2.2) verwendet, da auch hier eine explizite Repräsentation der Datenstrukturen, zur Beschreibung des internen Zustandes, verwendet wird

[vgl. Wooldridge, 2006]. Eine Implementation der BDI-Architektur ist beispielsweise das Agentensystem Jadex.

Reaktive Agenten Im Vergleich zu deliberativen Agenten ist der einzelne reaktive Agent wesentlich einfacher aufgebaut. Sie besitzen keine symbolische Repräsentation ihres Zustandes oder ihrer Umwelt und speichern in der Regel auch keine Informationen für eine spätere Verwendung ab. Reaktive Architekturen streben eine direkte Umsetzung der Sensordaten in Aktionen an. Dadurch können die Agenten sehr schnell auf dynamische Umgebungen reagieren. In einem reaktiven Agenten findet kein Prozess zur Entscheidungsfindung statt, da keine explizite Repräsentation von Zielen existiert. Die Ziele, die diese Art von Agenten verfolgt, sind implizit in den Verhaltensregeln enthalten. Trotz des einfachen Aufbaus des einzelnen Agenten kann eine bestimmte Art von Intelligenz als emergentes Verhalten durch die Interaktion mit der Umwelt entstehen [vgl. Wooldridge, 2006, S. 89]. Durch das Zusammenspiel vieler einfacher reaktiver Agenten kann ein sehr komplexes System hervorgehen. Beispielsweise können die Knoten in neuronalen Netzen als eine Art reaktive Agenten betrachtet werden. Ihr Zusammenwirken ruft sehr komplexe Eigenschaften hervor.

Hybride Agenten Diese Art von Agenten ist eine Mischung der beiden oben genannten Architekturen. Dabei ist es das Ziel, die Vorteile beider Architekturen zu vereinen. Soweit es möglich ist, soll das reaktive Verhalten angewendet werden. Das hat zum Vorteil, dass bei unmissverständlichen Zuständen sehr schnell reagiert werden kann. Für komplexes Verhalten, wie zum Beispiel die Kommunikation und Kooperation mit anderen Agenten, wird dann Deliberation verwendet. In einem hybriden System sind die verschiedenen Subsysteme in hierarchischen Schichten angeordnet [vgl. Wooldridge, 2006, S. 97].

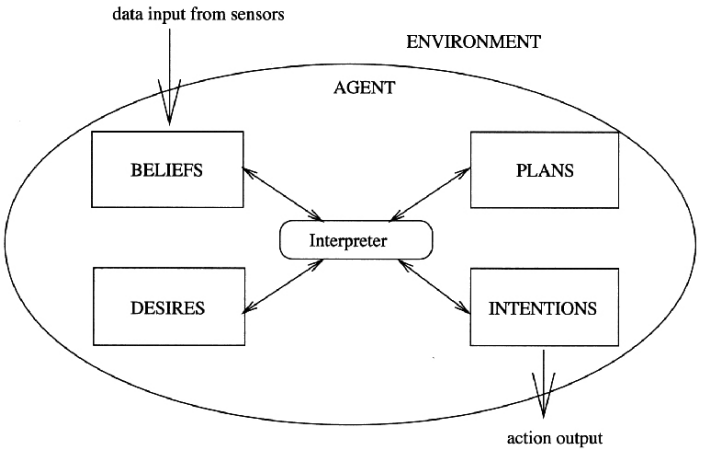

Abbildung 2.2: Procedural Reasoning System (PRS) [vgl. Wooldridge, 2006, S. 83]

In der Masterarbeit sollen die deliberativen Architekturen als Grundlage der Verifikation dienen, da an ihnen die speziellen Eigenschaften von Multiagentensystemen zum Tragen kommen. Reaktive Agenten sind zwar einfacher aufgebaut als deliberative Agenten, zeigen aber aus diesem Grund nicht diese speziellen Eigenschaften. Bei reaktiven Architekturen steht eher das globale Verhalten im Vordergrund. Dieses kann zwar analysiert, aber nur sehr schwer verifiziert werden. Für deliberativen Architekturen soll die symbolische Repräsentation von Wissen und Zielen eine Grundlage für die Verifikation sein.

2.3 Szenario

In diesem Abschnitt soll eine erste Idee vorgestellt werden, wie das Konzept für die Verifikation eines Multiagentensystems aussehen kann. Abbildung 2.3 zeigt den generellen Aufbau dieses Konzepts. Die Idee für ein mögliches Vorgehen zur Verifikation von Multiagenten-

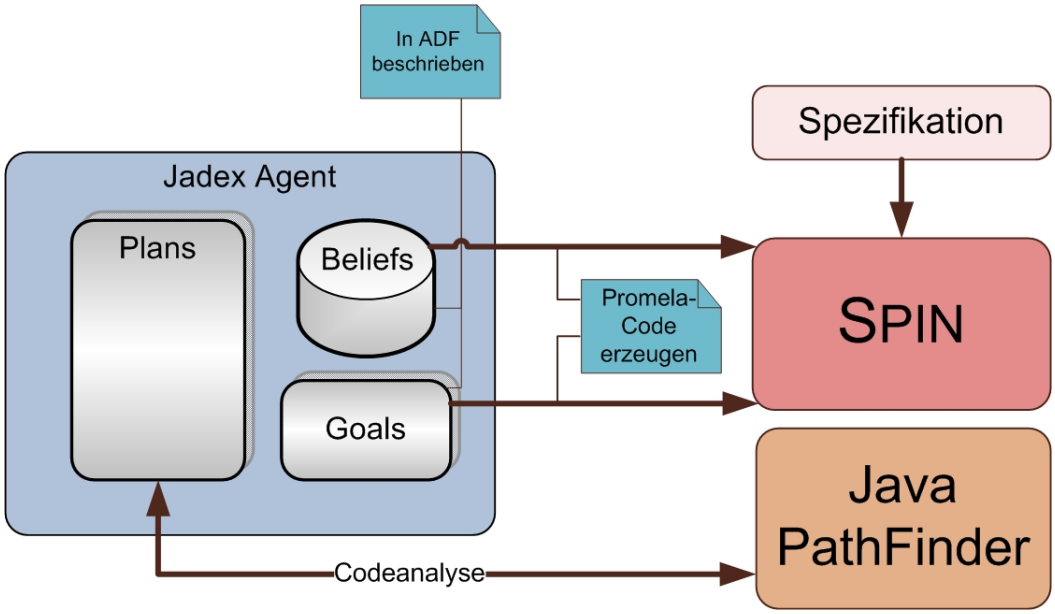

Abbildung 2.3: Konzept für die Verifikation von MAS

systemen ist, auf Jadex aufzubauen. Dieses Agentensystem ermöglicht es BDI-Agenten zu entwickeln und auszuführen (siehe Abschnitt 2.2.2). Die Ziele (Goals) und das initiale Weltwissen (Beliefs) eines Agenten werden im Agent Description File (ADF) in XML deskriptiv formuliert. Die Pläne (Plans) werden in Java entwickelt. Ziel dieses Vorgehens ist es, aus dem ADF PROMELA-Code zu extrahieren. Aus dem ADF können bereits die wichtigsten Informationen extrahiert werden. Dazu gehören die Ziele, die initialen Beliefs und Informationen

über die Struktur und den Inhalt der möglichen eingehenden und ausgehenden Nachrichten. Um den erzeugten PROMELA-Code zu verifizieren muss eine Spezifikation, die die zu verifizierenden Eigenschaften enthält, entwickelt werden. Mit Hilfe von SPIN kann der PROMELA-Code dann gegen die Spezifikation geprüft werden.

Da die Pläne in Jadex in Java entwickelt werden, ist es möglich, sie mit dem Java PathFinder zu analysieren. Der Java PathFinder analysiert den compilierten Java Bytecode, indem alle Ausführungspfade durchlaufen werden. Auf diese Weise können diverse Eigenschaften des Java Programms verifiziert werden. Wahrscheinlich ist eine Anpassung des Java PathFinder an Jadex nötig.

Dieses Vorgehen beschreibt einen möglichen Weg einen einzelnen Jadex-Agenten zu verifizieren. Um ein komplettes Multiagentensystem zu verifizieren, müssen alle Agenten des Systems in PROMELA-Code überführt und in SPIN parallel ausgeführt werden. Um eine vollständige Repräsentation des Multiagentensystems in PROMELA zu haben, müssten zusätzlich auch die Pläne übersetzt werden. Inwieweit dies möglich ist, muss in der Masterarbeit erforscht werden.

3 Risiken

Im folgenden Kapitel werden die Risiken, die bei der Behandlung des Themas auftreten könnten, erläutert und bewertet. Außerdem werden Möglichkeiten für deren Vermeidung aufgezeigt.

Grundsätzlich besteht das Risiko, dass die beiden Themenschwerpunkte, Verifikation und Multiagentensysteme, zu gegensätzlich sind, um vernünftig vereint werden zu können. Multiagentensysteme sind von Natur aus sehr chaotisch und entwickeln emergentes Verhalten. Es ist sehr wahrscheinlich, dass der Zustandsraum eines solchen Systems unendlich ist und dadurch nicht vollständig betrachtet werden kann. Dem gegenüber steht ein sehr formales Verfahren, welches versucht, den gesamten Zustandsraum zu erfassen, um die Gültigkeit bestimmter Eigenschaften zu prüfen. Dabei ist es wichtig, dass dieser Nachweis für alle möglichen Konstellationen (also Zustände) des Systems gilt. Um dieses Risiko zu vermeiden, muss herausgefunden werden, wie sich andere Projekte diesem Problem genährt haben. Es existieren bereits Ergebnisse, in denen die Machbarkeit der Verifikation von Multiagentensystemen belegt wurde. Diese Ergebnisse werden in der Ausarbeitung im Fach AW2 genauer beschrieben. Eine weitere Maßnahme, um dieses Risiko zu minimieren ist, Mechanismen zur Abstraktion anzuwenden, die es ermöglichen, das System in ein Modell mit endlichem Zustandsraum zu überführen.

Ein weiteres Risiko stellt die Komplexität des Themas dar. Um das gewählte Thema umfassend bearbeiten zu können, müssten zu viele Bereiche beleuchtet werden. Dadurch kann es schnell zu umfangreich für eine Masterarbeit werden. Um dieses Risiko ausschließen zu können, dürfen nur die wichtigsten Aspekte betrachtet werden. Dazu ist es nötig, dass das Thema nach intensiver Einarbeitung klar abgegrenzt wird. Außerdem sollten die erarbeiteten Mechanismen zur Verifikation nicht zwangsweise verallgemeinert werden. Es ist bei bestimmten Aspekten ausreichend, sie auf einen konkreten Sachverhalt anzuwenden.

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

Die Verifikation von Multiagentensystemen ist ein sehr interessanter Bereich, der noch relativ schwach ergründet ist. Die starke Verbreitung von Multiagentensystemen und deren Paradigmen erfordern weitere Forschungen auf diesem Gebiet. Die Verifikation dieser Systeme könnte Skeptiker überzeugen und sie noch weiter etablieren. Die bisherigen Ergebnisse anderer Studien und Projekte zeigen, dass die Umsetzung des Problems trotz der bestehenden Risiken prinzipiell möglich ist. Die Umsetzung dieser Vision kann neue Wege eröffnen, sichere Software mit neuartigen Methoden zu entwerfen. Dennoch kann nicht ausgeschlossen werden, dass noch weitere, bis jetzt noch nicht bedachte, Probleme im Laufe der Arbeit auftreten können.

4.2 Ausblick

Das weitere Vorgehen wird es sein, bestehende Ergebnisse weiter zu erfassen und auszuwerten. Um das Thema erfolgreich bearbeiten zu können, muss es noch weiter eingegrenzt und konkretisiert werden. Außerdem müssen noch weitere Techniken zur Verifikation betrachtet werden. Einen interessanten Ansatz bietet in diesem Fall die Sprache *AgentSpeak*. Bei *AgentSpeak* handelt es sich um eine Regelbasierte Programmiersprache für BDI-Agenten. Der formalisierte Aufbau dieser Sprache eröffnet Möglichkeiten zur Verifikation. Es existieren bereits Ansätze, die auf dieses Ziel hinarbeiten. Auf diesem Weg kann der Bereich der direkten Codeverifikation weiter erschlossen werden.

Literaturverzeichnis

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

[Bordini u. a. 2004] 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

[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