Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Milen Koychev

Software-Agenten-Paradigma
am Beispiel einer Simulationsumgebung für Indoornavigation
im Flughafenkontext

Fakultät Technik und Informatik
Stidiendepartment Informatik

Faculty of Engineering and Computer Science
Departmentt of Computer Science

Milen Koychev

Software-Agenten-Paradigma am Beispiel einer Simulationsumgebung für Indoornavigation im Flughafenkontext

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Birgit Wendholt
Zweitgutachter: Prof. Dr. rer.nat. Kai von Luck

Abgegeben am 24. Juni 2008

Milen Koychev

Thema der Masterarbeit

Software-Agenten-Paradigma am Beispiel einer Simulationsumgebung für Indoornavigation im Flughafenkontext

Stichworte

Software-Agenten-Paradigma, Software-Agenten, Software-Agentensimulation, Fußgängernavigation, Fußgängerverhalten, Flughafenkontext

Kurzzusammenfassung

Ziel dieser Arbeit ist das Design und Realisierung einer Simulationsumgebung für Indoornavigation auf Grundlage der Software-Agententechnologie. Die Simulationsumgebung unterstützt den Fortschritt der Systeme auf dem Gebiet der Fußgängernavigation, indem sie die Erprobung dieser Systeme in virtuell abgebildeten, umfangreichen und dynamischen Szenarien ermöglicht. Das Abbilden von Fußgängerverhalten und Fußgängerumgebung wird als Schwerpunkt der Simulation festgelegt.

Milen Koychev

Title of the paper

Paradigm of Software Agents on the Model of a Simulation Environment for Indoor Navigation within the Context of an Airport

Keywords

software-agent paradigm, software-agents, software-agent simulation, pedestrian navigation, pedestrian behavior, airport environment

Abstract

The aim of this work is to design and implement a simulation environment for indoor navigation on the basis of a software-agent technology. The simulation environment assists the development of the systems for navigation of pedestrians, which puts into effect the testing of these systems in visually-presented, extensive and dynamic cases. The simulation lays special emphasis upon the presentation of the pedestrians' behaviour and pedestrians' environment.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einführung

In der Gegenwart leben wir in einer sehr dynamischen Welt, in der die Zeit für den Einzelnen immer knapper zu werden scheint und die Anzahl der Aufgaben und besonders deren Komplexität für das einzelne Individuum rasch wachsen. Der Mensch sieht sich ständig gezwungen, seine Aktivitäten besser zu ordnen und seinen Alltag optimal zu planen, um der Zeitknappheit in der sich ändernden Welt für die eigene Person entgegen wirken zu können. Diese Entwicklung wird durch die Informatik berücksichtigt. Neue Szenarien werden definiert, erforscht und als komplexe IT-Systeme realisiert, in denen die Aufgaben des menschlichen Individuums neu festgelegt werden. Der Mensch verlässt die passive Rolle eines üblichen Benutzers bzw. Beobachters des IT-Systems und nimmt immer mehr die Rolle als aktives Teil eines umfangreichen, technischen Gesamtsystems an. Diese Systeme werden in der Fachliteratur (s. [Barrett01]) als sozio-technische Systeme bezeichnet.

Mit Hilfe solcher sozio-technischen Systeme kann der berufliche und private Alltag für den Menschen durch seine aktive Beteiligung sicherer, flexibler und komfortabler gestaltet werden. In [Ducatel01] werden vier Zukunftsszenarien definiert, darunter „Road Warrior“ und „Digital Me“. Im „Road Warrior“-Szenario wird die Vorstellung eines elektronischen Geschäftsreiseassistenten entwickelt, der alle Formalitäten (z.B. automatische Visabeschaffung) und die Planung (z.B. Hotelbuchung, Mietwagenreservierung) übernimmt sowie die Durchführung der eigentlichen Geschäftsreise unterstützt (z.B. durch vereinfachte Dokumentenverwaltung, automatische Konferenzraumverwaltung usw.). Das „Digital Me“-Szenario entwickelt die Vorstellung von einem virtuellen „Ich“. „Digital Me“ entlastet seinen Inhaber, indem es viele seiner Aufgaben (z.B. Führen von Telefongesprächen, Kontaktplanung, Einkaufsvorbereitungen, Gesundheitsüberwachung) übernimmt. Wie diese Aufgaben zu bewältigen sind, lernt „Digital Me“ von seinem Inhaber selbst, indem er seine Handlungen überwacht und analysiert. Solche Szenarien können dem heutigen Leser futuristisch erscheinen, dennoch geben sie erste Denkanstöße, wie der Mensch durch die Weiterentwicklung der heutigen Technik in nächster Zukunft (laut [Ducatel01] im Jahr 2010) bei seinen täglichen Aufgaben unterstützt werden kann. Wenn wir die laufenden Projekte des sechsten Rahmenprogramms der Europäischen Union betrachten (s. [cordis.europa.eu]), werden wir feststellen, dass viele Aspekte der bei [Ducatel01] dargestellten Szenarien schon in der heutigen Realität zum Einsatz kommen. Das Projekt „WEARIT@WORK“ (s. [wearitatwork.com]) verwirklicht die Vision eines tragbaren persönlichen elektronischen Assistenten, der seinen Besitzer bei den alltäglichen Aufgaben im Berufsleben unterstützt bzw. entlastet. Der Benutzer dieses Assistenten kann mehrere Aufgaben gleichzeitig abarbeiten, ohne sich explizit auf die Bedienung des

elektronischen Helfers zu konzentrieren. Der Assistent ist in die Kleidung des Benutzers integriert, er beobachtet seinen Inhaber durch Sensoren und stellt selbst aus dem Kontext die Aktivitätsliste zusammen (z.B. Präsentation vorbereiten, Dokument versenden usw.). Das Projekt „HIGHWAY“ (s. [ist-highway.org]) hat das Ziel, durch intelligente Fahrzeugnavigation die Sicherheit des Straßenverkehrs zu erhöhen. Kritische Verkehrssituationen (z.B. Stau, Straßenschäden, bevorstehender Unfall) werden erkannt, der Autofahrer und das Fahrzeug werden auf solche Situationen vorbereitet, indem z.B. ankommende Anrufe verspätet gemeldet werden, Geschwindigkeit automatisch im Vorfeld reduziert wird, Korrektur der Fahrtrichtung eingeleitet wird usw. Dadurch wird das Unfallrisiko minimiert. Neben dem „HIGWAY“-Projekt sind viele Forschungsarbeiten auf dem Gebiet der Fahrzeugnavigation durchgeführt worden. Ergebnis dieser Forschungsarbeiten ist es, dass Fahrzeugnavigationssysteme heutzutage erfolgreich im Straßenverkehr eingesetzt werden. Diese Systeme führen den Fahrzeugfahrer mittels Outdoor-Navigation sicher durch das Straßennetz.

1.1 Motivation

In dieser Arbeit wird der nächste Schritt betrachtet, der von der Forschung im Bereich „Navigation“ gegangen wird - Systeme für Fußgängernavigation. Auf diesem Gebiet sind in den letzten Jahren verschiedene Navigationssysteme entstanden (s. [Meng05]). Trotz der zahlreichen Ergebnisse erscheint eine weitere Entwicklung laut [Meng05] und [Schwinger05] notwendig. Die beiden Quellen bemängeln, dass nach abgeschlossener Berechnung einer Route die bestehenden Systeme nur unzureichend oder gar nicht auf Abweichungen von der Route oder Kontextänderungen reagieren. Diese immer noch vorhandene Schwachstelle der Systeme ist durch den Hauptunterschied der Fußgängernavigation im Vergleich zur Fahrzeugnavigation zu erklären. Der Fußgänger bewegt sich nicht auf genau vordefinierten Bahnen (wie z.B. dem Straßennetz bei Fahrzeugnavigation). Abweichungen von dem vorgegebenen Weg sind oftmals nicht einfach festzustellen. Eine weitere schwerwiegende Herausforderung bedeutet die Tatsache, dass der Fußgänger nicht nur durch Gebiete mittels Outdoor-Navigation sondern auch durch Gebäude mittels Indoor-Navigation geführt werden muss (s. [Gregor06]).

Diese Arbeit greift die oben beschriebenen Verbesserungsvorschläge von [Meng05] und [Schwinger05] auf und unterstützt die Entwicklung eines verbesserten Indoor-Fußgängernavigationssystems in Form einer Simulationsumgebung. Routenabweichungen und Kontextänderungen werden erkannt und aktiv in die Fußgängernavigation mit einbezogen, indem sie in bestimmten dynamisch definierten Grenzen toleriert werden. Werden diese Grenzen

überschritten, können Ablaufpläne und Routen neu berechnet bzw. optimiert werden. Um genaue Aussagen über ein solches System treffen zu können und um die Qualität des Systems während des Entwicklungsprozesses zu sichern, muss die Möglichkeit zu umfangreichen Systemtests gegeben sein. Da es sich bei einem Indoor-Fußgängernavigationssystem um ein komplexes sozio-technisches System handelt, erfordern solche Tests aufwändige Hardware-Installationen (beispielsweise für Indoor-Positionsbestimmung), das Generieren bzw. Simulieren von Kontext und Kontextänderungen, beliebig konfigurierbare Räumlichkeiten und den Einsatz von mehreren Testpersonen (Fußgängern). Eine Testumgebung mit geforderten Parametern aufzustellen, ist kostspielig und nicht immer möglich (s. [Koychev07], [Kutak07], [Napitupulu07], [Schumann07]). Aus diesem Grund wird der Fokus dieser Arbeit auf die Entwicklung einer Software-Simulationsumgebung für das oben genannte sozio-technische Fußgängernavigationssystem gelegt, in deren Rahmen sich umfangreiche Tests durchführen lassen.

1.2 Ziele der Arbeit

Die konkreten Ziele dieser Arbeit sind:

- Entwicklung eines Modells zur Simulation des Indoor-Kontextes sowie zur Simulation des menschlichen Verhaltens innerhalb von Gebäuden,
- Entwurf einer Softwarearchitektur und Sprachspezifikation für die geforderte Simulationsumgebung,
- Realisierung der Softwarearchitektur und Abbildung der relevanten Modelle mittels eines Prototyps. Der Prototyp stellt ein Rahmenwerk zur Verfügung, auf dessen Basis eine Simulationsumgebung für das im Kapitell 1.1 beschriebene Szenario erstellt werden kann.

Im Verlauf der Arbeit werden die oben genannten Aufgaben und Ziele detaillierter analysiert. Die Analyse und der Weg, wie die gestellten Aufgaben zu lösen und die angestrebten Ziele zu erreichen sind, werden in weiteren Kapiteln beschrieben.

1.3 Gliederung der Arbeit

Diese Masterarbeit besteht aus sieben Kapiteln. Nach der Einleitung werden im Kapitel „Grundlagen“ Konzepte zu den Themen Agentensimulation, Software-Agenten und standardisierte Software-Agentensysteme vorgestellt. Im Kapitel „Vergleichbare Arbeiten“ wird die Arbeit im Gesamtkontext der Wissenschaft eingeordnet. Dazu werden vorhandene wissenschaftliche Arbeiten und Konzepte zum Abbilden von Gebäudestrukturen als Basis für das Simulieren von Fußgängerverhalten beschrieben. Kapitel „Anforderungsanalyse“ ermittelt anhand eines konkreten Szenarios die

grundlegenden funktionalen wie nicht funktionalen Anforderungen an eine Simulationsumgebung für Indoornavigation im Flughafenkontext. Kapitel „Konzeption“ wählt eine geeignete Entwicklungsplattform und stellt eine tragfähige Architektur der künftigen Simulationsumgebung vor. Im Kapitel „Realisierung“ werden die Konzepte der Simulationsumgebung für Indoornavigation durch Entwicklung eines Prototyps veranschaulicht. Anschließend wird die Implementierung des Prototyps bewertet. Im abschließenden Kapitel „Zusammenfassung und Ausblick“ werden die Ergebnisse der Arbeit kurz zusammenfassend dargestellt und ein Ausblick auf zukünftige Entwicklungen auf dem Gebiet der Simulation von sozio-technischen Systemen geboten.

2. Grundlagen

Dieses Kapitel stellt die Grundlagen zu den Themen Agentensimulation, Software-Agenten und standardisierte Software-Agentensysteme vor. und präsentiert relevante Design-Ansätze, Technologien, Architekturen und den FIPA-Standard (s. [fipa.org]) zur Entwicklung von Softwaresystemen innerhalb des Software-Agentenparadigmas. Dadurch wird eine technische Basis für das Erreichen der definierten Ziele dieser Masterarbeit aufgebaut.

2.1 Agentensimulation

Voraussetzung für jede erfolgreiche Simulationsstudie ist es, dass das gewählte Modell den für das Simulationsziel relevanten Teil der Wirklichkeit ausreichend genau wiederspeigelt. Es bestehen verschiedene Paradigmen und Techniken zur Formulierung eines Modells. Die Agentensimulation eignet sich als eine moderne Simulationstechnik zur Abbildung und Untersuchung von komplexen dynamischen Systemen, deren Akteure (meist Menschen) sowohl soziales als auch autonomes Verhalten aufweisen (s. [Mengistu07]). Beispiele für solche Systeme sind der Aktienmarkt inklusive der Käufer- und Verkäufer-Verhalten, Fußgänger in Gebäuden, Populationen der Tierwelt usw. Die Agentensimulation erweitert die Standardsimulationstechniken in Bereichen wie Wirtschaftslehre, Soziologie, Biologie, Ökologie usw., indem sie einen Einblick in soziale Phänomene durch deren Abbildung mittels Agenten-basierten Rechenmodellen ermöglicht. Die Agentenmodellierung erleichtert die Untersuchung von sozialen Phänomenen, bei denen das unterschiedliche Verhalten der beteiligten Individuen und die Interaktion unter ihnen zur Entstehung von Mustern und Strukturen führen können. Das reale menschliche Verhalten wird durch das virtuelle Verhalten der Software-Agenten abgebildet. Die Agenten können (ähnlich wie Menschen in der realen Welt) eigene Entscheidungen treffen, mit anderen Agenten kommunizieren, soziale Strukturen bilden und auf Veränderungen der Außenwelt reagieren.

2.2 Software-Agenten

Wie im Kapitel 2.1 beschrieben ist, ist die Modellierung mittels Agenten ein wichtiges Paradigma in der Softwareentwicklung. Den Grundbaustein dieses Paradigmas bilden die sogenannten Agenten. Daher müssen, die Fragen, was ist ein Agent und wie Agentenarchitekturen aufgebaut werden können, geklärt werden, was im Folgenden geschehen soll.

2.2.1 Was ist ein Agent?

Im Allgemeinen wird ein Agent als ein Vertreter angesehen, der in fremdem Auftrag selbständig handelt (s [Burkhard98]). Ein solcher Agent wird dabei anhand seiner eigenen Problemlösungskompetenz entscheiden, wie genau und unter Einsatz welcher der ihm verfügbaren Ressourcen er tätig wird. Der Auftraggeber ist an dem Ergebnis und nicht am Weg zu diesem Ergebnis interessiert. Diese Methode der Delegierung wird in der Informatik durch das Paradigma der Software-Agenten abgebildet. Software-Agenten sind in erster Linie Softwarekomponenten, die eine gewünschte Aktion im Auftrag ausüben und deren Ergebnis an den Auftraggeber zurückmelden. Sie agieren dabei selbständig und reagieren auf Änderungen ihrer Umgebung. Die Software-Agenten versuchen, die Ziele ihres Auftraggebers in einem komplexen, dynamischen Umfeld umzusetzen. Im Gegensatz zu konventioneller Software, die durch direkte Manipulation gesteuert wird, arbeiten Software-Agenten asynchron zu ihren Auftraggebern, d.h. ohne direkte Intervention des Nutzers. Darüber hinaus sind sie in der Lage, aus der Beobachtung der Umgebung, der Analyse des Problemlösungsprozesses oder aus den Ergebnissen ihrer Aktionen ihr künftiges Verhalten an die Präferenzen des Benutzers bzw. an die Gegebenheiten der Umgebung anzupassen und somit einen gewissen Lernprozess abzubilden.

2.2.2 Agentenarchitekturen

Mit der Entwicklung des Software-Agentenparadigmas haben sich verschiedene Architekturen für Agentensysteme etabliert (s. [Bellifemine07]). Zu den wichtigsten Architekturen zählen:

- **Logik-basierte Architekturen:** Diese Architekturform hat ihren Ursprung in wissensbasierten Systemen und beruht darauf, dass die Realität in einer Art Expertensystem zentral deklarativ abgebildet und mittels regelbasierten Mechanismen verändert wird. Das Expertensystem bzw. die beteiligten Software-Agenten treffen auf Grundlage der abgebildeten Realität logische Schlüsse und Entscheidungen, wie sich das weitere Verhalten des Gesamtsystems entwickeln soll. Der Vorteil eines solchen Ansatzes besteht im unkomplizierten Charakter der gewählten statischen Realität und der relativen Einfachheit, diese komplett abzubilden bzw. zu berechnen. Die Logik in einer solchen Repräsentation ist für den Menschen (z.B. der Systementwickler oder Systembenutzer) verhältnismäßig gut durchschaubar. Bei der Abbildung von dynamischen und komplexen Zusammenhängen stoßen die logik-basierten Architekturen an ihre Grenzen, da solche Zusammenhänge sich umständlich und in bestimmten Fällen unvollständig deklarativ beschrieben lassen.
- **Reaktive Architekturen:** Bei dieser Architekturform werden sehr einfache hierarchisch angeordnete Agenten für die Lösung eines Problems eingesetzt (anstelle umfassender

symbolischer bzw. logischer Abbildungen der Realität und Deklaration von regelbasierten Mechanismen). Dabei stehen die Agenten mit Hilfe von Sensoren ständig mit der Realität in Kontakt und reagieren auf Veränderungen. Auf diese Weise wird das weitere Verhalten des Systems mittels einer direkten Überführung von Sensor-Information in Aktion bzw. Reaktion eines Agenten bestimmt. Eine solche Architektur weist mehrere Vorteile gegenüber der Logik-basierten Architektur auf. Die reaktiven Systeme sind einfach aufgebaut, trotzdem kann aus dieser Einfachheit ein komplexes Verhalten konstruiert werden. Eine verstärkte Form der Aufgabenteilung wird ermöglicht, indem verschiedene Module (Gruppen von Agenten) autonom an jeweils nur einer Aufgabe arbeiten. Es gibt kein Modell für ein globales Verhalten. Die reaktiven Systeme arbeiten sehr viel näher an der Realität, da sie ohne komplexe Umweltmodelle auskommen. Der Hauptnachteil reaktiver Agentensysteme ist jedoch die Schwierigkeit, wegen der zahlreichen im System aktiven Agenten das Verhalten des Gesamtsystems vorherzusagen. Daraus resultiert die Problem, das System so zu entwickeln, dass es Ziele in vordefinierter Art und Weise erreicht.

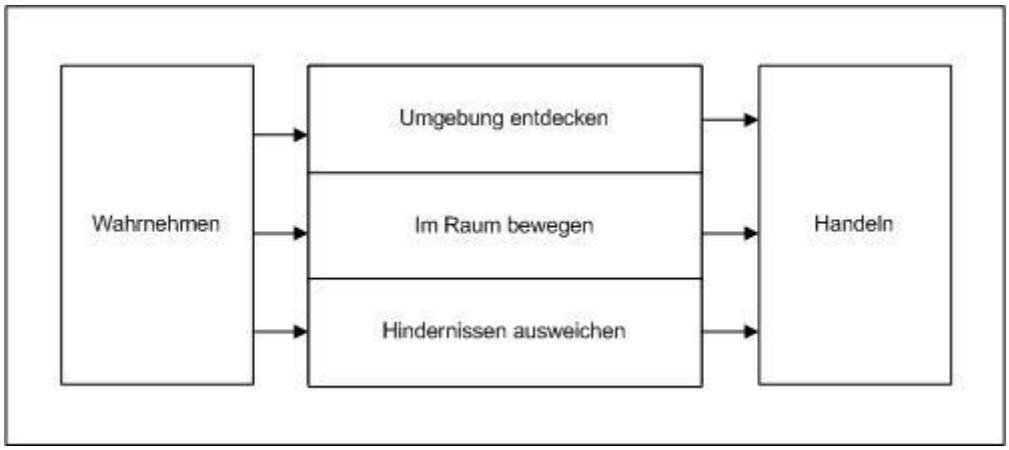

Abbildung 2-1: Reaktive Architektur nach [Brooks91]

Ein Beispiel für reaktive Architektur ist die Subsumption-Architektur (s. [Brooks91]) von Rodney Brooks für Navigation mobiler Roboter. Die Architektur ist in der Abbildung 2-1 vorgestellt. Roboter nehmen durch Sensoren die Umgebung wahr. Die Sensor-Information wird dann durch die drei Agenten-Gruppen (s. Abbildung 2-1) verarbeitet, die für das Gesamtverhalten des Roboters verantwortlich sind. Dadurch werden entsprechende Aktivitäten ausgelöst.

- **BDI-Architekturen:** Die Belief-Desire-Intention-Architekturen basieren auf Theorien der Entscheidungsfindung beim Menschen (s. [Bellifemine07]). Die BDI-Systeme stellen eine Weiterentwicklung der reaktiven Systeme dar. BDI-Agenten sind mit Sensoren ausgestattet

und nehmen die Realität über diese Sensoren wahr. Darüber hinaus können BDI-Agenten über Aktionen die Realität verändern. Welche Aktionen durch welche Sensor-Informationen und auf welche Weise genau ausgelöst werden, entscheidet der Interpreter eines BDI-Agenten. Der Interpreter arbeitet auf Basis folgender drei Strukturen: Beliefs (Wissen), Desires (Ziele), Intentions (Absichten) (s. Abbildung 2-2).

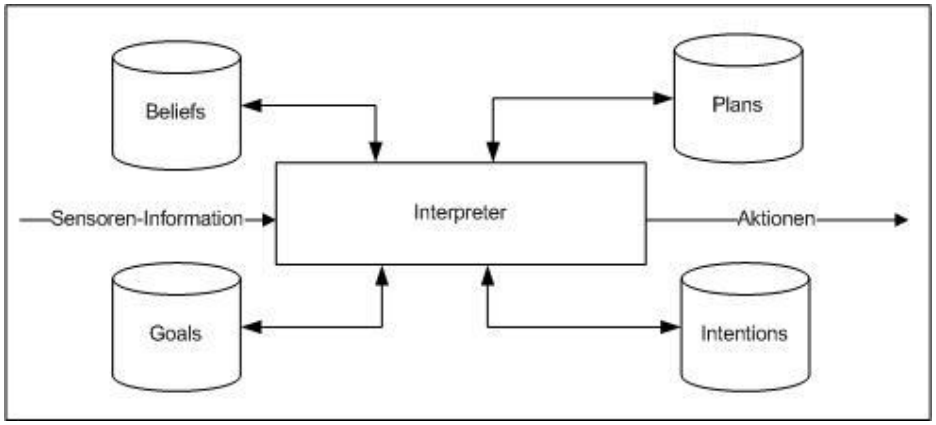

Abbildung 2-2: BDI-Architektur nach [Bellifemine07]

Beliefs bilden die Information über die Realität ab, in der sich der Agent befindet. Die Information wird in einer Wissensdatenbank gespeichert und muss ständig aktualisiert werden, beeinflusst von der Wahrnehmung des Agenten und durch interne Schlussfolgerungen. So wird gewährleistet, dass das Modell der Realität weitestmöglich entspricht.

Desires enthalten die Ziele eines Agenten, die sein weiteres Verhalten beeinflussen. Aus einer Menge von Optionen wird ein Ziel ausgewählt, das über eine bestimmte Zeit verfolgt wird. Sollte eine Aktion während der „Zielverfolgung“ fehlschlagen, kann das ausgewählte Ziel trotzdem weiter verfolgt werden, z.B. durch einen alternativen Lösungsansatz oder einen späteren erneuten Versuch.

Intentions enthalten die aktiven Pläne eines BDI-Agenten. Diese werden in einer hierarchisch organisierten Plandatenbank verwaltet. Darüber hinaus werden die vorhandenen Pläne auf die mögliche Annäherung an das gewünschte Ziel analysiert und der besttreffende Plan als nächster Schritt gewählt. Durch die hierarchisch organisierten Pläne findet bei BDI-Agenten das Treffen von Entscheidungen und Gewinnen von Rückschlüssen statt: zuerst wird ein Plan ausgewählt, der direkt vom Startzustand zum Zielzustand führt. Ein Plan besteht im Normalfall aus Teilzielen, für die dann jeweils ein eigener, hierarchisch untergeordneter Plan

ausgewählt wird. Dies passiert bis zu Ebene, wo Pläne nur aus elementaren atomaren Aktionen bestehen (z.B. „Bewegung anhalten“, „links abbiegen“ usw.).

- **Schicht-Architekturen:** Diese Architekturkategorie ermöglicht es, die Vorteile mehrerer Architektur-Arten (z.B. Logik-basierte und reaktive) miteinander zu kombinieren (s. [Bellifemine07]). Dazu wird das Agentensystem in Schichten aufgeteilt. Jede Schicht des Systems wird nach einer bestimmten Architektur-Art aufgebaut. Bei den Schicht-Architekturen spielt die Frage nach dem Kontrollfluss eine zentrale Rolle. Es werden zwei Typen des Kontrollflusses bei dieser Agentenarchitektur unterschieden: horizontal und

  Bei horizontalem Kontrollfluss erhalten alle Schichten direkt Sensoren-Informationen und dürfen die Realität durch Auslösen von Aktionen verändern (s. Abbildung 2-3). Bei diesem Szenario agiert jede Schicht des Systems als ein selbständiger Agent. Der Vorteil dabei ist das einfache Design. Z.B. können N Schichten durch N Agenten mit unterschiedlichem Verhalten modelliert werden. Nachteil der Architektur ist es, dass jede Schicht selbständig auf Sensor-Informationen reagiert. Dies kann zu Situationen führen, in denen mehrere (im schlechtesten Fall N) unterschiedliche Reaktionen auf einen Input generiert werden und das System aus diesem Grund einen inkonsistenten Zustand annimmt. Für solche Fälle muss die Architektur einen Vermittler vorsehen, der diese Situation auflösen kann. Ein weiterer Nachteil des horizontalen Kontrollflusses ist die hohe Anzahl (M^N, M ist die Anzahl Aktionen pro Schicht und N ist die Anzahl der Schichten im System) an maximal möglichen Schichten-Interaktionen (s. Abbildung 2-3).

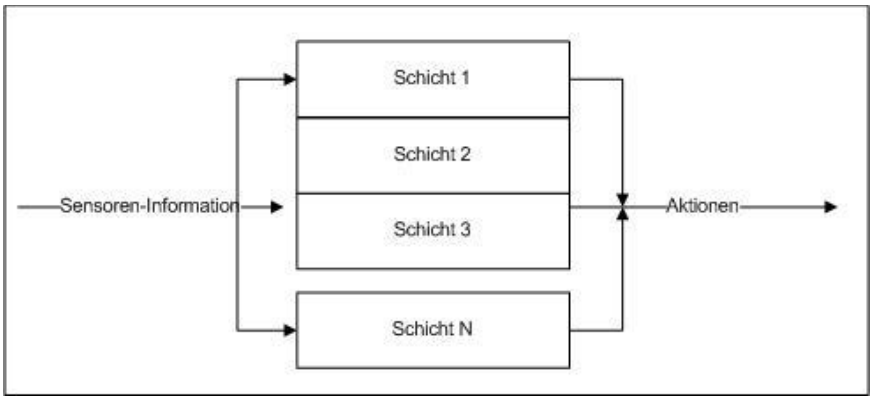

Abbildung 2-3: Schichten-Architektur mit horizontalem Kontrollfluss nach [Bellifemine07]

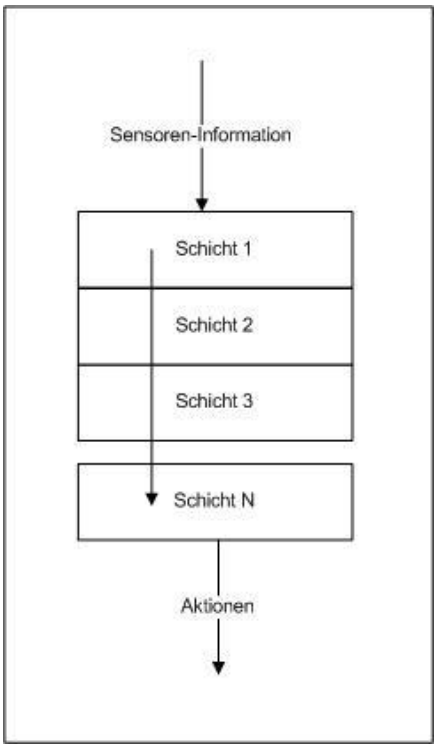

Abbildung 2-4: Schichten-Architektur mit vertikalem Kontrollfluss nach [Bellifemine07]

Der vertikale Kontrollfluss begrenzt die Anzahl der möglichen Schichten-Interaktionen auf ein Maximum von $2 * M(N-1)$, indem jede Schicht nur mit den Nachbarschichten des Kontrollflusses kommunizieren darf (s. Abbildung 2-4). Dabei wird in diesem Szenario die Sensoren-Information nur von der obersten Schicht aufgenommen und nur die unterste Schicht kann Aktionen auslösen. Der Nachteil solcher Architektur ist es, dass sie nicht fehlertolerant ist. Sollte eine der Schichten ausfallen, führt dies zum Ausfall des Gesamtsystems.

Die Architektur legt die Hauptmerkmale des jeweiligen Agentensystems festgelegt. Dennoch bedarf es zur näheren Beschreibung eines Agentensystems weiterer Spezifikationsmaßnahmen. Sie werden in den folgenden Kapiteln der Arbeit ausführlicher vorgestellt.

2.3 Agentensysteme nach FIPA

Die FIPA-Organisation (Foundation for Intelligent Physical Agent) verfolgt das Ziel, agentenbasierte Technologie zu standardisieren und so die Interoperabilität der Agentensysteme sicherzustellen. Die Organisation definiert mehrere Spezifikationen, die sich in die folgenden Kategorien einordnen lassen (s. [fipa.org]).

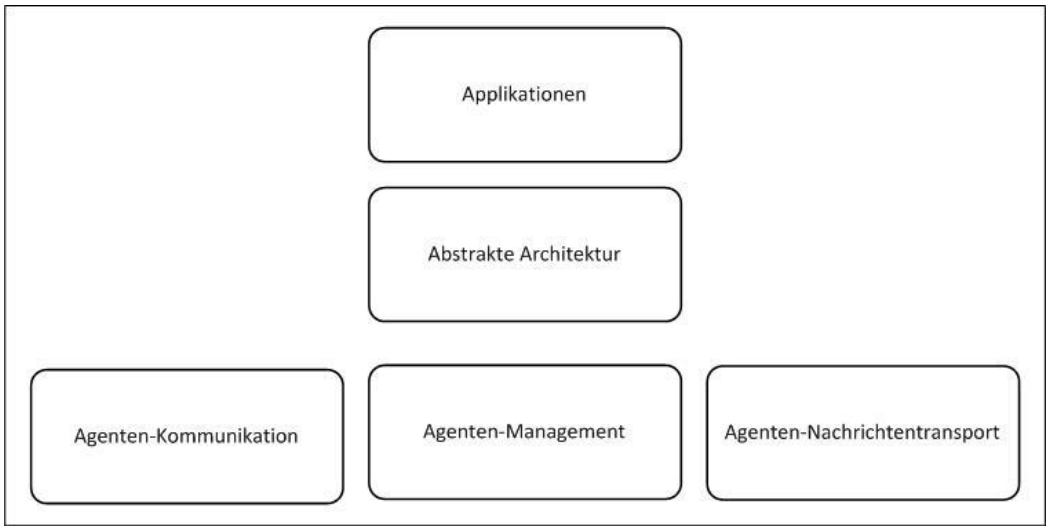

Abbildung 2-5: FIPA-Spezifikationskategorien nach [fipa.org]

Die Kategoerie „Applikationen“ gliedert auf einer abstrakten Ebene den Einsatz von FIPA-standardisierten Agentensystemen in verschiedenen Anwendungsbereichen auf. Die Spezifikationen enthalten Ontologie-Definitionen sowie Dienstbeschreibungen für den jeweiligen Anwendungsbereich. Die Kategorie „Abstrakte Architektur“ spezifiziert die abstrakten Entitäten und deren Zusammenhänge, die notwendig sind, um Agentenfunktionalität nach FIPA realisieren zu können. Die Kategorie „Agenten-Kommunikation“ ist das „Herz“ der FIPA-Spezifikationen. Die Kategorie definiert den Verlauf des Nachrichtenaustausches in Agentensystemen. Die Kategorie „Agenten-Management“ befasst sich mit der Steuerung und Kontrolle der Agenten innerhalb eines Agentensystems sowie über seine die Grenzen hinaus. Die Kategorie „Agenten-Nachrichtentransport“ regelt die Übertragung der Nachrichten, die zwischen Agenten und Agentensystemen mittels unterschiedlichen Übertragungsprotokollen ausgetauscht werden.

Im Weiteren werden die wesentlichen Details der FIPA-Spezifikationen vorgestellt.

2.3.1 Abstrakte Architektur

Die durch FIPA spezifizierte abstrakte Architektur entwickelt das Leitbild eines Agentensystems, bei dem die Interoperabilität im Vordergrund steht. Es werden nur die notwendigsten Entitäten definiert und deren Zusammenhänge beschrieben. Die abstrakte Architektur kann dann mittels unterschiedlicher Technologien konkret realisiert und um zusätzliche Funktionalitäten erweitert werden. Die Interoperabilität der auf dieser Basis entstandenen Agentensysteme ist durch den gemeinsamen Kern sichergestellt.

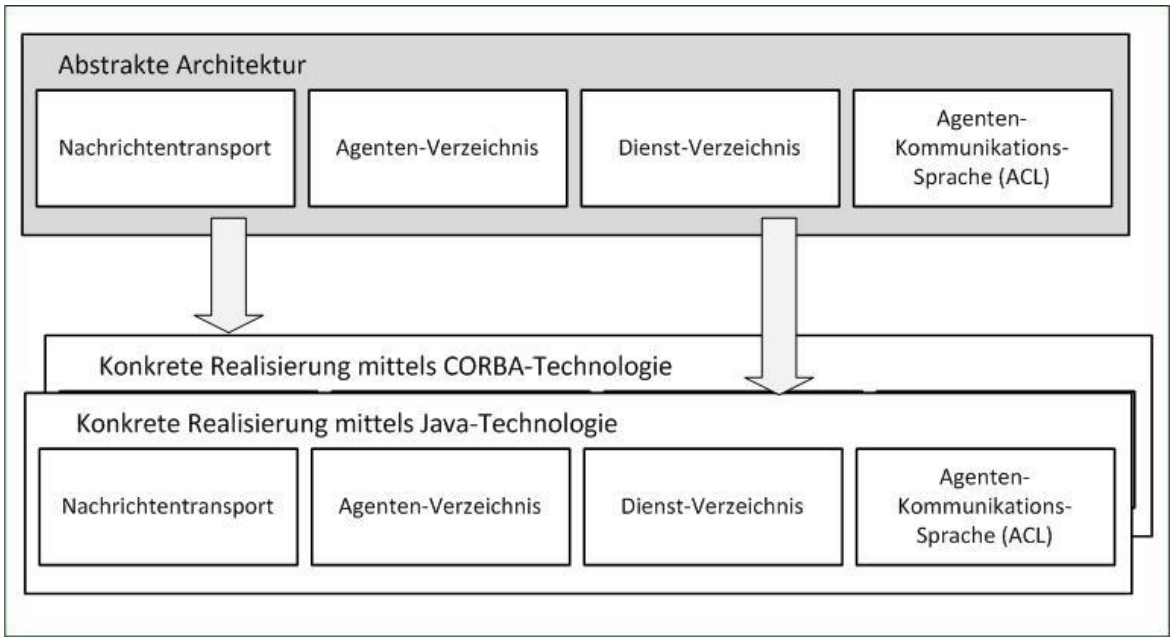

Abbildung 2-6: Abstrakte FIPA-Architektur und konkrete Realisierungen nach [fipa.org]

Die in der Abbildung 2-6 dargestellte Architektur wird im Folgenden näher erläutert. Die in der FIPA-Architektur vorgesehenen Prozesse und grundsätzliche Abläufe werden beschrieben, so dass ein einfacher Überblick über die FIPA-standardisierten Systeme entsteht. Besonderer Wert wird auf die für diese Arbeit relevanten Aspekte der FIPA-Architektur gelegt. Eine Ausführliche Beschreibung der Architektur kann der Quelle [fipa.org] entnommen werden.

2.3.1.1 Agenten und Dienste

Dienst und Agent sind in der FIPA-Architektur zwei unterschiedliche Entitäten, die sehr eng miteinander agieren. Aus diesem Grund ist der Unterschied zwischen Dienst und Agent nicht immer deutlich. Aus Sicht der FIPA-Architektur ist das Hauptmerkmal eines Dienstes seine ständige Erreichbarkeit im System und seine Bereitschaft, an ihn gestellte Anfragen zu bearbeiten. Ein Agent dagegen ist nicht zwingend immer im System erreichbar und hat die Freiheit, über die Ausführung der ihm gestellten Anfragen selbst zu entscheiden ggf. diese auch abzulehnen.

Die Dienste haben im FIPA-Kontext haben die Rolle, unterstützende Funktionalität für die Agenten zur Verfügung zu stellen. Der FIPA-Standard sieht neben den Standard-Diensten wie Agenten-Verzeichnis-Dienste (zur Auflistung der im System vorhandenen Agenten) und Nachrichten-Transport-Dienste (zur Beschreibung der Kommunikationsmöglichkeiten im System) auch einen Dienste-Verzeichnis-Dienst (zur Auflistung der im System vorhandenen Dienste) vor.

Die abstrakte Architektur trifft keine Aussage, wie die oben genannten Dienste realisiert werden müssen. Sie können sowohl als Agenten als auch als Extrasoftware ohne Agentenfunktionalität im System implementiert werden. Wenn Dienste als Agenten realisiert werden, entsteht eine besondere „Zusammensetzung“ – der sogenannte Dienst-Agent. Diese Agenten sind begrenzter in ihrem Verhalten als die Standardagenten eines Agentensystems. Die Dienst-Agenten müssen als vollwertige Dienste im System auftreten und somit, wie oben beschrieben, immer erreichbar sein und die angebotenen Funktionalitäten müssen immer zur Verfügung gestellt werden können. Bei solcher Realisierung können Probleme beim Suchen nach Diensten und der Dienstkommunikation auftreten.

Bevor Lösungsansätze für diese Probleme vorgestellt werden können, werden im Weiteren die Hauptprozesse und Entitäten eines FIPA-Systems beschrieben.

2.3.1.2 Agentenstart

Wenn ein Agent gestartet wird, muss er mit Grunddiensten (Dienste-Verzeichnis-Dienste, Agent-Verzeichnis-Dienste, Nachrichten-Transport-Dienste) des Systems verbunden werden. Erst dadurch wird seine Ausführung im Gesamtkontext möglich. Diese Aufgabe wird von einem speziellen Objekt im System, dem Wurzeldienst, übernommen. Dieser liefert dem neugestarteten Agenten entweder die Beschreibung, wie die Grunddienste in der Umgebung zu erreichen sind, oder er leitet den Agenten an einen anderen Dienst, der diese Aufgabe erfüllen kann.

2.3.1.3 Agenten-Verzeichnis-Dienste

Die Aufgabe der Agenten-Verzeichnis-Dienste ist es, eine Plattform zu schaffen, auf der sich Agenten mit ihrer Beschreibung registrieren können. Andere Agenten können dann diese Plattform durchsuchen, und auf Grund der oben genannten Beschreibung Agenten finden, mit denen sie agieren möchten. Wenn ein Agent sich in einem FIPA-Agentensystem registriert, muss er sich zunächst an einen Transport-Typen binden. Dies kann über die Nachrichten-Transport-Dienste geschehen oder der Agent kann sich selbst für bestimmte Nachrichten-Typen (z.B. mittels Object-Request-Brokers [ORB] oder Remote-Method-Invocation [RMI]) registrieren. Nach dieser Aktion kann der Agent über einen oder mehrere Transport- bzw. Nachrichten-Typen adressiert werden. Danach muss der Agent seine Existenz im System bekanntgeben, indem er sich bei den Agenten-Verzeichnis-Diensten mit den zugehörigen Informationen (s. Tabelle 2-1) registriert. Sollte ein Agent nach weiteren Agenten im System suchen, schickt er seine Anfrage an die Agenten-Verzeichnis-Dienste. Entsprechend der Anfrage werden an den Agenten die passenden Einträge aus dem Agenten-Verzeichnis weitergeleitet. Damit ist der Agent imstande, sich auf Grund der Verzeichnis-Daten zu

entscheiden, welche Agenten und wie er diese kontaktieren kann, um die eigenen Aufgaben abarbeiten zu können.

Vom Agenten-Verzeichnis müssen mindestens folgende Informationen in Form von Schlüssel-Werte-Paaren verwaltet werden:

Agenten-Name	Globaleindeutiger Agenten-Name, der zur Identifikation des Agenten dient.
Agent-Lokator	ermöglicht den Zugriff auf den Agenten, dabei enthält der Lokator eine oder mehrere Transport-Beschreibungen. Jede Transport-Beschreibung enthält den Transport-Typ, eine spezifische Transport-Adresse und null oder mehrere spezifische Transport-Eigenschaften, die die Kommunikation mit dem Agenten beschreiben.

Tabelle 2-1: Agenten-Verzeichnis-Daten nach [fipa.org]

Darüber hinaus können auch zusätzliche Informationen vom Agenten-Verzeichnis verwaltet werden. Beispiel dafür sind Dienste, die ein Agent anbietet, oder Kosten, die beim Benutzen des Agenten entstehen können, auch weitere Restriktionen, die die Nutzung des Agenten genauer spezifizieren.

2.3.1.4 Dienst-Verzeichnis-Dienste

Die Aufgabe der Dienst-Verzeichnis-Dienste ist es, eine Plattform zu schaffen, auf der Agenten und Dienste nach Diensten suchen können. Das Dienste-Verzeichnis kann mit dem Agenten-Verzeichnis verglichen werden, doch ist der Schwerpunkt bei diesem Verzeichnis auf die Dienste gelegt - es verwaltet im Gegensatz zum Agenten-Verzeichnis ausschließlich Dienst-Informationen (s. Tabelle 2-2) und keine Agenten-Informationen. Um den Unterschied zu verdeutlichen, wird nochmals betont, dass ein Agent mehrere Dienste anbieten kann. Dies führt zu einem Eintrag im Agenten-Verzeichnis sowie zu mehreren in dem Dienst-Verzeichnis.

Dienst-Name	Globaleindeutiger Dienst-Name, der zu Identifikation des Dienstes eingesetzt wird.
Dienst-Typ	Kategorisierte Dienstbeschreibung (z.B. „Drucker-Dienst“).
Dienst-Lokator	ermöglicht den Zugriff auf den Dienst, dabei enthält der Lokator Dienst-Signatur-Typ, Dienst-Signatur sowie die Dienstadresse, über die der Dienst erreicht werden kann.

Tabelle 2-2: Dienst-Verzeichnis-Daten nach [fipa.org]

Neben den in der Tabelle 2-2 aufgelisteten Informationen können auch zusätzliche Daten vom Dienst-Verzeichnis verwaltet werden z.B. Kosten, die beim Benutzen des Dienstes entstehen können, auch weitere Restriktionen, die die Nutzung des Dienstes genauer spezifizieren.

2.3.1.5 Nachrichtentransport

In FIPA-Systemen kommunizieren die Agenten untereinander mittels Versenden von Nachrichten. Die Kommunikation wird durch drei fundamentale Aspekte beschrieben: Nachrichten-Struktur, Nachrichten-Transport und Nachrichten-Repräsentation.

Die Nachrichten-Struktur basiert auf Schlüssel-Wertepaaren, deren Inhalt in einer Agenten-Kommunikationssprache (ACL) wie z.B. FIPA-ACL (s. [fipa.org]) definiert wird. Die Beschreibung des Inhaltes wird in einer Content-Sprache wie z.B. FIPA-SL (s. [fipa.org]) vorgenommen. Zusätzlich kann der Nachrichteninhalt auf definierte Ontologien verweisen. Weiterhin enthalten Nachrichten Absender und Empfänger der Nachricht in Form von eindeutigen Agenten-Namen. Eine Nachricht muss immer einen Absender und null oder mehrere Empfänger enthalten. Im Falle von null Empfänger, kann die Nachricht als Broadcast in Ad-hoc-Netzwerken betrachtet werden. Eine Nachricht kann rekursiv weitere Nachrichten enthalten.

Der Nachrichten-Transport-Mechanismus funktioniert folgendermaßen – jede Nachricht, die ein Agent verschickt, wird in eine spezielle Transport-Nachricht verpackt. Dabei wird die ursprüngliche Nachricht in ein für das jeweilige Transport-Medium optimales Format transformiert. Neben der ursprünglichen Nachricht werden auch weitere Informationen wie z.B. Transport-Beschreibung und Kodierungs-Repräsentation in die Transport-Nachricht verpackt. Erst dann kann eine Nachricht in Form einer Transport-Nachricht versandt werden.

Das eigentliche Versenden im Sinne von Übertragen der Transport-Nachrichten kann über unterschiedliche Transport-Medien (z.B. HTTP oder SMTP usw.) erfolgen. Wie die Transport-Nachricht genau versandt wird, ist in der Transport-Beschreibung festgelegt. Ein Agent kann über mehrere Transport-Medien kommunizieren, welche das sind, ist im Feld Agent-Locator mit der Eigenschaft Transport-Typ (s. Tabelle 2-1) definiert. Es besteht auch die Möglichkeit, dass Transport-Medien auch während einer laufenden Kommunikation geändert werden können, ohne dass die Kommunikation unterbrochen wird.

Als Extrafunktionalität sieht die abstrakte FIPA-Architektur einfache Mechanismen zur sicheren Nachrichtenübertragung vor. Transport-Nachrichten können so versandt werden, dass jede Modifikation während der Übertragung identifiziert werden kann. Auch können die Nutz-Daten (die ursprüngliche Agenten-Nachricht) der Transport-Nachrichten durch gängige Verfahren verschlüsselt werden, wodurch der Inhalt der Nachrichten ausschließlich für Teilnehmer der Kommunikation zugänglich ist.

Ein wichtiger Aspekt der FIPA-Kommunikation ist die Sicherung der Interoperabilität des Nachrichtenaustausches und dadurch des Gesamtsystems. FIPA bietet die Plattform-Interoperabilität in zwei Schritten an – im ersten Schritt durch die Interoperabilität der verschiedenen Transport-Medien, im zweiten Schritt durch die Interoperabilität der Nachrichten-Repräsentationen.

2.3.2 Management

Das FIPA-Agenten-Management definiert ein Ausführungsmuster für FIPA-Software-Agenten basierend auf der abstrakten FIPA-Architektur (s. Abbildung 2-6). Dabei wird ein abstraktes Referenzmodell (s. Abbildung 2-7) aufgestellt, indem logische Operationen (wie z.B. Erzeugen, Registrierung, Auffinden, Kommunikation, Migration und Entfernen von Agenten) und logische Entitäten definiert werden. Jede der Entitäten des Modells stellt eine bestimmte Menge an Funktionalitäten im Gesamtkontext zur Verfügung.

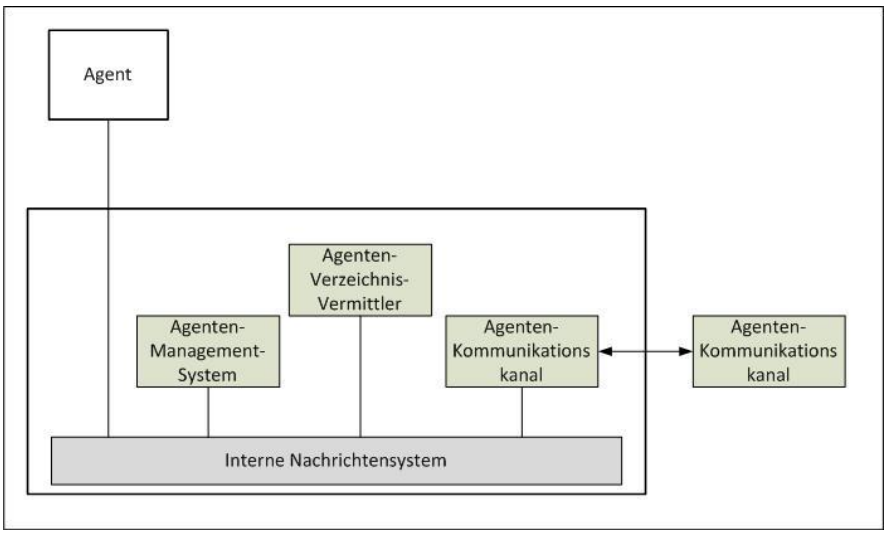

Abbildung 2-7: FIPA-Management-Modell nach [fipa.org]

Im Folgenden werden die Entitäten des FIPA-Management-Modells mit deren Funktionalitäten vorgestellt:

- **Agent:** Der Agent ist der Hauptakteur eines Agentensystems. Er stellt mehrere Funktionalitäten (wie z.B. Interaktion mit den Benutzern des Systems [Menschen], Informationsaustausch mit externer Software usw.) zur Verfügung. Ein Agent hat stets einen oder mehrere Besitzer. Der Agent ist mittels seines eindeutigen Agentennamens (s. Kapitel 2.3.1 „Abstrakte Architektur“) über alle FIPA-Domänen adressierbar. Dabei kann der Agent an mehreren Adressen in einem Agentensystem angemeldet und somit über diese erreichbar sein. Das Agenten-Zustandsdiagramm (s. Abbildung 2-8) bildet alle Zustände ab, die im Lebenszyklus eines Agenten möglich sind.

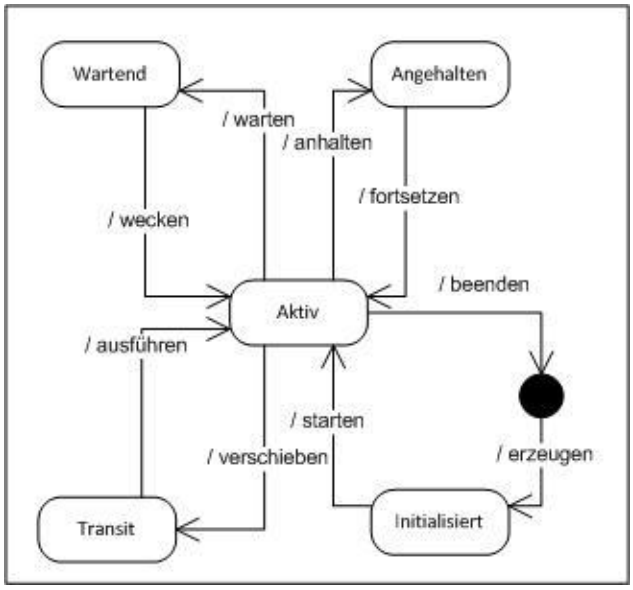

Abbildung 2-8: Agenten-Zustandsdiagramm nach [fipa.org]

Bevor ein Agent ausgeführt werden kann, muss er erzeugt werden. Während dieses Vorgangs wird der Agent initialisiert, danach befindet er sich in dem Zustand „Initialisiert“. In diesem Zustand kann der Agent zur Ausführung gebracht werden. Die aktive Ausführung eines Agenten ist durch den Zustand „Aktiv“ dargestellt. Ein Agent kann auf Aufforderung des Management-Systems (s.u.) beendet werden. Dabei kann es sich um eine so genannte zwingende oder um eine „einfache“ Aufforderung handeln. Im ersten Fall muss der Agent die Aufforderung befolgen und seine Ausführung beenden. Im zweiten Fall kann der Agent die Aufforderung ignorieren und weiterhin im aktiven Zustand bleiben. Ein Agent kann auf Aufforderung eines anderen Agenten oder des Management-Systems seine Ausführung anhalten (unterbrechen). Das Fortsetzen einer unterbrochenen Ausführung kann ausschließlich auf Aufforderung des Management-Systems

erfolgen. Ein Agent kann seine Ausführung unterbrechen und sich in einen wartenden Zustand (z.B. warten auf Nachrichten) versetzen. Auch das Aufwachen aus dem Zustand „Wartend“ kann nur das Management-System initiieren (z.B. beim Eintreffen von Nachrichten). Mobile Agenten (Agenten, die zwischen Plattformen migrieren können) können sich in einen speziellen Zustand (Migration-Zustand) versetzen. In diesem Zustand kann der Agent die Grenzen einer Agenten-Plattform verlassen (wobei auch sein aktueller Zustand gespeichert und über das Netzwerk verschickt wird). Das Management-System kann die Ausführung eines Agenten aus einem Migration-Zustand fortsetzen.

- **Verzeichnis-Vermittler:** Der Verzeichnis-Vermittler (engl. Directory Facilitator) ist die Realisierung eines Verzeichnis-Dienstes in Form eines Agenten (s. abstrakte FIPA-Architektur).
- **Management-System:** Das Management-System ist eine zentrale Entität der Agenten-Plattform. Diese Entität ist durch einen speziellen Agenten (MS-Agent) realisiert. Der Agent überwacht und kontrolliert den Zugang zu Plattformfunktionen (wie z.B. Starten eines neuen Agenten, Beenden eines Agenten usw.). Außerdem verwaltet der MS-Agent das Agenten-Verzeichnis (s. Kapitel 2.3.1 „Abstrakte Architektur“).
- **Kommunikationskanal:** Jeder Agent hat Zugriff auf mindestens einen Kommunikationskanal. Kommunikationskanäle sind das Standardkommunikationsmittel zwischen Agenten in unterschiedlichen Agenten-Plattformen.
- **Agenten-Plattform:** Eine Agenten-Plattform besteht aus der konkreten physikalischen Infrastruktur, in der die Software-Agenten ausgeführt werden. Sie beinhaltet Rechner, Betriebssysteme, Agentensoftware, FIPA-Management-Komponenten, das interne Nachrichtensystem (s.u.) und die eigentlichen Agenten. Die interne Realisierung der Agenten-Plattform wird vom FIPA-Standard nicht definiert, sondern von den Entwicklern einer Agenten-Plattform durch die freie Wahl der Technologien verantwortet.
- **Internes Nachrichtensystem:** Das interne Nachrichtensystem verantwortet den Nachrichtenaustausch sowohl innerhalb als auch nach außerhalb einer Agenten-Plattform.
- **Agenten-Domäne:** Eine Agenten-Domäne ist eine logische Gruppierung von Agenten, die durch eine entsprechende Anmeldung beim Verzeichnis-Vermittler definiert wird. Jeder Domäne darf genau ein Verzeichnis-Vermittler zugeordnet sein, welcher neben der ausführlichen Beschreibung der jeweiligen Domäne auch die komplette Liste der zugehörigen Agenten verwaltet. Ein Agent kann in einer oder in mehreren Domänen Mitglied sein, eine Agenten-Plattform kann eine oder mehrere Domänen besitzen.
- **Agenten-Universum:** Ein Agenten-Universum ist eine Zusammensetzung von allen im jeweiligen Kontext bekannten Agenten-Domänen.

Die hier beschriebenen Management-Komponenten ermöglichen ein optimales Agenten-Management im Rahmen des FIPA-Standards.

2.3.3 Kommunikation

Die Spezifikation der Agentenkommunikation ist der Schwerpunkt des FIPA-Standards. Dieser sieht den Einsatz einer speziellen Sprache zur Abwicklung der Kommunikation (Agent Communication Language, kurz. ACL) zwischen den Software-Agenten vor. Diese Sprache basiert auf der Annahme, dass Aussagen bzw. Nachrichten Handlungen dadurch auslösen, dass die Nachrichten einfach versandt werden. Die FIPA-ACL definiert mehrere Nachrichtentypen (z.B. Query-If – Anfrage, Request -Aufforderung zu Aktion, Agree - Bestätigung einer Aktion, Refuse - Ablehnen einer Aktion usw.) und legt entsprechend die damit verbundenen Handlungen des sendenden und des empfangenden Agenten fest (s. [fipa.org]). Neben dem Typ einer ACL-Nachricht ist naturgemäß ihr Inhalt für die Kommunikation maßgebend. Dieser wird in der dafür vorgesehenen Semantik-Sprache FIPA-SL ausgedruckt. Bei der Übergabe des Inhaltes kann eine Ontologie als Referenz eingegeben werden. Auf diese Weise wird sichergestellt, dass die beiden Seiten der Kommunikation den Inhalt der Nachricht im richtigen Kontext interpretieren, unabhängig davon, ob Sender und Empfänger dieselbe oder verschiedene Technologien zum Kommunizieren einsetzen. Dieser Zusammenhang ist in der Abbildung 2-9 vorgestellt.

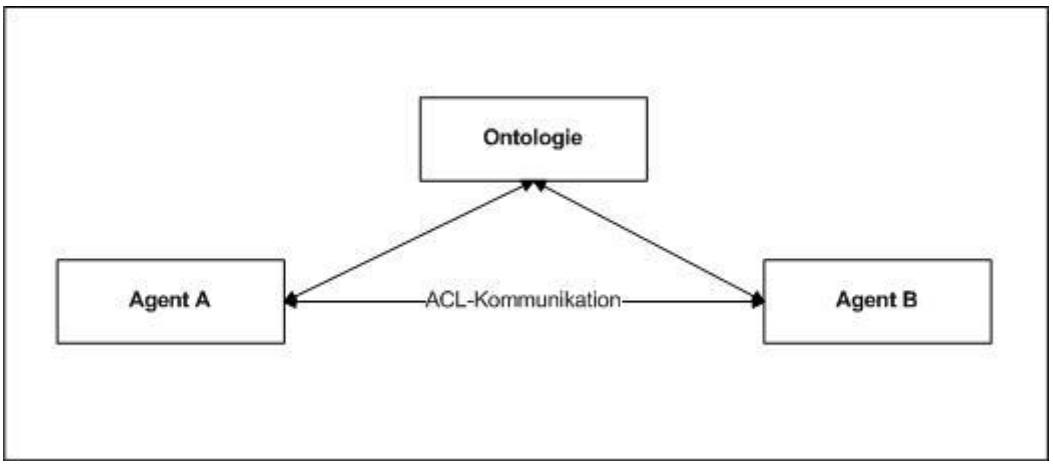

Abbildung 2-9: FIPA-Ontologie-Einsatz nach [fipa.org]

Neben der Kommunikationssprache FIPA-ACL, der Semantiksprache FIPA-SL und dem Einsatz von Ontologien spezifiziert FIPA im Rahmen der Agentenkommunikation auf einer höheren Ebene mehrere Interaktionsprotokolle. Diese bilden für etablierte Szenarien Kommunikationsmuster ab, die

den genauen Ablauf einer Agenteninteraktion im jeweiligen Szenario definieren. Im Folgenden werden die wesentlichen dieser Protokolle vorgestellt.

- **FIPA Propose Interaction Protocol:** Dieses Protokoll ermöglicht einem Agenten (Initiator), einen anderen Agenten (Empfänger) zur Ausführung einer oder mehreren festgelegten Aktionen aufzufordern. Der Initiator sendet eine Propose-Nachricht an den Empfänger-Agenten. Die Propose-Nachricht beschreibt genau (z.B. mittels Ontologie) die gewünschte Aktion. Nachdem der Empfänger die Aufforderung erhalten hat, kann er sich entscheiden, ob er die Aktion ausführt oder die Ausführung ablehnt. Die Entscheidung teilt er dem Initiator mittels geeigneter Protokoll-Nachrichten mit (s. Abbildung 2-10).

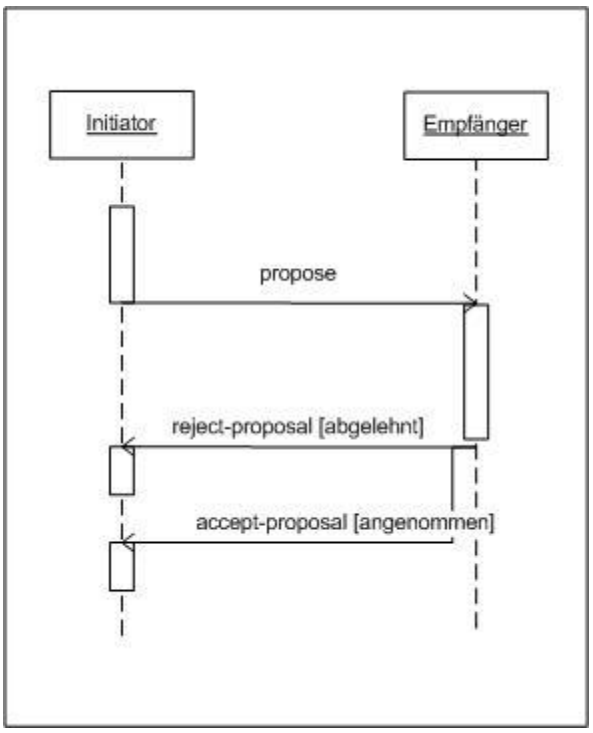

Abbildung 2-10: FIPA-Propose-Protocol nach [fipa.org]

- **FIPA Request Interaction Protokol:** Auch dieses Protokoll ermöglich einem Agenten (Initiator), einen anderen Agenten (Empfänger) zur Ausführung einer oder mehreren festgelegten Aktionen aufzufordern. Im Vergleich zu dem FIPA-Propose verfügt der Initiator in diesem Szenario über bessere Übersicht und Kontrolle über den zeitlichen Verlauf der gewünschten Aktion sowie deren Ergebnis. Der Empfänger-Agent erhält die Aufforderung und kann sich entscheiden, ob er sie ausführt oder ablehnt. Seine Entscheidung teilt er mittels des Protokolls dem Initiator mit. Wenn der Empfänger die Aufforderung angenommen hat, wird er den Initiator informieren, ob die Aktion erfolgreich ausgeführt werden konnte. Die Failure-Nachricht signalisiert, dass die Aktion nicht erfolgreich ausgeführt werden konnte. Die Inform-Done-Nachricht informiert, dass die Aktion erfolgreich ausgeführt wurde. Die Inform-Result-Nachricht informiert, dass die Aktion erfolgreich ausgeführt wurde und liefert das Ergebnis der Aktion zurück (s. Abbildung 2-11).

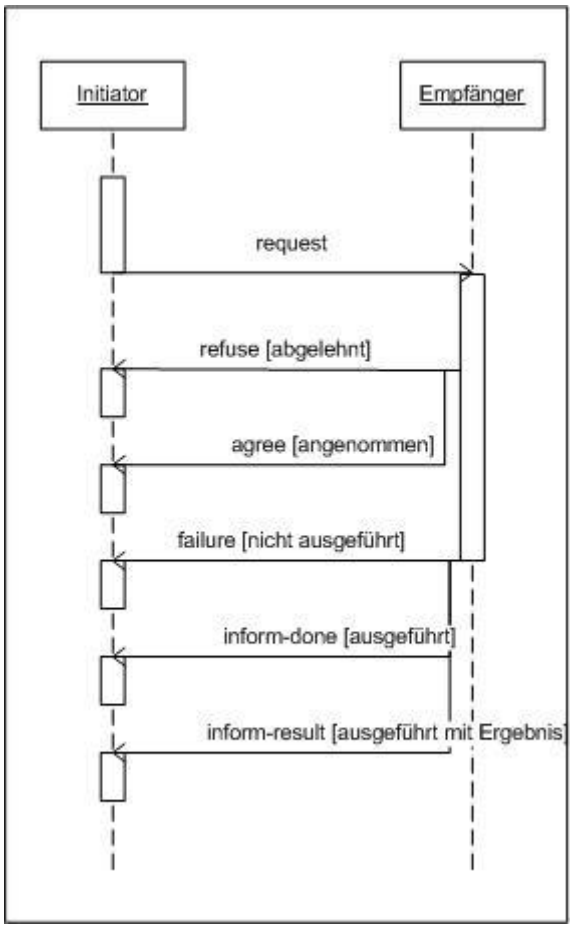

Abbildung 2-11: FIPA-Request-Protocol nach [fipa.org]

- **FIPA Subscribe Protocol:** Dieses Protokoll ermöglicht es einem Agenten (Initiator), einen anderen Agenten (Empfänger) zu beobachten, indem sich der Initiator für Veränderungen der Empfänger-Objekte registriert. Die Registrierung wird durch eine Subscribe-Nachricht ausgelöst. Die Nachricht enthält Referenzen derjenigen Objekte, an denen der Initiator interessiert ist. Der Empfänger kann der Registrierung entweder zustimmen oder sie ablehnen. Falls der Empfänger die Registrierung angenommen hat, teilt er jede Zustandsänderung der beobachteten Objekte dem Initiator mittels einer Inform-Result-Nachricht mit. Der Empfänger kann die Registrierung zu jeder Zeit aufheben, indem er eine Failure-Nachricht an den Initiator versendet (s. Abbildung 2-12).

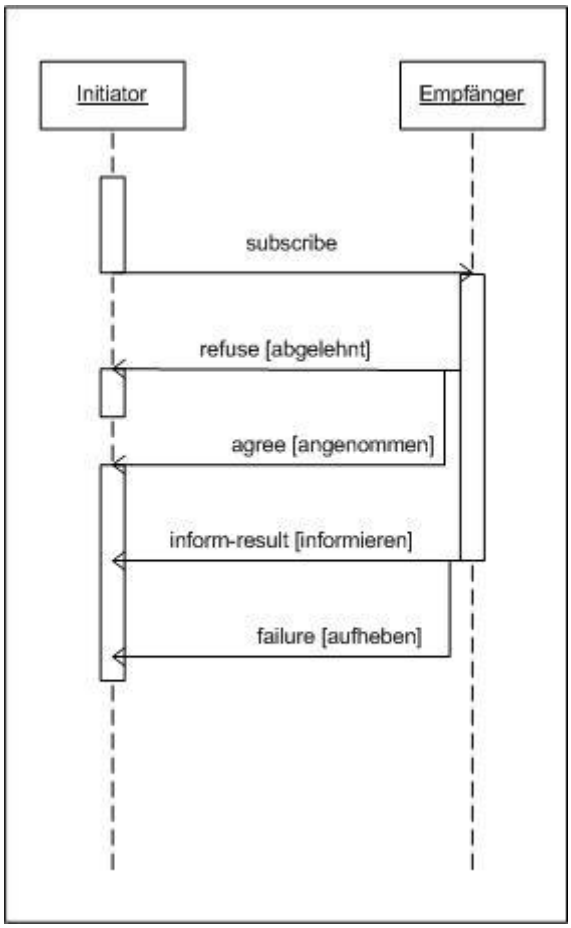

Abbildung 2-12: FIPA-Subscribe-Protocol nach [fipa.org]

- **FIPA Cancel Metha Protocol:** Dieses Protokoll wird in der Agentenkommunikation dann eingesetzt, wenn ein aktives Interaktionsprotokoll (z.B. FIPA Request Protocol) unterbrochen werden muss. Dadurch signalisiert der Initiator des momentan aktiven Protokolls, dass er nicht mehr an der Kommunikation interessiert ist und diese auf eine für beide Seiten günstige Art und Weise beendet werden soll (s. Abbildung 2-13).

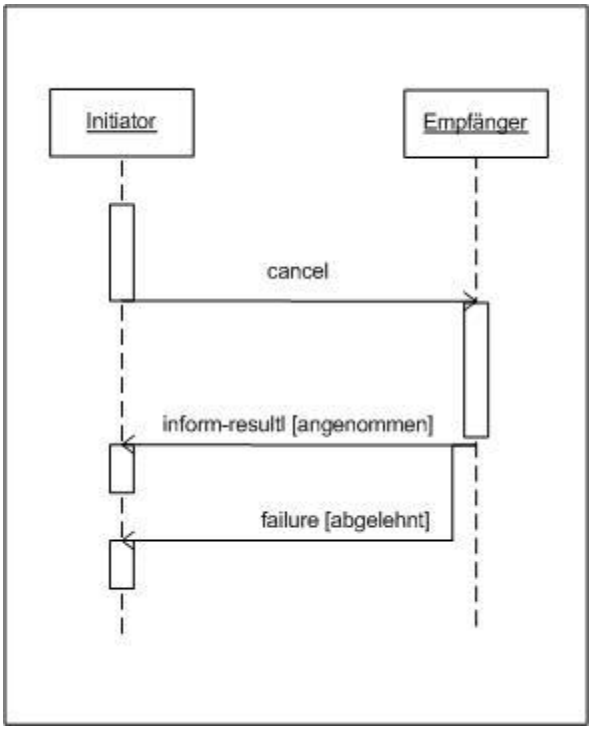

Abbildung 2-13: FIPA Cancel Metha Protocol nach [fipa.org]

Der FIPA-Standard definiert auch weitere Interaktionsprotokolle, die in umfangreicheren Kommunikationsszenarien eingesetzt werden können. Eine ausführliche Beschreibung dieser Protokolle ist unter [fipa.org] zu finden.

Durch den gemeinsamen Kern (die abstrakte FIPA-Architektur), die Standardisierung der Nachrichtenstruktur (FIPA-ACL), die Art und Weise, wie Inhalte weitergegeben werden (FIPA-SL), den Einsatz festgelegter Interaktionsprotokolle und das wohldefinierte Agenten-Management erreicht der FIPA-Standard die Interoperabilität der Agentensysteme unabhängig von konkret eingesetzten Technologien. Trotz der zahlreichen Vorteile des FIPA-Standards muss dennoch gefragt und untersucht werden, ob nicht andere Agentensysteme besser als die FIPA-standardisierten zur Lösung der in dieser Arbeit gestellten Aufgaben besser geeignet sind.

2.4 Fazit

In diesem Kapitel wurden die zum weiteren Verständnis der Arbeit notwendigen Grundlagen zu den Themen Agentensimulation, Software-Agenten und FIPA-standardisierte Software-Agentensysteme vorgestellt. Dabei wurden relevante Design-Ansätze, Technologien und Architekturen zur Entwicklung von Softwaresystemen innerhalb des Software-Agentenparadigmas präsentiert.

3. Vergleichbare Arbeiten

Das Thema „Fußgängersimulation in Gebäuden“ steht im Mittelpunkt dieser Arbeit. Dabei sind besonders zwei Aspekte im Rahmen der Arbeit relevant: die Fußgängerumgebung und das typische Fußgängerverhalten. Es wird dabei angenommen, dass die Fußgängerumgebung eine beliebige Gebäudestruktur ist und dass das typische Fußgängerverhalten auf die Aktivitäten Bewegung entlang einer Route innerhalb der Umgebung und Verhalten am Ziel reduziert werden kann (s. [Narasimhan07]). Zu den oben getroffenen Annahmen werden im Laufenden relevante wissenschaftliche Arbeiten mit Konzepten zum Abbilden von Gebäudestruktur vorgestellt, die als Basis für das Erstellen einer Simulationsumgebung für Indoornavigation im Flughafenkontext dienen soll.

3.1 Abbilden von Gebäudestruktur

Es existieren mehrere Modellierungsansätze, um die Gebäude- bzw. Raumstruktur (repräsentatives Beispiel dafür sind CAD-Systeme für den Architektur-Sektor) auf unterschiedlichen Abstraktionsebenen abzubilden. Diese Ansätze lassen sich jedoch nicht unmittelbar auf das spezielle Problem der Fußgängersimulation anwenden (s. [Narasimhan07]) sondern müssen erst für diesen besonderen Fall optimiert werden. Dazu wird Anwendungsfall der Fußgängersimulation im Folgenden genauer analysiert.

Das zu simulierende Fußgängerverhalten umfasst die Fußgängerbewegung entlang einer Route bis zu einem vorgegebenen Ziel und das Ausüben einer Tätigkeit am Zielort. Diese Tätigkeit kann beliebig sein (z.B. Kauf eines Artikels, Weitergabe einer Information, Genießen einer Speise usw.). Der Zielort wird nur durch die räumliche Positionierung definiert. Seine Funktion (z.B. Kaufhaus, Informationsschalter, Restaurant usw.) wird in dem geometrischen Modell nicht betrachtet (s. [Narasimhan07]). Beim Abbilden des gewünschten Verhaltens in einem geometrischen Kontext werden vorhandene Architekturdetails wie Wendedichte, Fensterposition, verlaufende Rohre usw. nicht benötigt und können ignoriert werden. Die relevanten Gebäudedetails sind Informationen über die möglichen räumlichen Positionen der Zielorte, die Menge der möglichen Wege durch das Gebäude sowie deren Eigenschaften (z.B. maximale Geschwindigkeit, maximaler Durchsatz, Richtung usw.). Diese Informationen werden nicht direkt von den bekannten geometrischen Modellen angeboten, sie müssen durch spezielle Mechanismen abgeleitet bzw. generiert (z.B. aus existierenden CAD-Daten eines Gebäudes), und in geeigneter Form zur Verfügung gestellt werden.

3.1.1 Manuelles Abbilden von Gebäudestruktur

Für das Ableiten bzw. Generieren der oben genannten Informationen werden zwei Verfahrensarten eingesetzt. Im ersten Fall handelt es sich um ein manuelles Abbilden und Generieren von Wege- und Zielort-Informationen auf Grund vorhandener geometrischer Gebäudeinformationen. Beispiel für einen solchen Ansatz ist die Raumsuche der Universität Stuttgart (s. [uni-stuttgart.de]):

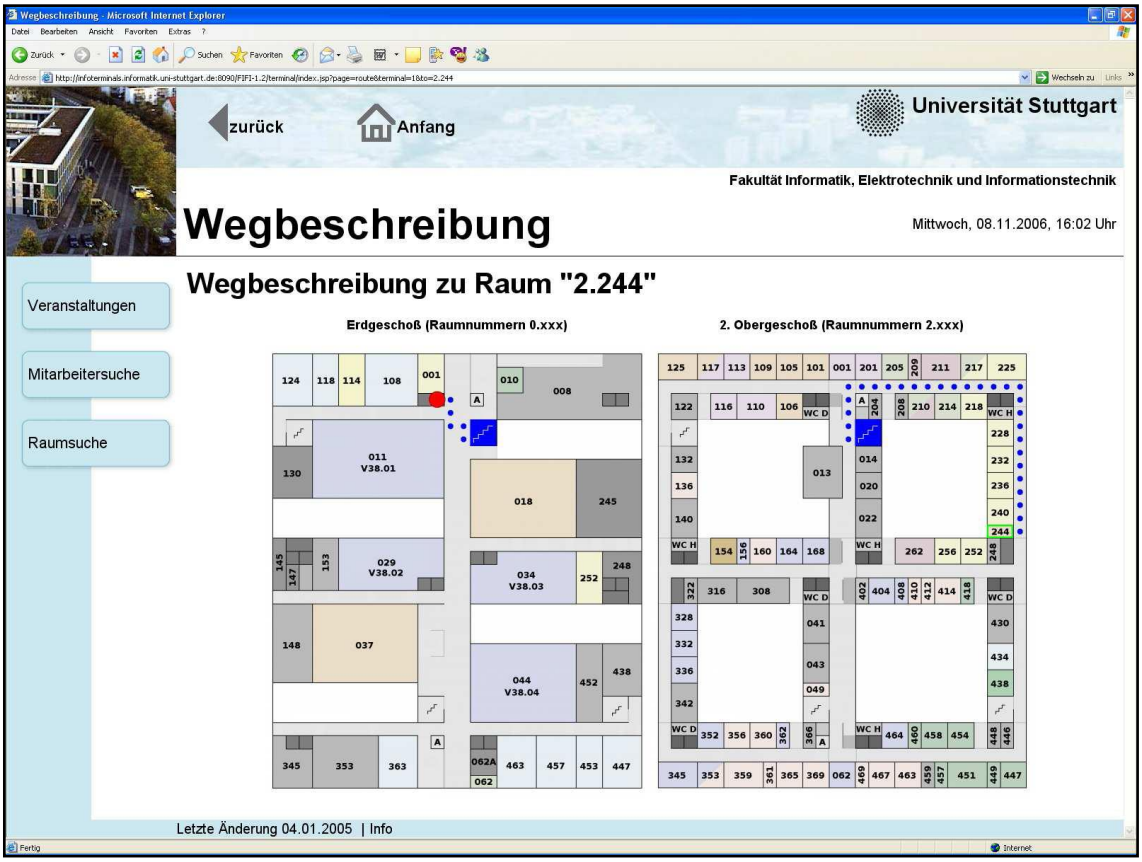

Abbildung 3-1: Raumsuche der Universität Stuttgart nach [Narasimhan07]

Die Raumsuche ermöglicht einem Besucher der Universität Stuttgart Wegbeschreibungen durch Teile des Universitätsgebäudes abzufragen bzw. anzufordern (s. Abbildung 3-1). Bei diesem System werden die Weginformationen in Form eines Graphs, der alle Räume miteinander verbindet, abgebildet und manuell in das System eingegeben. Die Knoten des Graphs stellen die möglichen Zielorte dar, die ein Besucher erreichen kann (in diesem Fall die einzelnen Räume). Die Kanten des Graphs bezeichnen die tatsächlichen Strecken, die ein Besucher innerhalb des Gebäudes zurücklegen kann. Die Knoten und Kanten des Graphs werden durch entsprechende Koordinaten im 2D- bzw. 3D-Raum positioniert. Der so entstandene Graph wird bei der Generierung der Wegbeschreibung zwischen zwei Punkten (z.B. Gebäude-Eingang und Raum 2.244) als Basis für eine Wegsuche verwendet (s. [Giesecke04]).

3.1.2 Dynamisches Abbilden von Gebäudestruktur

Eine Verbesserung der manuellen Beschreibung aller möglichen Wege in einem Gebäude bietet [Drexl03]. In der Arbeit wird eine Methode entwickelt und als Softwareprogramm realisiert, die aus vorhandenen CAD-Daten eines Gebäudes einen Gebäudegraph mit entsprechenden Zielorten als Knoten und möglichen Strecken als Kanten automatisch durch spezielle mathematische und Bildbearbeitungsverfahren generiert [Drexl03] (s. Abbildung 3-2). Die Kanten und Knoten des Graphs werden auf Grund des ursprünglichen CAD-Modells im 3D-Raum durch errechnete Koordinaten positioniert.

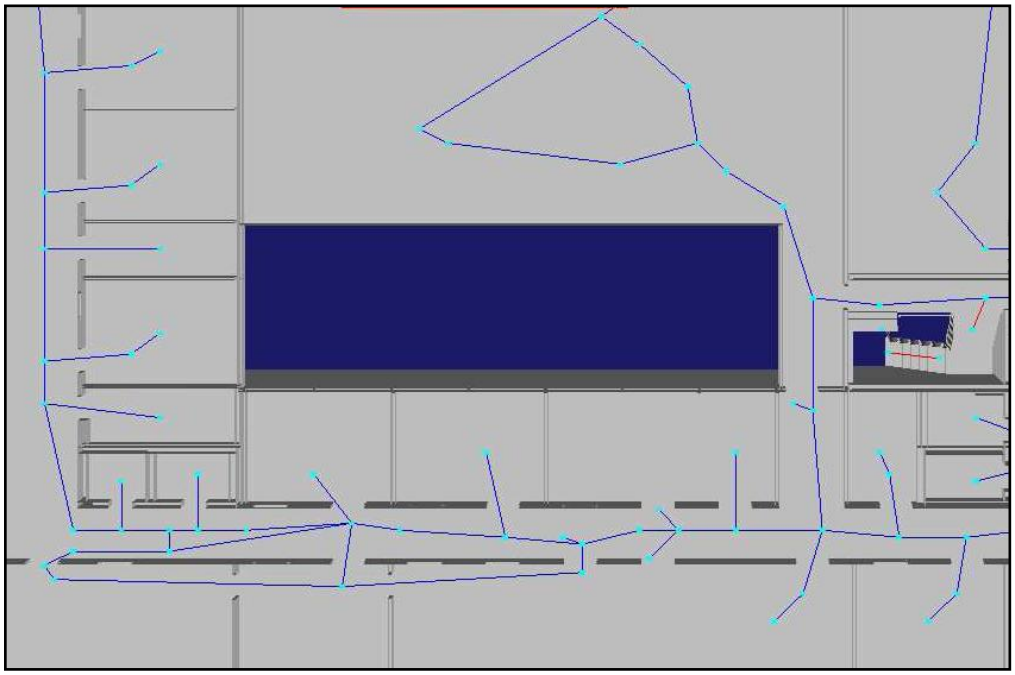

Abbildung 3-2: Automatische Generierung des Gebäudegraphs nach [Drexl03]

Der so entstandene Graph beschreibt sehr detailiert die Gebäudestruktur und kann auf Grund seiner Größe und Unregelmäßigkeit für die Zwecke einer Fußgängersimulation nicht direkt eingesetzt werden [Narasimhan07]. Ausmaß und Unregelmäßigkeit des Graphs sind bedingt durch die Tatsache, dass die Methode alle möglichen Wege durch den modellierten Raum ermittelt und alle Nachbarknoten miteinander verbindet. Eine Simulation auf Basis eines derart umfangreichen Graphs ist zu rechenintensiv und nicht immer sinnvoll. In [Narasimhan07] wird eine Verbesserung der in [Drexl03] beschriebenen Methode entwickelt. Der automatisch generierte Graph wird optimiert, indem Kanten und Knoten, die ein begrenztes Gebiet beschreiben, zusammengelegt werden (s. Abbildung 3-3). Die Variation der Größe eines solchen begrenzten Gebietes zieht auch die Änderung der Größe des Graphs und dann der Abbildgenauigkeit der Gebäudestruktur nach sich. Trotz der

reduzierten Größe bleiben die geometrischen Eigenschaften des abzubildenden Gebäudes in für die Simulation akzeptablen Grenzen erhalten.

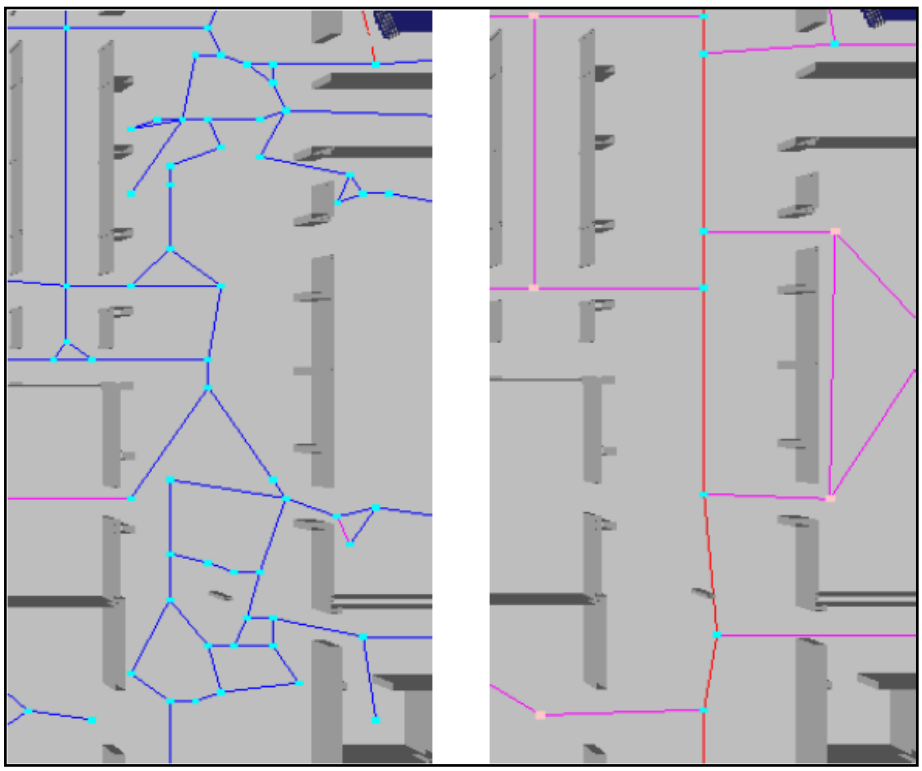

Abbildung 3-3: Reduzieren eines Gebäudegraphs nach [Narasimhan07]

Für eine Fußgängersimulation werden neben der Menge der möglichen Wege und Zielorte mit ihren Koordinaten, die aus einem CAD-Modell extrahiert werden können, weitere zusätzliche Informationen wie Weg-Eigenschaften, Entfernungen, Raumnummern, Nutzungspläne usw. benötigt. Die Weg-Eigenschaften (z.B. Typ, Kapazität und Länge) können teilweise aus den geometrischen Gegebenheiten des CAD-Modells errechnet werden. Der Typ (z.B. Treppe, Korridor oder Rampe) werden durch die Methode von Dexel erkannt [Drexl03]. Erkennen von Aufzügen dagegen ist nicht möglich, derartige Informationen müssen manuell hinzugefügt werden. Informationen wie Raumnummer und Nutzungsplan sind nicht im CAD-Modell enthalten und müssen in weiteren Schritten aus dem Simulationskontext generiert bzw. manuell eigepflegt werden [Narasimhan07]. Da sie nicht unmittelbar zu den geometrischen Daten gehören, werden sie in einem erweiterten Umgebungsmodell verwaltet.

Die im Rahmen der Arbeit von [Drexl03] gefertigte Softwarerealisierung erlaubt es, die entsprechenden Ergebnisse einer Graphgenerierung als XML-Datei zu exportieren, was eine Weiterverarbeitung erleichtert (s. [Narasimhan07]). Dadurch wird eine effiziente Möglichkeit zur

semiautomatischen Erstellung von beliebigen geometrischen Kontexten für Fußgängersimulationen geschaffen.

3.1.3 Zusammenfassung

Der Nachteil einer wie in [Giesecke04] beschriebenen Lösung ist es, dass die Basis der Modellierung ein statisch erzeugter Graph ist. Dieser Graph ist ausschließlich für ein konkretes Gebäude gültig. Zudem wird die Größe des Graphs unmittelbar von der Komplexität des abzubildenden Gebäudes bestimmt, was ab einem gewissen Maß die manuelle Pflege unmöglich macht. Dennoch kann diese Lösung bei einer Prototypen-Entwicklung der gewünschten Simulationsumgebung wegen ihrer Einfachheit und des geringen Aufwandes sinnvoll eingesetzt werden. Für die Abbildung eines realen Flughafenszenarios auf Grund des sich oft ändernden Nutzungsplans und der umfangreichen Gebäudestruktur kann das von [Drexl03] entwickelte Verfahren eingesetzt werden.

3.2 Simulation von Fußgängerverhalten

Nachdem im vorigen Kapitel Methoden zur Abbildung der Fußgängerumgebung in dem festgelegten Szenario vorgestellt wurden, werden im Folgenden Modelle des Fußgängerverhaltens mit dem Schwerpunkt der Bewegung in Gebäuden entlang einer Route bis zu einem vorgegebenen Ziel und das Ausüben einer Tätigkeit am Zielort analysiert.

Nach [May89] kann Fußgängerverhalten bzw. Fußgängerbewegung analog zum Autoverkehr entweder auf einer makroskopischen oder einer mikroskopischen Ebene modelliert werden. Im Fall der makroskopischen Abbildung des Fußgängerverhaltens wird diese als Bewegung von Fußgängerströmen modelliert. Dabei beruhen solche Verfahren auf den Ähnlichkeiten zwischen Fußgängerströmen und Flüssigkeiten oder Gasen. Die Mobilitätseigenschaften werden durch ein von Flüssigkeiten bzw. Gasen abgeleitetes numerisches Modell abgebildet, indem bestimmte für Fußgänger typische Eigenschaften (z.B. Kollisionsvermeidung, variable Beschleunigung usw.) berücksichtigt werden (s. [Helbing90] und [Helbing96]). Die Makromodellierung wird für die Feststellung von Dichte und Durchsatz bei Fußgängerströmen eingesetzt. Da das Verhalten des einzelnen Individuums bei der hier gewünschten Simulation im Vordergrund steht, bietet die Makromodellierung keine geeignete Basis für die Entwicklung der entsprechenden Software-Simulationsumgebung. Einen besseren Ansatz bietet die Modellierung auf einer mikroskopischen Ebene, realisiert auf Basis von zellularen Automaten, Warteschlangen und Software-Agenten.

3.2.1 Fußgängersimulation auf Basis von zellularen Automaten

Fußgängersimulationen, die auf zellularen Automaten basieren, modellieren die Fußgängerbewegung durch ein Gitter von gleichartigen Zellen. Der Zustand jeder einzelnen Zelle wird nach im System wohldefinierten Regeln in jedem Simulationsschritt berechnet. Bei der Berechnung des neuen Zustandes einer Zelle werden deren frühere Zustände sowie die der Nachbarzellen berücksichtigt. In [Gipps85] werden nur zwei Zustände, die von einer Zelle angenommen werden können, zur Abbildung der Fußgängerbewegung definiert. Die Zelle ist entweder durch einen Fußgänger besetzt oder frei. Die Bewegung wird dadurch modelliert, dass Fußgänger die von ihnen besetzte Zelle in jedem Simulationsschritt verlassen und eine freie Nachbarzelle besetzen können(s. Abbildung 3-4).

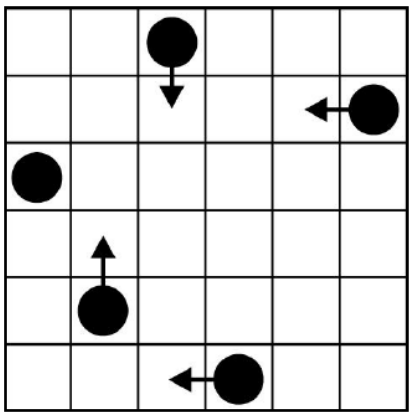

Abbildung 3-4: Fußgängersimulation mit zellularen Automaten

Um die Modellierung der Fußgängerbewegung zu verfeinern, werden in [Gipps87] weitere Zellzustände sowie Berechnungsmethoden eingeführt. Es werden nicht mehr einzelne Zellenzustände berechnet sondern es werden Zellenrelationen gebildet, deren Zellenzustände zusammenhängen. Wenn etwa eine Zelle durch einen Fußgänger belegt ist, werden die Zellenzustände der entsprechenden Relation die Nähe zu der besetzten Zelle abbilden. Zusätzlich werden Zustände für nicht begehbare Zellen (z.B. Hindernisse) definiert.

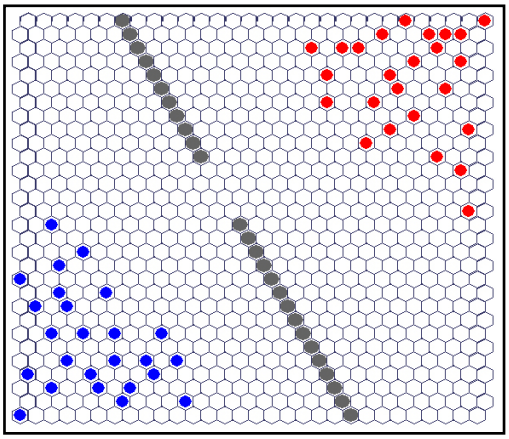

Abbildung 3-5: Fußgängersimulation mit erweiterten zellularen Automaten

Mit Hilfe der genannten Erweiterungen können typische Phänomene wie Kollisionsvermeidung, Hindernisausweichen und Bilden von Fußgängergruppen abgebildet werden (s. Abbildung 3-5). Es muss berücksichtigt werden, dass ein wie von [Gipps87] beschriebenes umfangreiches Modell eine komplexe Parametrierung erfordert, was die Fußgängersimulation durch zellulare Automaten erschwert.

Die Fußgängersimulation mit zellularen Automaten kann als Basis für die Entwicklung der hier zu erarbeitenden Simulationsumgebung nicht eingesetzt werden. Die Gründe dafür sind das zu starre Berechnungsmodell sowie die Fokussierung allein auf das Abbilden der physischen Bewegung. Dennoch wäre es denkbar, dass zusätzliche Funktionalität der Simulationsumgebung zum Abbilden von Phänomenen wie Kollisionsvermeidung, Hindernisausweichen und Bilden von Fußgängergruppen mit Hilfe dieses Modells realisiert werden kann.

3.2.2 Fußgängersimulation auf Basis von Warteschlangen

Ein weiteres Modell der Fußgängerbewegung basiert auf Simulation durch Warteschlangen (s. [Lovas94] und [Okazaki93]). Mittels solcher Simulationen werden oft Evakuierungsszenarien abgebildet. In diesen Szenarien haben alle Fußgänger dasselbe Ziel, ein Gebäude bzw. Räumlichkeiten durch entsprechende Ausgänge schnellstmöglich zu verlassen. Die Abbildung erfolgt über einen Graph. Die Knoten des Graphs sind z.B. Räume und Durchgänge bzw. Ausgänge, die ein Fußgänger erreichen möchte. Die Knoten sind durch die Kanten des Graphs verbunden. Jede Kante stellt eine Warteschlange dar. Um von einem Knoten zum anderen zu gelangen, muss ein Fußgänger eine Warteschlange betreten. Nachdem er sie betreten hat, muss er eine gewisse Zeit in der Warteschlange verbleiben, bevor er den anderen Knoten erreicht. Sollten aus einem Knoten mehrere Kanten (Warteschlangen) ausgehen, wird zufällige Auswahl aus den von diesem Punkt aus

„begehbaren“ Kanten getroffen. Diese Wahl bestimmt, welche von den möglichen Kanten als nächste genutzt wird. Die Gewichtung der Wahl wird durch die Fußgängeranzahl in den einzelnen Warteschlangen bestimmt. Die Auswahl und das Betreten von Warteschlangen werden für einen Fußgänger so oft wiederholt, bis er das Gebäude verlassen hat. Bei dieser Form der Simulation wird die Zeit gemessen, die ein Fußgänger braucht, um das Gebäude zu verlassen, hier die Summe aller Zeiten, die er auf seinem Weg in den jeweiligen Warteschlangen verblieben ist.

Warteschlangen eignen sich mit bestimmten Modifizierungen als Basis für die Entwicklung der gewünschten Simulationsumgebung: analog zu einem Evakuierungsszenario haben alle Fluggäste in einem Flughafenszenario auch ein gemeinsames Ziel (am richtigen Gate zu sein), wobei der Faktor Zeit von entscheidender Bedeutung ist. Dennoch ist das Verfahren (ähnlich wie das der zellularen Automaten) allein auf Bewegung ausgerichtet.

3.2.3 Fußgängersimulation auf Basis von Software-Agenten

Fußgängersimulationen, die auf Software-Agenten (s. Kapitel 2) basieren, werden für komplexe Szenarien eingesetzt, bei denen das zu modellierende Fußgängerverhalten nicht durch nummerische Modelle auf einer Makroebene oder mit Berechnungsverfahren auf einer Mikroebene (wie z.B. zellulare Automaten oder Warteschlangen) abgebildet werden kann. Das ist insbesondere dann der Fall, wenn nicht nur die rein mechanische Bewegung der Fußgänger sondern auch weitere für den jeweiligen Anwendungsfall relevante Aspekte seines individuellen Verhaltens (wie z.B. psychischer bzw. physischer Zustand, bestimmtes soziales Handeln usw.) simuliert werden müssen. In einem solchen Modell werden die Fußgänger mittels Software-Agenten als selbständig agierende Entitäten mit zugehörigen Eigenschaften abgebildet. Dadurch wird das individuelle Verhalten jedes einzelnen Fußgängers im Rahmen der Simulation definiert (s. [Narasimhan07]). Ein solches Simulationsmodell entspricht in gewisser Weise der Vorstellung einer künftigen Simulationsumgebung für ein Indoor-Fußgängernavigationssystem. Zur genauen Beurteilung der Möglichkeiten, die eine solche Simulation bietet, soll hier als ein repräsentatives Beispiel für den Einsatz von Software-Agenten im Bereich der Fußgängersimulation das Exodus-Softwareprodukt[1] zur dynamischen Simulation von Evakuierungsprozessen vorgestellt werden. Der Schwerpunk der Exodus-Software ist die Modellierung und Darstellung von Bewegungen großer Personenströme in komplexen

[1] Das Produkt wurde von der Fire Safety Engineering Group unter der Leitung von Professor Ed Galea an der University of Greenwich (s. [fseg.gre.ac.uk]) in unterschiedlichen Versionen für den Einsatz in Fachgebieten wie Gebäudebau, Flugzeugbau, Schiffbau und Eisenbahn entwickelt. Die Software wird durch ein zusätzliches Modul zur Erstellung dreidimensionaler Ansichten der Modellierungsergebnisse vervollständigt.

Räumlichkeiten im Kontext einer Raumevakuierung. Dieser Schwerpunkt kann mit dem Szenario dieser Masterarbeit verglichen werden.

Beim Abbilden einer Evakuierung berücksichtigt das Programm die Interaktionen zwischen den beteiligten Personen untereinander und zwischen ihnen und der Umgebung. Jede Person wird mittels eines Software-Agenten modelliert. Verhalten und Bewegung des Agenten werden durch eine Anzahl heuristischer Regeln bestimmt. Den einzelnen Regeln sind fünf verschiedene Modelle zugeordnet. Diese Modelle agieren auf einem der Simulation zugrunde liegenden Gitter, das die aus CAD-Zeichnungen importierte Raumgeometrie abbildet (s. Abbildung 3-6). Das zweidimensionale Gitter setzt sich aus Knoten zusammen, die von den Agenten besucht bzw. besetzt werden können. Zwischen diesen Knoten werden Verbindungen definiert, die die Bewegungen der Agenten ermöglichen.

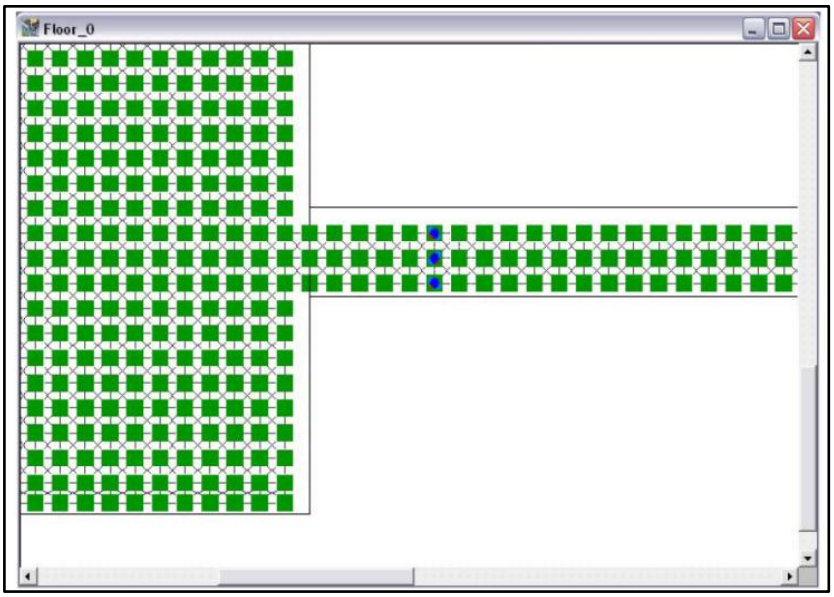

Abbildung 3-6: Beispiel für Abbildung einfacher Gebäudegeometrie aus [Ehm04]

Die Bewegungstrajektorien der Agenten sind dabei maßgeblich Potentialfeld eines Knotens bestimmt. Der Wert des Potentialfeldes ist mit der Entfernung vom nächstgelegenen Ausgang (Rettungspunkt) verknüpft. Je näher sich ein Knoten zu einem Ausgang befindet, desto wahrscheinlicher wird er von einem Agenten besucht. Auf dieser Weise wird ein Maß für „Attraktivität“ eines Knotens bzw. einer Bewegung in Richtung eines Knotens abgebildet. Neben dem Wert für das Potential besitzt jeder Knoten weitere Attribute wie etwa seinen Typ (Free-Space, Stair, Seats) (s. [fseg.gre.ac.uk]) oder auch im Falle eines Brandes die Konzentration bestimmter toxischer Gase. Diese Merkmale haben zusätzlichen Einfluss auf das Verhalten und die Bewegung der virtuellen

Personen. Neben den Knoten können auch die Verbindungen sowie die Agenten selber parametriert werden und somit den Verlauf der Simulation beeinflussen. Die Software bietet z.B. die Möglichkeit, sowohl für abgebildete Personengruppen als auch für jede individuelle virtuelle Person umfangreiche Merkmale betr. der physischen und psychischen Fähigkeiten vorzugeben. Dies kann sowohl explizit als auch in einem bestimmten Rahmen erfolgen. Im Allgemeinen werden diese Attribute einer bestimmten Verteilung folgend der Agenten-Population zugewiesen. Weitere Realisierungsdetails der Software sind in den Quellen [fseg.gre.ac.uk] und [Ehm04] vorgestellt.

Umfang und Komplexität des Modells ermöglichen es der Exodus Software, umfassende Räumlichkeiten (z.B. Gebäude, Flugzeuge, Schiffe usw.) auf ein Evakuierungsszenario vorzubereiten, indem dieses Szenario Simuliert und untersucht wird. Bei der Simulation werden der Ablauf sowie die Ergebnisse grafisch animiert. Auf dieser Weise können Phänomene wie Gruppenverhalten (s. Abbildung 3-7) und Stauungen (in Abbildungen 3-8, 3-9, 3-10 durch die rote Farbe dargestellt) schnell erkannt werden.

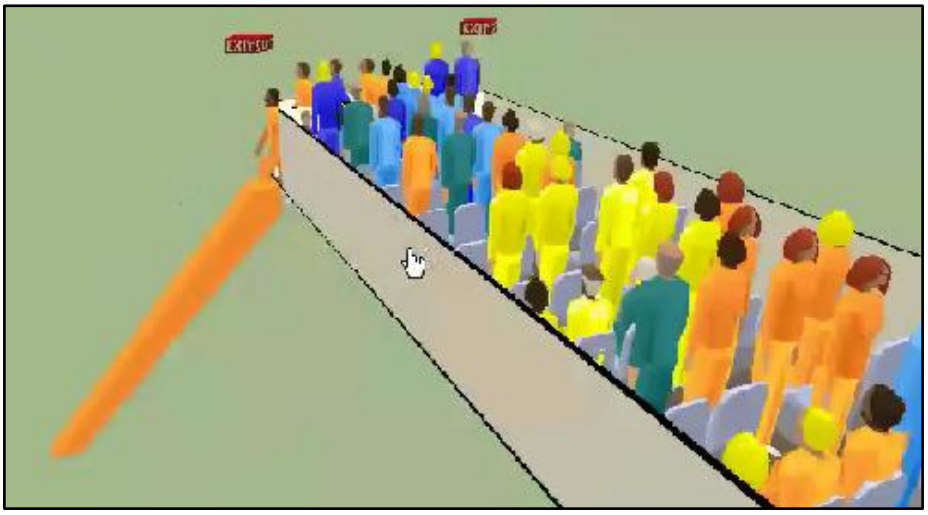

Abbildung 3-7: 3D-Modell einer Flugzeugevakuierung aus [fseg.gre.ac.uk]

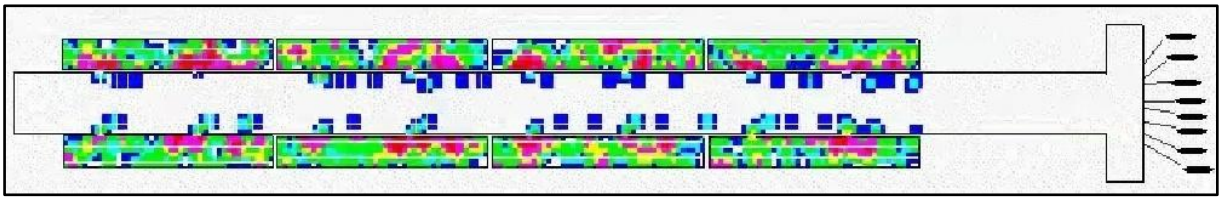

Abbildung 3-8: 2D-Modell einer Gebäudeevakuierung (Anfangsphase) aus [fseg.gre.ac.uk]

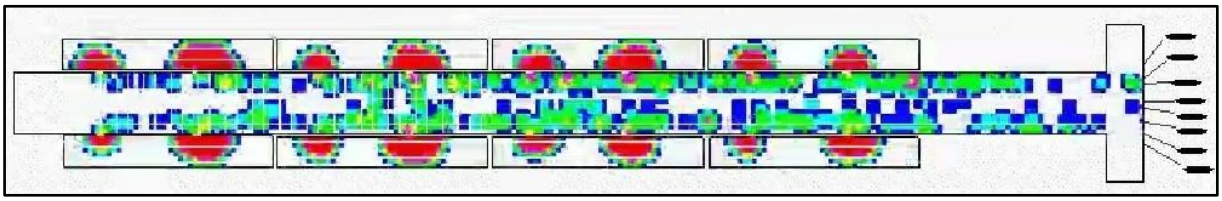

Abbildung 3-9: 2D-Modell einer Gebäudeevakuierung (Zwischenphase) aus [fseg.gre.ac.uk]

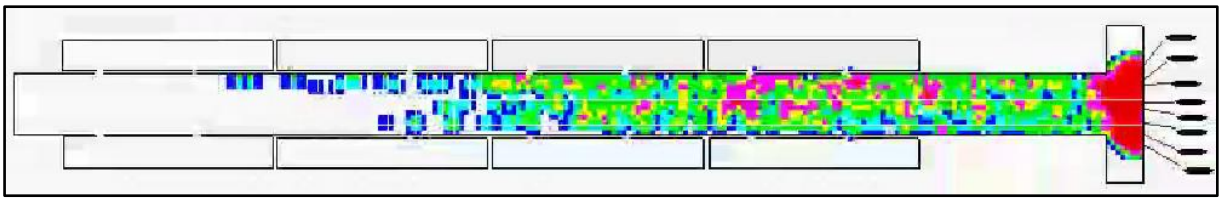

Abbildung 3-10: 2D-Modell einer Gebäudeevakuierung (Endphase) aus [fseg.gre.ac.uk]

Das Beispiel Exodus zeigt, dass das Software-Agentenparadigma im Bereich der komplexen Fußgängersimulationen sehr gut eingesetzt werden kann. Viele der Ansätze von Exodus können in das Szenario dieser Masterarbeit übernommen und hier weiterentwickelt werden.

3.3 Fazit

Die wissenschaftliche Literatur bietet mehrere Ansätze zum Abbilden von Gebäudestrukturen als Basis der Simulation von Fußgängerverhalten. Für das Abbilden von Gebäudestruktur im Rahmen des Szenarios dieser Masterarbeit können sowohl das vorgestellte manuelle Verfahren von [Giesecke04] für einen vereinfachten Prototypenbau als auch das semiautomatische Verfahren von [Drexl03] für komplexe Flughafenszenarien eingesetzt werden. Für die Simulation von Fußgängerverhalten im Kontext der Masterarbeit ist die Methode der Software-Agenten geeignet. Mit ihrer Hilfe kann nicht nur die Bewegung der Fußgänger sondern auch deren soziales Verhalten modelliert werden. Darüber hinaus besteht die Möglichkeit, die Funktionalität der gewünschten Simulationsumgebung durch den Einsatz von zellularen Automaten zu erweitern, womit Fußgänger-Phänomene wie Kollisionsvermeidung, Hindernisausweichen und Bilden von Fußgängergruppen abgebildet werden

können. Einen solchen Ansatz zu untersuchen, wird den Rahmen dieser Masterarbeit sprengen. Vielmehr sollte dieser Hinweis Denkanstöße für zukünftige Arbeiten auf dem Gebiet geben.

4. Anforderungsanalyse

Im Folgenden werden die Eigenschaften der zu entwickelnden Simulation des sozio-technischen Kontextes eines in Kapitel 1.1 definierten Indoor-Fußgängernavigationssystems analysiert. Zur Analyse wird ein entsprechendes Modell für einen konkreten Einsatz des Navigationssystems aufgestellt. Auf Grund der Analyse werden funktionale und nicht funktionale Anforderungen an die zukünftige Software-Simulationsumgebung definiert. Die funktionalen Anforderungen beschreiben, was die Simulationsumgebung leisten muss und welche Funktionen von der künftigen Umgebung erfüllt werden müssen. Die nicht funktionalen Anforderungen spezifizieren die technischen Randbedingungen, unter denen die geforderte Funktionalität zu realisieren ist.

4.1 Szenario

Um eine konkrete Simulationsumgebung entwickeln zu können, wird in dieser Arbeit der Einsatz eines in Kapitel 1.1 definierten Indoor-Fußgängernavigationssystems im Bereich eines Flughafengeländes betrachtet. In dem ausgewählten Zusammenhang ist es die Aufgabe des Navigationssystems, den Benutzer (in diesem Szenario den Fluggast, der das Flughafengelende eine gewisse Zeit vor dem geplanten Abflug betritt) durch ein komplexes Flughafengebäude bzw. Flughafengebäuden zu einem entsprechenden Gate zu führen (navigieren). Dabei müssen mehrere Aspekte von dem System berücksichtigt werden. Der Fluggast muss rechtzeitig zu seinem Gate gelangen, um seinen Abflug nicht zu verpassen. Er möchte auf seinem Weg zum Gate für ihn interessante eventuell Dienste in Anspruch nehmen oder muss dringende Ad-hoc-Aufgaben erledigen (z.B. ankommende Anrufe entgegennehmen, auf wichtige E-Mails reagieren usw.). Während der Navigation können mehrere unvorhersehbare Ereignisse auftreten und der Benutzer kann so von der vorgeschlagenen Route (zeitlich oder räumlich) abweichen. Flugplanänderungen können auftreten und die Zeit zum Abflug kann sich aus diesem Grund verlängern oder verkürzen. Und das Dienstangebot des Flughafens kann sich dynamisch ändern. Daher erweisen sich laufende Anpassungen der Navigation als notwendig.

4.1.1 Komponenten der Simulation

Auf Grund des definierten Flughafenszenarios lassen sich folgende Komponenten und deren Aspekte der zukünftigen Simulationsumgebung definieren:

- **Fluggastverhalten:** Diese Komponente bildet den Benutzer des Indoor-Fußgängernavigationssystems und dessen typisches Verhalten im Kontext einer üblichen Flughafenumgebung ab. Wesentlich für die Abbildung ist das Fußgängerverhalten in

Gebäuden. Ähnlich wie in Kapitel 3.1 wird die Bewegung entlang einer vordefinierten Route bis zu einem Zielort und das Ausüben einer Tätigkeit am Zielort betrachtet. Dabei müssen folgende zusätzliche Aspekte genauer abgebildet werden:

  - **Reagieren auf Navigationsanweisungen:** Der Fluggast wird von dem Indoor-Fußgängernavigationssystem in der Flughafenumgebung geführt. Dazu werden ihm Routen durch das Flughafengebäude vorgeschlagen, denen er in vorgegebenem Zeitrahmen folgen muss. Es kann bei Verfolgen dieser Routen von der Vorgaben zeitlich (etwa durch Verspätung) oder räumlich (z.B. durch Verlassen der Route) abweichen.
  - **Konsumieren von Diensten:** Der Fluggast kann während seines Aufenthalts auf dem Flughafengelände Dienste des Flughafens in Anspruch nehmen. Während einer solchen Aktion ist der Fluggast gebunden und kann nicht mit anderen agieren.
  - **Erledigen von Ad-hoc-Aufgaben:** Während seines Aufenthalts auf dem Flughafengelände kann der Fluggast mit dringenden Aufgaben konfrontiert werden. Um diese schnellstmöglich zu erledigen, greift er auf die Unterstützung des Indoor-Fußgängernavigationssystems zurück.

- **Flughafenumgebung:** Diese Komponente bildet die Flughafenumgebung auf einer für die Simulation geeigneten Abstraktionsebene ab. Dabei werden folgende Aspekte in den Vordergrund gestellt:
  - **Gebäudestruktur**: Die Gebäudestruktur bildet den geometrischen Kontext bzw. die Raumgegebenheiten eines Flughafengeländes ab und schafft die Grundlage zur Erstellung des entsprechenden Nutzungs- und Fortbewegungsplans des Flughafens.
  - **mögliche Wege**: Die möglichen Wege bilden die Strecken, die ein Fluggast zu Fuß oder mit Hilfe eines Fortbewegungsmittels innerhalb der Gebäudestruktur zurücklegen kann.
  - **Fluggastposition**: Um eine sinnvolle realistische Abbildung des Indoor-Kontextes zu schaffen, müssen die aktuellen Positionen innerhalb der Gebäudestruktur aller in der Simulation aktiven Fluggäste abgebildet werden.
  - **Dienstangebot**: Das Dienstangebot bildet die verschiedenen Dienste, die auf dem Flughafen dem Passagier zugänglich sind, mit ihren Eigenschaften ab. Es werden physische und virtuelle Dienste unterschieden. Die physischen Dienste sind durch ihre räumliche Ausdehnung charakterisiert, innerhalb derer der Fluggast diesen Dienst nutzen kann (z.B. Raucherecke). Virtuelle Dienste

dagegen können unabhängig von der Position des Fluggastes in Anspruch genommen werden (z.B. WLAN-Zugang). Die Sichtbarkeit der Dienste (die Bekanntgabe, dass ein Dienst angeboten wird) kann in unterschiedlicher Weise erfolgen. Manche Dienste werden auf dem gesamten Flughafengelände bekannt gegeben, manche nur innerhalb bestimmter Werbezonen, andere (insbesondere die virtuellen Dienste) sind nur in einem beschränkten Zeitraum bekannt und nutzbar.

- **Ereignisse**: Die Ereignisse bilden die Dynamik (z.B. Flugplanänderungen, Auftreten von Ad-hoc-Aufgaben usw.) des Flughafenszenarios ab.

Die oben definierten Hauptkomponenten der Simulation beschreiben die Grundbausteine, die bei der weiteren Analyse der Simulationsumgebung beachtet werden müssen.

4.1.2 Zusammenfassung

Die Aufgabe der Simulationsumgebung ist es, das oben beschriebene Flughafengeschehen realitätsnah für das Indoor-Fußgängernavigationssystem auf einer geeigneten Abstraktionsebene abzubilden und somit das Aufstellen von umfangreichen Testszenarien zu ermöglichen. Dadurch sollen die Weiterentwicklung eines gemäß Kapitel 1.2 verbesserten Indoor-Fußgängernavigationssystems sowie dessen Untersuchung innerhalb mehrerer modellierter virtueller Situationen ermöglicht werden.

4.2 Funktionale Anforderungen

Im Weiteren werden die funktionalen Anforderungen beschrieben und in Form von Anwendungsfalldiagrammen in Unified Modeling Language (s. [Kahlbrand98]) vollständig definiert. Die denkbaren Anwendungsfälle werden zum besseren Verständnis in Sichten auf die Simulationsumgebung gruppiert. Jede Sicht schildert, wie die beteiligten Akteure die Softwaresimulation wahrnehmen und wie die Akteure untereinander agieren. Die Beschreibung der unterschiedlichen Sichten und der dazu gehörigen Anwendungsfälle erfolgt nach dem in [Kahlbrandt98] vorgeschlagenen Schema.

4.2.1 Sicht Simulation

Diese Gruppe von Anwendungsfällen beschreibt die Interaktion zwischen den Akteuren Indoor-Fußgängernavigationssystem und Simulationsumgebung. Das Indoor-Fußgängernavigationssystem ist

eine Software, die Fußgänger in einem Indoor-Szenario navigiert. Die Simulationsumgebung ist die in dieser Arbeit zu entwickelnde Software, die Indoor-Szenarien sowie das Fußgängerverhalten in solchen Szenarien für das Indoor-Fußgängernavigationssystem simuliert. Abbildungen 4-1 und 4-2 stellen die einzelnen Anwendungsfälle grafisch dar.

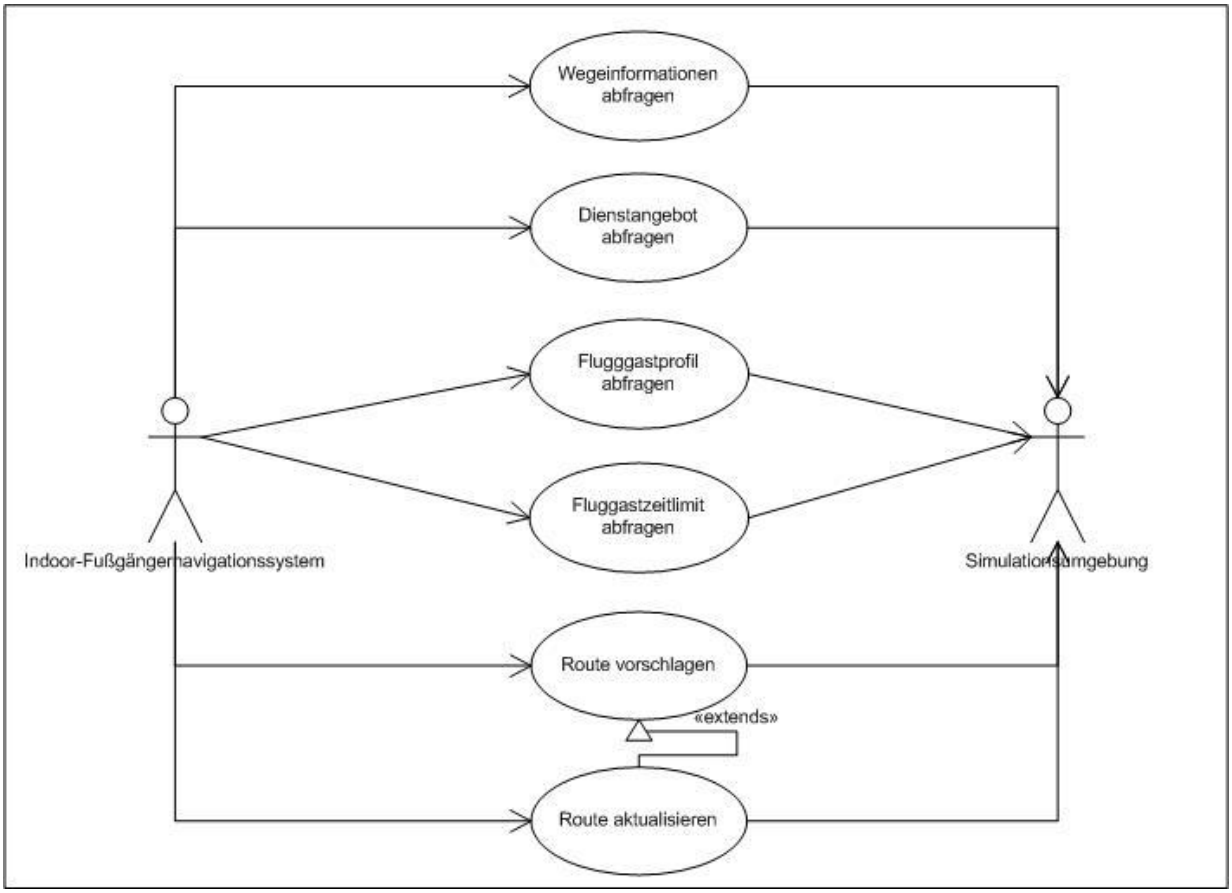

Abbildung 4-1: Anwendungsfälle - Sicht Simulation (Initiator Indoor-Fußgängernavigationssystem)

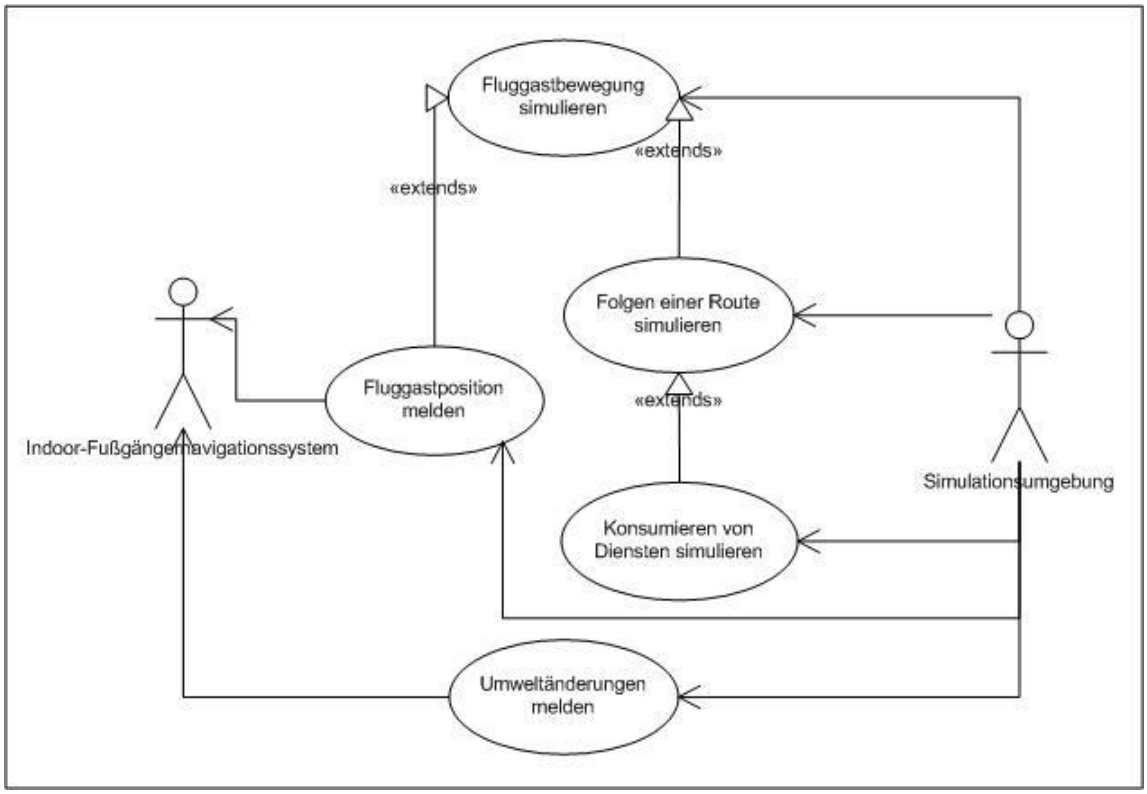

Abbildung 4-2: Anwendungsfälle - Sicht Simulation (Initiator Simulationsumgebung)

Die abgebildeten Anwendungsfälle lassen sich bezüglich des Akteurs, der einen Anwendungsfall initiiert hat, in zwei Untergruppen einordnen. In der Abbildung 4-1 sind nur Fälle abgebildet, bei denen das Indoor-Fußgängernavigationssystem der Initiator ist. In der Abbildung 4-2 ist die Simulationsumgebung der Initiator. Im Weiteren wird jeder Anwendungsfall präzise definiert.

4.2.1.1 Wegeinformationen abfragen

Kurzbeschreibung: Das Indoor-Fußgängernavigationssystem fordert bei der Simulationsumgebung die Informationen über alle möglichen Wege in der aktuell simulierten Flughafengebäudestruktur an.

Vorbedingungen: Die Flughafengebäudestruktur mit allen möglichen Wegen muss von der Simulationsumgebung abgebildet worden sein.

Nachbedingungen: Das Indoor-Fußgängernavigationssystem verfügt über Informationen, die aussagen, wie sich ein Mensch in der abgebildeten Struktur bewegen kann.

Präzise Beschreibung: Um einen Fluggast durch das Flughafengelände navigieren zu können, benötigt das Indoor-Fußgängernavigationssystem mehrere Informationen und unter anderem auch

alle möglichen Wege innerhalb der aktuell simulierten Flughafengebäudestruktur. Die Wegeinformationen bestehen aus allen Teilstrecken, die ein Fluggast zurücklegen kann. Diese sind in einem 2D- bzw. 3D-Raum geordnet. Jede Teilstrecke muss zusätzlich eine mittlere Fortbewegungsgeschwindigkeit definieren, die auf dieser Strecke zu Fuß oder durch Benutzung von anderen Fortbewegungsmöglichkeiten (wie z.B. Rolltreppe, Ausflug, Minibus usw.) vom Fluggast erreicht werden kann.

4.2.1.2 Dienstangebot abfragen

Kurzbeschreibung: Das Indoor-Fußgängernavigationssystem fordert bei der Simulationsumgebung das Dienstangebot des aktuell simulierten Flughafens an.

Vorbedingungen: Das Flughafendienstangebot muss von der Simulationsumgebung abgebildet worden sein.

Nachbedingungen: Das Indoor-Fußgängernavigationssystem verfügt über Informationen, welche Dienste ein Fluggast innerhalb des abgebildeten Flughafens aktuell in Anspruch nehmen kann.

Präzise Beschreibung: Während der Navigation kann ein Indoor-Fußgängernavigationssystem einem Fluggast für ihn interessante Dienste anbieten bzw. den Fluggast zu diesen Diensten navigieren. Dafür benötigt das Navigationssystem Informationen über das Dienstangebot des aktuell simulierten Flughafens. Das Dienstangebot beschreibt, wo welche Dienste (z.B. Raucherecke, Restaurant, Ruheraum usw.) momentan für den jeweiligen Fluggast aktiv und sichtbar sind und wie lange jeder dieser Dienste üblicher Weise von den Fluggästen in Anspruch genommen wird. Um das Auftreten von ortsgebundenen (Dienste, die nur in einer eingeschränkter Umgebung bekannt sind) und zeitgesteuerten (Dienste, die nur zu einer bestimmten Zeit bekannt sind) Diensten mit dieser Abfrage berücksichtigen zu können, kann sie durch Änderungen in der simulierten Umwelt angestoßen werden (s. Anwendungsfall „Umweltänderungen melden“).

4.2.1.3 Fluggastprofil abfragen

Kurzbeschreibung: Um bei der Navigation für den Fluggastes interessante Dienste berücksichtigen zu können, fragt das Indoor-Fußgängernavigationssystem das Profil des aktuell geführten Fluggast bei der Simulationsumgebung ab.

Vorbedingungen: Ein Benutzerprofil wird von der Simulationsumgebung verwaltet.

Nachbedingungen: Das Indoor-Fußgängernavigationssystem kennt auf Grund des Fluggastprofils die Präferenzen des zu führenden Gastes und kann ihn interessierende Dienste anbieten.

Präzise Beschreibung: Um bei der Navigation für den Fluggast interessante Dienste berücksichtigen zu können, fragt das Indoor-Fußgängernavigationssystem das Benutzerprofil des aktuell geführten Fluggastes bei der Simulationsumgebung ab. Das Profil speichert in geeigneter Form die Präferenzen des Fluggastes in Bezug auf Dienstkonsumierung. Diese Informationen werden vom Navigationssystem ausgewertet und bei der Fluggastführung beachtet.

4.2.1.4 Fluggastzeitlimit abfragen

Kurzbeschreibung: Das Indoor-Fußgängernavigationssystem fragt bei der Simulationsumgebung die Zeit ab, die dem aktuell geführten Fluggast bis zu seinem Abflug zu Verfügung steht.

Vorbedingungen: Der Abflugzeitpunkt des Fluggastes ist bekannt.

Nachbedingungen: Das Indoor-Fußgängernavigationssystem hat eine genaue Zeitvorgabe, die bei der Navigation berücksichtigt werden muss.

Präzise Beschreibung: Das Indoor-Fußgängernavigationssystem fragt bei der Simulationsumgebung die Zeit ab, die dem aktuell geführten Fluggast bis zu seinem Abflug zu Verfügung steht. Bei der Navigation muss das Indoor-Fußgängernavigationssystem dieses Zeitlimit berücksichtigen und den Gast pünktlich zu seinem Gate führen.

4.2.1.5 Route vorschlagen

Kurzbeschreibung: Das Indoor-Fußgängernavigationssystem führt den Fluggast durch den Flughafen, indem es durch die Simulationsumgebung dem Fluggast eine Route vorschlägt.

Vorbedingungen: Alle Informationen zu Erstellung der Route sind vorhanden (s. Anwendungsfälle „Wegeinformationen abfragen“, „Dienstangebot abfragen“, „Fluggastprofil abfragen“, „Fluggastzeitlimit abfragen“).

Nachbedingungen: Die Simulationsumgebung erhält eine Route, die von dem Fluggast zu folgen ist.

Präzise Beschreibung: Das Indoor-Fußgängernavigationssystem führt den Fluggast durch den Flughafen, indem es mit Hilfe der Simulationsumgebung dem Fluggast eine Route vorschlägt. Die Route enthält die genauen Strecken, die ein Fluggast zurücklegen soll, die Orte, die der Fluggast besuchen soll und die Dienste, die für den Fluggast interessant sein können. Die Routen-Strecken sind eine Teilmenge der möglichen Wege durch die Flughafengebäudestruktur (s. Anwendungsfall „Wegeinformationen abfragen“). Die Simulationsumgebung nimmt die Route entgegen und startet die Simulation des Benutzerverhaltens (s. Anwendungsfälle „Folgen einer Route simulieren“ und „Konsumieren von Diensten simulieren“).

4.2.1.6 Route aktualisieren

Kurzbeschreibung: Das Indoor-Fußgängernavigationssystem ändert eine aktuelle Fluggastführung durch den Flughafen, indem es mit Hilfe der Simulationsumgebung dem Fluggast eine aktualisierte Route vorschlägt.

Vorbedingungen: Mindestens eine der Informationen, auf deren Basis die aktuelle Route erstellt wurde, hat sich geändert (s. Anwendungsfälle „Wegeinformationen abfragen“, „Dienstangebot abfragen“, „Fluggastzeitlimit abfragen“).

Nachbedingungen: Die Simulationsumgebung erhält eine aktualisierte Route, die von dem Fluggast zu folgen ist.

Präzise Beschreibung: Auf Grund verschiedener Ereignisse in der Simulationsumgebung können sich für die Navigation relevante Daten ändern (s. Anwendungsfall „Umweltänderungen melden“). Dies kann dazu führen, dass die aktuelle Route vom Indoor-Fußgängernavigationssystem geändert wird. Nach solchen Änderungen wird dem Fluggast eine aktualisierte Route durch die Simulationsumgebung bekannt gegeben.

4.2.1.7 Fluggastbewegung simulieren

Kurzbeschreibung: Die Simulationsumgebung simuliert die Fluggastbewegung durch die abgebildete Flughafengebäudestruktur.

Vorbedingungen: Die Flughafengebäudestruktur mit allen möglichen Wegen muss von der Simulationsumgebung abgebildet worden sein.

Nachbedingungen: Fluggastposition hat sich geändert.

Präzise Beschreibung: Die Simulationsumgebung simuliert die Fluggastbewegung durch die abgebildete Flughafengebäudestruktur. Der Fluggast bewegt sich auf in der Gebäudestruktur abgebildeten möglichen Wegen. Die Fortbewegungsgeschwindigkeit des Fluggastes wird durch die mittlere Fortbewegungsgeschwindigkeit auf der aktuellen Strecke bestimmt. Dabei kann die Fluggastgeschwindigkeit von der mittleren Geschwindigkeit auf dieser Strecke in gewissen Grenzen abweichen. Eine Bewegung außerhalb der möglichen Wege ist nicht möglich.

4.2.1.8 Folgen einer Route simulieren

Kurzbeschreibung: Die Simulationsumgebung simuliert, wie ein Fluggast der ihm von dem Indoor-Fußgängernavigationssystem vorgeschlagenen Route (s. Anwendungsfälle „Route vorschlagen“ und „Route aktualisieren“) durch die Flughafengebäudestruktur folgt.

Vorbedingungen: Die Flughafengebäudestruktur mit allen möglichen Wegen muss von der Simulationsumgebung abgebildet worden sein. Ein Routenvorschlag muss existieren (s. Anwendungsfälle „Route vorschlagen“ und „Route aktualisieren“).

Nachbedingungen: Fluggastposition hat sich geändert.

Präzise Beschreibung: Die Simulationsumgebung simuliert, wie ein Fluggast der ihm vom Indoor-Fußgängernavigationssystem vorgeschlagene Route (s. Anwendungsfälle „Route vorschlagen“ und „Route aktualisieren“) durch die Flughafengebäudestruktur folgt. Dabei wird die Fluggastbewegung (s. Anwendungsfall „Fluggastbewegung simulieren“) entlang der vorgeschlagenen Route simuliert. Der Fluggast kann mit einer gewissen Wahrscheinlichkeit von der vorgegebenen Route abweichen. Es gibt zwei Arten von Abweichungen – räumlich und zeitlich. Die räumliche Abweichung besteht darin, dass sich der Fluggast während seiner Bewegung entlang der Route für andere Strecken entscheidet, die nicht von der Route berücksichtigt sind, aber in der Menge aller möglichen Wege durch die Flughafenstruktur enthalten sind, und dadurch die vorgeschlagene Route verlässt. Die zeitliche Abweichung besteht darin, dass die Fortbewegungsgeschwindigkeit des Fluggastes bei seiner Bewegung entlang der Route von der mittleren Fortbewegungsgeschwindigkeit der Routen-Strecken abweicht. Daher kann der Fluggast später oder früher, als vom Indoor-Fußgängernavigationssystem geplant, einen Zielort erreichen.

4.2.1.9 Konsumieren von Diensten simulieren

Kurzbeschreibung: Die Simulationsumgebung simuliert, wie ein Fluggast Dienste des Flughafens während seiner Bewegung entlang ihm vorgeschlagener Route (s. Anwendungsfall „Folgen einer Route simulieren“) nutzt.

Vorbedingungen: Die Flughafengebäudestruktur mit allen möglichen Wegen muss von der Simulationsumgebung abgebildet worden sein. Ein Routenvorschlag muss existieren (s. Anwendungsfälle „Route vorschlagen“ und „Route aktualisieren“). Das Flughafendienstangebot muss von der Simulationsumgebung abgebildet worden sein.

Nachbedingungen: keine

Präzise Beschreibung: Die Simulationsumgebung simuliert, wie ein Fluggast Dienste des Flughafens während seiner Bewegung entlang der ihm vorgeschlagenen Route (s. Anwendungsfall „Folgen einer Route simulieren“) konsumiert. Dabei ist davon auszugehen, dass alle Dienste entlang der vorgeschlagenen Route für den Fluggast interessant sind. Somit muss der Fluggast alle diese Dienste nutzen. Während der Konsumierung von Diensten kann der Fluggast keine weiteren Aktivitäten ausüben. Jeder Dienst hat eine mittlere Zeitdauer, die die Konsumierung des Dienstes in Anspruch nimmt. Der Fluggast konsumiert den Dienst über die mittlere Zeitdauer des Dienstes. Erst wenn der Fluggast mit der Konsumierung eines Dienstes fertig ist, kann er seine Bewegung fortsetzen (s. Anwendungsfall „Folgen einer Route simulieren“).

4.2.1.10 Fluggastposition melden

Kurzbeschreibung: Die Simulationsumgebung meldet die Änderungen der Fluggastposition an das Indoor-Fußgängernavigationssystem.

Vorbedingungen: Der Fluggast hat sich bewegt (s. Anwendungsfall „Fluggastbewegung simulieren“).

Nachbedingungen: Das Indoor-Fußgängernavigationssystem kennt die aktuelle Position des Fluggastes innerhalb der Flughafengebäudestruktur.

Präzise Beschreibung: Die Simulationsumgebung meldet die Änderungen der Fluggastposition an das Indoor-Fußgängernavigationssystem. Ahand der aktuellen Position kann das Indoor-Fußgängernavigationssystem feststellen, inwieweit der Fluggast der vorgeschlagenen Route folgt

oder von dieser Route zeitlich bzw. räumlich abweicht. Bei Abweichungen kann das Indoor-Fußgängernavigationssystem entsprechende Routenänderungen vorschlagen (s. Anwendungsfall „Route aktualisieren“).

4.2.1.11 Umweltänderungen melden

Kurzbeschreibung: Die Simulationsumgebung meldet ereignisbedingte Änderungen in der simulierten Umwelt an das Indoor-Fußgängernavigationssystem.

Vorbedingungen: Eine ereignisbedingte Änderung in der simulierten Umwelt ist aufgetreten.

Nachbedingungen: Das Indoor-Fußgängernavigationssystem ist über Änderungen in der simulierten Umwelt informiert und kann entsprechende Aktionen einleiten.

Präzise Beschreibung: Die Simulationsumgebung meldet ereignisbasierte Änderungen in der simulierten Umwelt an das Indoor-Fußgängernavigationssystem. Ereignisbasierte Änderungen sind zeitliches oder ortsbezogenes Auftreten von Diensten, die als Änderungen des Dienstangebotes betrachtet werden, Änderungen des Fluggastzeitlimits, verursacht durch z.B. Änderungen im Flugplan, Konfrontation des Fluggastes mit Adhoc-Aufgaben, die dringend erledigt werden müssen.

4.2.2 Sicht Systembetreiber

In dem angegangenen Unterkapitel wurde beschrieben, wie die zukünftige Simulationsumgebung und das Indoor-Fußgängernavigationssystem untereinander agieren und welche Funktionalitäten von welcher Seite zur Verfügung gestellt werden sollen. Das Ziel der Simulationsumgebung ist es, die Beobachtung und Auswertung des Einsatzes des Indoor-Fußgängernavigationssystems in mehreren Testszenarien zu ermöglichen. Dabei werden insbesondere die Reaktionen des Indoor-Fußgängernavigationssystems auf Abweichungen von der vorgeschlagenen Route und ereignisbasierte Veränderungen in der abgebildeten Umgebung betrachtet. Aus diesem Grund wird neben den zwei bereits definierten Akteuren ein zusätzlicher benötigt – der Systembetreiber. Dieser Akteur ist ein Mensch, der Testszenarien erstellt und anschließend den Ablauf des Simulationsprozesses kontrolliert, beobachtet und auswertet. Die Abbildung 4-3 stellt die einzelnen Anwendungsfälle grafisch dar.

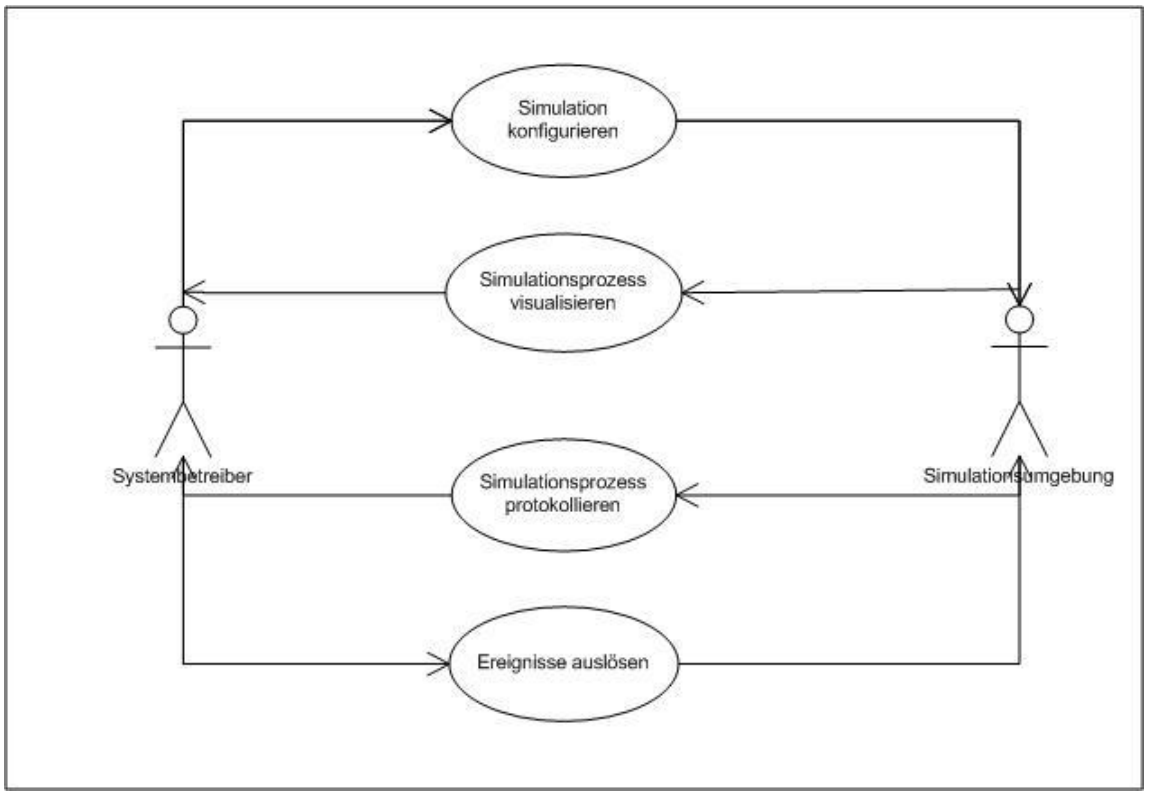

Abbildung 4-3: Anwendungsfälle - Sicht Systembetreiber

4.2.2.1 Simulation konfigurieren

Kurzbeschreibung: Der Systembetreiber konfiguriert ein Testszenario.

Vorbedingungen: keine

Nachbedingungen: Ein Testszenario ist vorhanden und eine Simulation kann durchgeführt werden.

Präzise Beschreibung: Damit eine Simulation des gewünschten Flughafenkontextes möglich ist, muss dieser Kontext in Form eines Testszenarios vorliegen. Das Testszenario definiert die Flughafengebäudestruktur, die Wegeinformationen, das Dienstangebot, die Fluggäste und deren Positionen, Fluggastprofile und Fluggastzeitpläne. Diese Informationen konfiguriert der Systembetreiber durch geeignete Eingabemöglichkeiten in der Simulationsumgebung.

4.2.2.2 Simulationsprozess visualisieren

Kurzbeschreibung: Die Simulationsumgebung visualisiert den aktiven Simulationsprozess.

Vorbedingungen: keine

Nachbedingungen: Die Zustandsänderungen und der aktuelle Zustand des aktiven Simulationsprozesses sind visuell abgebildet.

Präzise Beschreibung: Die Simulationsumgebung stellt die aktuellen Zustandsänderungen sowie den aktuellen Zustand des Simulationsprozesses grafisch dar. Der Schwerpunkt bei der Darstellung sind die realistische Abbildung der Fluggastbewegung durch die Flughafengebäudestruktur und das Konsumieren von Diensten. Während einer laufenden Simulation wird dieser Anwendungsfall ständig wiederholt.

4.2.2.3 Simulationsprozess protokollieren

Kurzbeschreibung: Die Simulationsumgebung protokolliert den aktiven Simulationsprozess.

Vorbedingungen: keine

Nachbedingungen: Alle Zustandsänderungen des aktiven Simulationsprozesses sind dauerhaft dokumentiert.

Präzise Beschreibung: Die Simulationsumgebung protokolliert jede Zustandsänderung des Simulationsprozesses (z.B. Positionsänderung eines Fluggastes, Aktivieren eines Dienstes, Auftreten eines Ereignises) sowie die Kommunikation mit dem Indoor-Fußgängernavigationssystem (z.B. versenden von Navigationsanweisungen). Dabei wird die Uhrzeit des jeweiligen Ereignisses festgehalten. Auf Grund des so erstellten Protokolls kann der Systembetreiber den genauen Ablauf des Simulationsprozesses zu einem späteren Zeitpunkt nachvollziehen bzw. untersuchen.

4.2.2.4 Ereignisse auslösen

Kurzbeschreibung: Der Systembetreiber beeinflusst den aktuellen Zustand des Simulationsprozesses, indem er unterschiedliche Ereignisse auslöst.

Vorbedingungen: keine

Nachbedingungen: Der Zustand des Simulationsprozesses ist verändert.

Präzise Beschreibung: Der Systembetreiber kann den aktuellen Zustand des Simulationsprozesses ereignisbasiert beeinflussen, indem er z.B. das Dienstangebot des simulierten Flughafens bzw. das Fluggastzeitlimit ändert oder Fluggäste mit Adhoc-Aufgaben konfrontiert.

4.3 Nicht funktionale Anforderungen

Nachdem die funktionalen Anforderungen an das System beschrieben worden sind, werden in diesem Kapitel die nicht funktionalen Anforderungen definiert. Diese stellen die Qualitätsmerkmale des zukünftigen Systems dar. Zum besseren Überblick werden die Anforderungen im Folgenden in zwei Gruppen als Anforderungen an die Entwicklungsplattform und solche an die zukünftige Simulationsumgebung vorgestellt.

4.3.1 Entwicklungsplattform

Als Ergebnis der Auswertung der in Kapitel 3 aufgeführten wissenschaftlichen Veröffentlichungen mit gleicher oder ähnlicher Aufgabenstellung wie diese Arbeit wird hier zur Entwicklung der gewünschten Simulationsumgebung der Einsatz von Software-Agenten vorgeschlagen. Dieser Vorschlag wird durch die in Kapitel 4.2 beschriebenen funktionalen Anforderungen an die zukünftige Softwaresimulationsumgebung und insbesondere den Schwerpunkt der Simulation – das Abbilden des menschlichen Verhaltens - bekräftigt. Dieser Einsatz geschieht mit Hilfe des Software-Agentenparadigmas, das wiederum in mehrerer Plattformen und Umgebungen für Software-Agenten realisiert wird (s. Kapitel 5.1). Dabei setzt jede Plattform ihren eigenen Schwerpunkt (z.B. Interoperabilität, einfache Erweiterbarkeit usw.). Aus diesem Grund werden im Weiteren an die Entwicklungsplattform Anforderungen aufgestellt, die die Qualität der zukünftigen Softwaresimulationsumgebung sichern. In Kapitel 5.1 wird auf Grund dieser Anforderungen eine geeignete Entwicklungsplattform ausgewählt. Die notwendigen Anforderungen sind folgende:

- **Offene Schnittstellen:** Es ist abzusehen, dass die Simulationsumgebung mit externen Systemen (z.B. Indoor-Fußgängernavigationssystem, externen Datenbanken, usw.) agieren wird. Daher muss die Plattform die einfache Anbindung heterogener externer Systeme durch offene Schnittstellen unterstützen.
- **Einfacher Zugriff auf GUI- und 2D-Grafik-Bibliotheken:** Die zu entwickelnde Simulationsumgebung muss komplexe Simulationsprozesse (z.B. Bewegung im 2D-Raum, Konsumieren von Diensten) durch geeignete grafische Benutzeroberflächen darstellen. Das erfordert innerhalb der Entwicklungsumgebung den einfachen Zugriff auf GUI- und 2D-Grafik-Bibliotheken.

- **Geeignete Werkzeugpalette:** Es ist vorauszusehen, dass die Simulationsprozesse (besonders in einem realen Szenario) zunehmend komplexer werden. Aus diesem Grund muss die Plattform den Entwicklungsprozess und die Testphase durch geeignete Werkzeuge (z.B. für Analyse, Debugging, Logging, Administration usw.) unterstützen.
- **Standardisierte Technologien:** Es ist von Vorteil, wenn die Entwicklungsplattform den Einsatz von heute vorhandenen Standardtechnologien (z.B. Programmiersprachen, Entwicklungsumgebungen, weitere Entwickler-Werkzeuge) ermöglicht.

4.3.2 Simulationsumgebung

Im Folgenden werden zusätzliche nicht funktionale Anforderungen an die zu entwickelnde Simulationsumgebung definiert, die die Komplexität der Simulationsumgebung festlegen:

- **Visualisierung und Zeitaspekt:** Die Visualisierung und der Zeitaspekt der Simulationsprozesse müssen von der künftigen Umgebung realitätsgetreu abgebildet werden. Das bedeutet, dass simulierte Ereignisse eine realistische Zeitdauer haben. Der Bezug auf die Realität kann für Simulationszwecke eingestellt werden. Die Visualisierung (grafische Abbildung) muss die realen Größenverhältnisse zwischen allen abgebildeten Objekten richtig darstellen. Der Maßstab der Darstellung kann zu Simulationszwecken verändert werden.
- **Evaluierung:** Alle Ereignisse der abgebildeten Simulationsprozesse müssen protokolliert werden. An den Schnittstellen der verschiedenen Komponenten der Simulationsumgebung sind die Aufrufe zu protokollieren, um nachzuweisen, welche Informationen bereitgestellt werden. Die Protokollierung muss eine genaue Evaluierung während und auch nach Beendigung eines Simulationsprozesses ermöglichen.
- **Schnittstellen:** Es ist abzusehen, dass die Simulationsumgebung mit heterogenen externen Systemen über Schnittstellen agiert (s. Kapitel „Paradigma und Entwicklungsplattform“).Um die Möglichkeit der Wartbarkeit und die Interoperabilität des Gesamtsystems auf Dauer zu gewährleisten, ist es nutwendig, dass die von der Simulationsumgebung angebotenen Schnittstellen geeignete Standards unterstützen.
- **Verteilung:** Es ist vorauszusehen, dass die simulierten Testszenarien relativ komplex sein werden. Um komplexe Szenarien abbilden zu können, werden viele Rechenressourcen (im Sinne von Speicher, Prozessor-Kapazität usw.) benötigt, die nicht immer von einem System zur Verfügung gestellt werden können. Aus diesem Grund muss die Architektur der Simulationsumgebung eine sinnvolle physikalische Verteilung des Gesamtsystems auf mehrere Rechner über Intra- bzw. Internet ermöglichen.

- **Erweiterbarkeit:** Um die Simulationsumgebung nicht nur auf das definierte Szenario (s. Kapitel 4.1) zu begrenzen und effizient neue funktionale Anforderungen implementieren zu können, ist es notwendig, dass die Anwendungsarchitektur die Möglichkeit der Erweiterbarkeit des Systems anbietet.

4.4 Fazit

Mit den in diesem Kapitel formulierten funktionalen bzw. nicht funktionalen Anforderungen werden der genaue Leistungskatalog und die Qualitätsmerkmale der künftigen Simulationsumgebung festgelegt. Dabei konnte der Vorschlag aus Kapitel „Vergleichbare Arbeiten“, das Software-Agentenparadigma als Basis für Konzeption und Entwicklung der zukünftigen Softwaresimulationsumgebung einzusetzen, bekräftigt werden. Daher werden das Software-Agentenparadigma zusammen mit den hier definierten Anforderungen in Kapitel 5 bei der Auswahl einer geeigneten Plattform für Software-Agenten und Erstellung einer tragfähigen Architektur für die gewünschte Simulationsumgebung berücksichtigt. Anschließend in Kapitel 6 wird auf Grund der definierten Architektur eine konkrete Realisierung der Simulationsumgebung in Form eines Prototyps vorgestellt.

5. Konzeption

In diesem Kapitel wird gemäß den definierten Anforderungen ein abstrakter System-Entwurf der geforderten Software-Simulationsumgebung konzipiert. Dazu wird eine geeignete Entwicklungsplattform eine tragfähige Architektur der künftigen Simulationsumgebung aufgestellt. Die Ergebnisse der Konzeption werden dokumentiert und anschließend hinsichtlich der Systemanforderungen analysiert und bewertet. Das Endergebnis wird dann in Kapitel 6 verwendet, um eine konkrete Realisierung der Softwaresimulationsumgebung in Form eines Prototyps zu erstellen.

5.1 Auswahl der Entwicklungsplattform

Wie bei der Anforderungsanalyse (s. Kapitel 4.3) beschrieben, wird das Software-Agentenparadigma bei der Entwicklung der Simulationsumgebung eingesetzt, um das typische Fluggastverhalten im definierten Flughafenszenario optimal abbilden zu können. Das Software-Agentenparadigma findet Ausprägung in mehreren Entwicklungsplattformen für Software-Agenten. Dabei wird bei jeder einzelnen Plattform festgelegt, in welcher Form und mit welchem Schwerpunkt (z.B. Simulationsentwicklung, Realisierung von BDI-Agenten usw.) das Software-Agentenparadigma realisiert wird und wie es mit anderen Programmier-Paradigmen (z.B. objektorientierter bzw. funktionaler Programmierung) interagieren kann. Dadurch werden Eigenschaften (z.B. Interoperabilität, Wartbarkeit, Erweiterbarkeit, Verfügbarkeit usw.) und denkbare Anwendungsbereiche (z.B. Robotersteuerung bzw. Bewegungssimulation) der Software-Agentensysteme, die auf Basis der jeweiligen Entwicklungsumgebung realisiert wurden, beeinflusst bzw. definiert. Drei solcher Plattformen werden im Folgenden analysiert. Gründe für die Auswahl genau dieser Plattformen werden im Einzelnen bei der Beschreibung gegeben.

Als ein besonderer Vertreter oben genannter Plattformen wird SeSAm im ersten Schritt vorgestellt. Diese Plattform bietet eine abgeschlossene generische Entwicklungs- und Laufzeit- sowie Experimentierumgebung für Agentensimulationen. Im zweiten Schritt wird JADE als Entwicklungs- bzw. Laufzeitumgebung für javabasierte Agentensysteme beschrieben. Im dritten Schritt wird JADEX als eine Erweiterung der JADE-Plattform vorgestellt. Anschließend wird eine Auswahl getroffen, welche der drei Plattformen für Erstellung einer Simulationsumgebung gemäß den definierten Anforderungen (s. Kapitel 4.2 und 4.3) am besten geeignet ist. Dabei muss berücksichtigt werden, dass die Auswahl der Plattform in gewissem Maße Auswirkungen auf das Aufstellen einer tragfähigen Architektur haben wird.

5.1.1 SeSAm

SeSAm (Shell for Simulated Agent Systems) ist eine generische Entwicklungs- und Ablauf- sowie Experimentierumgebung für Agentensimulationen. Die Umgebung ist hauptsächlich ausgerichtet auf die einfache Konstruktion komplexer Bewegungsmodelle und ermöglicht eine vereinfachte Bedienung ohne umfangreiche Programmierkenntnisse, indem die Modelle visuell definiert werden. Der Ablauf jeder SeSAm-Simulation kann mit geringem Entwicklungsaufwand grafisch dargestellt werden. Da die Umgebung zahlreiche Werkzeuge zur genauen Verfolgung des simulierten Szenarios (z.B. Abbildung des Agentenzustandes bzw. Umgebungszustandes) zur Verfügung stellt (s. [Klügl03]), kann der Beobachter jederzeit und im Einzelnen die Simulation verfolgen. Abbildung 5-1 gibt den ersten Überblick über die Gestaltung der SeSAm-Umgebung.

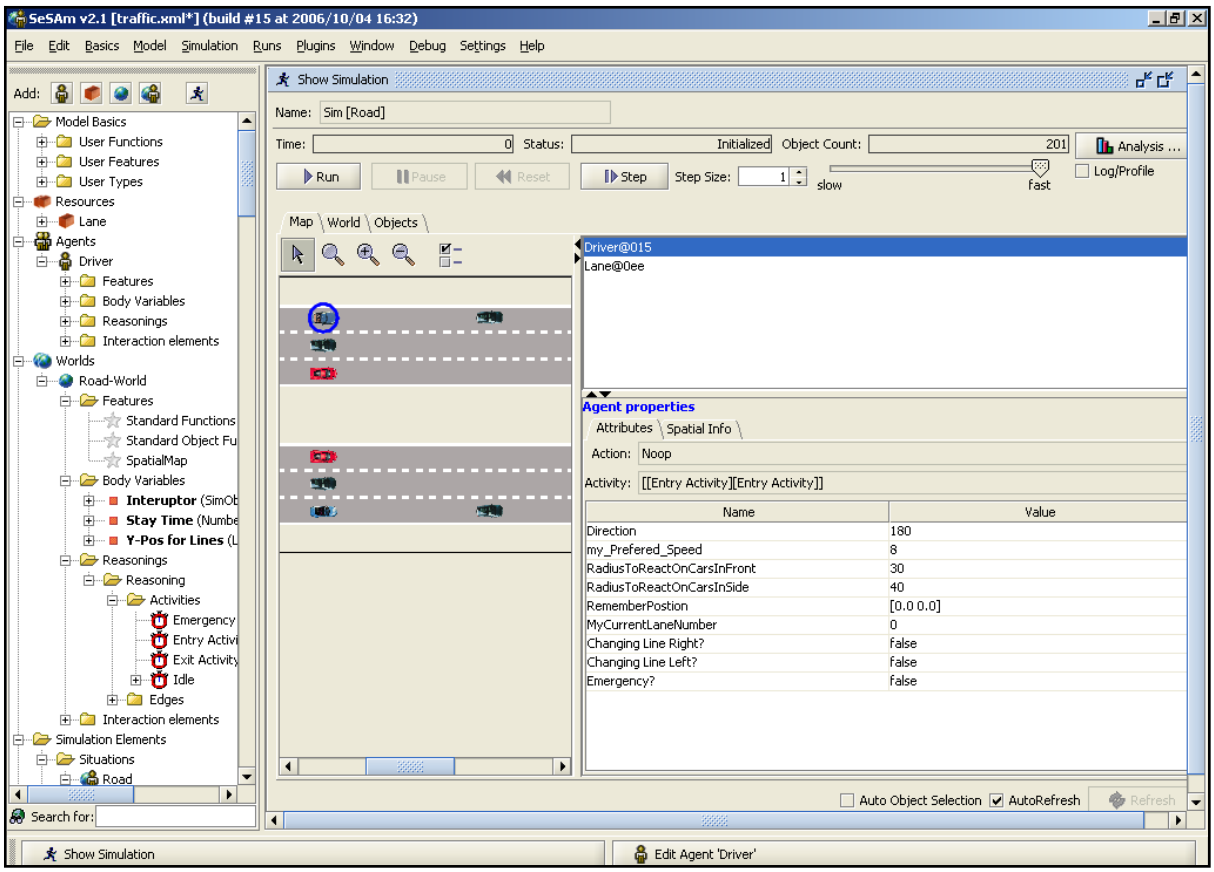

Abbildung 5-1: SeSAm - Überblick

5.1.1.1 Plattformdetails

Das SeSAm-Simulationsmodell basiert auf drei Entitäten: Agenten, Ressourcen und Welt.

- **Agent:** Die Agenten werden in SeSAm mittels Entscheidungsregeln und interner Variablen definiert. Die Entscheidungsregeln beschreiben das Agentenverhalten. Sie werden durch spezielle Aktivitätsdiagramme definiert (ähnlich UML-Aktivitätsdiagrammen (s. [Kahlbrandt98])). Ein Agent befindet sich zu jedem Zeitpunkt innerhalb einer Aktivität und führt deren Aktionen aus. Aktivitäten werden durch Regeln verbunden (grafisch und logisch). Die Aktionen und Regeln werden in einem vordefinierten SeSAm-Format formuliert. Eine Aktivität wird entweder, nachdem alle Aktionen abgearbeitet sind, beendet oder gewechselt, indem eine Regel aktiviert wird. Die internen Variablen bilden den aktuellen Agentenzustand ab.

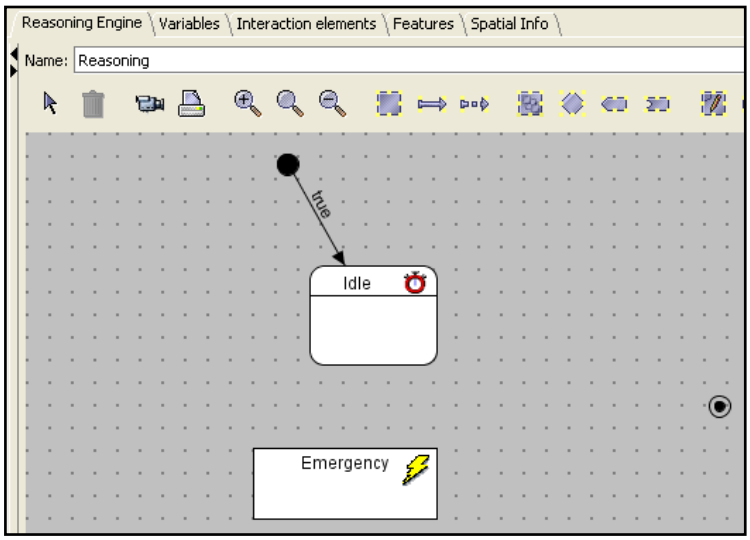

Abbildung 5-2: Einfacher SeSAm-Agent

In der Abbildung 5-2 ist ein einfacher SeSAm-Agent mit seinem grafisch definierten Verhalten abgebildet. Der Agent besitzt eine leere Aktivität, die sofort nach Erzeugung des Agenten durch die Regel mit Bedingung „true“ aktiviert wird. Die Aktivität „Idle“ definiert keine Aktionen und wird daher sofort sie abgearbeitet. Da diese Aktivität mit keiner anderen Aktivität oder mit dem Ende des Diagramms verbunden ist, verursacht ihre Ausführung eine endlose Schleife – die Idle-Aktivität wird immer wieder ausgeführt. Durch wenige Maus-Klicks kann das Diagramm und damit das Verhalten des Agenten so erweitert werden, dass der Agent seine Aufgabe nach einem Simulationsschritt beendet. Das resultierende Verhalten ist in der Abbildung 5-3 vorgestellt. Die Abbildung 5-4 stellt ein komplexes Aktivitätsdiagramm eines Agenten grafisch dar, der ein vereinfachtes Autofahrerverhalten simuliert.

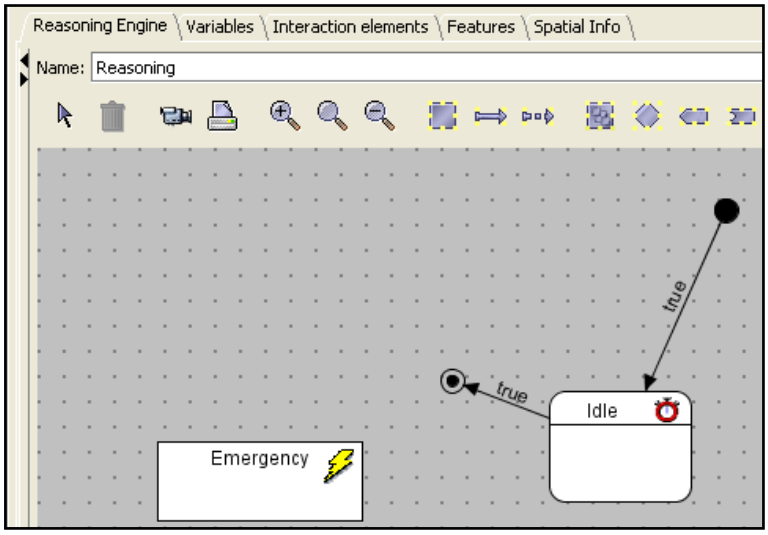

Abbildung 5-3: Erweiterter SeSAm-Agent

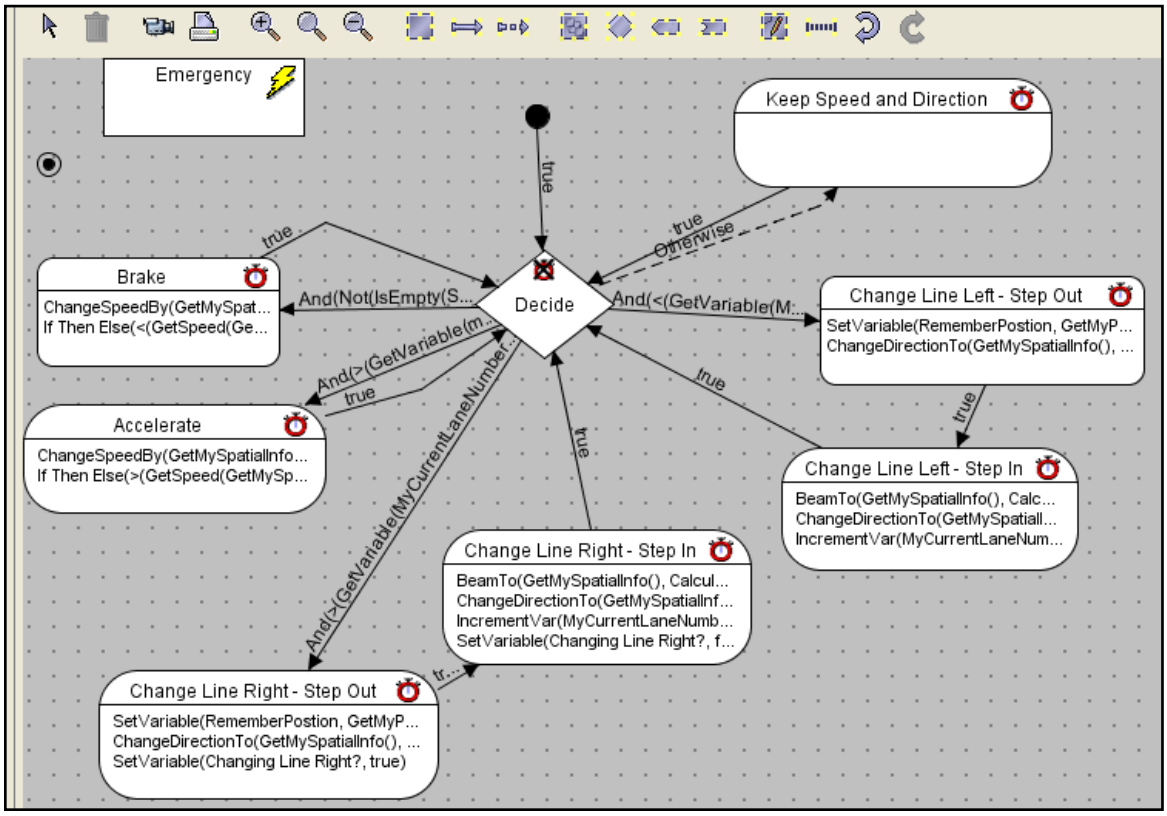

Abbildung 5-4: Komplexer SeSAm-Agent

In der Abbildung 5-5 wird dargestellt, wie die internen Variablen der Software-Agenten in SeSAm definiert und bearbeitet werden. Der Benutzer hat eine umfangreiche Auswahl zwischen unterschiedlichen Variablentypen und Funktionen. Die zahlreichen vordefinierten

Funktionen können zur Berechnung von Start- bzw. Folgewerten der internen Variablen benutzt werden.

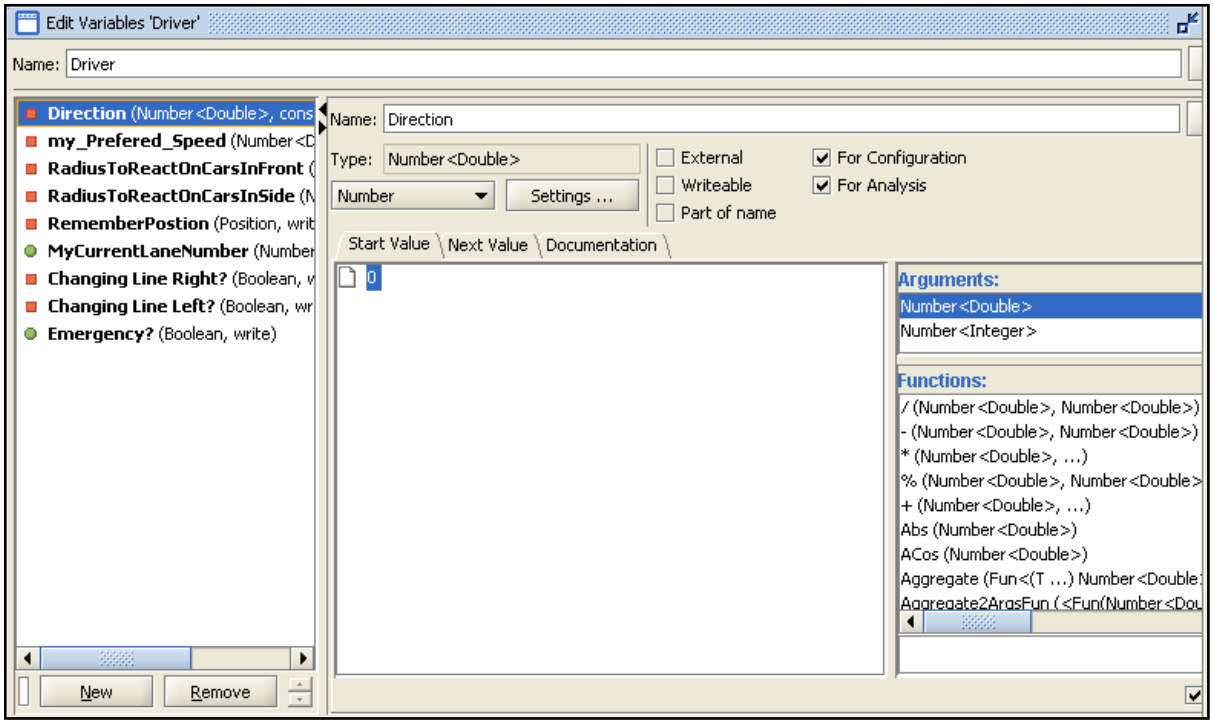

Abbildung 5-5: Variablendefinition in SeSAm

- **Ressource:** Im Unterschied zu Agenten, sind Ressourcen ausschließlich durch Variablen und nicht durch Aktionen gekennzeichnet. Die Ressourcen treten als passive Elemente auf und können von Agenten manipuliert (z.B. aufgebraucht, erzeugt oder verändert) werden.

- **Welt:** Durch die Modellierung der Agenten und Ressourcen unter SeSAm werden das Verhalten der einzelnen Elemente der Simulation und deren Kontext definiert. Um vollständige Simulation abbilden zu können, müssen das Verhalten bzw. die Eigenschaften der Umgebung, in der sich Agenten und Ressourcen befinden, modelliert werden. Dafür ist in SeSAm eine besondere Entität (die Welt, realisiert als Welt-Agent) vorgesehen. Ähnlich wie bei den „normalen“ SeSAm-Agenten werden das Verhalten und die Eigenschaften des Welt-Agenten durch Aktivitätsdiagramme und interne Variablen abgebildet. Der Welt-Agent übernimmt eine besondere Rolle in der Simulation, indem er ein globales Kommunikationsmedium, den Simulations-Agenten durch seine Variablen bereitstellt. Der Welt-Agent ist für die Generierung des Anfangszustandes der Simulation verantwortlich. Er legt z.B. fest, wie viele Agenten und Ressourcen zum Beginn einer Simulation gestartet bzw. erzeugt werden und in welchem Zustand diese sich befinden. Darüber hinaus kann der Welt-Agent die Rolle eines globalen Synchronisators übernehmen.

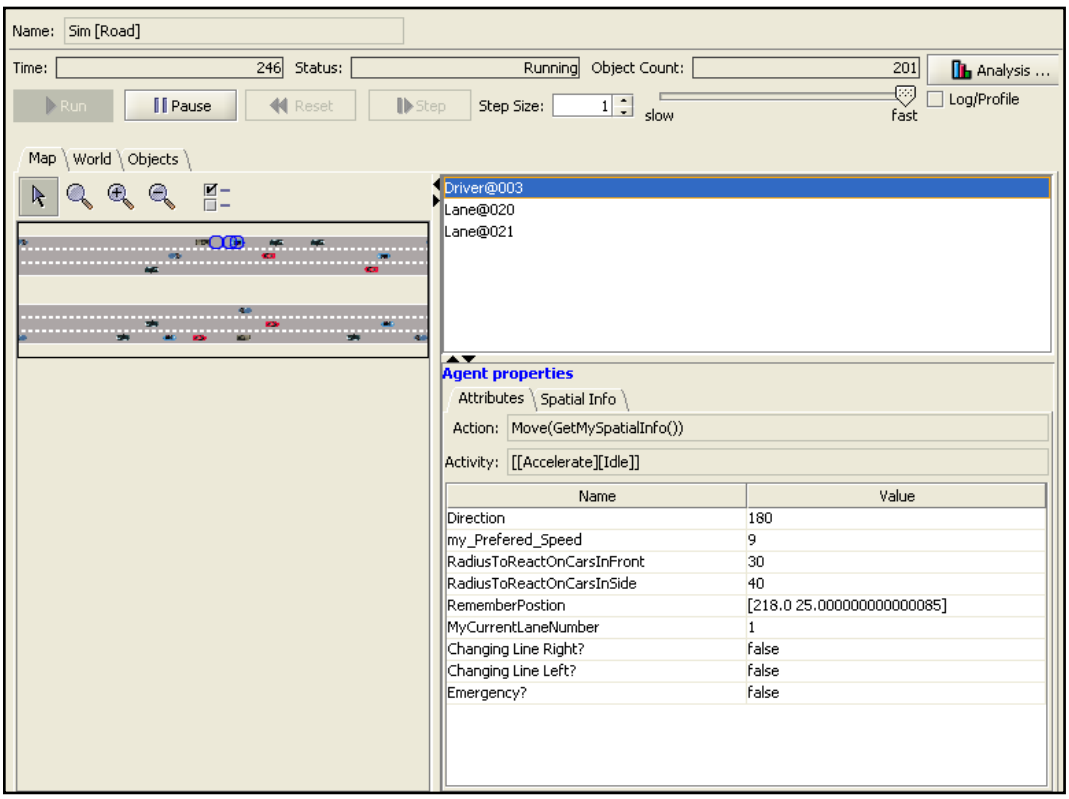

Abbildung 5-6: SeSAm-Simulation zur Laufzeit

Jede SeSAm-Simulation kann grafisch visualisiert werden. Das Visualisierungsmodell ist besonders für Bewegungssimulationen gut geeignet. Dies kann die Analyse einer Simulation sehr verdeutlichen und vereinfachen. Weiterhin kann der aktuelle Zustand (Belegung der internen Variablen sowie die aktuelle Aktivität und Aktion) der Agenten bzw. der Umgebung beobachtet werden (s. Abbildung 5-6). Die Geschwindigkeit der Simulation kann leicht eingestellt werden. Um komplexe Berechnungen durchführen zu können, bietet SeSAm zahlreiche vordefinierte Funktionen. Die Funktionen können beliebig kombiniert und zu weiteren neuen Funktionen zusammengefasst werden.

5.1.1.2 Analyse und Bewertung

Da SeSAm ursprünglich zur Simulation in sich geschlossener und vollständig in der Entwicklungsumgebung repräsentierter Agentensysteme konzipiert war, bietet es zunächst keine offenen Schnittstellen zur Interaktion mit externen Systemen (z.B. Datenbanken oder anderen Agentensystemen usw.) an. Schnittstellen sowie weitere neue Funktionalitäten können durch

Erstellung von SeSAm-Plugins (Erweiterungen), die spezielle Java-Klassen sind, bereitgestellt werden. Diese erfordern eine Konfiguration mittels unübersichtlicher und schwierig wartender XML-Dateien.
Die Plattform ermöglicht neben dem Einsatz des Software-Agentenparadigmas auch einen begrenzten Einsatz der objektorientierten Programmierung im Rahmen der Plugin-Entwicklung. Die Entwicklung wird in der Sprache Java realisiert. Da jedoch keine der heute bekannten Java-Entwicklungsumgebungen (wie z.B. J-Builder oder Eclipse) die Plugin-Entwicklung unterstützen, sind der Test und das Debugging der Plugins nicht auf direktem Wege möglich. Daher sind nur relativ einfache Erweiterungen der vordefinierten SeSAm-Funktionalitäten bzw. –Mechanismen möglich. Die physikalische Verteilung der SeSAm-Simulationen erfordert einen sehr hohen zusätzlichen Aufwand.

5.1.2 JADE

JADE (Java Agent DEvelopment Framework) ist eine Entwicklungs- und Laufzeitumgebung für javabasierte Agentensysteme. JADE ermöglicht eine vereinfachte Entwicklung und den Einsatz von verteilten Agentensystemen durch ihren betriebssystemunabhängigen Java-Kern (eine den FIPA-Standard unterstützende Middleware) und mittels grafischer Werkzeuge, die das Ausführen und die Analyse der Agentensysteme erleichtern.

5.1.2.1 Plattformdetails

Die Hauptelemente der JADE-Architektur und deren Zusammenhänge sind in der Abbildung 5-7 dargestellt.

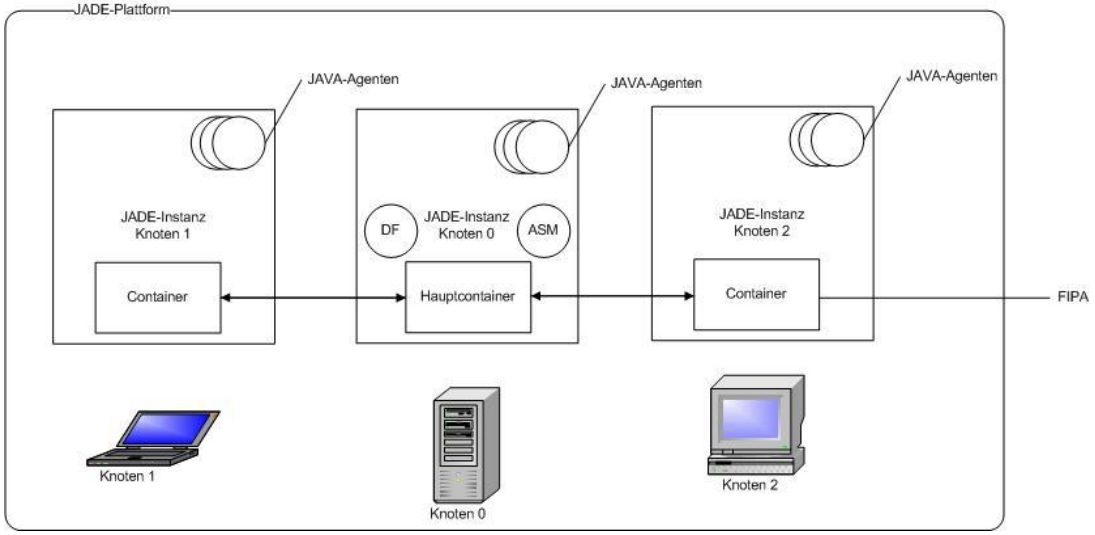

Abbildung 5-7: JADE-Architektur nach [Bellifemine07]

Die verteilte JADE-Plattform besteht aus mehreren JADE-Instanzen. Einer JADE-Instanz werden ein Container und mehrere Agenten zugeordnet. Jeder Agent ist mit genau einem Container verbunden. Der Container ist ein Java-Prozess, der die Ausführung „seiner“ Agenten initiiert und steuert und ihnen Schnittstellen zu Grundfunktionalitäten der JADE-Plattform (wie z.B. Nachrichtentransport, Migration usw.) zur Verfügung stellt. Eine besondere Rolle in der JADE-Architektur übernimmt der Hauptcontainer. Er ist der Startpunkt der JADE-Plattform. Er muss als erster gestartet werden und alle weiteren Container müssen sich bei ihm registrieren. Zu den Aufgaben des Hauptcontainers gehören folgende Aktivitäten:

- Verwaltung der Container-Information (Container Table - CT) bzw. Infrastruktur-Information. Diese Information enthält die physikalischen Adressen aller im System vorhandenen Container-Knoten.
- Verwaltung der Agenten-Information (Global Agent Descriptor Table - GADT). Diese Information beschreibt alle im System vorhandenen Agenten sowie deren Status und aktuelle Lokation bzw. Container-Zuordnung. Durch diesen Mechanismus wird die Ortstransparenz der Agenten sichergestellt – ein Agent kann jederzeit angesprochen werden unabhängig davon in, welchem Container er sich momentan befindet.
- Ausführen von ASM- und DF-Agenten. Der ASM-Agent (Agent Management System, s. Kapitel Agenten-Management) bietet Funktionalität zu Steuerung und Kontrolle der gesamten JADE-Plattform. Jeder weitere Agent muss sich beim ASM-Agenten registrieren, um in der bzw. mit Plattform agieren zu können. Der DF-Agent (Directory Facillator s. Kapitel Agenten-Management) bietet die Funktionalität eines Dienstverzeichnisses (ähnlich wie UUDI-Registry im Webservice-Szenario, s. [Dustdar03]), in dem die JADE-Agenten deren Dienste registrieren bzw. nach Diensten anderer Agenten suchen können.

Wie jede zentrale Komponente eines verteilten Systems ist der Hauptcontainer eine Schwachstelle für die Fehlertoleranz des Gesamtsystems. Sollte der Hauptcontainer ausfallen, können keine Agenten adressiert werden. Um dieser Situation entgegen wirken zu können, wird die Agenten-Information (GADT) von den einzelnen Containern dezentral gespeichert und bei Bedarf aktualisiert. Zusätzlich ist eine Verteilung der Hauptcontainer-Rolle auf mehrere Knoten möglich (s. [Bellifemine07]).

Wie oben beschrieben, werden Container und Agenten in der JADE-Plattform eng verbunden. Die Agenten übernehmen im Gegensatz zu den Containern eine aktive Rolle. Durch die JADE-Agenten wird die gewünschte Funktionalität des Agentensystems realisiert, indem das Verhalten und die Eigenschaften jedes Agenten in der Programmiersprache Java definiert werden. Dabei muss die Architektur des JADE-Agenten (s. Abbildung 5-8) berücksichtigt werden.

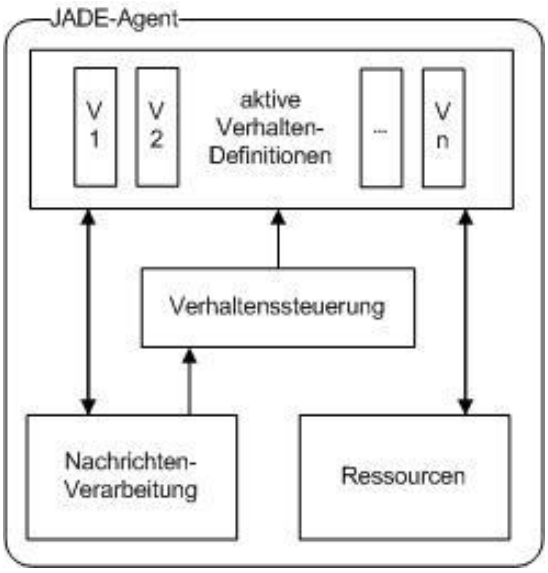

Abbildung 5-8: JADE-Agent-Architektur nach [Bellifemine07]

Die Aktivitäten eines JADE-Agenten werden durch sogenannte Verhaltensdefinitionen modelliert. Diese Definitionen bestehen aus Befehlsreihenfolgen, die von dem jeweiligen Agenten ausgeführt werden können. Die Komponente Verhaltenssteuerung stellt Schnittstellen zur Registrierung der Verhaltensdefinitionen eines Agenten zur Verfügung. Erst wenn eine Verhaltensdefinition registriert ist, kann sie aktiviert bzw. ausgeführt werden. Zusätzlich kontrolliert die Komponente „Verhaltenssteuerung“ die genaue Ausführung der einzelnen Verhaltensdefinitionen.

Die Verhaltenssteuerung unterscheidet zwischen zwei Gruppen von Verhaltensdefinitionen (s. Abbildung 5-9). Zu der ersten Gruppe „SimpleBehaviour“ (einfache oder atomare) gehören Verhaltensdefinitionen mit folgenden Eigenschaften:

- **OneShotBehaviour:** Diese Verhaltensdefinition hat die Eigenschaft, dass sie nur einmal aktiviert wird. Die Aktivierung erfolgt sofort. Nach ihrem Ausführen, wird diese

Verhaltensdefinition verworfen. Solche Definitionen eignen sich zum Implementieren von einmaligen Aktionen im Agentenkontext.

- **WakerBehaviour:** Diese Verhaltensdefinition basiert auf der OneShotBehaviour-Verhaltensdefinition und erweitert diese, indem das Aktivieren erst nach einem vordefinierten Zeitintervall stattfindet.
- **CyclicBehaviour:** Eine Definition dieser Art wird cyclisch von der Verhaltenssteuerung aktiviert. Diese Art der Verhaltensdefinitionen wird bei sich ständig wiederholenden Agentenaktionen eingesetzt.
- **TickerBehaviour:** Diese Verhaltensdefinition basiert auf der CyclicBehaviour-Verhaltensdefinition und erweitert diese, indem das Wiederholen der Aktivitäten in vordefinierten Zeitabständen stattfindet.

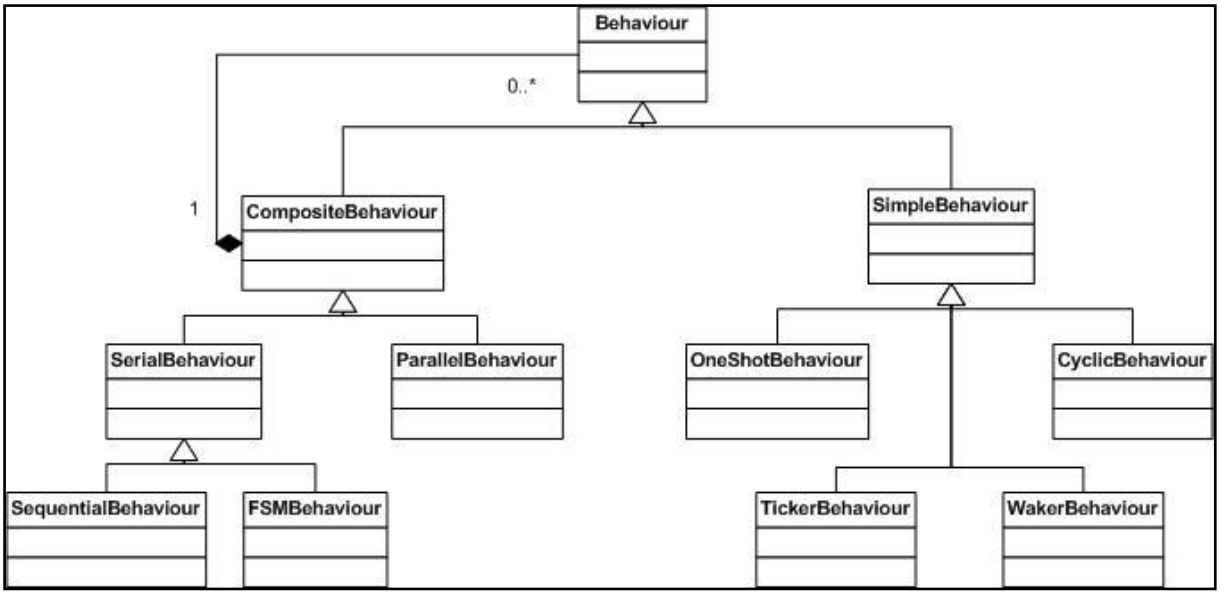

Abbildung 5-9: Hierarchie der JADE-Verhaltensdefinitionen nach [Bellifemine07]

Die Gruppe CompisiteBehaviour enthält zusammengesetzte Verhaltensdefinitionen mit komplexen Eigenschaften.

- **SerialBehaviour:** Diese Verhaltensdefinition ist eine Zusammensetzung von mehreren Verhaltensdefinitionen, die in einer seriellen Reihenfolge abgearbeitet werden. Dabei kann die Reihenfolge einfach (SequentialBehaviour) oder durch einen Automaten (FSMBehaviour) definiert werden.
- **ParallelBehaviour:** Die parallele Verhaltensdefinition besteht aus mehreren Definitionen (ähnlich wie SerialBehaviour), die parallel abgearbeitet werden. Es müssen alle parallelen

Verhaltensdefinitionen abgearbeitet worden sein, bevor die ParallelBehaviour als abgearbeitet gilt.

Die von JADE realisierte Hierarchie von Verhaltensdefinitionen ist offen und es können bei Bedarf weitere Arten von Verhaltensdefinitionen realisiert werden, um komplexere Agentenaufgaben abbilden zu können.

Neben der Art der Verhaltensdefinition kann die Nachrichtenverarbeitung für die Ausführung der Verhaltensdefinitionen eine Rolle spielen. Jede Verhaltensdefinition kann (während sie ausgeführt wird) Nachrichten aus der Nachrichtenverwaltung entnehmen bzw. Nachrichten über die Nachrichtenverwaltung an andere Agenten senden. Der Nachrichtenaustausch verläuft asynchron, indem die Verhaltenssteuerung durch Schnittstellen die Möglichkeit anbietet, dass sich Verhaltensdefinitionen für ausgewählte Nachrichtenarten registrieren können. Eine Verhaltensdefinition kann ihre eigene Ausführung unterbrechen und auf Nachrichten warten. Beim Eintreffen einer „passenden“ Nachricht wird die Verhaltenssteuerung von der Nachrichtenverwaltung informiert. Die Verhaltenssteuerung setzt dann das Ausführen der betroffenen unterbrochenen Verhaltensdefinition fort. Durch diesen Registrierungsmechanismus werden das Entwickeln und die Kontrolle von umfangreichen Agenten-Kommunikationspfaden vereinfacht. Da der Nachrichtenaustausch die FIPA-Vorgaben erfüllt, ist die Kommunikation auch mit weiteren FIPA-standardisierten Agentensystemen möglich.

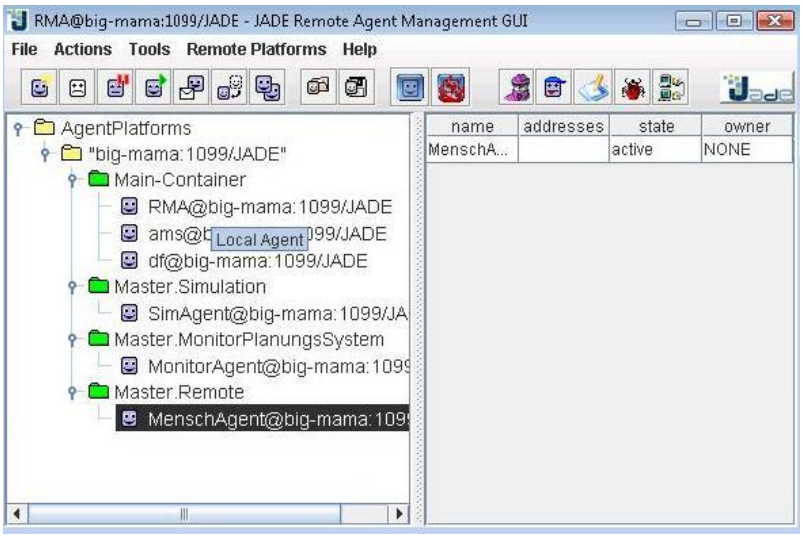

Abbildung 5-10: JADE-Hauptbenutzeroberfläche

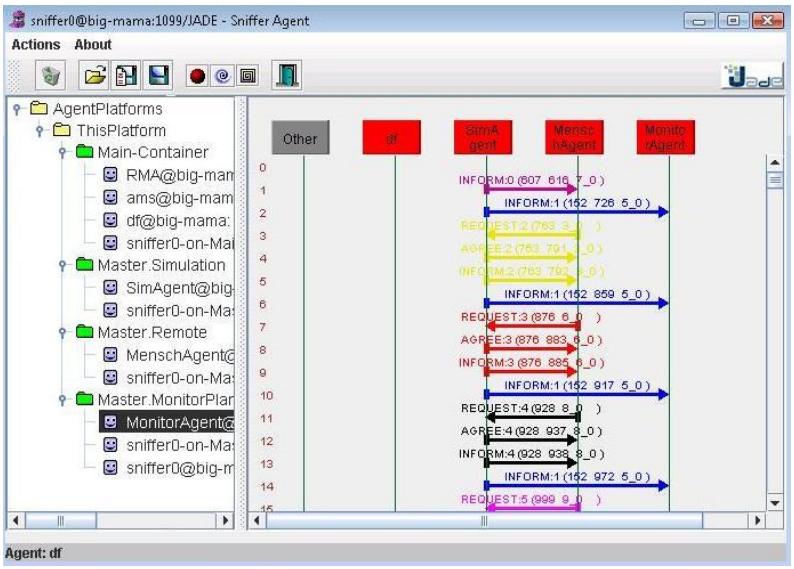

Abbildung 5-11: JADE-Nachrichtenverfolgung

Die JADE-Entwicklungsumgebung bietet mehrere grafische Werkzeuge zur Steuerung und Analyse der JADE-Plattform, Beobachtung und Kontrolle der Agentenzustände sowie Nachrichtenverfolgung. Da die JADE-Plattform und -Entwicklungsumgebung wie oben beschrieben javabasiert sind, werden die Agenten sowie die Verhaltensdefinitionen mittels JAVA-Klassen implementiert. Dadurch bietet JADE neben einer umfangreichen Werkzeugpalette einen zusätzlichen Komfort durch Einsatz der heute verfügbaren JAVA-Entwicklungsumgebungen und deren zusätzlichen Möglichkeiten (wie. Kode-Verwaltung, Debugging usw.).

5.1.2.2 Analyse und Bewertung

Die JADE-Plattform bietet nicht nur umfangreiche Funktionalitäten zur Entwicklung von Agentensystemen sondern auch eine vollständige Laufzeitumgebung. FIPA-standardisierte Basis-Dienste wie z.B. standardisiertes asynchrones Nachrichtensystem, Dienstverzeichnis (DF-Agent), Systemmanagement (ASM-Agent) usw. ermöglichen eine einfache Konzeption von komplexen Agentensystemen. Die Plattform kann leicht durch Implementieren weiterer Funktionalitäten, Agenten und / oder Verhaltensdefinitionen der Programmiersprache Java erweitert werden. Daneben ist der Zugriff auf die komplette Java-Funktionalität (z.B. 2D- b zw. 3D-Grafik, GUI-Schnittstelen usw.) gewährleistet. Neben dem Software-Agentenparadigma ist der vollwertige Einsatz von objektorientierter Programmierung möglich. Die Implementierung kann in den heute verfügbaren Java-Entwicklungsumgebungen (wie z.B. J-Builder oder Eclipse) vorgenommen werden, was die Entwicklung nachhaltig erleichtert und deren Qualität sichert. Das standardisierte Nachrichtensystem und die komplette Java-Funktionalität erlauben eine optimale Anbindung von externen Systemen. Die umfangreichen JADE-Werkzeuge reduzieren den Aufwand für die Entwicklung besonders von komplexen Systemen in bedeutendem Maße.

5.1.3 JADEX

JADEX (Java Agent DEvelopment Framework Extension) wurde als Erweiterung der JADE-Agenten-Plattform an der Universität Hamburg entwickelt (s. [Pokahr07]). JADEX baut auf den von JADE (s. Kapitel 5.1.2) angebotenen Funktionalitäten auf und realisiert eine BDI-Plattform (s. Kapitel 2.2.2) für Java-Software-Agenten. Ab Version 0.96 kann JADEX (laut [Pokahr07]) nicht nur im Kontext von JADE sondern auch im Kontext jeder beliebigen Agenten-Plattform eingesetzt werden. Dadurch stellt JADEX eine Abstraktionsebene für effiziente Entwicklung von BDI-Software-Agenten zur Verfügung. Dadurch wird jedoch das Agentenparadigma auf die BDI-Architektur begrenzt.

5.1.3.1 Plattformdetails

Entsprechend der BDI-Architektur spielen bei JADEX die Entitäten Plans, Beliefs und Goals eine zentrale Rolle. Diese werden als Objekte realisiert, die innerhalb von Agenten erzeugt und geändert werden können. Dabei repräsentieren die Goals die konkreten zu erreichenden Ziele von einem JADEX-Agenten. Um seine Zeile zu erreichen, muss der Agent bestimmte Handlungen bzw. Plans ausführen. Diese werden in der Sprache Java formuliert. Beliefs repräsentieren das Wissen eines Agenten über seine Umgebung. Sie werden durch beliebige Java-Objekte abgebildet und in einer Beliefs-Datenbank des JADEX-Agenten gespeichert. Der Agent kann zur Laufzeit die Beliefs-Daten abfragen bzw. ändern.

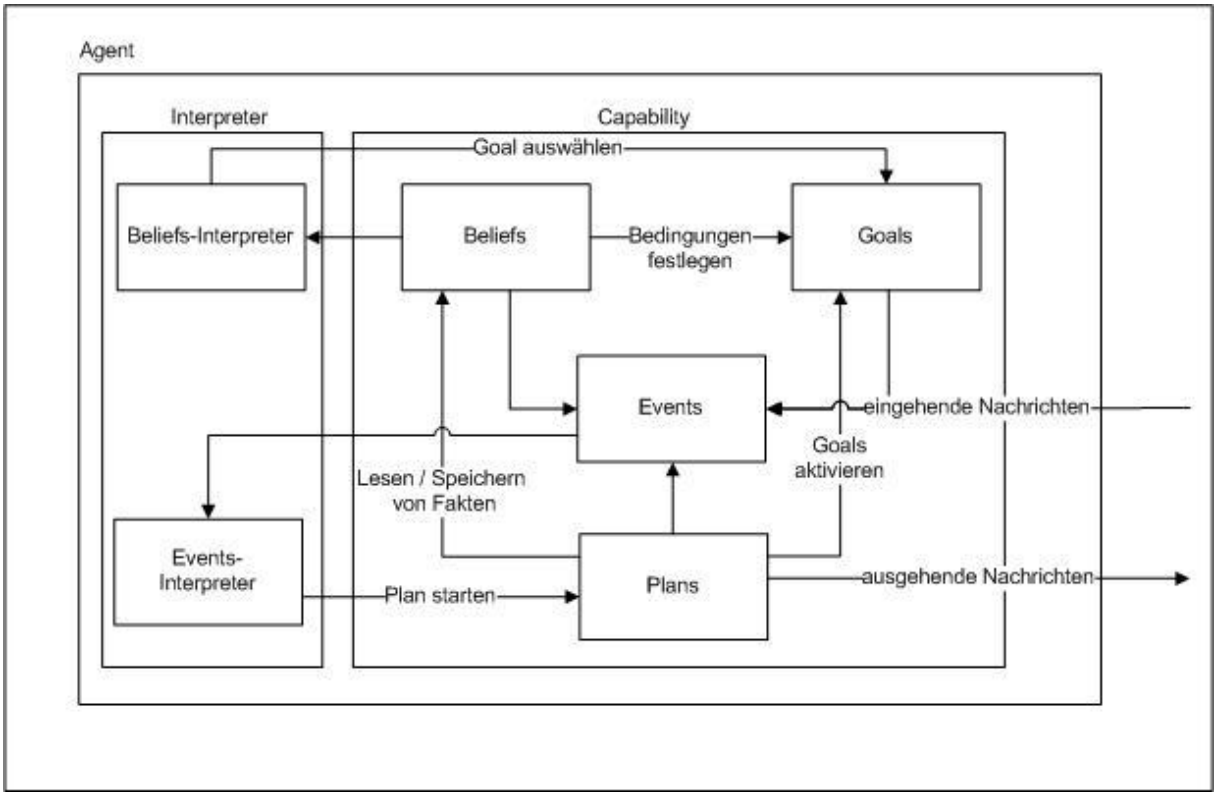

Abbildung 5-12: Abstrakte JADEX-Architektur nach [Pokahr07]

Eine wichtige Komponente der BDI-Architektur, der Interpreter, ist laut der JADEX abstrakten Architektur (s. Abbildung 5-12) Teil jedes JADEX-Agenten. Er wird durch zwei untergeordnete Komponenten (Beliefs-Interpreter und Events-Interpreter) realisiert. Der Events-Interpreter kontrolliert auf Grund von aufgetretenen Ereignissen (z.B. eingegangenen Nachrichten, erreichte Ziele, geänderte Beliefs usw.) die Ausführung aktueller Pläne (Plans) eines JADEX-Agenten. Der Beliefs-Interpreter entscheidet auf Grund seines Wissens (Beliefs-Daten) über die Auswahl der aktuellen Ziele (Goals) eines Agenten.

Ähnlich wie in JADE muss auch in JADEX das Agentenverhalten vom Entwickler des Agentensystems definiert werden. Erst danach kann ein Agent ausgeführt werden. Das Agentenverhalten wird in JADEX durch die Definition der schon oben erwähnten drei Komponenten einer BDI-Architektur Plans, Goals und Beliefes sowie zusätzlicher JADEX-spezifischer Informationen festgelegt. Die Agentenverhaltensdefinitionen werden in speziellen XML-Dateien (Agent Definition File) sowie zusätzlichen Java-Code-Dateien (für die Pläne eines Agenten) verwaltet (s. Abbildung 5-13).

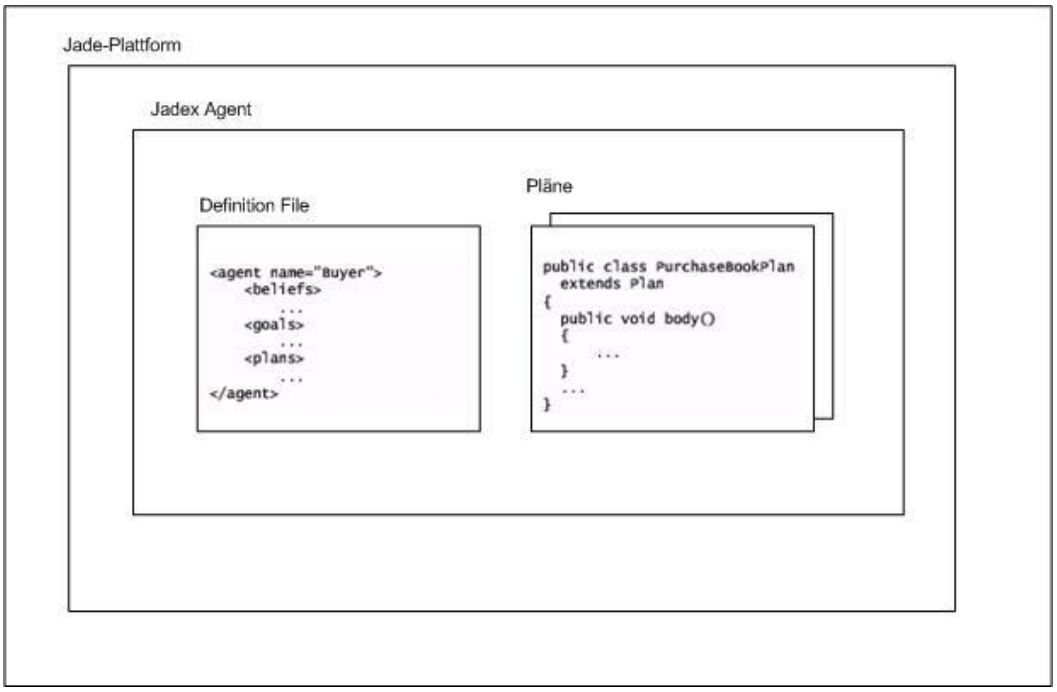

Abbildung 5-13: JADEX-Agentenstruktur nach [Pokahr07]

In der Abbildung 5-14 ist der Inhalt einer ADF-Datei (Agent Definition File) vorgestellt. Zu dem definierten Verhalten gehören mehrere Pläne, Beliefs sowie zusätzliche Informationen, die bei der Ausführung des Agenten von der JADEX-Plattform ausgewertet werden. In der Abbildung 5-15 ist eine entsprechend der in Abbildung 5-14 vorgestellten Konfiguration Planimplementierung abgebildet.

```
- <agent xmlns="http://jadex.sourceforge.net/jadex" xmlns:xsi="http://www.w3.org/2001/
    xsi:schemaLocation="http://jadex.sourceforge.net/jadex http://jadex.sourceforge.net
    package="jadex.examples.puzzle.humanplayer">
  + <imports>
  - <beliefs>
    + <!--  -->
      <belief name="board" class="IBoard" />
      <!-- The gui of the game board.  -->
    - <belief name="board_gui" class="BoardGui">
        <fact>new BoardGui($agent.getExternalAccess(), $beliefbase.board, true)</fact>
      </belief>
    </beliefs>
  - <plans>
    - <plan name="make_move">
        <body>new MakeMovePlan()</body>
      - <trigger>
          <messageevent ref="request_move" />
        </trigger>
      </plan>
    - <plan name="takeback_move">
        <body>new TakebackPlan()</body>
      - <trigger>
          <messageevent ref="request_takeback" />
        </trigger>
      </plan>
    </plans>
```

Abbildung 5-14: Beispiel einer JADEX-Verhaltensdefinition

```
package jadex.examples.puzzle.humanplayer;

import jadex.runtime.*;
import jadex.examples.puzzle.*;
import jadex.adapter.fipa.Done;

/**
 *  Takeback a requested move.
 */
public class TakebackPlan extends Plan
{
	/**
	 * The body method is called on the
	 * instatiated plan instance from the scheduler.
	 */
	public void body()
	{
		IMessageEvent me = (IMessageEvent)getInitialEvent();
		RequestTakeback rt = (RequestTakeback)me.getContent();

		IBoard board = (Board)getBeliefbase().getBelief("board").getFact();
		if(board.takeback())
		{
			sendMessage(me.createReply("inform_action_done", new Done(rt)));
		}
		else
		{
			System.out.println("Cannot takeback move.");
			sendMessage(me.createReply("failure"));
		}
	}
}
```

Abbildung 5-15: Beispiel eines JADEX-Plans

Um den Aufwand bei Entwicklung von komplexen Agentensystemen zu reduzieren, können häufig verwendete Agentenverhaltensdefinitionen wiederverwendet bzw. weitergegeben oder gruppiert werden, indem sie zu so genannten Capabilities zusammengefasst werden. Die Capabilities sind besondere Art von Verhaltensdefinitionen und werden auch durch XML und Java-Code-Dateien abgebildet. Sie können dann in weiteren Agentendefinitionen leicht angebunden werden.

Als Unterstützung für den Agentensystementwickler bietet JADEX neben den vorhandenen Jade-Werkzeugen zusätzliche Tools, die das BDI-Agentenparadigma (s. Abbildung 5-16 – 5-18) unterstützen.

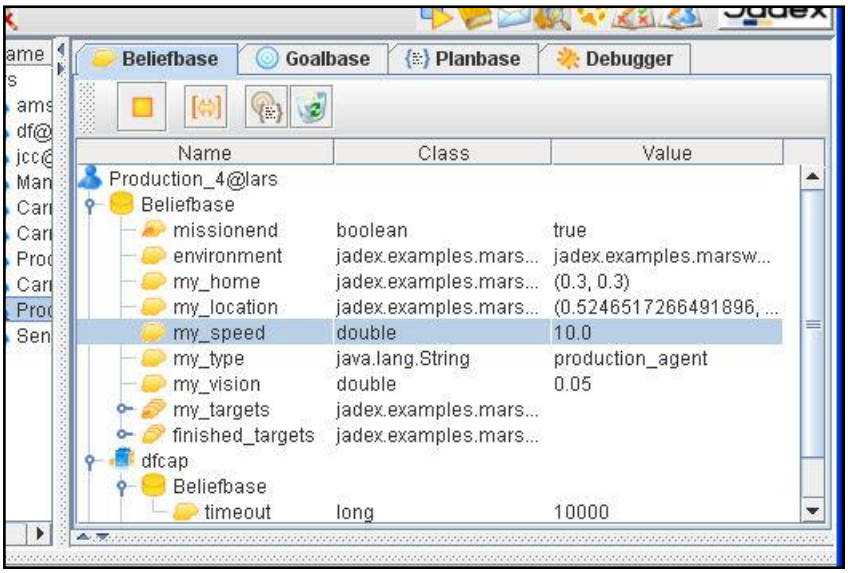

Abbildung 5-16: Ausschnitt aus Beliefs-Daten eines JADEX-Agenten

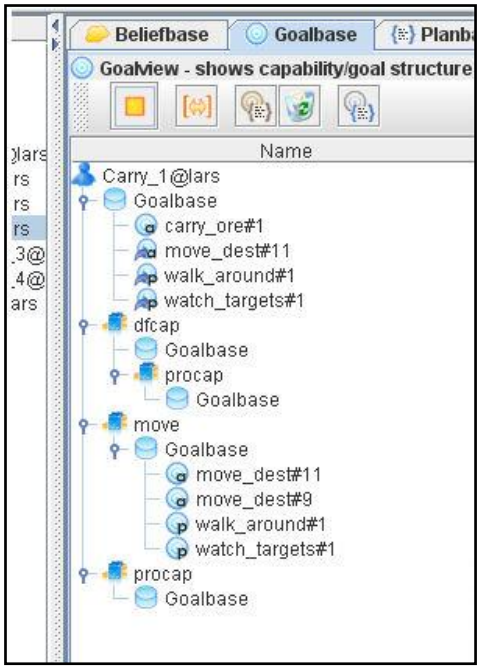

Abbildung 5-17: Ziele eines JADEX-Agenten

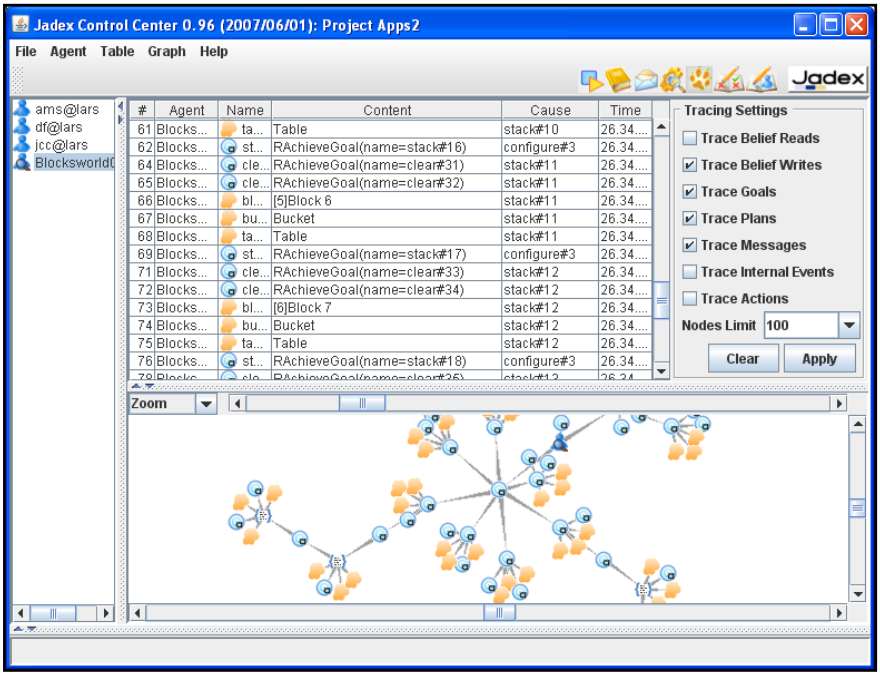

Abbildung 5-18: Ablaufverfolgung eines JADEX-Agenten

5.1.3.2 Analyse und Bewertung

Der Schwerpunkt der JADEX-Agentenumgebung ist die Erweiterung der JADE-Plattform in Hinsicht auf eine einfache Realisierung von Agentensystemen mit BDI-Architektur. So ist auch die komplexe

Struktur von JADEX-Agenten zu erklären, bei der das Agentenverhalten durch Java-Klassen und zusätzliche umfangreiche XML-Definitionen realisiert wird. Entwicklung und Test sind,wie erwähnt, nicht in vorhandenen Java-Entwicklungsumgebungen möglich, sondern nur mit Hilfe spezieller JAEX-Werkzeuge möglich, die für das BDI-Agentenparadigma entwickelt wurden. Darüber hinaus ist zu berücksichtigen, dass nach [Bellifemine07] BDI-Agentensysteme optimal zur Modellierung von Robotersteuerung, weniger jedoch zur Beschreibung von menschlichem Verhalten (Fluggastverhalten) geeignet sind. Dies lässt den Einsatz von JADEX für das definierte Szenario (s. Kapitel 4.1) ausgesprochen ungünstig erscheinen, da laut Aufgabenstellung ja gerade das menschliche Verhalten modelliert werden soll. Von daher ist die Entwicklung der gewünschten Simulationsumgebung auf Basis der JADEX-Plattform abzulehnen.

5.1.4 Zusammenfassung

In einem ersten Schritt wurde im Rahmen dieser Arbeit die SeSAm-Entwicklungsplattform für Software-Agenten als Möglichkeit, Fluggastverhalten zu simulieren, in Betracht gezogen, da es umfangreiche Bewegungsmodelle abbilden kann. Da SeSAm jedoch keinen offenen Schnittstellen zur Interaktion mit externen Systemen bietet, ist die Plattform nur begrenzt erweiterbar.Das Agentenverhalten kann nur innerhalb der SeSAm-Plattform definiert werden, Test und Entwicklung von Agenten sind nicht in vorhandenen Java-Entwicklungsumgebungen möglich. Trotz des mit der komplexen Struktur verbundenen hohen Rechenaufwand, ist eine physikalische Verteilung der SeSAm-Plattform ist nicht vorgesehen. Aus den genannten Gründen wurde entschieden, die Entwicklung der geforderten Software-Simulationsumgebung auf Basis von SeSAm nur nachrangig in Betracht zu ziehen.

Im Folgeschritt wurde nach einer Alternative zu SeSAm gesucht und der Einsatz der JADE-Plattform für Software-Agenten analysiert. JADE unterstützt zwar nicht direkt das Abbilden von Bewegungsmodellen, besitzt jedoch mehrere Eigenschaften, die bezogen auf die definierten Anforderungen (s. Kapitel 4.3.2) bei der Entwicklung der komplexen Simulationsumgebung vorteilhaft erscheinen. JADE bietet umfangreiche Funktionalitäten für Entwicklung von verteilten Agentensystemen. Die Plattform verfügt über einen standardisierten Nachrichtenaustausch. JADE kann leicht erweitert werden und bietet offene Schnittstellen zur Interaktion mit externen Systemen. Das Agentenverhalten wird durch flexible Verhaltensdefinitionen in Form von Java-Klassen festgelegt. Dabei ist der Zugriff auf die komplette Java-Funktionalität gewährleistet. Test und Entwicklung von Agenten sind innerhalb von vorhandenen Java-Entwicklungsumgebungen möglich.

Diese insgesamt „positiven“ Eigenschaften sprechen für einen aussichtsreichen Einsatz von JADE als Entwicklungsbasis der geforderten Simulationsumgebung.

Im letzten Schritt wurde der Einsatz der JADEX-Plattform für Software-Agenten auf Eignung für die geforderte Systementwicklung analysiert. Der Ausgangsgedanke dazu war, dass JADEX die Möglichkeiten der JADE-Plattform besonders im Bereich BDI erweitert. Auf Grund des starken BDI-Schwerpunktes lässt sich JADEX jedoch eigentlich besser zum Modellieren komplexer Ablaufpläne, (etwa im Bereich Robotersteuerung) als für das menschlichen Bewegungsverhalten einsetzen. Das ergab einen Widerspruch mit dem definierten Szenario der Arbeit. Dieser Widerspruch zum definierten Szenario dieser Arbeit wird besonders bei der Agentenstruktur deutlich. Das Agentenverhalten wird durch begrenzte Java-Funktionalität in Form eng spezialisierter Java-Klassen und durch zusätzliche umfangreiche XML-Definitionen modelliert. Die eingesetzte Java-Funktionalität und die XML-Definitionen sind für BDI-Strukturen optimiert und sehr eingeschränkt für das Abbilden von Fluggastverhalten geeignet.

Nach Vergleich und Analyse der drei Agentensysteme wurde entschieden, die JADE-Plattform als Basis für die Entwicklung der geforderten Simulationsumgebung einzusetzen. Diese Entscheidung wird sowohl bei der Konzeption der fachlichen Architektur als auch der Realisierung des Systems berücksichtigt.

5.2 Fachliche Architektur

In diesem Kapitel wird die fachliche Architektur der künftigen Software-Simulationsumgebung auf Basis der ausgewählten JADE-Entwicklungsplattform aufgestellt. Die einzelnen Komponenten der Architektur werden mit ihren Zuständigkeiten und Schnittstellen definiert. Anschließend wird das Komponentenzusammenspiel auf Grund von Prozessen der Fluggastnavigation und Fluggastbewegung erläutert.

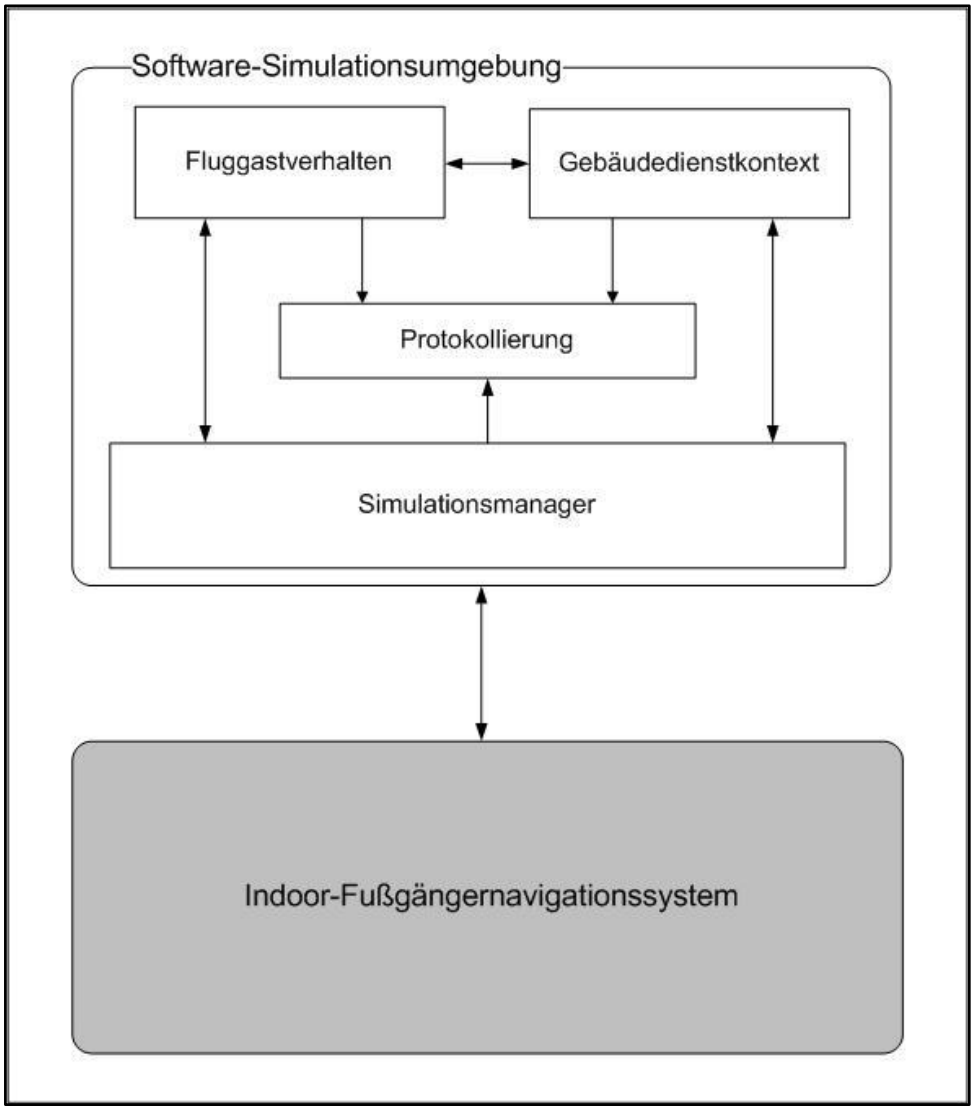

Abbildung 5-19: Fachliche Architektur

In der Abbildung 5-19 wird eine erste Vision des Gesamtsystems vorgestellt. Dieses besteht aus der zu entwickelnden Software-Simulationsumgebung und dem zu evaluierenden Indoor-Fußgängernavigationssystem. Um die gestellten Anforderungen (s. Kapitel 4.2 und 4.3) erfüllen zu können, besteht die Software-Simulationsumgebung aus vier Komponenten: „Fluggastverhalten“, „Gebäudedienstkontext“, „Protokollierung“ und „Simulationsmanager“. Das Fußgängernavigationssystem wird im Gesamtszenario ein externer Akteur gesehen, seine genaue interne Struktur wird daher von der fachlichen Architektur nicht betrachtet. Naturgemäß nuss die Architektur geeignete Anbindungsmöglichkeiten zwischen der Software-Simulationsumgebung und dem Navigationssystems sichern.

5.2.1 Fluggastverhalten

Die Komponente Fluggastverhalten (s. Abbildung 5-20) ist für die Abbildung des typischen Verhaltens der Fluggäste in einer repräsentativen Flughafenumgebung (s. Kapitel 4.1) verantwortlich.

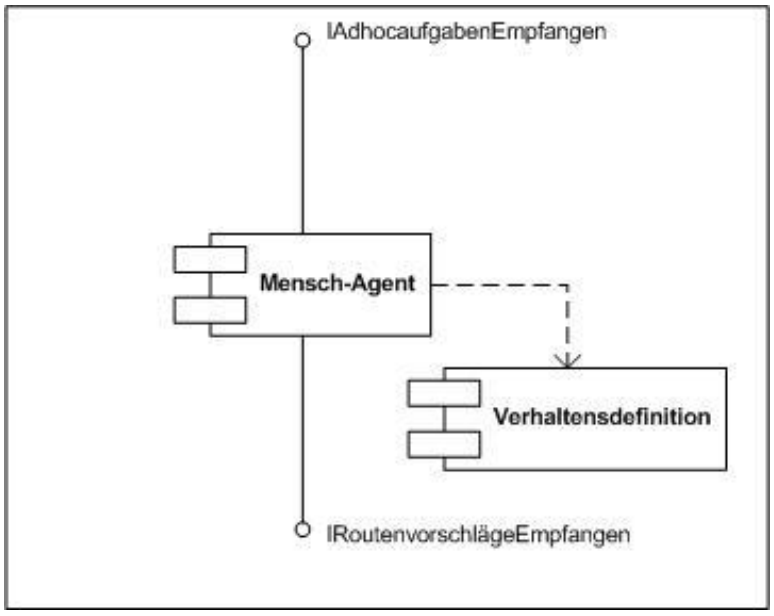

Abbildung 5-20: Komponente Fluggastverhalten

Die Komponente wird mittels Mensch-Agenten und dessen Verhaltensdefinitionen abgebildet. Der Mensch-Agent ist ein auf JADE-Basis realisierter Software-Agent. Dieser bildet eine Basis für das Simulieren des gewünschten Menschenverhaltens. Neben der Standard-Agentenfunktionalität (s. Kapitel 2) bietet die Basis insbesondere zwei Schnittstellen zum Datenaustausch mit weiteren Systemkomponenten. Die Schnittstelle IRoutenvorschlägeEmpfangen ermöglicht den Empfang von Routenvorschlägen (s. Kapitel 4.2) von der Komponente „Indoor-Fußgängernavigationssystem“. Die Schnittstelle IAdhocaufgabenEmpfangen ermöglicht den Empfang von Adhoc-Aufgaben (s. Kapitel 4.2) von der Komponente „Simulationsmanager“.

Die Reaktionen eines Mensch-Agenten auf empfangene Routenvorschläge und Adhoc-Aufgaben sowie die Ausführung weiterer Aktivitäten (z.B. Dienstkonsumierung) im abgebildeten Flughafenkontext bilden die Schwerpunkte der Verhaltensmodellierung. Diese Schwerpunkte werden durch die Verhaltensdefinitionen abgebildet, indem jede Verhaltensdefinition durch Befehlsreihenfolgen die genauen Aktivitäten eines Mensch-Agenten auf Grund der Kontextinformationen festlegt. Dabei werden nicht nur die Daten der Schnittstellen IRoutenvorschlägeEmpfangen bzw. IAdhocaufgabenEmpfangen betrachtet, sondern auch die

Informationen des Gebäudedienstkontextes (Gebäudestruktur, begehbare Wege im Gebäude, aktuelles Dienstangebot usw.) ausgewertet.

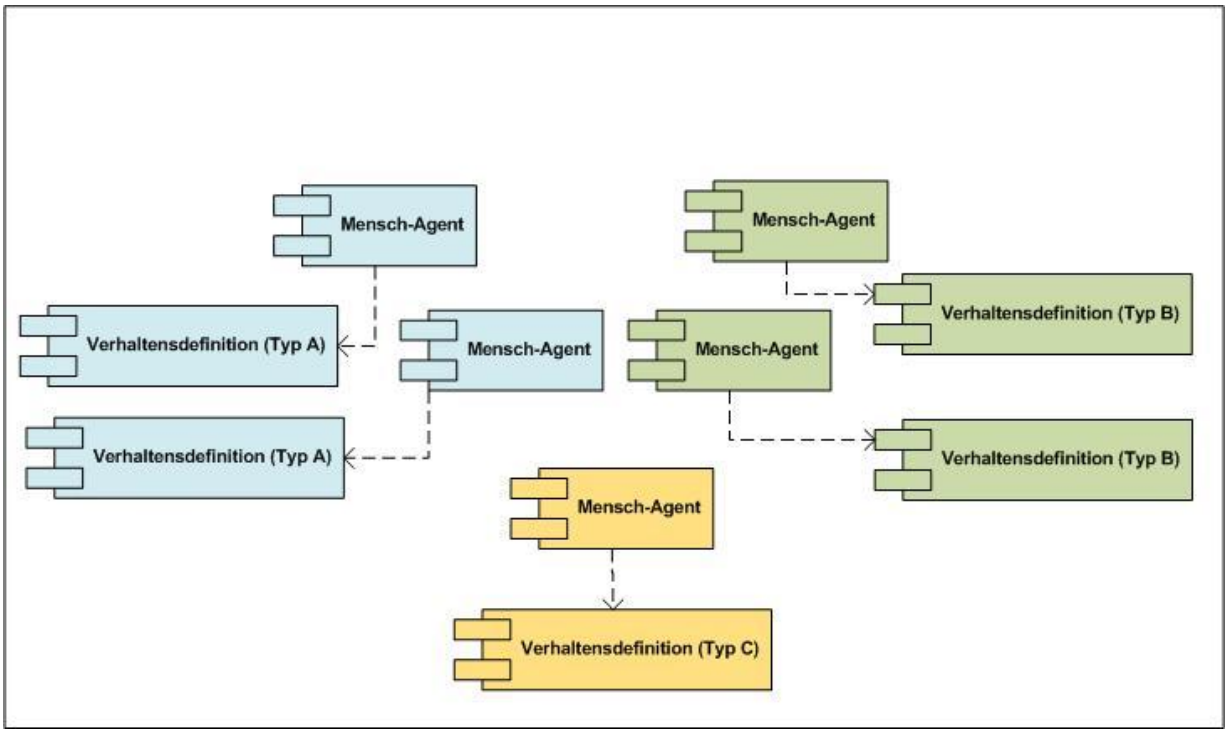

Abbildung 5-21: Komponente Fluggastverhalten – Abbildung mehrerer Verhaltensarten

Es ist denkbar, dass mehrere Verhaltensarten abgebildet werden müssen. Dies gilt etwa für Menschen, die immer genau die Anweisungen des Indoor-Fußgängernavigationssystem befolgen, oder für Menschen, die mit einer gewissen Wahrscheinlichkeit von der vorgeschlagenen Route abweichen und dadurch das Navigationssystem zu einer neuen Routenplanung zwingen. Es ist weiterhin denkbar, dass unterschiedliche Verhaltensarten nicht getrennt voneinander sondern gleichzeitig im System existieren. Ein solches Szenario kann von der Komponente abgebildet werden, indem für jede Verhaltensart eine spezifische Verhaltensdefinition vorgesehn wird. Damit die gleichzeitige Existenz mehrerer Verhaltensarten bzw. Verhaltens-Typen im System abgebildet werden kann, müssen mehrere Instanzen der Komponente mit entsprechenden Verhaltensdefinitionen eingesetzt werden (s. Abbildung 5-21). Sollte sich das Verhalten eines Menschen (z.B. vom Typ A zum Typ B) im Verlauf der Zeit ändern müssen, kann das durch die Komponente abgebildet werden, indem die aktiven Verhaltensdefinitionen eines Mensch-Agenten durch weitere dynamisch ersetzt werden.

5.2.2 Gebäudedienstkontext

Die Komponente Gebäudedienstkontext (s. Abbildung 5-22) ist für die Modellierung der Flughafengebäudestruktur und des damit verbundenen Nutzungsplans bzw. des Dienstangebots sowie die Modellierung möglicher Wege durch die Flughafengebäudestruktur verantwortlich.

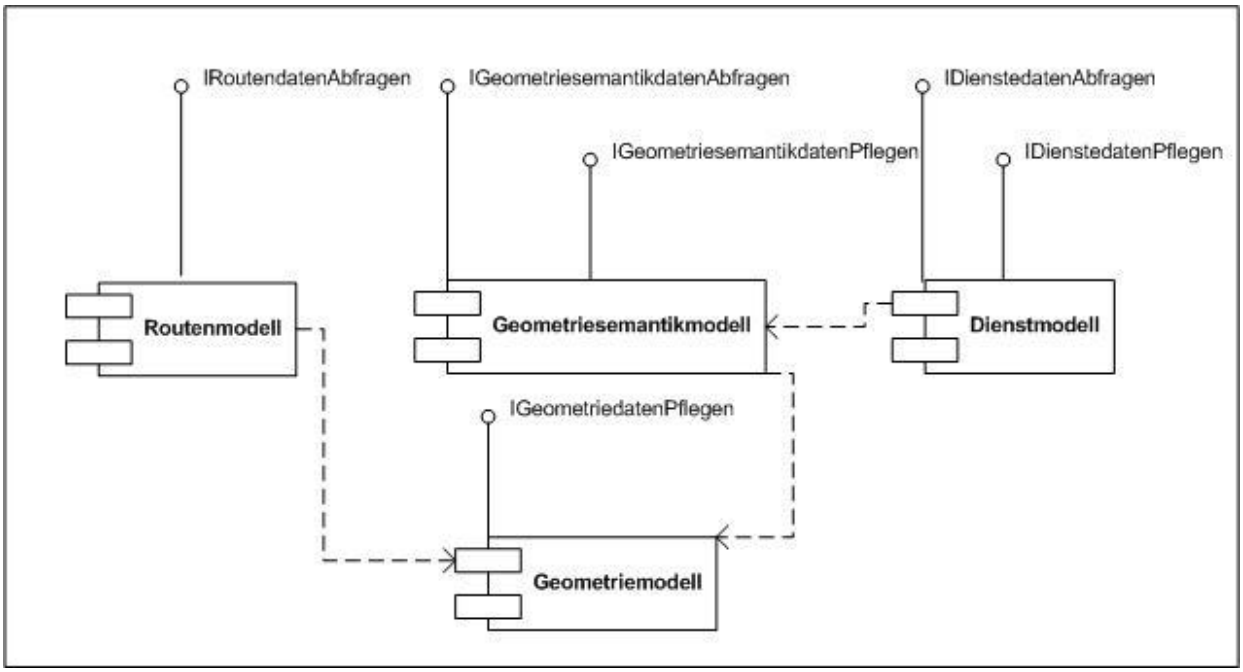

Abbildung 5-22: Komponente Gebäudedienstkontext

Um den geforderten Kontext modellieren zu können, enthält die Komponente folgende Unterkomponenten: Geometriemodell, Geometriesemantikmodell, Dienstmodell, Routenmodell.

Die Aufgabe des Geometriemodells zusammen mit dem Geometriesemantikmodell ist es, ein vereinfachtes Gebäudeinformationssystem nach dem von [Jurgeit03] vorgeschlagenem Beispiel zu realisieren. Das so entstandene System verwaltet die Gebäudestrukturdaten des simulierten Flughafens. Das Geometriemodell bildet die rein geometrischen Daten der Gebäudestruktur ab, indem es die Räumlichkeiten des Flughafens durch Polygone beschreibt. Für die Konzeption und Umsetzung im Rahmen dieser Arbeit werden nur Polygone bzw. Punkte in einem zweidimensionalen Koordinatensystem betrachtet. Über das Geometriesemantikmodell wird jeder geometrischen Fläche die entsprechende Bedeutung im Kontext eines Gebäudes zugewiesen, z.B. ob es sich um eine Ebene, um einen Raum oder um einen Durchgang handelt. Damit eine grafische Darstellung der Flughafengebäudestruktur möglich ist, stellt das Geometriesemantikmodell die Schnittstelle IGeometriesemantikdatenAbfragen zur Weitergabe von Geometriesemantikdaten an die

Komponente „Simulationsmanager“ zur Verfügung. Für das Ändern bzw. Erzeugen von Geometrie und Geometriesemantik-Daten (durch die Komponente „Simulationsmanager“) sind die Schnittstellen IGeometriedatenPflegen und IGeometriesemantikdatenPflegen vorgesehen.

Das Dienstmodell bildet das Dienstangebot eines Flughafens ab. Dienste werden durch ihre eindeutige ID, Name, Beschreibung, Kategorisierung und mittlere Nutzungsdauer beschrieben. Dabei unterscheidet das Dienstmodell zwei Arten von Diensten – physische und virtuelle. Die physischen Dienste (z.B. Raucherecke) werden mittels des Geometriesemantikmodells Räumen zugewiesen und haben somit eine definierte räumliche Ausdehnung. Das bedeutet, dass ein physischer Dienst nur innerhalb des entsprechenden Raumes genutzt werden kann (z.B. der Dienst „Raucherecke“ im Raucherraum). Virtuelle Dienste (z.B. WLAN-Zugang) haben keine räumliche Ausdehnung und können überall (soweit diese verfügbar oder bekannt sind) konsumiert werden. Ein weiterer Aspekt der physischen und der virtuellen Dienste ist deren Sichtbarkeit bzw. deren Annotation. Die Sichtbarkeit der Dienste kann mittels Geometriesemantikmodells mit Räumlichkeiten verbunden werden. Beispielsweise können Dienste auf dem gesamten Flughafengelände oder nur in bestimmten Werbezonen (Räumen) sichtbar sein (ortsbezogen auftreten). Außerdem kann die Sichtbarkeit durch Zeitereignisse gesteuert werden. Damit die grafische Darstellung der Flughafendienste und die Weiterverarbeitung der Daten möglich sind, stellt das Dienstmodell die Schnittstelle IDienstedatenAbfragen (für Weitergabe von Dienstdaten an die Komponente „Simulationsmanager“) zur Verfügung. Für das Ändern bzw. Erzeugen von Dienstdaten (durch die Komponente „Simulationsmanager“) ist die Schnittstelle IDienstdatenPflegen vorgesehen.

Die Komponente Routenmodell verwaltet alle möglichen Wege durch die Flughafengebäudestruktur in Form eines Gebäudegraphs (s. Kapitel 3.1). Die Knoten des Graphs sind die Orte, die von einem Fluggast besucht werden können. Die Kanten des Graphs sind die Strecken, die ein Fluggast innerhalb der Gebäudestruktur zurücklegen kann. Die Grundlage zur Erstellung des Gebäudegraphs ist das Geometriemodell. Die Verfahren zur Graphengenerierung auf Grund von Gebäudegeometriedaten wurden im Kapitel 3.1.1 beschrieben. Welches dieser Verfahren am besten für das Szenario dieser Arbeit geeignet ist, muss in der Realisierungsphase entschieden werden. Neben den Daten, die aus dem Geometriemodell zu gewinnen sind, wird im Gebäudegraph für jede Kante ein Transportmittel verwaltet. Dieses beschreibt die Art und Weise, mit der Strecke von einem Fluggast zurückgelegt werden kann (z.B. zu Fuß, mittels einer Rolltreppe oder eines Caddies). Insbesondere definiert das Transportmittel die Geschwindigkeit auf der Strecke, die von einem Fluggast erreicht werden kann. Dadurch kann die Anforderung nach einer realistischen Abbildung des Zeitaspektes (s. Kapitel 4.3.2) berücksichtigt werden, indem für jede Strecke eine entsprechende Geschwindigkeit gepflegt wird.

Damit eine grafische Darstellung der Routen und die Weiterverarbeitung der Daten möglich sind, stellt das Routenmodell die Schnittstelle IRoutendatenAbfragen (für Weitergabe von Routendaten an die Komponente „Simulationsmanager“) zur Verfügung.

5.2.3 Protokollierung

Die Komponente Protokollierung (s. Abbildung 5-23) ist für das Dokumentieren der Simulationsprozesse zuständig.

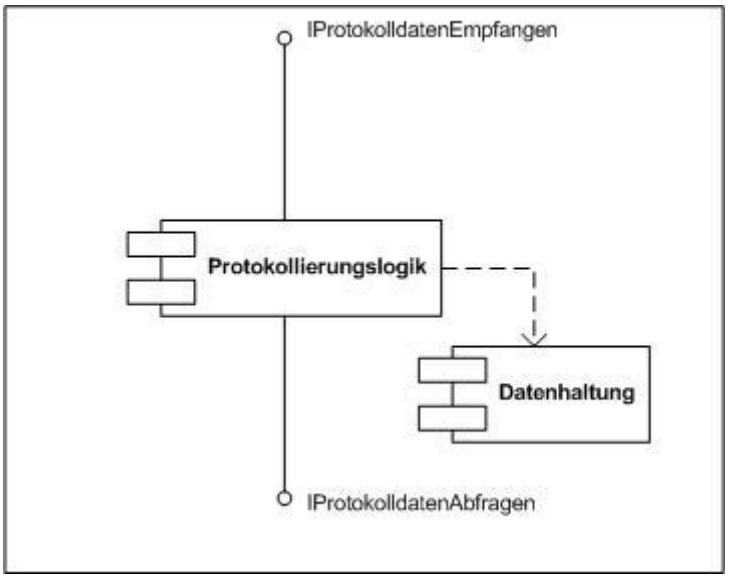

Abbildung 5-23: Komponente Protokollierung

Die Komponente besteht aus Protokollierungslogik und Datenhaltung. Die Protokollierungslogik kapselt spezifische Funktionalität, die bei dem Dokumentieren der Prozesse benötigt wird. Die Datenhaltung speichert die entsprechenden Prozessdaten dauerhaft. Damit der Empfang von relevanten Prozessdaten (z.B. Fluggastpositionsänderung, Aktivieren eines Dienstes, Versenden eines Routenvorschlags usw.) von den Systemkomponenten Fluggastverhalten, Gebäudedienstkontext, Simulationsmanager und Indoor-Fußgängernavigationssystem möglich ist, wird die Schnittstelle IProtokolldatenEmpfangen zur Verfügung gestellt. Für das Abfragen bzw. Visualisieren von Protokolldaten durch die Komponente „Simulationsmanager“ ist die Schnittstelle IProtokolldatenAbfragen vorgesehen.

5.2.4 Simulationsmanager

Die Komponente Simulationsmanager (s. Abbildung 5-24) ist für das Zusammenführen der einzelnen Funktionalitäten der Simulationsumgebung verantwortlich. Der Simulationsmanager besteht aus fünf Unterkomponenten: Visualisierung, Steuerung, Ereignisgenerierung, Umgebungsmodell und Umgebungsproxy.

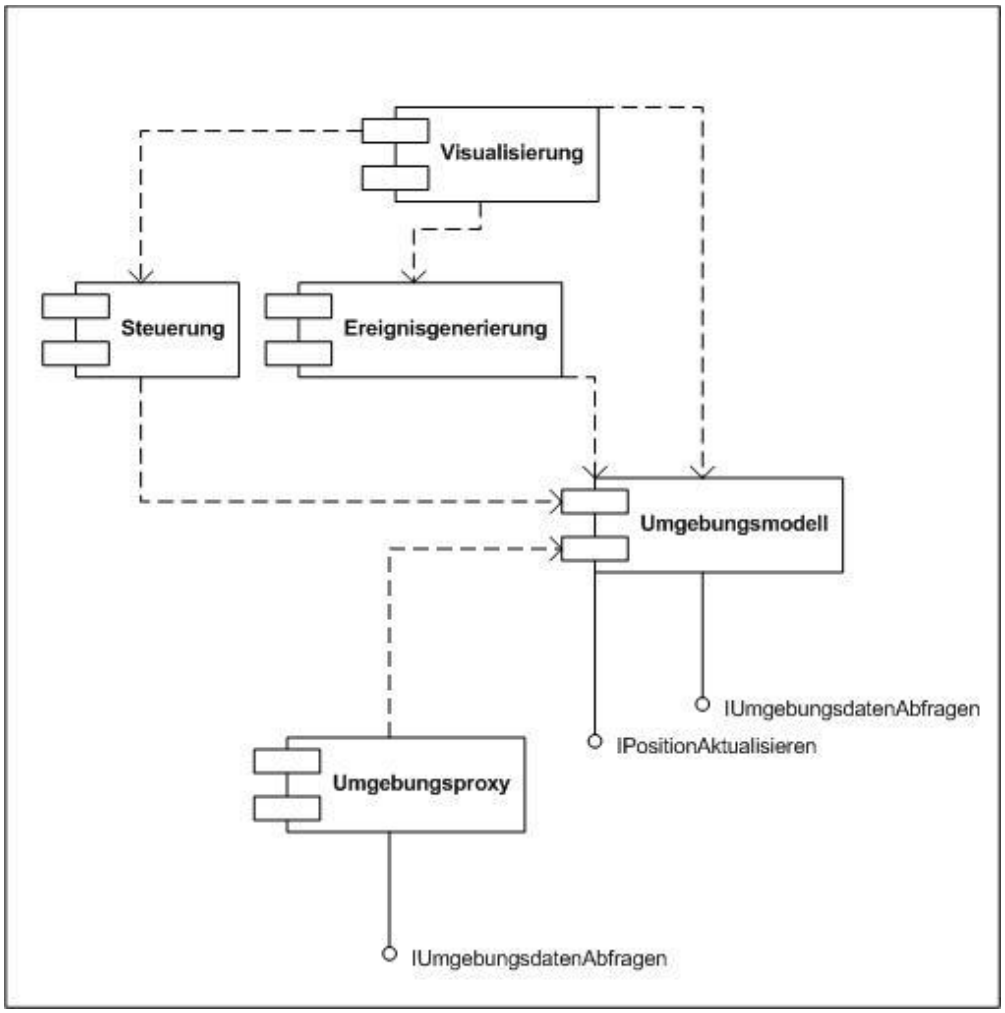

Abbildung 5-24: Komponente Simulationsmanager

Die Komponente Visualisierung bietet dem Systembetreiber der Simulationsumgebung ein geeignetes GUI (Graphic User Interface), um eine grafische Darstellung der Simulationsprozesse sowie die Interaktion zwischen dem Systembetreiber und der Simulationsumgebung zu ermöglichen.

Die Komponente Steuerung bietet entsprechende Funktionalität, um die Simulationsprozesse der Umgebung zu steuern bzw. Daten der simulierten Umgebung zu ändern.

Die Komponente Ereignisgenerierung bietet entsprechende Funktionalität, um Ereignisse in der Simulationsumgebung auszulösen.

Die Komponente Umgebungsmodell bildet den aktuellen Zustand des simulierten Flughafens ab, indem die Daten der Komponenten Gebäudedienstkontext und Fluggastverhalten zusammenführt werden. Zusätzlich bietet die Komponente die Schnittstelle IPositionAktualisieren für das Abbilden der aktuellen Fluggastposition im Kontext des Flughafens an (durch die Komponente „Fluggastverhalten“). Über die Schnittstelle IUmgebungsdatenAbfragen werden ortsbezogene Umgebungsdaten gleichfalls durch die Komponente „Fluggastverhalten“ abgefragt.

Die Komponente Umgebungsproxy ist für die kontrollierte Weitergabe der Umgebungsmodelldaten und Funktionalität an das externe Indoor-Fußgängernavigationssystem verantwortlich. Das Umgebungsproxy verbirgt die Komplexität des Umgebungsmodells und veröffentlicht über die Schnittstelle IUmgebungsdatenAbfragen nur die Funktionalität des Modells, die vom Indoor-Fußgängernavigationssystem benötigt wird.

5.2.5 Indoor-Fußgängernavigationssystem

Das Fußgängernavigationssystem ist eine externe Komponente, die mit Hilfe der Simulationsumgebung getestet werden kann. Dies erfordert eine intensive Interaktion zwischen Simulationsumgebung und Fußgängernavigationssystem. Um eine solche Interaktion optimal zu gestalten, werden die Komponenten Navigationssystemadapter und Monitor (s. Abbildung 5-25) benötigt.

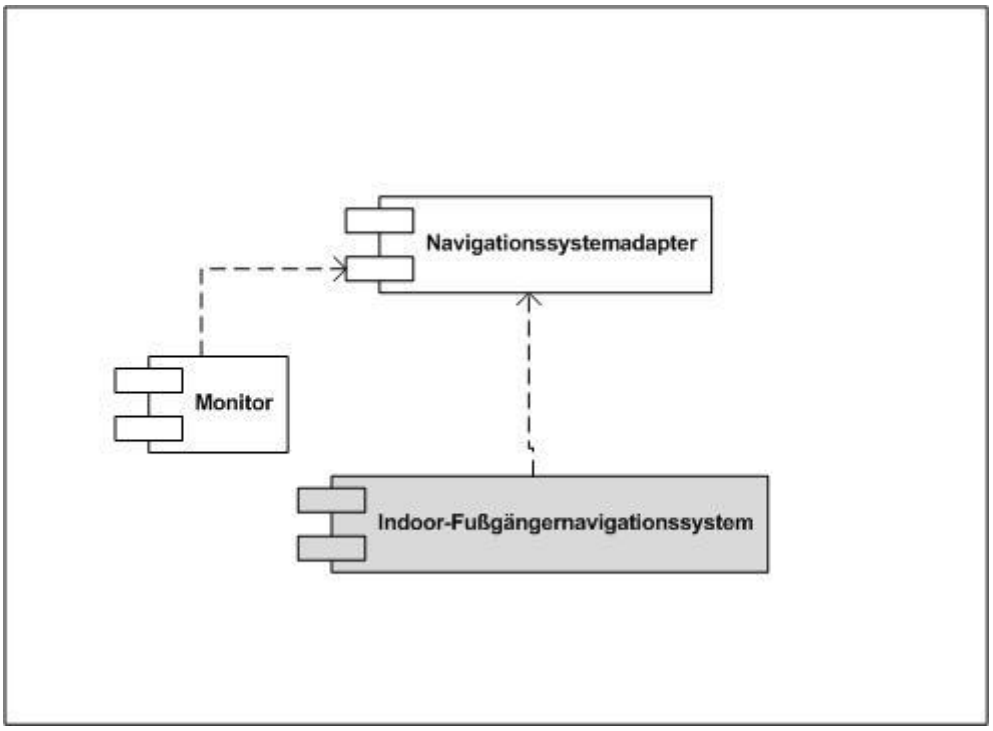

Abbildung 5-25: Komponente Indoor-Fußgängernavigationssystem

Der Navigationssystemadapter ist für die Anbindung des Indoor-Fußgängernavigationssystems an die Simulationsumgebung verantwortlich. Das Navigationssystem fordert vom Simulationsmanager Daten des Umgebungsmodells und sendet Routenvorschläge an den Mensch-Agenten über den Navigationssystemadapter. Der Navigationssystemadapter wird auf Basis des Adapter-Design-Musters (s. [Gamma96]) realisiert. Dabei werden die Schnittstellen der Simulationsumgebung an die vom Fußgängernavigationssystem erwarteten Schnittstellen adaptiert. Somit werden die Schnittstellen der beiden Komponenten voneinander entkoppelt. Durch diese Entkopplung kann die Simulationsumgebung mit unterschiedlichen Fußgängernavigationssystemen interagieren, indem für jedes System ein passender Adapter entwickelt wird.

Der Monitor ist für die Überwachung der Interaktionen des Fußgängernavigationssystems mit der Simulationsumgebung verantwortlich. Damit eine genaue Analyse der Simulationsprozesse durchgeführt werden kann, verfolgt der Monitor, wie die Routenvorschläge des Navigationssystems von dem Mensch-Agenten befolgt werden. Dabei werden aufgetretene Routenabweichungen dokumentiert und dem Indoor-Fußgängernavigationssystem gemeldet.

5.2.6 Komponentenzusammenspiel

Im Folgenden wird das Zusammenspiel einzelner Komponenten der künftigen Simulationsumgebung dargestellt. Dabei werden zwei grundlegende Prozesse dieser Simulationsumgebung vorgestellt: Fluggastnavigation und Fluggastbewegung. Der Prozess Fluggastnavigation beschreibt die Aktivitäten, die ausgeführt werden, um dem virtuellen Fluggast eine Route durch die abgebildete Umgebung vorschlagen zu können. Der Prozess Fluggastbewegung beschriebt die Aktivitäten, die ausgeführt werden, um eine Fluggastbewegung in der virtuellen Umgebung entlang der vorgeschlagenen Route simulieren zu können. Als Grundlage bei der Prozessbeschreibung dient die detaillierte fachliche Architektur (Abbildung 5-26). Diese stellt die bisher beschriebenen Systemkomponenten im gemeinsamen Kontext dar (auf die Abbildung der Komponente Protokollierung wurde wegen ihrer Standardfunktionalität verzichtet) und ermöglicht damit einen besseren Systemüberblick.

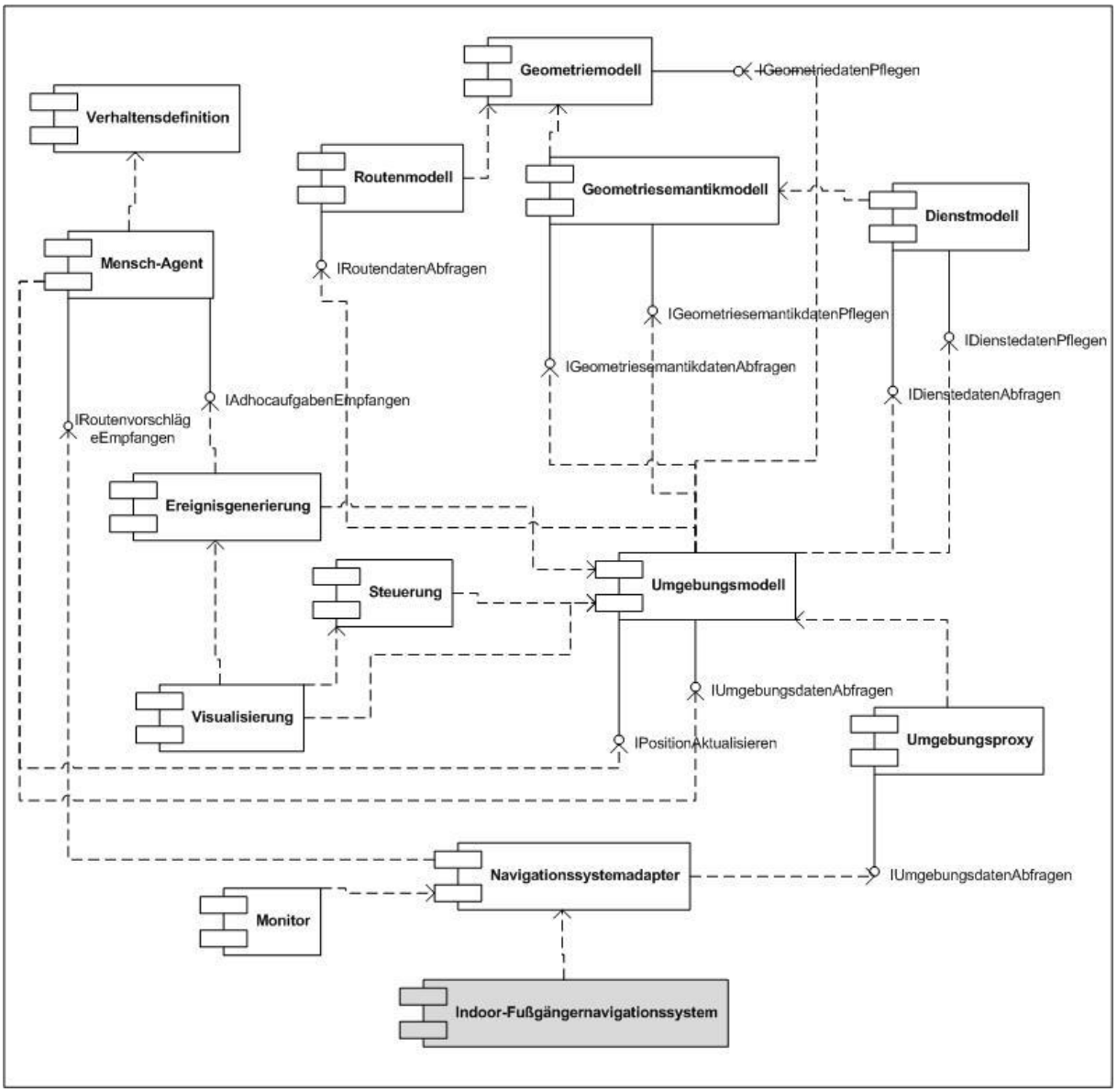

Abbildung 5-26: Detaillierte fachliche Architektur

5.2.6.1 Fluggastnavigation

Damit das Indoor-Fußgängernavigationssystem einen virtuellen Fluggast mit Berücksichtigung seiner Präferenzen durch die abgebildete Flughafenumgebung navigieren kann, benötigt das System die Umgebungsdaten (z.B. Gebäudegeometrie, vorhandene Dienste, Menge aller möglichen Wege) sowie die Fluggastdaten (z.B: Fluggast-Profil bzw. -Zeitlimit). Das Navigationssystem, vertreten durch den Navigationssystemadapter, muss die oben genannten Daten bei der Softwaresimulationsumgebung abfragen.

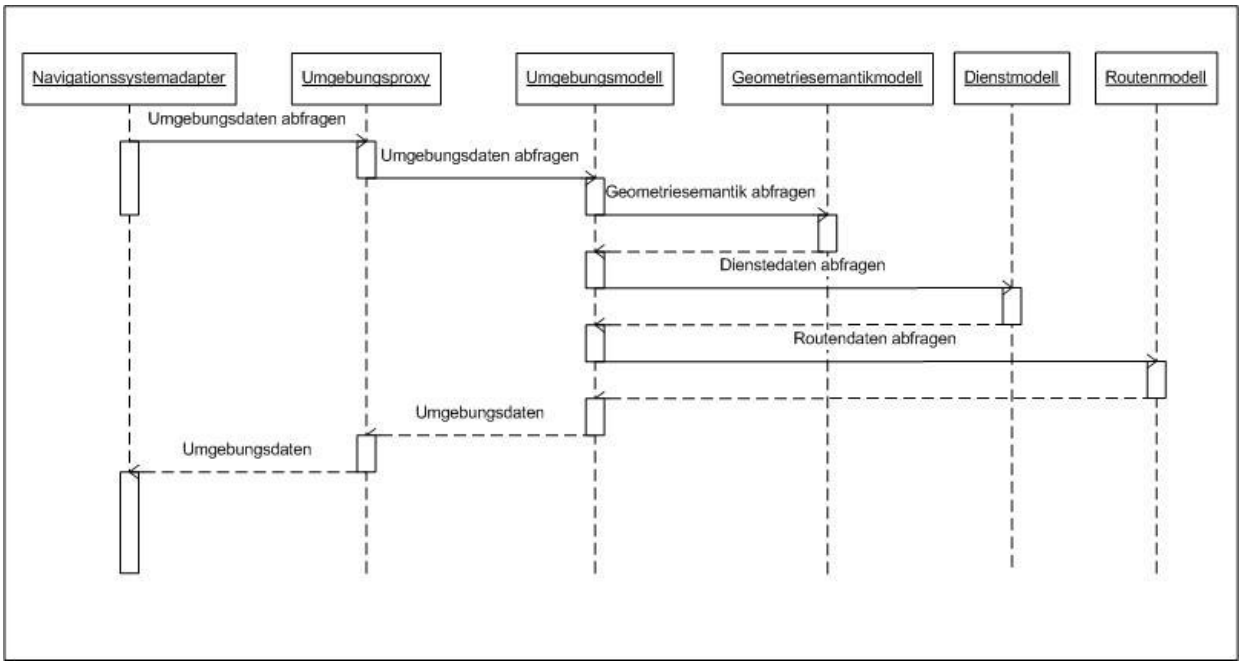

Abbildung 5-27: Umgebungsdaten abfragen

Die Abbildung 5-27 stellt den zeitlichen Ablauf einer Umgebungsdatenabfrage dar. Das Navigationssystem startet die Abfrage durch den Navigationssystemadapter. Dieser leitet die Abfrage durch die Umgebungsproxy und das Umgebungsmodell an die zuständigen Systemkomponenten: das Geometriesemantikmodell (verantwortlich für das Abbilden von Gebäudegeometrie und Gebäudesemantik), das Dienstmodell (verantwortlich für das Abbilden des aktuellen Dienstangebotes) und das Routenmodell (verantwortlich für das Abbilden der möglichen Wege durch die Gebäudestruktur). Nachdem die Abfrage von der Simulationsumgebung bearbeitet worden ist, werden die Umgebungsdaten an den Navigationsadapter weitergeleitet. Dieser übergibt die Daten an das Navigationssystem.

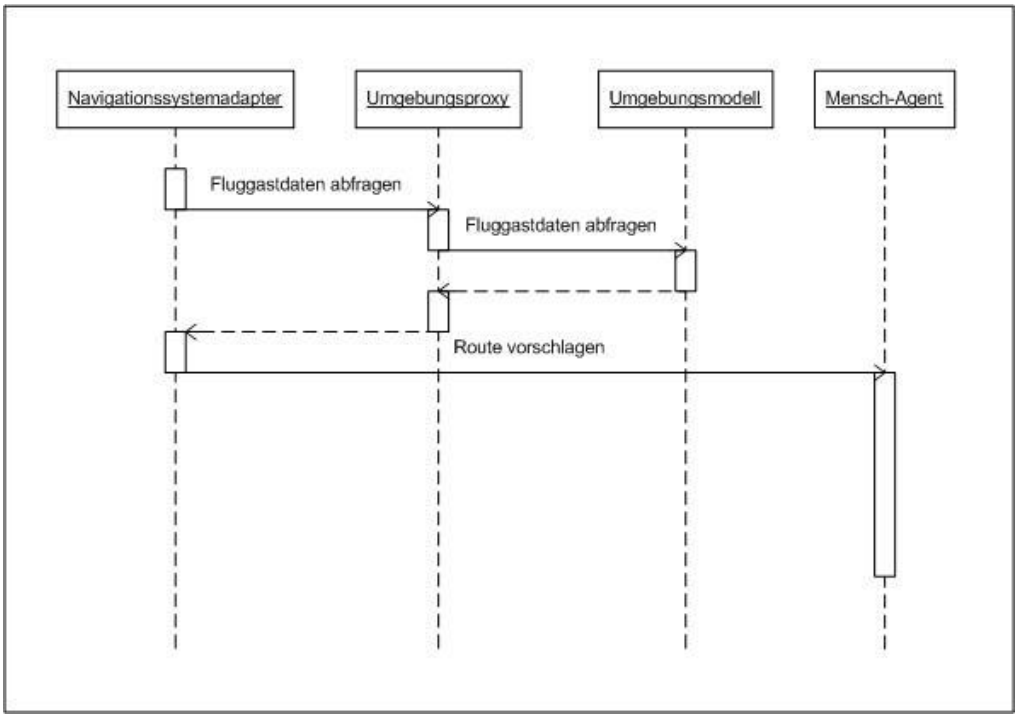

Abbildung 5-28: Navigationsanweisungen erteilen

Nachdem das Navigationssystem über Umgebungsdaten verfügt, muss es zusätzlich Fluggastdaten abfragen, um eine sinnvolle Navigation durchführen zu können. Die Abbildung 5-28 stellt diesen Prozess dar. Das Navigationssystem startet die Abfrage durch den Navigationssystemadapter. Dieser leitet die Abfrage durch die Umgebungsproxy an das Umgebungsmodell (verantwortlich für das Abbilden von Fluggastprofil und Fluggastzeitlimit). Nachdem die Abfrage von der Simulationsumgebung bearbeitet worden ist, werden die Umgebungsdaten an den Navigationsadapter weitergeleitet. Dieser übergibt die Daten an das Navigationssystem. Danach startet das Navigationssystem die Fluggastführung, indem eine Route direkt dem Mensch-Agenten vorgeschlagen wird. Die Route enthält eine Reihenfolge von Punkten (Knoten des GebäudeGraphs), die den Fluggast zu seinem Ziel führen werden. Nach der Unterbreitung des Routenvorschlages überwacht das Navigationssystem, wie genau dieser vom Mensch-Agenten befolgt werden.

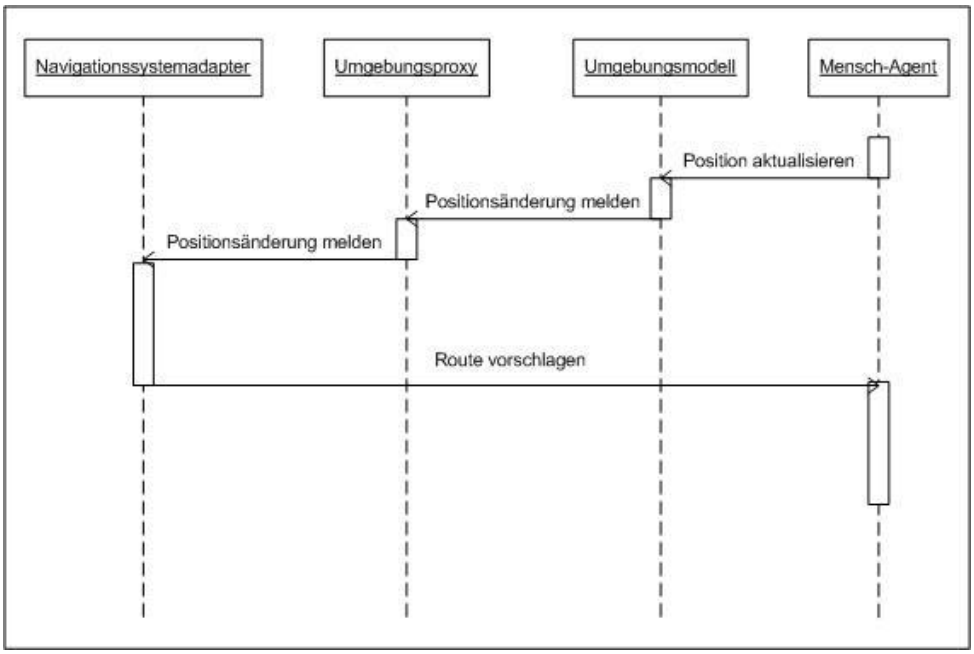

Abbildung 5-29: Position aktualisieren

Damit das Navigationssystem überwachen kann, inwieweit ein Mensch-Agent die vorgeschlagene Route befolgt, benötigt das System die aktuellen Positionsdaten des Agenten. Die Abbildung 5-29 stellt die Übermittlung von Positionsdaten eines Mensch-Agenten dar. Sobald sich die Position eines Mensch-Agenten innerhalb der virtuellen Umgebung ändert, wird diese Positionsänderung vom Agenten an das Umgebungsmodell gemeldet. Das Umgebungsmodell übermittelt die Positionsänderung an das Navigationssystem durch die Umgebungsproxy und den Navigationssystemadapter. Bei Bedarf (z.B. bei Abweichungen) kann das Navigationssystem nach Positionsänderungen eine neue Route vorschlagen.

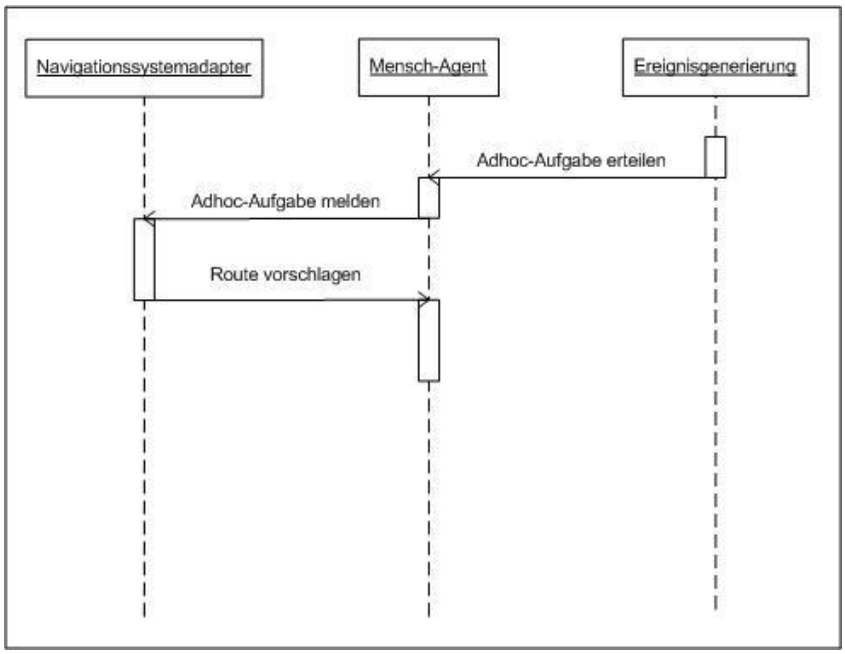

Abbildung 5-30: Adhoc-Aufgaben melden

Während einer Fluggastführung können einem Mensch-Agenten Adhoc-Aufgaben erteilt werden. Die Abbildung 5-30 stellt diesen Prozess dar. Die Ereignisgenerierung erteilt eine Adhoc-Aufgabe direkt dem Mensch-Agenten. Dieser leitet die erteilte Adhoc-Aufgabe durch den Navigationssystemadapter an das Navigationssystem weiter. Somit kann das Navigationssystem die Adhoc-Aufgabe entsprechend des Fluggastzeitlimits berücksichtigen und bei Bedarf die aktuelle Route ändern und eine neue Route vorschlagen.

5.2.6.2 Fluggastbewegung

Das Indoor-Fußgängernavigationssystem schlägt dem virtuellen Fluggast eine Route durch die abgebildete Umgebung vor und überwacht, in wieweit der Fluggast den Routenvorschlag befolgt. Der virtuelle Fluggast, abgebildet durch den Mensch-Agenten und seine Verhaltensdefinitionen, nimmt den Routenvorschlag an und beginnt mit der Bewegung entlang der vorgeschlagenen Route.

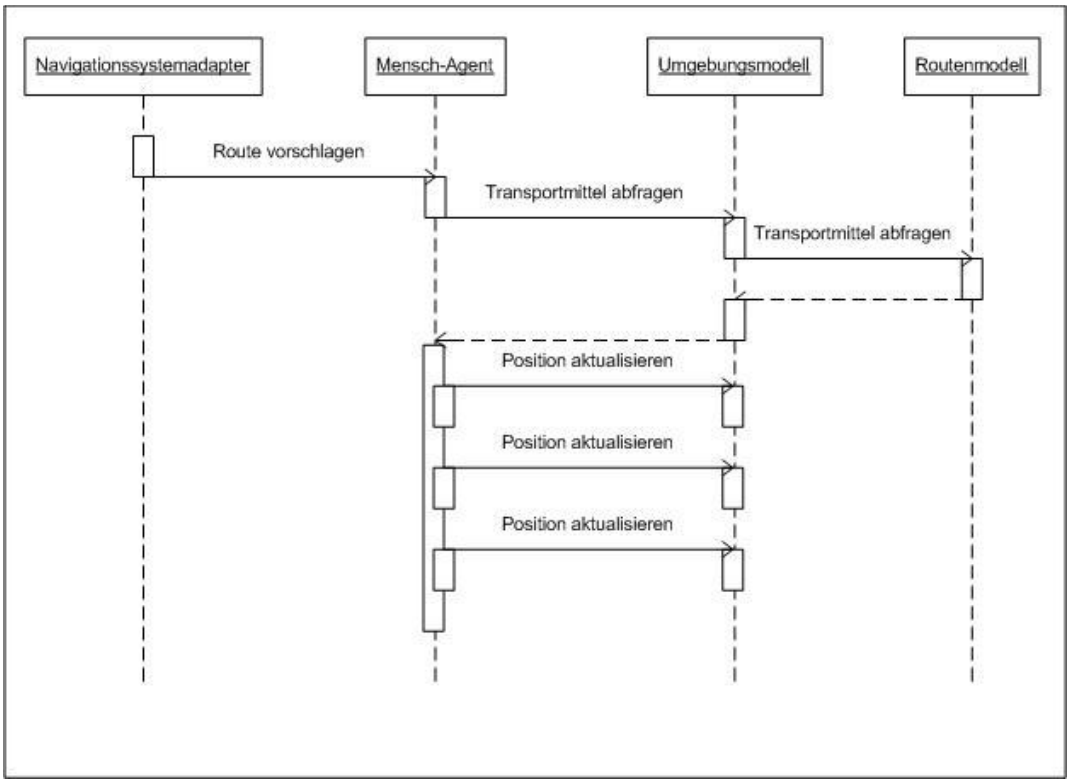

Abbildung 5-31: Bewegung entlang einer Route

Die Abbildung 5-31 stellt den zeitlichen Ablauf der Fluggastbewegung dar. Der Routenvorschlag enthält eine Reihenfolge von Punkten, die der Mensch-Agent aufsuchen soll. Die Punkte sind Knoten des Gebäudegraphs der abgebildeten Flughafenumgebung. Die Reihenfolge ist so vom Navigationssystem gewählt, dass die Punkte durch Kanten im Gebäudegraph verbunden sind und dass eine Bewegung entlang der so entstandenen Route möglich ist. Um die Strecke zwischen zwei Punkten zurücklegen zu können, benötigt der Mensch-Agent die Information darüber, wie er sich auf dieser Strecke bewegen kann (z.B. zu Fuß, mittels einer Rolltreppe oder eines Caddies). Diese Information erhält er durch die Abfrage des Transportmittels einer Strecke. Die Abfrage wird beim Umgebungsmodell gestartet und vom Routenmodell beantwortet. Nachdem die Abfrage abgearbeitet ist, startet der Mensch-Agent die simulierte Fluggastbewegung, indem er die Fluggastposition in angemessener Geschwindigkeit (beschrieben durch das Transportmittel) entlang

der aktuellen Strecke ändert. Dieser Vorgang wird wiederholt, bis der Mensch-Agent den nächsten Punkt erreicht hat. In jedem Punkt kann der Mensch-Agent von der vorgeschlagenen Route abweichen, indem er andere Strecken, die nicht von der Route vorgegeben sind, für seine nächste Bewegung auswählt. Entsprechende Streckeninformationen erhält der Mensch-Agent vom Routenmodell.

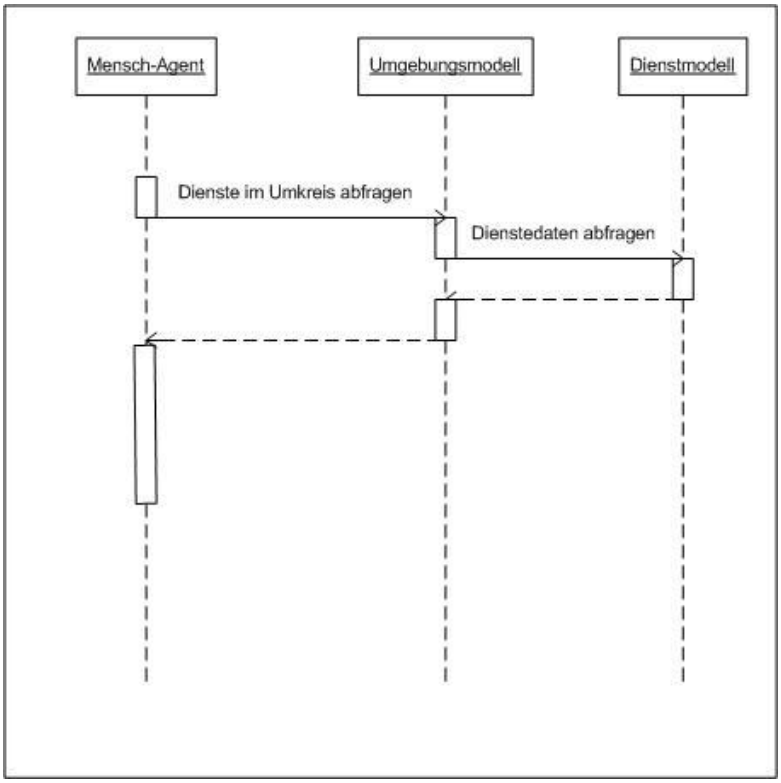

Abbildung 5-32: Konsumieren von Diensten

Das Indoor-Fußgängernavigationssystem hat bei der Erstellung des Routenvorschlags die Präferenzen des virtuellen Fluggastes berücksichtigt und interessante Dienste in die Route zum Ziel mit aufgenommen. Neben der Bewegungssimulation bildet der Mensch-Agent das Konsumieren von Diensten ab. Beim Erreichen eines Routenpunktes fragt der Mensch-Agent beim Umgebungsmodell die Daten aller Dienste, die für ihn erreichbar sind. Die Abfrage wird vom Dienstmodell beantwortet. Der Mensch-Agent simuliert das Konsumieren von erreichbaren Diensten, indem er die für jeden Dienst angemessene Zeit (beschrieben durch Eigenschaften eines Dienstes) verbraucht. Nachdem alle Dienste konsumiert worden sind, kann der Mensch-Agent seine Bewegung fortsetzen.

Bei der hier beschriebenen Fluggastbewegung können zwei Arten von Abweichungen auftreten: räumliche und zeitliche. Die räumliche Abweichung kann dadurch entstehen, dass der Mensch-Agent sich auf anderen Strecken als auf der vorgeschlagenen Route bewegt. Eine zeitliche Abweichung kann

dadurch entstehen, dass der Mensch-Agent bei seiner Bewegung von der Streckengeschwindigkeit abweicht oder Dienste nicht in der vorgeschriebenen Zeit konsumiert. Die Handlungen der Mensch-Agenten werden vom Navigationssystem durch seine aktuelle Position überwacht. Dabei entscheidet das Navigationssystem, ob die aktuell vorgeschlagene Route noch rechtzeitig zum Ziel führen kann oder nicht. Bei Bedarf kann das Navigationssystem eine neue (z.B. verkürzte) Route vorschlagen.

5.3 Fazit

In diesem Kapitel wurde gemäß den definierten Anforderungen ein abstrakter System-Entwurf der zukünftigen Software-Simulationsumgebung konzipiert. Die grundlegende Entscheidung der Anforderungsanalyse (s. Kapitel 4.3) zum Einsatz von Software-Agenten fand Berücksichtigung damit, dass die Systemvision im Rahmen des Software-Agentenparadigmas entwickelt wurde. Die Entwicklungsplattformen SeSAm, JADE und JADEX wurden auf ihre Eignung als Entwicklungsgrundlage der gewünschten Agenten-Funktionalität analysier und bewertet. SeSAm erwies sich dazu wegen ihrer schlechten Erweiterbarkeit und fehlenden Schnittstellen zu externen Systemen als ungeeignet. Die JADEX-Plattform kann mit ihrem Schwerpunkt auf BDI-Agentenarchitekturen die gestellten Anforderungen zum menschlichen (Fluggast-)Verhalten nur unzureichend erfüllen, was zur Entscheidung gegen JADEX führte. Die Analyse der JADE-Plattform für Software-Agenten kam wegen deren Flexibilität bzw. Erweiterbarkeit und standardisierten Nachrichtenaustausches zum Entschied für diese Software.

Im Folgeschritt wurde eine fachliche Systemarchitektur unter Berücksichtigung der Modellierungseigenschaften von JADE bei der Abbildung des Fluggastverhaltens durch Software-Agenten aufgestellt. Auf Basis dieser Architektur und der ausgewählten JADE-Entwicklungsplattform für Software-Agenten wird im Folgekapitel eine konkrete Realisierung der künftigen Software-Simulationsumgebung vorgestellt.

6. Realisierung

In diesem Kapitel wird eine konkrete Realisierung der Simulationsumgebung für Indoornavigation durch die Implementierung eines Prototyps vorgestellt. Der Prototyp muss alle in Kapitel 4 definierten Systemanforderungen erfüllen. Er wird auf Basis der definierten Systemarchitektur (s. Kapitel 5.2) entworfen und anhand eines Beispielszenarios erprobt. Die wichtigsten Entwurfsentscheidungen, Erfahrungen und die Bewertung der Implementierung werden in den folgenden Unterkapiteln beschrieben.

6.1 Auswahl der Schnittstellentechnologie

Das Thema Schnittstellentechnologie bzw. Schnittstelleneigenschaften wurde bereits in der Anforderungsanalyse (s. Kapitel 4.3) diskutiert, wobei Ansprüche an die Schnittstellen des zukünftigen Systems definiert wurden. Die Schnittstellen müssen eine standardisierte komponentenübergreifende Interaktion ermöglichen. In diesem Kontext kann auch die Forderung nach einer physikalischen Verteilung der Softwaresimulationsumgebung, die erst durch komponentenübergreifende standardisierte Schnittstellen möglich ist, betrachtet werden. Die aufgestellten Forderungen können durch den Einsatz mehrerer standardisierter Technologien für Interprozesskommunikation erfüllt werden. Zwei Standardtechnologien für verteilte Anwendungen - Web-Services und CORBA - könnten für dieses Szenario besonders geeignet sein. Dennoch, um eine richtige Auswahl treffen zu können, muss neben den oben beschriebenen Forderungen auch die fachliche Architektur der Softwaresimulationsumgebung betrachtet werden. Die Simulationsumgebung besteht aus einer externen und vier internen Grundkomponenten, die aufeinander durch Schnittstellen zugreifen. Dabei wird die Komponente Fluggastverhalten als Software-Agent auf Basis von JADE realisiert. Eine der Grundfunktionalitäten von JADE-Software-Agenten ist die Kommunikation untereinander mittels FIPA-standardisierten systemübergreifenden Nachrichtenaustauschs, was den Einsatz des FIPA-Nachrichtenaustauschs als Schnittstellentechnologie für die Softwaresimulationsumgebung zur Bedingung macht.

Aus Gründen der Übersichtlichkeit und optimierter Wartung des künftigen Systems ist es wünschenswert, dass nur oder zumindest überwiegend nur eine Schnittstellentechnologie für die Komponenteninteraktion eingesetzt wird. Dabei ist es zu entscheiden, welche der Möglichkeiten für eine komponentenübergreifende Kommunikation eingesetzt werden sollte:

- **Standardtechnologien für Interprozesskommunikation**: Der Einsatz einer Standardtechnologie für Interprozesskommunikation wie z.B. Web-Services oder CORBA

wird das Schnittstellendesign erwartungsgemäß vereinfachen und eine komponentenübergreifende Kommunikation erleichtern. Dafür müssen die Schnittstellen der Systemkomponenten in der jeweiligen Technologie realisiert werden. Da aber die Komponente Fluggastverhalten den FIPA-Nachrichtenaustausch automatisch implementiert, muss dieser zusätzlich in die jeweilige Standardtechnologie überführt werden. Dadurch werden zwei Schnittstellentechnologien (FIPA-Nachrichtenaustausch und die eingesetzte Standardtechnologie) in einem System nebeneinander koexistieren. Dies hätte zu Folge, dass zwei unterschiedliche Infrastrukturen vorgesehen und gepflegt werden müssen.

- **FIPA-standardisierter Nachrichtenaustausch**: Beim Einsatz des FIPA-standardisierten Nachrichtenaustauschs als komponentenübergreifende Schnittstellentechnologie wird die vorhandene JADE-Infrastruktur in Anspruch genommen. Um den FIPA-Nachrichtenaustausch implementieren zu können, müssen die Schnittstellen der Systemkomponenten durch JADE-Software-Agenten abgebildet werden. Das kann das Design und die Realisierung der vorgesehenen Schnittstellen erschweren.

Nach Abwägung der Argumente für oder gegen die Realisierung einer der beiden hier betrachteten Alternativen wird vom Autor der Einsatz des FIPA-standardisierten Nachrichtenaustauschs als Schnittstellentechnologie bevorzugt. Auf dieser Weise würde nur eine einzige Schnittstellentechnologie von allen Systemkomponenten implementiert und keine zusätzliche Infrastruktur benötigt, was die Möglichkeiten zur Wartung und die Übersichtlichkeit des Systems erleichtern wird. Der zusätzliche Aufwand bei Design und Realisierung der Systemschnittstellen wird in Kauf genommen, da bei dieser Lösung keine zusätzliche Überführung von Schnittstellentechnologien notwendig wird, zumal der Aufwand für eine solche Überführung momentan nicht abzuschätzen ist.

6.2 Technische Architektur

Eine wichtige konzeptionelle Entscheidung, die getroffen werden musste, ist die, wie die fachliche Systemarchitektur auf Basis der JADE-Laufzeitumgebung umgesetzt werden kann. Bei der Umsetzung wurden neben den Systemanforderungen (s. Kapitel 4.2) die technischen Realisierungsmöglichkeiten der JADE-Plattform (s. Kapitel 5.1.2) sowie die in Kapitel 6.1 getroffene Auswahl des FIPA-Nachrichtenaustausches als übergreifende Schnittstellentechnologie berücksichtigt. Die Abbildung 6-1 stellt das Ergebnis die technische Architektur der Umsetzung vor. Im Folgenden werden die einzelnen Schritte beschrieben, in denen die Architektur konzipiert wurde.

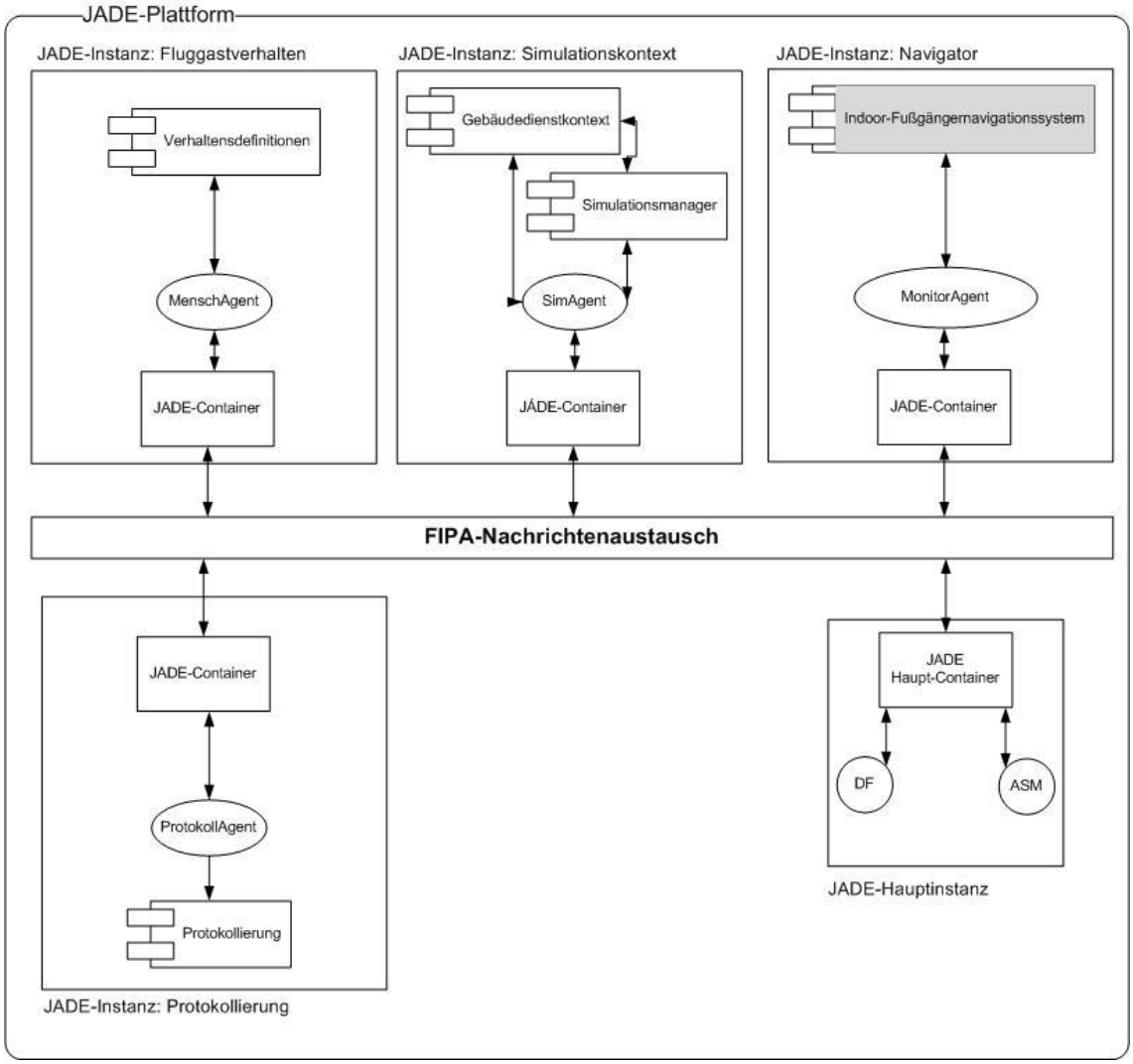

Abbildung 6-1: Technische Architektur der Simulationsumgebung

Zunächst war zu bestimmen, wie die fachliche Funktionalität auf JADE-Instanzen und JADE-Agenten verteilt werden kann. Es wurde entschieden, dass jede JADE-Instanz nur eng zusammenhängende Komponentenfunktionalität kapselt. Der kontrollierte Zugriff auf die gekapselte Funktionalität wird über einen Software-Agenten ermöglicht, der die Funktion übernimmt, indem er mittels FIPA-Nachrichten mit anderen JADE-Instanzen bzw. Agenten kommuniziert und ausgehende und eingehende Abfragen von und an die lokalen Instanz-Komponenten weiterleitet. Auf diese Weise kann die fachliche Architektur durch folgende Instanzen im JADE-Kontext abgebildet werden (s. Abbildung 6-1):

- **Fluggastverhalten:** Diese JADE-Instanz setzt das gewünschte Fluggastverhalten um. Die Instanz beinhaltet einen Menschagenten und die zugehörigen Verhaltensdefinitionen zum Abbilden des Fluggastverhaltens im Flughafenkontext. Der Agent dieser JADE-Instanz muss nicht nur die Kommunikation mit anderen JADE-Instanzen verantworten, sondern auch eine

fachliche Aufgabe erfüllen, die Simulation des Fluggastverhaltens durch Ausführung entsprechender Verhaltensdefinitionen.

- **Simulationskontext:** Diese JADE-Instanz kapselt die Fachkomponenten Gebäudedienstkontext und Simulationsmanager. Der außerdem enthaltene SimAgent hat ausschließlich die Kommunikation mit anderen JADE-Instanzen zur Aufgabe.
- **Navigator:** Diese JADE-Instanz bindet das externe Indoor-Fußgängernavigationssystem an die Simulationsumgebung an, indem der MonitorAgent die Kommunikation zwischen dem Navigationssystem und den Komponenten der Simulationsumgebung (beheimatet in anderen JADE-Instanzen) verantwortet und zusätzlich die Routenabweichungen des Menschagenten überwacht.
- **Protokollierung:** Diese JADE-Instanz kapselt die Fachkomponenten der Protokollierung. Der ProtokollAgent ermöglicht den Empfang der relevanten Prozessdaten, die protokolliert werden müssen.

Neben den vier Instanzen für die Umsetzung der fachlichen Architektur wurde eine zusätzliche Instanz (JADE-Hauptinstanz) für die Ausführung des Haupt-Containers sowie die benötigten Plattformagenten (ASM und DF) vorgesehen. Durch die so gebildeten Instanzen ermöglicht die JADE-Laufzeitumgebung eine flexible physikalische Verteilung des Systems, wobei die Anzahl der Knoten (Rechner) im System zwischen eins (alle Instanzen werden auf einem Knoten ausgeführt) und fünf (jede Instanz wird auf einem eigenen Knoten ausgeführt) variieren kann. So kann je nach Bedarf schnell und einfach eine optimale Verteilung der Simulationsumgebung konfiguriert werden.

Eine weitere Überlegung beim Design der künftigen JADE-Plattform war, die FIPA-Kommunikation und somit die Schnittstellen zwischen den einzelnen JADE-Instanzen bzw. Komponenten transparent und technologieunabhängig zu gestalten. Daher wurde entschieden, die Semantik Nachrichten zwischen Agenten durch eine anwendungsspezifische Ontologie zu beschreiben und diese bei der Komponentenkommunikation als Referenz einzugeben. Die Ontologie wurde mit Hilfe des Werkzeugs Protégé (s. [stanford.edu]) erstellt.

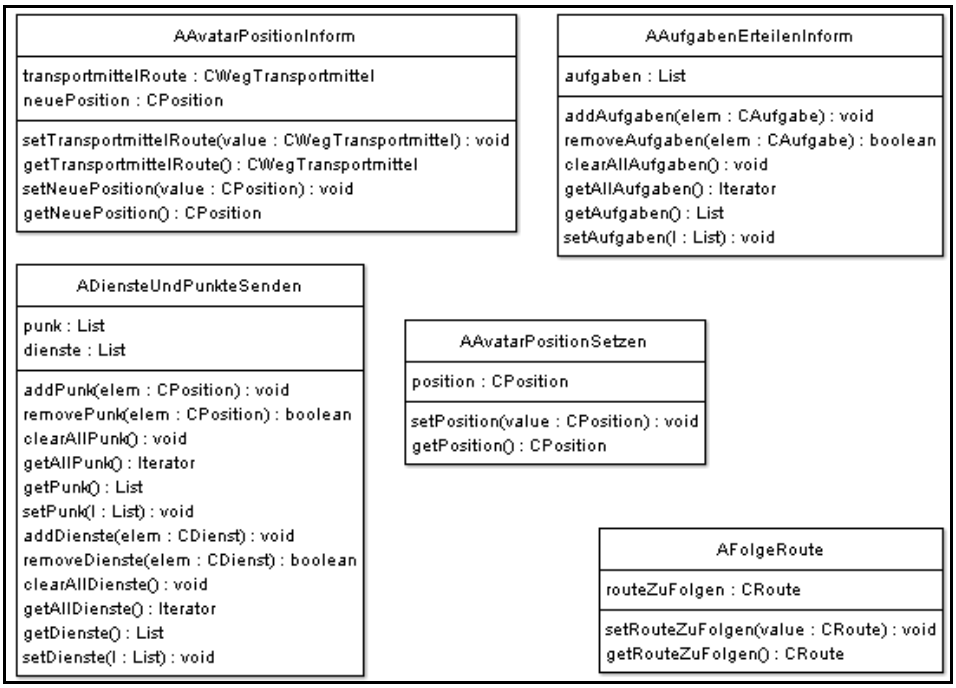

Abbildung 6-2: Ausschnitt aus der Applikationsontologie

Die Struktur der Ontologie (s. Abbildung 6-2) berücksichtigt die Tatsache, dass durch den Nachrichtenaustausch entsprechende Handlungen (Aktionen) bei den beteiligten Agenten ausgelöst werden. Daher definiert die Ontologie mehrere Aktionsklassen, die die Menge aller möglichen Anwendungsaktionen definieren. Jede Aktion kann zusätzliche Werte, die auch durch die Ontologie beschrieben sind, enthalten. Auf diese Weise können Agenten nicht nur Handlungen auslösen sondern auch komplexe Informationen austauschen.

6.3 Implementierungsdetails

Nachdem bisher eine komponentenübergreifende Schnittstellentechnologie ausgewählt und eine konkrete Umsetzung der fachlichen Architektur im JADE-Kontext beschrieben wurden, werden in diesem Kapitel die Implementierungsdetails der einzelnen Komponenten der Simulationsumgebung vorgestellt. Dabei werden die Komponenten basierend auf ihren Zusammenfassung zu JADE-Instanzen beschrieben (s. Abbildung 6-1).

6.3.1 Fluggastverhalten

Die JADE-Instanz „Fluggastverhalten“ enthält die Komponente gleichen Namens. Sie hat die Aufgabe, das typische Verhalten eines Fluggastes im Flughafenkontext und seine Reaktionen auf Routenvorschläge des Indoor-Fußgängernavigationssystems zu simulieren. Damit die Komponente diese Fähigkeit erhält, sieht die technische Architektur einen JADE-Agenten und mehrere Verhaltensdefinitionen vor. Die Abbildung 6-3 stellt die Klassenstruktur der konkret realisierten Komponente dar.

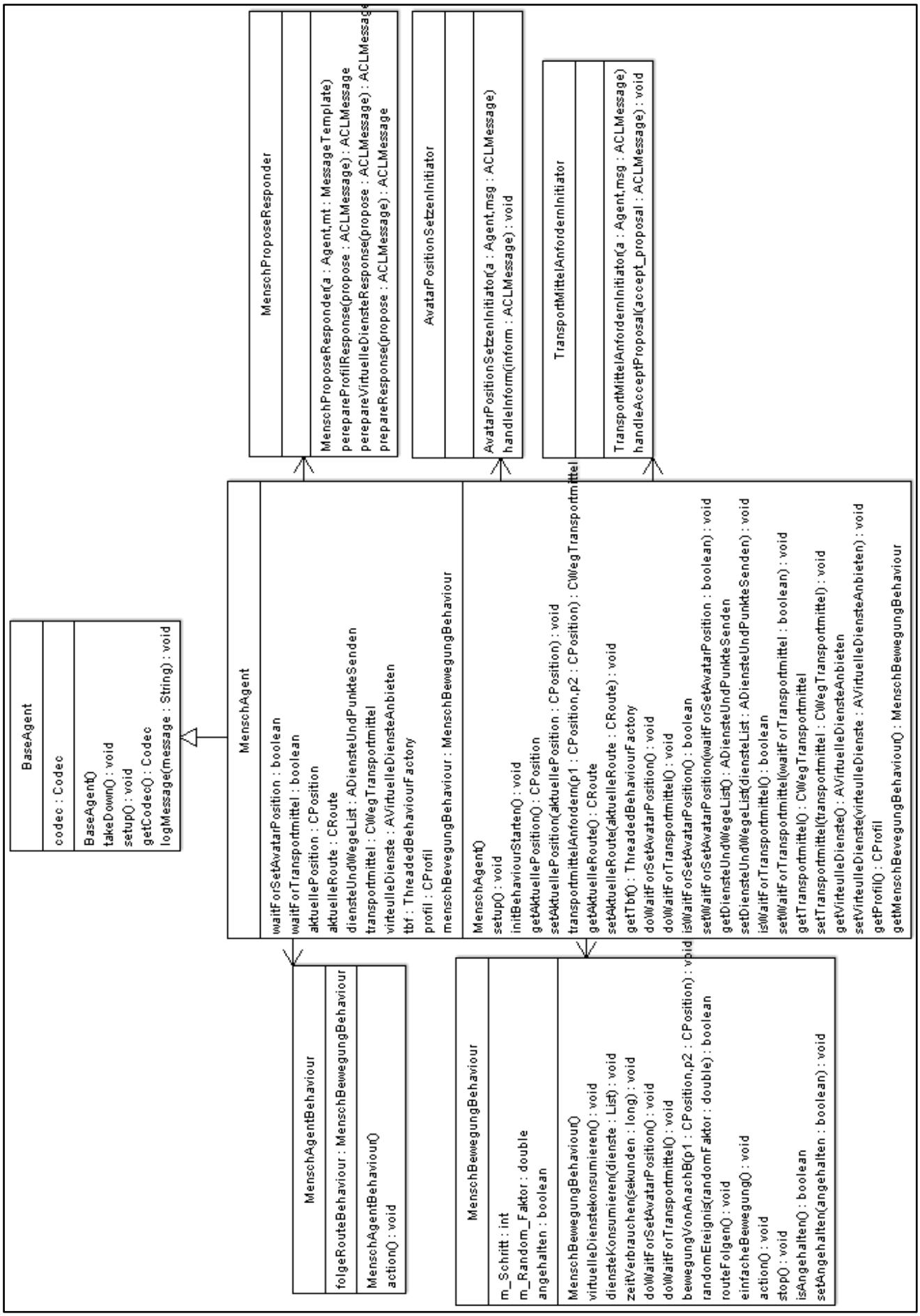

Abbildung 6-3: Klassendiagramm Fluggastverhalten

Die Klasse *MenschAgent* (im Folgenden werden alle Klassennamen durch eine kursive unterstrichene Schrift hervorgehoben) realisiert den geforderten JADE-Software-Agenten und bildet somit den Kern der Komponente. Die Klasse baut auf der Funktionalität eines Basisagenten (Klasse *BaseAgent*) auf und erweitert diese, indem sie die Ausführung der Verhaltensdefinitionen (realisiert durch die Klassen *MenschAgentBehaviour*, *MenschBewegungBehaviour*, *MenschProposeResponder*, *TransportMittelAnfordernInitiator* und *AvatarPositionSetzenInitiator*) steuert. Zusätzlich verwaltet der Menschagent das Fluggastdatenmodell (z.B. aktuelle Route, aktuelle Position, aktuelles Transportmittel und Fluggastprofil usw.), tritt dabei als zentraler Datencontainer auf und ermöglich einen kontrollierten Zugriff auf die entsprechenden Daten. Das Fluggastmodell ermöglicht das Zusammenspiel der einzelnen Verhaltensdefinitionen, indem letze Daten des Modells ändern bzw. auf Änderungen im Datenmodell reagieren. Die Änderungen werden von den Verhaltensdefinitionen durch aktives Abfragen der entsprechenden Informationen festgestellt (s. Abbildung 6-4).

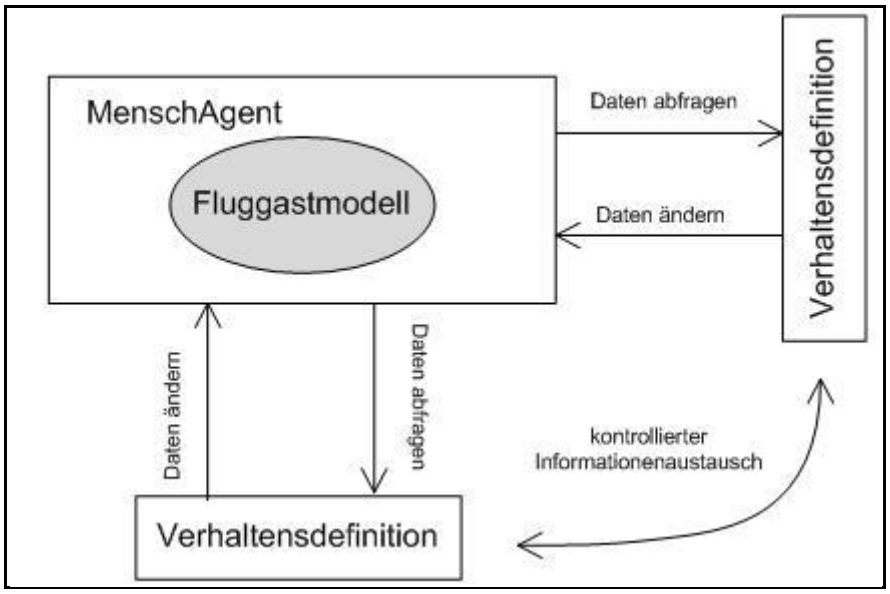

Abbildung 6-4: Datenaustausch zwischen Verhaltensdefinitionen

Die Verhaltensdefinitionen bilden das nach außen sichtbare Agentenverhalten ab. Dabei müssen zwei Gruppen unterschieden werden: die Klassen *MenschAgentBehaviour*, *MenschBewegungBehaviour* sind für das Abbilden des fachlich definierten Verhaltens eines Fluggastes (z.B. Bewegung entlang einer Route, Abweichen von vorgegebener Route, Konsumieren von Diensten usw.) zuständig, die Klassen *MenschProposeResponder*, *TransportMittelAnfordernInitiator* und *AvatarPositionSetzenInitiator* sind für die übergreifende Komponenteninteraktion (z.B. aktuelle Position des Fluggastes im System bekannt geben, aktuelles

Transportmittel anfordern usw.) zuständig und können somit als Hilfsverhaltensdefinitionen mit Schnittstellenfunktion betrachtet werden.

Ein tieferer Einblick in die Realisierung der Komponente kann gewonnen werden, wenn das modellierte fachliche Verhalten, realisiert in den Java-Klassen *MenschAgentBehaviour*, *MenschBewegungBehaviour,* näher betrachtet wird. Beide Verhaltensdefinitionen werden noch bei der Initialisierung bzw. beim Start des Menschagenten aktiviert (s. Abbildung 6-5).

```
protected void setup() {
    super.setup();
    this.initBehaviourStarten();
    this.menschBewegungBehaviour =new MenschBewegungBehaviour();
    tbf = new ThreadedBehaviourFactory();
    addBehaviour(new MenschAgentBehaviour());
    addBehaviour(getTbf().wrap(this.getMenschBewegungBehaviour()));
    addBehaviour(new MenschProposeResponder(this,MessageTemplate.and
    this.profil = new CProfil();
}
```

Abbildung 6-5: Initialisierung des Menschagenten

Die Verhaltensdefinition *MenschAgentBehaviour* ist vom Typ *CylclicBehaviour* (s. Kapitel 5.1.2) und hat die Aufgabe, Routenvorschläge des Indoor-Fußgängernavigationssystems zu empfangen. Der Empfang findet mittels FIPA-Nachrichtenaustausch statt, wobei die Routenvorschläge mittels Ontologie beschriebenen Nachrichten ausgetauscht werden. In Abbildung 6-6 ist die Empfangsroutine der Verhaltensdefinition vorgestellt.

```
public void action() {
    ACLMessage msg = null;
    msg = myAgent.receive(MessageTemplate.and(MessageTemplate.MatchOntology(
            ApplikationOntology.ONTOLOGY_NAME),
            MessageTemplate.MatchPerformative(ACLMessage.REQUEST)));
    if (msg!=null){
        ContentManager cm = myAgent.getContentManager();
        Action action;
        try {
            action = (Action) cm.extractContent(msg);
            if (action.getAction() instanceof AFolgeRoute){
                AFolgeRoute content = (AFolgeRoute)action.getAction();
                ((MenschAgent)myAgent).setAktuelleRoute(content.getRouteZuFolgen());
                ((BaseAgent)myAgent).logMessage("RoutenInfo");
                ((MenschAgent)myAgent).getMenschBewegungBehaviour().stop();
            } else {
                myAgent.putBack(msg);
            }
        } catch (UngroundedException ex) {
            ex.printStackTrace();
        } catch (OntologyException ex) {
            ex.printStackTrace();
        } catch (Codec.CodecException ex) {
            ex.printStackTrace();
        }
    }
    else block(10);
}
```

Abbildung 6-6: Empfang von Routenvorschlägen

Die Verhaltensdefinition reagiert auf ACL-Request-Nachrichten und überprüft, ob sich hinter der jeweils eintreffenden Nachricht eine durch die Applikationsontologie definierte *AFolgeRoute*-Aktion verbirgt. Wenn das der Fall ist, handelt es sich bei dieser Nachricht um einen Routenvorschlag, der als Objekt in der *AFolgeRoute*-Aktion enthalten ist. Dieser Vorschlag wird an den Menschagenten weitergegeben. Sollte der Menschagent zu diesem Zeitpunkt einen früheren Routenvorschlag durch die Verhaltensdefinition *MenschBewegungBehaviour* abarbeiten, signalisiert die *MenschAgentBehaviour*, dass dieser Vorgang unterbrochen werden muss, damit der neue Routenvorschlag abgearbeitet werden kann.

Parallel zu *MenschAgentBehaviour* ist auch die Verhaltensdefinition *MenschBewegungBehaviour* aktiv. Diese modelliert „freie Bewegung“ bzw. „Bewegung entlang einer vorgeschlagenen Route“ des virtuellen Fluggastes, dazu die Nutzung von Diensten und eventuelle Routenabweichungen. Die Abbildung 6-7 veranschaulicht das modellierte Verhalten.

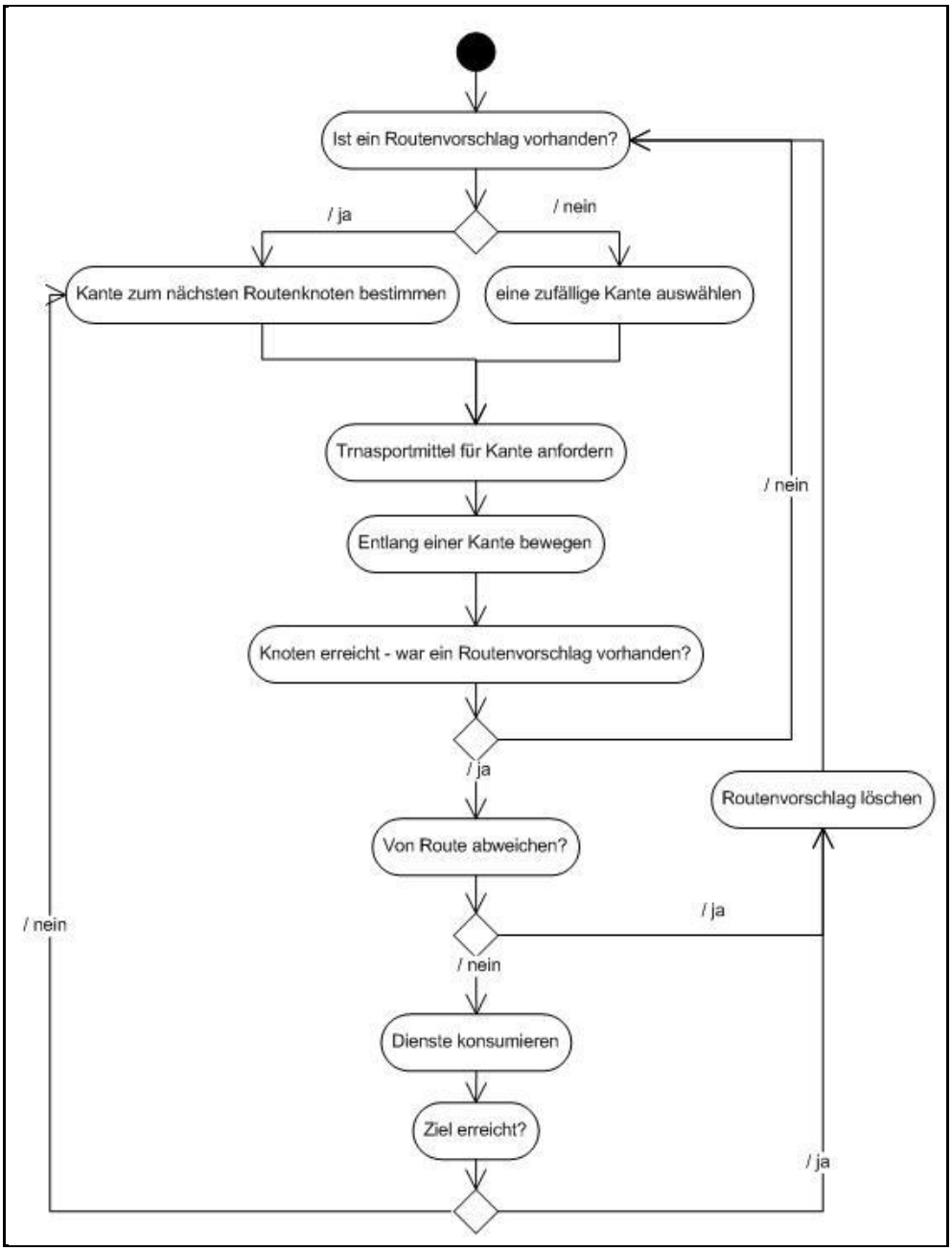

Abbildung 6-7: Verhaltensdefinition MenschBewegungBehaviour

Bei der Verhaltensdefinition *MenschBewegungBehaviour* handelt es sich um eine sehr komplexe Verhaltensdefinition vom Typ *CyclicBehaviour* (s. Kapitel 5.1.2). Wegen der großen Menge von Aktivitäten, die von der Verhaltensdefinition abgearbeitet werden, muss diese Verhaltensdefinition parallel (in einem zusätzlichen Thread des Menschagenten) zu allen anderen Verhaltensdefinitionen ausgeführt werden, damit deren Ausführung nicht blockiert wird. Aus diesem Grund wird diese Verhaltensdefinition durch ein spezielles Objekt (ThreadedBehaviourFactory) gestartet (s. Abbildung 6-5).

Einmal aktiviert überprüft, die Verhaltensdefinition, ob ein Routenvorschlag vorhanden ist. Ist das nicht der Fall, wird eine freie Bewegung simuliert, indem bei jedem Knoten des Gebäudegraphen eine beliebige ausgehende Kante für die bevorstehende Bewegung ausgewählt wird. Dadurch werden zufällige Strecken vom Menschagenten innerhalb des Gebäudegraphen zurückgelegt. Bei Positionsänderungen des Menschagenten wird die Struktur (Knoten und Kanten) des Gebäudegraphen mittels der Verhaltensdefinition *AvatarPositionSetzenInitiator* bei der JADE-Instanz „Simulationskontext“ abgefragt. Wenn ein Knoten erreicht wurde, wird eine zufällige Kante ausgewählt, die von diesem Knoten ausgeht. Für diese Kante wird von der JADE-Instanz „Simulationskontext“ das entsprechende Transportmittel durch die Verhaltensdefinition *TransportMittelAnfordernInitiator* angefordert. Das Transportmittel beschreibt, mit welcher Geschwindigkeit sich der Menschagent entlang einer Kante bewegen kann. Auf Grund dieser Geschwindigkeit wird die Bewegung entlang der Kante simuliert, indem eine Zeit berechnet wird, die der Fluggast bräuchte zum Zurückgehen der entsprechenden Strecke mit einer ähnlichen Geschwindigkeit. Der Begriff „ähnliche Geschwindigkeit“ basiert auf der Annahme, dass der Fluggast von der im Transportmittel angegebene Geschwindigkeit abweichen wird. Während der simulierten Bewegung wird die Position des Menschagenten ständig aktualisiert. Diese Aktualisierung wird durch die Verhaltensdefinition *AvatarPositionSetzenInitiator* an die JADE-Instanz „Simulationskontext“ weitergegeben. Nach der Bewegung entlang einer Kante wird der nächste Knoten des Gebäudegraphen erreicht. Im Fall freier Bewegung wiederholt sich der Vorgang nach diesem Punkt.

In dem Fall, dass ein Routenvorschlag an den Menschagenten unterbreitet worden ist, bewegt sich der Menschagent nicht mehr zufällig durch den Gebäudegraphen sondern folgt der Route, die von der Verhaltensdefinition *MenschAgentBehaviour* empfangen wurde. Der weitere Vorgang ist bis zu dem Zeitpunkt, an dem der nächste Knoten erreicht wird, mit dem Vorgang einer freien Bewegung identisch. Nachdem der Menschagent den nächsten Knoten erreicht hat und bis zu diesem Punkt ein Routenvorschlag vorhanden war, wird mit einer Wahrscheinlichkeit entschieden, ob der Routenvorschlag weiterhin abgearbeitet wird, oder ob der Menschagent von der vorgeschlagenen Route abweicht, den aktuellen Routenvorschlag löscht und zu einer freien Bewegung übergeht. Falls der Menschagent nicht abweicht, werden Dienste konsumiert, die vom Knoten aus erreichbar sind. Die Information darüber, welche Dienste genutzt werden sollen und wie lange das Nutzung jedes einzelnen Dienstes dauert, wird durch die Verhaltensdefinition *AvatarPositionSetzenInitiator* bereits bei Erreichen des Knotens von der JADE-Instanz „Simulationskontext“ abgefragt und zusätzlich durch die Verhaltensdefinition *MenschProposeResponder* (nur bei Zustandsänderung von virtuellen Diensten im Umgebungsmodell) empfangen. Beim Konsumieren von Diensten weicht die betreffende Dauer mit einer gewissen Wahrscheinlichkeit von der vordefinierten Dauer ab. Falls der

Menschagent sein Ziel (das Ende der Route) nicht erreicht hat, wird der gesamte Vorgang wiederholt, im anderen Fall geht die Verhaltensdefinition zu Simulation freier Bewegung über. Während des Folgens eines Routenvorschlages können zwei Arten von Abweichung (räumlich und zeitlich) auftreten. Diese werden vom Indoor-Fußgängernavigationssystem auf Grund der sich ständig aktualisierenden Position des Menschagenten wahrgenommen.

Wie zu Anfang dieses Kapitels erwähnt, werden Verhaltensdefinitionen nicht nur zum Abbilden des gewünschten Verhaltens eines Fluggastes sondern auch für übergreifende Komponenteninteraktion bzw. als Schnittstellen eingesetzt. Der Menschagent verfügt über drei Verhaltensdefinitionen mit Schnittstellenfunktionalität: *MenschProposeResponder*, *TransportMittelAnfordernInitiator* und *AvatarPositionSetzenInitiator*.

Die Verhaltensdefinition *MenschProposeResponder* (s. Abbildung 6-8) ist vom Typ *ProposeResponder*. Dadurch implementiert diese Verhaltensdefinition zum Datenaustausch mit der JADE-Instanz „Simulationskontext“ das FIPA-Propose-Interaction-Protokoll als Initiator (s. Kapitel 2.3.3).

```
protected ACLMessage prepareResponse(ACLMessage propose) throws NotUnderstoodEx
    ACLMessage reply = null;
    if (propose!=null){
        Action action;
        try {
            action = (Action) ((BaseAgent) myAgent).getContentManager().extract
            if (action.getAction() instanceof AProfilAnfordern){
                 reply = pereparеProfilResponse(propose);
            } else if (action.getAction() instanceof AVirtuelleDiensteAnbieten)
                reply = perepareVirtuelleDiensteResponse(propose);
            } else {}
        } catch (UngroundedException ex) {
            ex.printStackTrace();
            reply  = propose.createReply();
            reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
        } catch (Codec.CodecException ex) {
            ex.printStackTrace();
            reply  = propose.createReply();
            reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
        } catch (OntologyException ex) {
            ex.printStackTrace();
            reply  = propose.createReply();
            reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
        }
    }
    return reply;
}
```

Abbildung 6-8: Empfangsroutine MenschProposeResponder

Die Verhaltensdefinition reagiert auf zwei Aktionstypen (*AProfilAnfordern* und *AVirtuelleDiensteAnbieten*) aus der Applikationsontologie und bestätigt die Ausführung der entsprechenden Aktion. Im ersten Fall wird das Profil des simulierten Fluggastes durch den Content der Accept-Proposal-Nachricht an den Initiator (JADE-Instanz „Simulationskontext“) versandt. Im zweiten Fall erwartet die Verhaltensdefinition Informationen über aktive virtuelle Dienste im Content der Propose-Nachricht. Diese Informationen werden dann an den Menschagenten weitergegeben.

Die Verhaltensdefinition *TransportMittelAnfordernInitiator* (s. Abbildung 6-9) ist vom Typ *ProposeInitiator*. Dadurch implementiert diese Verhaltensdefinition zum Datenaustausch mit der JADE-Instanz „Simulationskontext“ das FIPA-Propose-Interaction-Protokoll als Initiator (s. Kapitel 2.3.3).

```
public class TransportMittelAnfordernInitiator extends ProposeInitiator{
    public TransportMittelAnfordernInitiator(Agent a, ACLMessage msg) {
        super(a,msg);
        ((MenschAgent)a).setWaitForTransportmittel(true);
    }
    protected void handleAcceptProposal(ACLMessage accept_proposal) {
        ACLMessage reply = null;
        if (accept_proposal!=null){
            Action action;
            try {action = (Action) ((BaseAgent) myAgent).getContentManager().extractConte
                ATransportmittelSenden content  = ((ATransportmittelSenden)action.getActi
                ((MenschAgent)myAgent).setTransportmittel(content.getTransportmittelRoute
                ((BaseAgent)myAgent).logMessage("Transportmitel: V=" + content.getTranspo
            } catch (UngroundedException ex) {
                ex.printStackTrace();
            } catch (Codec.CodecException ex) {
                ex.printStackTrace();
            } catch (OntologyException ex) {
                ex.printStackTrace();
            }
        }
        ((MenschAgent)myAgent).setWaitForTransportmittel(false);
    }
}
```

Abbildung 6-9: TransportMittelAnfordernInitiator

Die Aufgabe dieser Verhaltensdefinition ist, das Transportmittel für eine Kante des Gebäudegraphen beim Umgebungsmodell abzufragen. Die Abfrage wird durch die Aktion *ATransportmittelAnfordern* in der ACL-Propose-Nachricht formuliert. Das Ergebnis (die Aktion *ATransportmittelSenden*) wird in der ACL-Accept-Nachricht erwartet.

Die Verhaltensdefinition *AvatarPositionSetzenInitiator* (s. Abbildung 6-10) ist vom Typ AchieveREInitiator. Dazu implementiert diese Verhaltensdefinition das FIPA-Request-Interaction-Protokoll als Initiator (s. Kapitel 2.3.3) zum Datenaustausch mit der JADE-Instanz „Simulationskontext“.

```
public class AvatarPositionSetzenInitiator extends AchieveREInitiator{
    public AvatarPositionSetzenInitiator(Agent a, ACLMessage msg) {
        super(a,msg);
        ((MenschAgent)a).setWaitForSetAvatarPosition(true);
    }
    protected void handleInform(ACLMessage inform) {
        ACLMessage reply = null;
        if (inform!=null){
            Action action;
            try {action = (Action) ((BaseAgent) myAgent).getContentManager().
                ADiensteUndPunkteSenden content  = ((ADiensteUndPunkteSenden)
                ((MenschAgent)myAgent).setDiensteUndWegeList(content);
            } catch (UngroundedException ex) {
                ex.printStackTrace();
            } catch (Codec.CodecException ex) {
                ex.printStackTrace();
            } catch (OntologyException ex) {
                ex.printStackTrace();
            }
        }
        ((MenschAgent)myAgent).setWaitForSetAvatarPosition(false);
    }
```

Abbildung 6-10: AvatarPositionSetzenInitiator

Die Aufgabe der Verhaltensdefinition *AvatarPositionSetzenInitiator* ist es, die Positionsänderungen des Menschagenten an das Umgebungsmodell weiterzugeben, zusätzlich die Struktur des Gebäudegraphen sowie aktive Dienste nahe der aktuellen Position des Menschagenten zu ermitteln. Die Position wird durch die Aktion *AAvatarPositionSetzen* mit der ACL-Request-Nachricht versandt. Da es sich in unserem Fall um ein verteiltes System handelt, kann es zwischen der tatsächlichen Position des Menschagenten und der im Umgebungsmodell abgebildeten Position zu großen Abweichungen kommen. Da die im Umgebungsmodell abgebildete Position an das Indoor-Fußgängernavigationssystem weitergegeben wird und das Navigationssystem auf Grund dieses Wertes über das weitere Vorgehen entscheidet, führen große Abweichungen zu Inkonsistenz. Damit solche Situationen ausgeschlossen werden können und die Abweichungen der beiden Positionswerte minimiert werden, stoppt die Verhaltensdefinition die Ausführung des Menschagenten für die Zeit, bis die Positionsänderung durch eine ACL-Inform-Nachricht von der JADE-Instanz „Simulationskontext“ bestätigt wird. Durch die Inform-Nachricht werden auf dem Rückweg die

Informationen über aktive Dienste und mögliche Wege, bezogen auf die aktuelle Position des Agenten, bekannt gegeben.

6.3.2 Simulationskontext

Die JADE-Instanz „Simulationskontext“ kapselt die Fachkomponenten Gebäudedienstkontext und Simulationsmanager. Die Komponente Gebäudedienstkontext hat die Aufgaben, die Gebäudestruktur, das Dienstangebot und das Routenmodell des virtuellen Flughafengeländes abzubilden. Dafür wurde eine umfangreiche Klassenhierarchie entwickelt, die das in Kapitel 5.2.2 definierte Konzept realisiert. Bei der Realisierung musste entschieden werden, nach welchem Verfahren das Routenmodell und insbesondere der Gebäudegraph des Routenmodells aus den Geometriedaten des simulierten Flughafengeländes abgeleitet werden könnten. Das Konzept sah zwei Möglichkeiten (s. Kapitel 3.1) vor: ein semiautomatisches, dynamisches und ein manuelles, statisches Verfahren zur Generierung bzw. Erzeugung der benötigten Informationen. Um die Komplexität der Prototypenrealisierung zu begrenzen, wurde ein statisches Verfahren (ähnlich wie in Kapitel 3.1.1 beschrieben) implementiert. Das Verfahren ermöglicht eine manuelle Eingabe von Routen durch die abgebildete Flughafengebäudestruktur.

Die Komponente Simulationsmanager (s. Abbildung 6-11) hat die Aufgaben, die einzelnen Funktionalitäten der Simulationsumgebung zusammenzuführen, den aktiven Simulationsprozess grafisch zu visualisieren und eine Benutzerinteraktion durch geeignete grafische Schnittstellen zu ermöglichen.

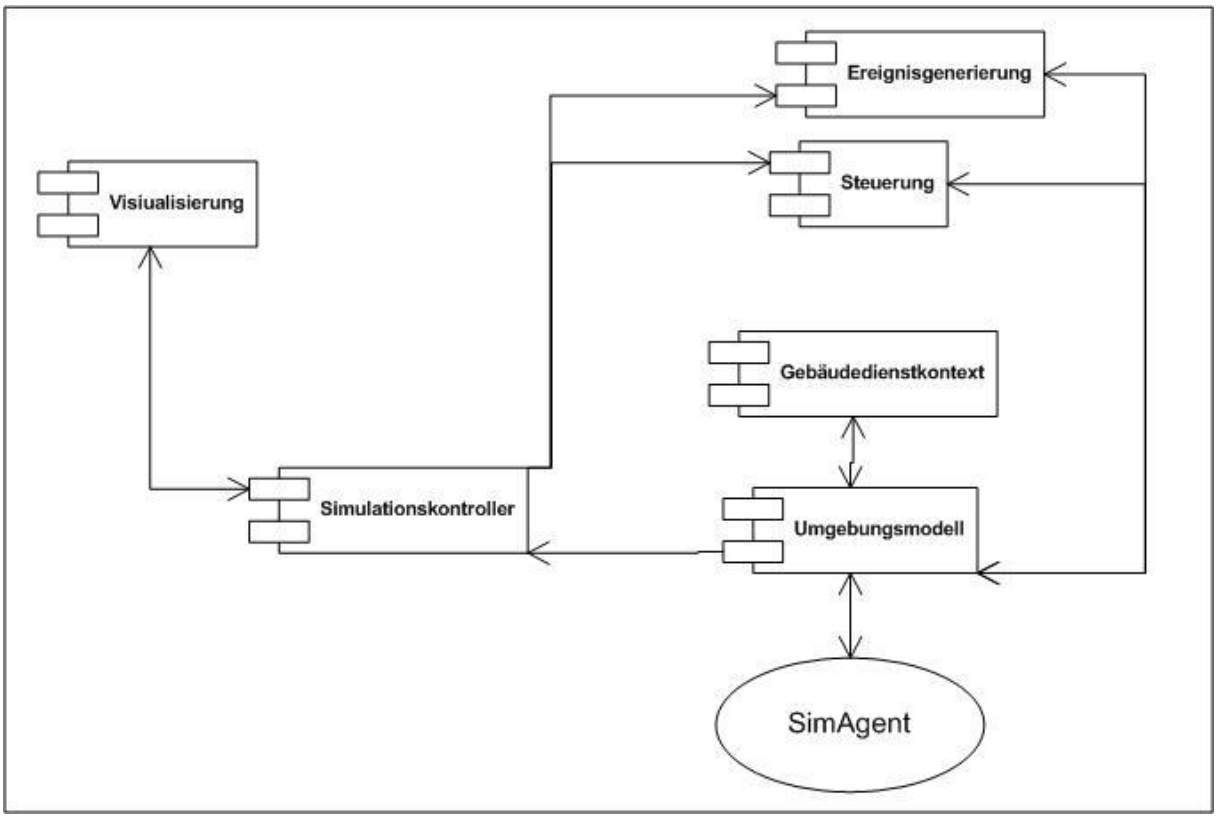

Abbildung 6-11: Komponentendiagramm des Simulationsmanagers

Die Komponente Simulationsmanager basiert auf dem Model-View-Controller-Muster (s. [Gamma96]). Das Modell (zu englisch Model) bildet durch die Unterkomponenten Umgebungsmodell, Gebäudedienstkontext, Steuerung und Ereignisgenerierung die unterschiedlichen Aspekte (aktuelle Position eines Fluggastes, aktuelles Dienstangebot, Gebäudestruktur usw.) des aktiven Simulationsprozesses in einem gemeinsamen Kontext ab. Die Unterkomponente Visualisierung (zu englisch View) stellt den durch das Modell abgebildeten Simulationsprozess grafisch dar. Zusätzlich ermöglicht die Visualisierung durch eine entsprechende GUI-Oberfläche die Benutzerinteraktion mit der Simulationsumgebung bzw. mit dem Modell des Simulationsprozesses. Die Unterkomponente Simulationskontroller übernimmt die Controller-Rolle, indem sie die Änderungen im Modell an die Visualisierung übermittelt sowie Benutzerinteraktionen von der Visualisierung entgegennimmt und auf das Modell überträgt. Abbildungen 6-12 und 6-13 zeigen, wie ein innerhalb dieser Arbeit aufgebautes Beispielszenario vom Prototyp grafisch dargestellt wird.

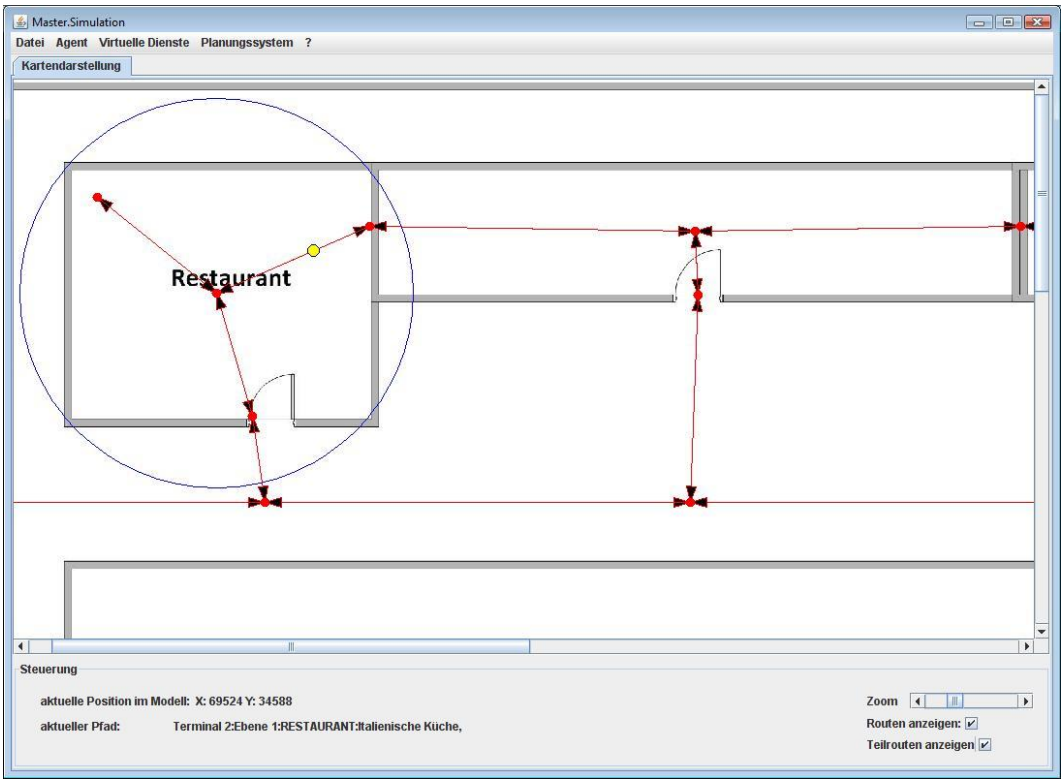

Abbildung 6-12: Grafische Teildarstellung des aufgebauten Beispielszenarios

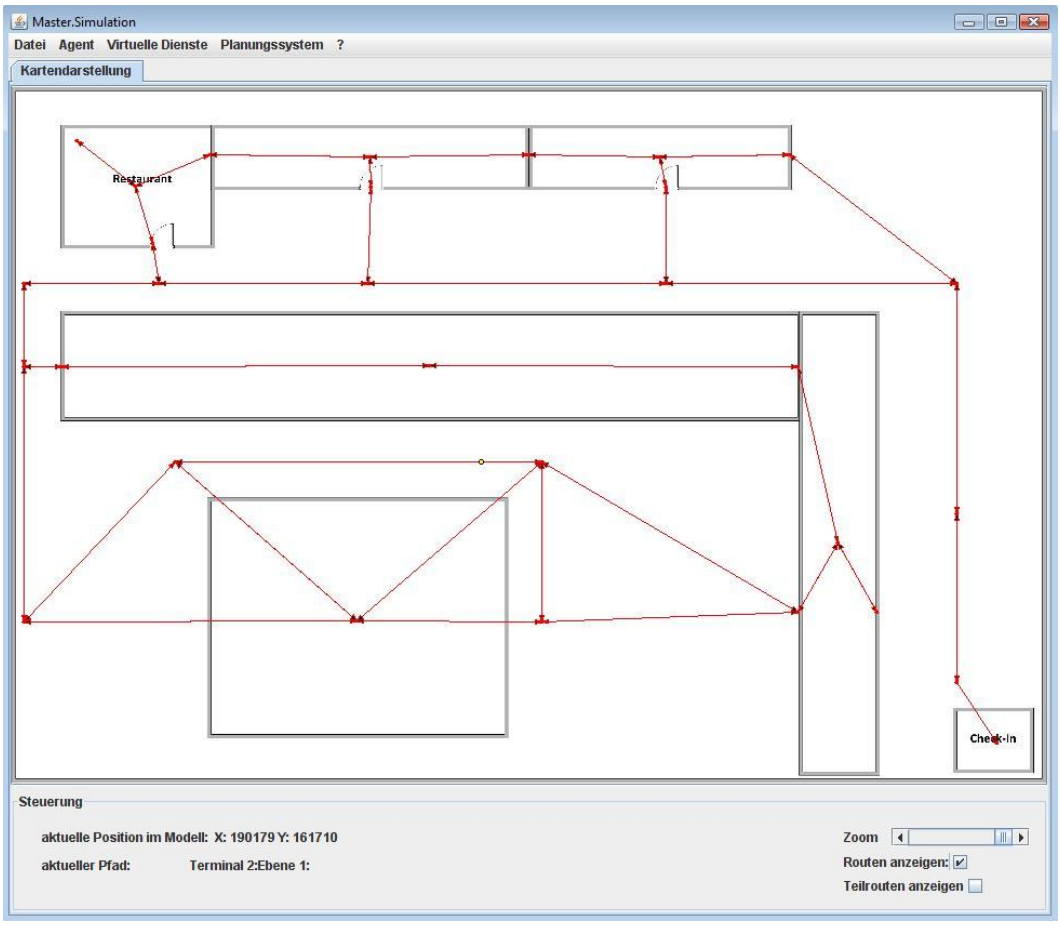

Abbildung 6-13: Grasfische Gesamtdarstellung des aufgebauten Beispielszenarios

Zusätzlich beinhaltet die Instanz den Simagenten (s. Abbildung 6-14). Dieser ermöglicht durch drei Verhaltensdefinitionen (die Klassen *SimRequestResponder*, *SimProposeResponder*, *SimAgentSubscriptionResponder*) übergreifende Kommunikation mit den anderen Systemkomponenten, indem er Zustandsänderungen im modellierten Simulationsprozess an das Indoor-Fußgängernavigationssystem weiterleitet, zusätzlich die Positionsänderungen des Menschagenten empfängt und diese im Modell abbildet.

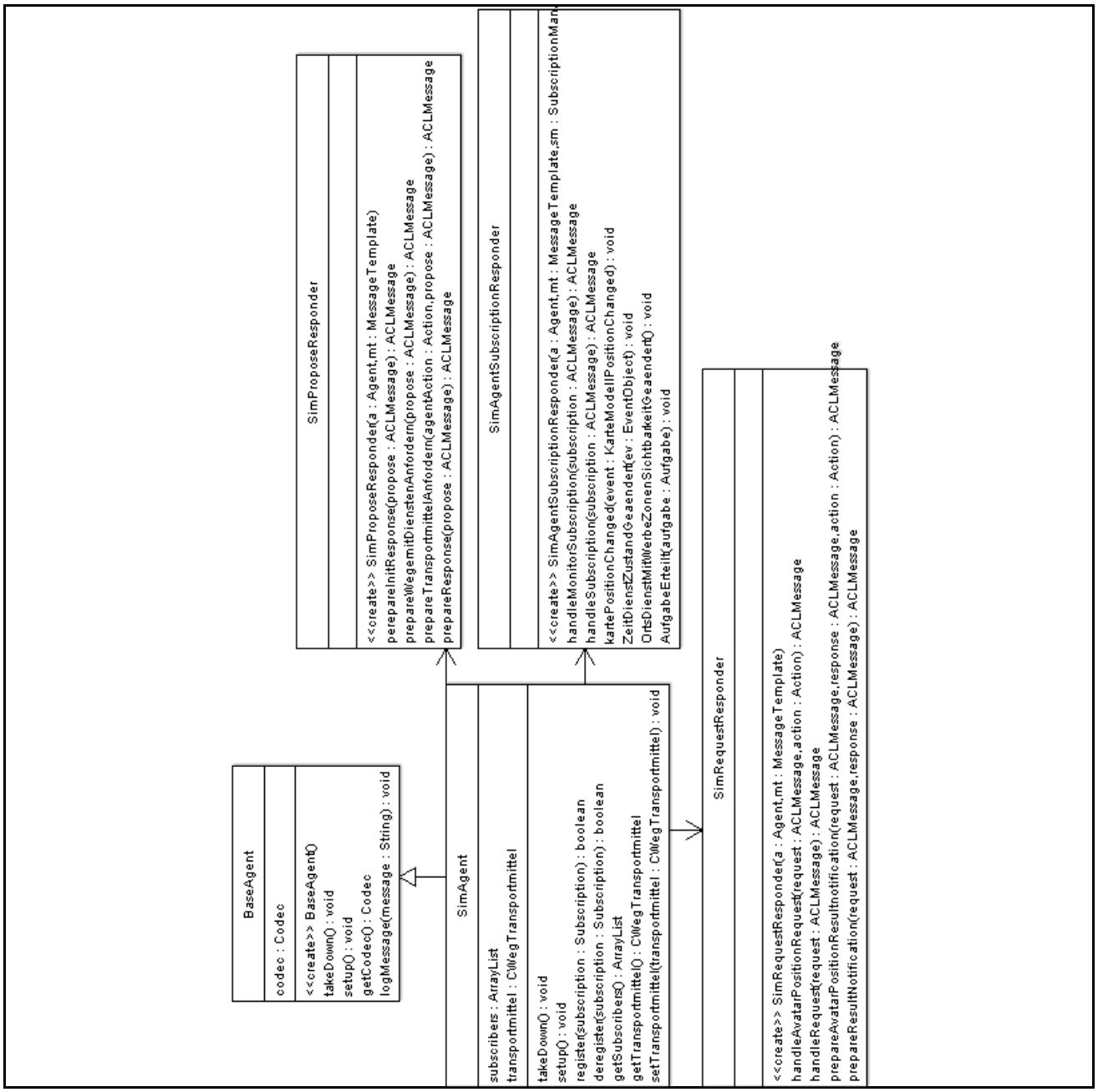

Abbildung 6-14: Klassendiagramm SimAgent

Die Verhaltensdefinition *SimRequestResponder* (s. Abbildung 6-15) ist vom Typ AchieveREResponder. Diese Verhaltensdefinition implementiert das FIPA-Request-Interaction-Protokoll als Empfänger (s. Kapitel 2.3.3) zum Datenaustausch mit der JADE-Instanz „Fluggastverhalten“.

```
protected ACLMessage handleAvatarPositionRequest(ACLMessage request,Action action){
    final AAvatarPositionSetzen content = (AAvatarPositionSetzen)action.getAction();
    Runnable runnable = new Runnable(){
        public void run(){
        SimManager.getInstance().getKarte().setAvatarPosition(
                content.getPosition().getX(),content.getPosition().getY());
        }
    };
    SimManager.getInstance().runOnGuiThread(runnable);
    ACLMessage reply = request.createReply();
    reply.setPerformative(ACLMessage.AGREE);
    return reply;
}

protected ACLMessage handleRequest(ACLMessage request) throws NotUnderstoodException
    ACLMessage reply = null;
    if (request!=null){
        reply = request.createReply();
        Action action;
        try {
            action = (Action) ((BaseAgent) myAgent).getContentManager().extractConte
            if (action.getAction() instanceof AAvatarPositionSetzen){
                reply = handleAvatarPositionRequest(request,action);
            } else {
                reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
            }
```

Abbildung 6-15: Verhaltensdefinition SimRequestResponder

Die Aufgabe der Verhaltensdefinition *SimRequestResponder* ist es, die vom Menschagenten durch die Verhaltensdefinition *AvatarPositionSetzenInitiator* (s. Kapitel 6.3.1) gemeldeten Positionsänderungen zu empfangen und diese in das Umgebungsmodell zu übertragen. Nachdem eine Positionsänderung vom Umgebungsmodell übernommen wurde, wird sie dem Menschagenten durch das Versenden einer ACL-Inform-Nachricht bestätigt. Zusätzlich übermittelt die Inform-Nachricht Informationen über aktive Dienste und mögliche Wege, bezogen auf die aktuelle Position des Menschagenten.

Die Verhaltensdefinition *SimProposeResponder* (s. Abbildung 6-16) ist vom Typ *ProposeResponder*. Diese Verhaltensdefinition implementiert das FIPA-Propose-Interaction-Protokoll als Empfänger (s. Kapitel 2.3.3) zum Datenaustausch mit den JADE-Instanzen „Fluggastverhalten“ und „Indoor-Fußgängernavigationssystem“.

```
protected ACLMessage prepareResponse(ACLMessage propose) throws NotUnderstoodExcep
    ACLMessage reply = null;
    if (propose!=null){
        Action action;
        try {
            action = (Action) ((BaseAgent) myAgent).getContentManager().extractCon

            if (action.getAction() instanceof AMenschAgentenInitialisieren){
                 reply = perepareInitResponse(propose);
            } else if (action.getAction() instanceof AWegeMitDienstenAnfordern){
                reply = prepareWegemitDienstenAnfordern(propose);
            } else if (action.getAction() instanceof ATransportmittelAnfordern){
                reply = prepareTransportmittelAnfordern(action,propose);
            } else {
            }
```

Abbildung 6-16: Verhaltensdefinition SimProposeResponder

Die Aufgabe der Verhaltensdefinition *SimProposeResponder* ist es, Abfragen von im Umgebungsmodell abgebildeten Transportmitteln und Wegen sowie von Dienstinformationen zu ermöglichen. Die Informationen werden aus dem Umgebungsmodell entgegengenommen und in entsprechende Aktionen (*ATransportmittelSenden* oder *AWegeMitDienstenSenden*) verpackt und an den Initiator des Protokolls (Fluggastverhalten oder Indoor-Fußgängernavigationssystem) zurückgeschickt.

Die Verhaltensdefinition *SimAgentSubscriptionResponder* (s. Abbildung 6-17) ist vom Typ *SubscriptionResponder*. Diese Verhaltensdefinition implementiert das FIPA-Subscribe-Interaction-Protokoll als Empfänger (s. Kapitel 2.3.3) zum Datenaustausch mit der JADE-Instanz „Indoor-Fußgängernavigationssystem“.

```
public SimAgentSubscriptionResponder(Agent a, MessageTemplate mt, Subscript
    super(a,mt,sm);
    Runnable runnable = new AddPositionListener(this);
    SimManager.getInstance().runOnGuiThread(runnable);
    runnable =  new AddZeitDienstZustandListener(this);
    SimManager.getInstance().runOnGuiThread(runnable);
    runnable =  new AddOrtsDiensteMitWerbezonenSichtbarkeitListener(this);
    SimManager.getInstance().runOnGuiThread(runnable);
    runnable =  new AddAufgabenListener(this);
    SimManager.getInstance().runOnGuiThread(runnable);
```

Abbildung 6-17: Verhaltensdefinition SimAgentSubscriptionResponder (Initialisierung)

Die Aufgabe der Verhaltensdefinition *SimAgentSubscriptionResponder* ist es, Zustandsänderungen im Umgebungsmodell zu überwachen und diese an das Indoor-Fußgängernavigationssystem weiterzuleiten. Die Verhaltensdefinition überwacht die Sichtbarkeit und den Zustand der Dienste, die aktuelle Position (s. Abbildung 6-18) und Aufgabenliste des Menschagenten.

```
public void kartePositionChanged(KarteModellPositionChanged event) {
    ArrayList<Subscription> subscribers = ((SimAgent)myAgent).getSubscribers();
    if (subscribers!=null){
        ACLMessage notification = new ACLMessage(ACLMessage.INFORM);
        notification.setOntology(ApplikationOntology.ONTOLOGY_NAME);
        notification.setLanguage(((BaseAgent)myAgent).getCodec().getName());
        AAvatarPositionInform content = new AAvatarPositionInform();
        CPosition position = new CPosition();
        position.setX(event.getPosition().x);
        position.setY(event.getPosition().y);
        content.setNeuePosition(position);
        content.setTransportmittelRoute(((SimAgent)myAgent).getTransportmittel
        Action action =  new Action();
        action.setActor(myAgent.getAID());
        action.setAction(content);
        try {
            ((BaseAgent) myAgent).getContentManager().fillContent(notification,
            for (int i=0; i<subscribers.size();i++){
            subscribers.get(i).notify(notification);
        }
```

Abbildung 6-18: Verhaltensdefinition SimAgentSubscriptionResponder (Reaktion auf Positionsänderung)

Sollten sich eine oder mehrere der oben genannten Informationen im Umgebungsmodell geändert haben, werden die aktuellen Werte durch entsprechende Aktionen an das Indoor-Fußgängernavigationssystem gesandt. Das Navigationssystem entscheidet auf Grund der übermittelten Daten über den weiteren Verlauf der Fluggastführung.

6.3.3 Navigator

Die JADE-Instanz „Navigator“ ermöglicht die Anbindung eines externen Indoor-Fußgängernavigationssystems an die Simulationsumgebung und zusätzlich die Überwachung des Menschagenten (s. Kapitel 6.3.1). Diese Funktionalität wird mittels eines JADE-Agenten und mehrerer Verhaltensdefinitionen realisiert. Die Abbildung 6-19 stellt die Klassenstruktur der realisierten Komponente dar.

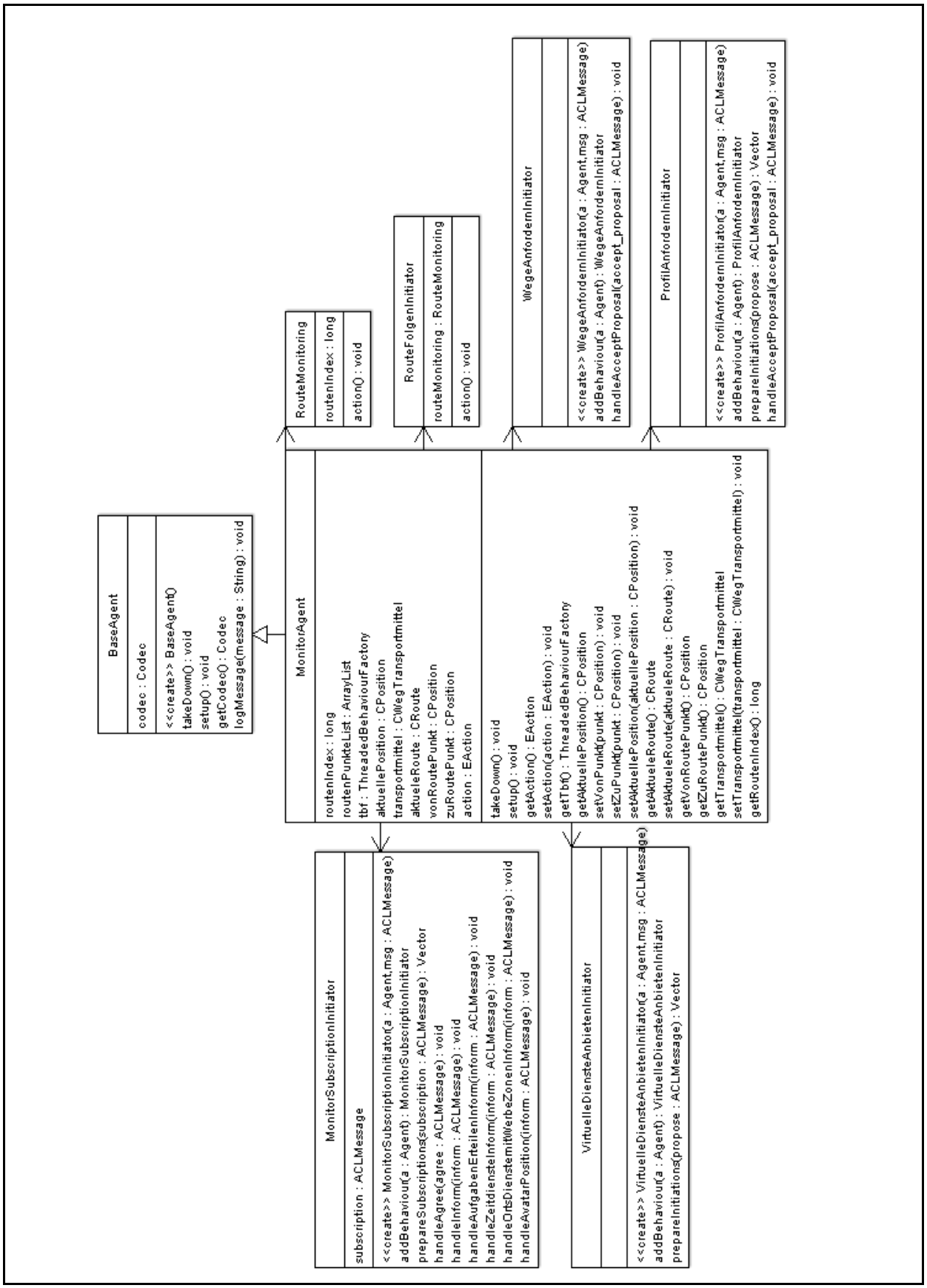

Abbildung 6-19: Klassendiagramm MonitorAgent

Die Klasse *MonitorAgent* bildet den geforderten JADE-Agenten ab. Die Klasse baut auf der Funktionalität eines Basisagenten (Klasse *BaseAgent*) auf und erweitert diese, indem sie die Ausführung der zuständigen Verhaltensdefinitionen (realisiert durch die Klassen *RouteFolgenInitiator*, *WegeAnfordernInitiator*, *ProfilAnfordernInitiator*, *VirtuelleDiensteAnbietenInitiator*, *MonitorSubscriptionInitiator*, *RouteMonitoring*) steuert und als zentraler Datencontainer (ähnlich wie der Menschagent, s. Kapitel 6.3.1) auftritt.

Die Verhaltensdefinitionen *RouteFolgenInitiator*, *WegeAnfordernInitiator*, *ProfilAnfordernInitiator*, *VirtuelleDiensteAnbietenInitiator* und *MonitorSubscriptionInitiator* realisieren die Fachkomponente Navigationssystemadapter (s. Kapitel 5.2.5). Dabei implementieren die Verhaltensdefinitionen die entsprechenden Schnittstellen in Form von FIPA-Protokollen, die eine Interaktion mit den Komponenten der Simulationsumgebung (Fluggast, Gebäudedienstkontext und Simulationsmanager) ermöglichen. Zusätzlich realisieren die Verhaltensdefinitionen geeignete Schnittstellen für die Kommunikation mit einem externen Indoor-Fußgängernavigationssystem und verantworten die notwendige Datentransformation für eine optimale Anbindung der beiden Systeme. Die Abbildung 6-20 stellt ein Beispiel dar, wie Routenvorschläge des Navigationssystems entgegengenommen und an die Komponente Fluggastverhalten weitergeleitet werden.

```
public void action() {
    ArrayList<AID> agents = BaseAgentManager.searchForServices(
            myAgent,"IMenschAgent");
    if (!agents.isEmpty()){
        AFolgeRoute content = new AFolgeRoute();
        content.setRouteZuFolgen(
                PlanungsSystemManager.getInstance().getPlaner().getRoute());
        ((MonitorAgent)myAgent).setAktueleRoute(content.getRouteZuFolgen());
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.setOntology(ApplikationOntology.ONTOLOGY_NAME);
        msg.setLanguage(((BaseAgent)myAgent).getCodec().getName());
        Action action =  new Action();
        action.setActor(myAgent.getAID());
        action.setAction(content);
        try {((BaseAgent)myAgent).getContentManager().fillContent(msg,action
        } catch (OntologyException ex) {
            ex.printStackTrace();
        } catch (Codec.CodecException ex) {
            ex.printStackTrace();
        }
        msg.addReceiver(agents.get(0));
        myAgent.send(msg);
    }
}
```

Abbildung 6-20: Verhaltensdefinition RouteFolgenInitiator

Die Verhaltensdefinition *RouteMonitoring* realisiert die Fachkomponente Monitor (s. Kapitel 5.2.5), indem sie die Positionsänderungen des Menschagenten überwacht und ständig mit den aktuellen Zeitvorgaben des externen Indoor-Fußgängernavigationssystems vergleicht. Sollte eine Abweichung von der Verhaltensdefinition festgestellt werden, wird sie an das Navigationssystem gemeldet und von der Komponente „Protokollierung“ dokumentiert (s. Abbildung 6-21).

```
public void action() {
    CPosition vonPunkt = ((MonitorAgent)myAgent).getVonRoutePunkt();
    CPosition zuPunkt = ((MonitorAgent)myAgent).getZuRoutePunkt();
    CPosition aktuellePosition = ((MonitorAgent)myAgent).getAktuellePosition();
    CWegTransportmittel tr = ((MonitorAgent)myAgent).getTransportmittel();
    if ((vonPunkt!=null)&&(zuPunkt!=null)&&(tr!=null)){
        ArrayList<ZeitplanEintrag> zeitplan =
                PlanungsSystemManager.getInstance().getPlaner().getZeitPlan();
        for (int i =0; i<zeitplan.size(); i++){
            if ((zeitplan.get(i).getPosition().getX()==zuPunkt.getX())
            &&(zeitplan.get(i).getPosition().getY()==zuPunkt.getY())
            && (((MonitorAgent)myAgent).getRoutenIndex()==i)){
                ((BaseAgent)myAgent).logMessage("Zeitplan:"+zeitplan.get(i).getZeit()
                long la = BerechnungsManager.getAbstand(vonPunkt,aktuellePosition);
                long lb = BerechnungsManager.getAbstand(zuPunkt,aktuellePosition);
                long lc = BerechnungsManager.getAbstand(vonPunkt,zuPunkt);
                long abstandZumZiel = 0;
                if ((Math.abs(lb+la-lc)<5) || (lc==0)){abstandZumZiel = lb;}
                else {abstandZumZiel = la+lc;}
                if (tr.getGeschwindigkeit()!=0){
                long zeit = BerechnungsManager.getVerbleibendeZeit(abstandZumZiel,tr)
                Date ankunftsZeit = java.util.Calendar.getInstance().getTime();
                ankunftsZeit.setTime(ankunftsZeit.getTime()+zeit);
                if (ankunftsZeit.getTime()>zeitplan.get(i).getZeit().getTime()){
                 ((BaseAgent)myAgent).logMessage(
                         "Zeitvorgaben werden nicht eingehalten.");
                 PlanungsSystemManager.getInstance().getPlaner().routeAbweichung();}
                }
                break;
            }
```

Abbildung 6-21: Verhaltensdefinition RouteMonitoring

6.3.4 Protokollierung

Die JADE-Instanz „Protokollierung“ enthält die Komponente gleichen Namens. Sie hat die Aufgabe das Dokumentieren bzw. Protokollieren des Simulationsprozesses zu ermöglichen, indem relevante Prozessdaten (z.B. Fluggastpositionsänderung, Aktivieren eines Dienstes, Versenden eines Routenvorschlags usw.) dauerhaft gespeichert werden. Die Komponente wird durch einen JADE-

Agenten und eine Verhaltensdefinition realisiert. Die Abbildung 6-22 stellt die Klassenstruktur der Komponente dar.

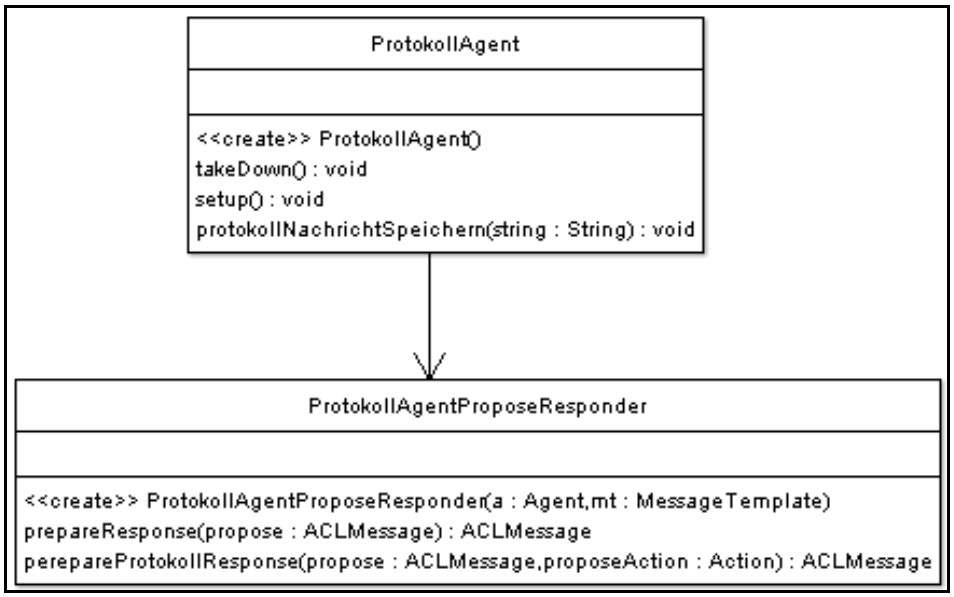

Abbildung 6-22: Klassendiagramm Protokollierung

Der ProtokollAgent speichert Daten des aktiven Simulationsprozesses, nachdem er sie über die Verhaltensdefinition *ProtokollAgentProposeResponder* von anderen Komponenten der Simulationsumgebung empfangen hat.

Die Verhaltensdefinition *ProtokollAgentProposeResponder* (s. Abbildung 6-23) ist vom Typ *ProposeResponder*. Diese Verhaltensdefinition implementiert das FIPA-Propose-Interaction-Protokoll als Empfänger (s. Kapitel 2.3.3) zum Empfang von Protokollinformationen von den JADE-Instanzen „Fluggastverhalten“ „Simulationskontext“ und „Navigationssystem“.

```
public class ProtokollAgentProposeResponder extends ProposeResponder{
    public ProtokollAgentProposeResponder(Agent a, MessageTemplate mt)
    {super(a,mt);}
    protected ACLMessage prepareResponse(ACLMessage propose) throws NotUndersto
        ACLMessage reply = null;
        if (propose!=null){Action action;
            try {
                action = (Action) ((BaseAgent) myAgent).getContentManager().ext
                if (action.getAction() instanceof AProtokollNachrichtSenden){
                     reply = pereparеProtokollResponse(propose, action);
                } else {}
            } catch (UngroundedException ex) {
                reply  = propose.createReply();
                reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
            } catch (Codec.CodecException ex) {
                reply  = propose.createReply();
                reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
            } catch (OntologyException ex) {
                reply  = propose.createReply();
                reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);}
        } return reply; }
    private ACLMessage pereparеProtokollResponse(ACLMessage propose, Action pro
        ((ProtokollAgent) myAgent).protokollNachrichtSpeichern(
        ((AProtokollNachrichtSenden)proposeAction.getAction()).getNachricht());
        Action action =  new Action();
        action.setActor(myAgent.getAID());
        ACLMessage reply  = propose.createReply();
        reply.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
        return reply;}
```

Abbildung 6-23: Verhaltensdefinition ProtokollAgentProposeResponder

Die Verhaltensdefinition *ProtokollAgentProposeResponder* erwartet beim Datenaustausch die Protokollinformationen in der ACL-Propose-Nachricht. Die Informationen müssen in Form einer *AProtokollNachrichtSenden*-Aktion formuliert sein. Nach Empfang werden die Protokollinformationen aus der Aktion extrahiert und an den ProtokollAgenten zum Speichern weiter gegeben.

Die JADE-Instanzen „Fluggastverhalten“ „Simulationskontext“ und „Navigationssystem“ und deren Komponenten versenden Protokollinformationen bzw. Prozessdaten an die Protokollierung durch die realisierten Agenten (MenschAgent, SimAgent, MonitorAgent). Jeder Agent basiert auf der Funktionalität eines Basisagenten (s. Abbildung 6-24).

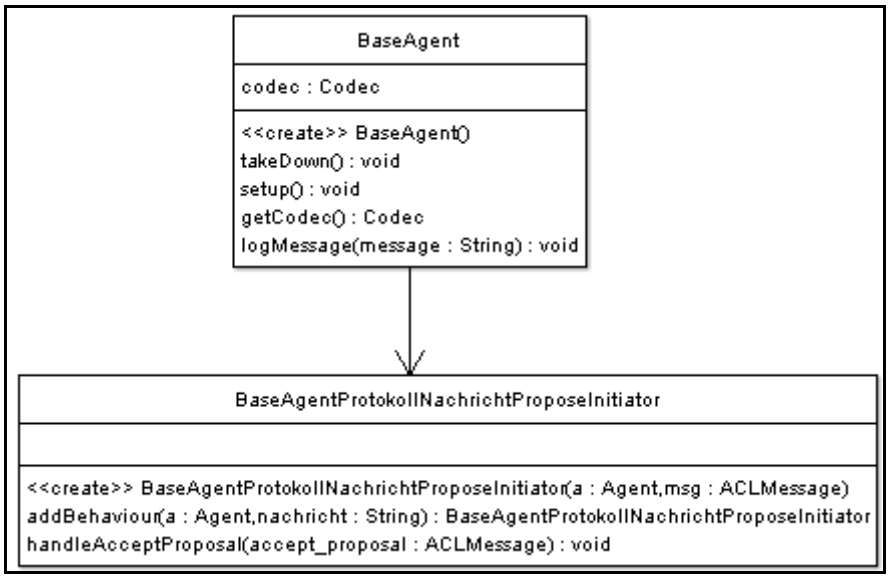

Abbildung 6-24: Klassendiagramm BaseAgent

Der Basisagent implementiert die Schnittstelle zu der Komponente Protokollierung mittels der Verhaltensdefinition *BaseAgentProtokollNachrichtProposeInitiator*. Zusätzlich bietet der Agent die Methode *logMessage* (s. Abbildung 6-25), die Informationen direkt an die Komponente Protokollierung versendet und in jeder JADE-Instanz bzw. von jedem Agent der Simulationsumgebung aufgerufen werden kann, um relevante Prozessdaten zu speichern.

```
public void logMessage(String message){
    Calendar cal = Calendar.getInstance();
    SimpleDateFormat sdf = new SimpleDateFormat("HH.mm.ss");
    String nachricht = this.getLocalName()+":"
            +(cal.getTime().toString()+":"+getLocalName()+":"+message);
    addBehaviour(BaseAgentProtokollNachrichtProposeInitiator.addBehaviour(
            this,nachricht));
}
```

Abbildung 6-25: Klasse BaseAgent

6.4 Bewertung der Realisierung

Der hier vorgestellte Prototyp konnte die Funktionalität der technischen Architektur der Simulationsumgebung, die das Abbild der im Kapitel 5.2 aufgestellten fachlichen Architektur im Kontext der JADE-Lauzeitumgebung realisiert, bestätigten. Es wurde erreicht gesichert, dass alle aufgestellten funktionalen und nicht funktionalen Anforderungen Berücksichtigung finden. Die logische Zusammenfassung der realisierten Funktionalitäten zu JADE-Instanzen erhöht die Modularität und ermöglicht eine optimale physikalische Verteilung der Simulationsumgebung. Die

physikalische Verteilung wird den sinnvollen Einsatz der Simulationsumgebung bei komplexen und rechenintensiven Simulationsszenarien zulassen.

Eine wichtige Entscheidung für den Einsatz des FIPA-Nachrichtenaustausches als Schnittstellentechnologie in Verbindung mit einer semantischen Beschreibung von Nachrichten zwischen Agenten mittels anwendungsspezifischer Ontologie konnte während der Implementierung des Prototyps bestätigt und erfolgreich umgesetzt. Dadurch wird die Komplexität der einzelnen Komponenten verborgen und die Erweiterbarkeit des Systems für weitere Entwicklungen garantiert.

In der Komponente „Fluggastverhalten“ wurden das Software-Agentenparadigma für die Simulierung menschlichen Verhaltens in komplexen Gebäuden und insbesondere die JADE-Entwicklungsplattform für Software-Agenten erfolgreich eingesetzt. In allen weiteren Komponenten der Simulationsumgebung, bei denen keine explizite Software-Agenten-Funktionalität gefordert ist, konnte der Übergang zwischen dem Software-Agentenparadigma und der objektorientierten Programmierung mittels zusätzlicher Software-Agenten geschaffen werden.

Für eine erleichterte Implementierung und Analyse des Prototyps wurden ein vereinfachtes Gebäude- und ein reduziertes Dienstmodell (s. Kapitel 6.3.2) sowie ein manuelles, statisches Verfahren zur Routengenerierung aus Gebäudemodelldaten entwickelt. Da keine echten Flughafengebäudestrukturen abgebildet werden, begrenzt das zwar die Einsatzmöglichkeiten des Prototyps. Jedoch ermöglicht die modulare Systemarchitektur und die entwickelten Schnittstellen bei einer Weiterentwicklung des Systems einfache Realisierung bzw. Anbindung von verbesserter Abbildungsverfahren oder Produkte (s. Kapitel 3.1.2) zum Modellieren von Gebäudestrukturen, Nutzungsplänen und Routen, ohne Beeinflussung anderer Systemkomponenten.

Um die simulierten Aktivitäten jederzeit formal rekonstruieren und nachprüfen zu können, werden die Daten des verteilen Simulationsprozesses durch die zentrale Komponente „Protokollierung“ dauerhaft gespeichert. Zusätzlich stellt der Prototyp den Simulationsprozess mit Hilfe der dazu entwickelten GUI-Oberfläche (s. Abbildungen 6-12 und 6-13) grafisch dar. Es werden die Gebäudestruktur, die aktuelle Position des Fluggastes, Dienste sowie Routen abgebildet. Die Oberfläche erleichtert dem Benutzer mit ihrer Zoom-Funktion das Abbilden von umfangreichen Räumlichkeiten. Ob die Oberfläche reale komplexe Flughafengebäudestrukturen sinnvoll darstellen kann, konnte auf Grund der eingesetzten vereinfachten Gebäude- und Routenmodelle nicht getestet

werden. Daher sind an dieser Stelle für den realen Einsatz der Simulationsumgebung Optimierungen zu erwarten.

Die zu Grunde liegende JADE-Laufzeitumgebung bietet, wie bereits erwähnt, die Möglichkeit, neben dem FIPA-Nachrichtenaustausch auch weitere Schnittstellentechnologien bzw. Standards zu realisieren und dadurch externe Systeme (wie z.B. das Indoor-Fußgängernavigationssystem) einfach anzubinden. Das erleichterte bei der Implementierung des Prototyps den Umstand zu berücksichtigen, dass jedes neu anzubindende Navigationssystem andere Schnittstellen zur Simulationsumgebung benötigt. Die Anbindung der Navigationssysteme und die damit verbundenen künftigen Änderungen konnten erfolgreich auf die JADE-Instanz „Navigator“ begrenzt werden. Auf diese Weise ist die Simulationsumgebung vom Indoor-Fußgängernavigationssystem unabhängig und kann für den Test mehrerer Navigationssysteme mit unterschiedlichen Schnittstellen leicht erweitert werden.

Im Anschluss an die Implementierung des Prototyps wurden die Tragfähigkeit der technischen Architektur und die fachliche Richtigkeit der aufgestellten Simulationsprozesse an einem Beispielszenario erfolgreich getestet. Dafür wurde ein exemplarisches Flughafenmodell (s. Abbildung 6-12 und 6-13) aufgebaut und die definierten Anwendungsfälle (s. Kapitel 4.2) der Simulationsumgebung erprobt.

6.5 Fazit

In diesem Kapitel wurde eine konkrete Realisierung der Simulationsumgebung für Indoornavigation durch die Implementierung eines Prototyps vorgestellt. Der Prototyp erfüllt alle in Kapitel 4 definierten Systemanforderungen, er wurde auf Basis der definierten Systemarchitektur (s. Kapitel 5.2) entworfen und anhand eines Beispielszenarios erprobt. Die Bewertung der Ergebnisse hat gezeigt, dass die Realisierung einer Simulationsumgebung für Indoornavigation mit den geforderten Eigenschaften und durch die zu Grunde liegende JADE-Laufzeitumgebung innerhalb des Software-Agentenparadigmas grundsätzlich möglich ist, doch muss der Prototyp für einen realen Einsatz der Simulationsumgebung im Bereich Gebäude- bzw. Raum-Modellierung optimiert werden.

7. Zusammenfassung und Ausblick

7.1 Zusammenfassung

Das Ziel dieser Masterarbeit war die Aufstellung eines Analysemodells zur Erstellung einer Simulationsumgebung im Rahmen des Software-Agentenparadigmas für Indoornavigation im Flughafenkontext. Die Simulationsumgebung sollte die Entwicklung und den Test von Indoor-Fußgängernavigationssystemen erleichtern, indem für diese Systeme eine virtuelle Teststrecke (z.B. ein virtueller Flughafen) mit dem zugehörigen Kontext (Gebäudestruktur, Nutzungsplan, mögliche Wege durch die Gebäudestruktur usw.) und das typische Verhalten der Testpersonen (hier der Fluggäste) innerhalb der Simulationsumgebung abgebildet werden.

Das Ziel der Masterarbeit wurde erreicht. Im ersten Schritt verallgemeinert die Arbeit den Schwerpunkt der Simulationsumgebung für Indoornavigation im Flughafenkontext und definiert ihn als „Fußgängersimulation in Gebäuden“ mit zwei ausgeprägten Aspekten: das typische Fußgängerverhalten und die Umgebung, in der sich der Fußgänger befindet. Mit Hilfe zweier Verfahren zum Abbilden von Gebäudestruktur und mehrerer Verfahren zur Simulation von Fußgängerverhalten baut die Masterarbeit eine methodische Grundlage zur Umsetzung der definierten Aspekte auf. Dabei wird die Software-Agententechnologie als eine denkbare Umsetzung der Simulationsumgebung etabliert. Als Nächstes wurde der konkrete Einsatz von Indoor-Fußgängernavigationssystemen innerhalb Flughafengeländes analysiert, woraufhin die Anforderungen an eine Simulationsumgebung gefolgert und definiert wurden, auf deren Grundlage schließlich ein Fachkonzept mit einer tragfähigen Architektur der künftigen Simulationsumgebung erarbeitet wurde. Die Auswahl zwischen mehreren näher vorgestellten Entwicklungsplattformen fiel auf die JADE-Technologie, deren Charakteristiken den genannten Bedingungen zur Realisierung der Simulationsumgebung im Rahmen des Software-Agentenparadigma am besten entsprechen. Zum Schluss wurde die Realisierung der Simulationsumgebung für Indoornavigation durch die Implementierung eines Prototyps vorgestellt. Die Funktionalität der Simulationsumgebung wird auf mehrere JADE-Instanzen logisch und physisch verteilt. Die Kommunikation zwischen den JADE-Instanzen geschieht mit Hilfe des FIPA-Nachrichtenaustausches in Verbindung mit einer anwendungsspezifischen Ontologie. Damit wird zunächst die Komplexität der einzelnen Komponenten verborgen und die Erweiterbarkeit des Systems für weitere Entwicklungen garantiert. Der Prototyp erfüllt alle definierten Systemanforderungen und zeigt damit, dass die Realisierung

einer Simulationsumgebung für Indoornavigation mit den geforderten Eigenschaften und durch die zu Grunde liegende JADE-Laufzeitumgebung möglich ist.

7.2 Ausblick

Sozio-technische Systeme gewinnen in der Gegenwart immer stärker an Bedeutung. Um komplexe Aktivitäten effizient abarbeiten zu können, ist der Mensch in seinem privaten und beruflichen Alltag immer mehr auf die aktive Mitarbeit als Teils eines umfangreichen, technischen Gesamtsystems angewiesen. Die Tatsache stellt laufen neue Anforderungen an die Entwicklung und Test derartiger Systeme: um deren Funktionalität in geeigneter Weise entwickeln und sichern zu können, muss im Vorfeld der soziale Faktor , das menschliche Verhalten, durch optimale Simulation abgebildet werden. In dieser Masterarbeit wurden dafür mehrere Lösungsansätze beschrieben und tiefergehend untersucht, indem eine Simulationsumgebung für Indoornavigation innerhalb des Software-Agentenparadigmas entwickelt wurde. Der Schwerpunkt lag dabei auf der Verhaltenssimulation einzelner Personen.

Wird das Thema dieser Masterarbeit unter dem Gesichtspunkt „kollaboratives Arbeiten“ betrachtet, kann das ihr zu Grunde liegende Modell menschlichen Verhaltens erweitert werden, wenn impliziert wird, dass in realistischen Szenarien der Mensch einer Gruppe zugehört, die das entsprechende sozio-technische System nutzt. Aus dieser Sicht können nicht nur das Verhalten eines einzelnen Individuums, sondern auch, bei angemessener Betrachtung zusätzlicher Bedingungen, das Gruppenverhalten bzw. die daraus entstehenden Gruppenprozesse simuliert werden. Damit lassen sich in der Perspektive die Entwicklung und der Test von sozio-technischen Systemen auch für Gruppenverhalten erweitern und auf aktuelle Szenarien wie Stauvermeidung oder Personenevakuierung innerhalb von Gebäuden anwenden.

Literaturverzeichnis

[Aslan06] I. Aslan, M. Schwalm, “Acquisition of spatial knowledge in location aware mobile pedestrian navigation systems” Mobile HCI, 2006

[Balfanz05] D. Balfanz, M. Etz, “TRAFFIC AND MOBILITY“, INI-GraphicsNet, 1. Auflage, Darmstadt, 2005

[Barrett01] C. Barrett, S. Eubank, “Science & engineering of large scale socio-technical simulations”, Los Alamos National Laboratory Report, LA-UR-01-6623, 2001

[Bellifemine07] F. Bellifemine, G. Caire, „Developing Multi-Agent System with Jade”, John Willey & Sons Ltd., Chichester, West Sussex, England, 2007, ISBN 978-0-470-05747-6

[Bidwell05] N. Bidwell, C. Lueg, “The territory is the map: designing navigational aids”. The 6th ACM SIGCHI New Zealand chapter’s international conference on Computer-human interaction, New York, NY, USA, 2005.

[Brooks91] R. Brooks, “Intelligence without Representation”, Artificial Intelligence, 47, 1991

[Burkhard98] H. Burkhard, „Einführung in die Agententechnologie“, in Informationstechnik und Technische Informatik, 1998

[Drexl03] T. Drexl, „Entwicklung intelligenter Pfadsuchsysteme für Architekturmodelle am Beispiel eines Kiosksystems (Info-Point) für die FMI in Garching“, Diplomarbeit, Institut für Informatik, Technische Universität München, 2003

[Ducatel01] K. Ducatel, M. Bogdanowizc, „Scenarios for Ambient Intelligence in 2010 “, IPTS-Seville, 2001

[Dustar03] S. Dustar, H. Gall, „Softwarearchitekturen für verteilte Systeme: Prinzipien, Bausteine und Standardarchitekturen für moderne Software“, Springer-Verlag, Berlin, Heidelberg, 2003, ISBN 3-540-43088-1

[Ehm04] M. Ehm, J.n Linxweiler, „Berechnung von Evakuierungszeiten bei Sonderbauten mit dem Programm buildingExodus“, Studienarbeit an der Technischen Universität Braunschweig Institut für Baustoffe, Massivbau und Brandschutz Prof. Dr.-Ing. D. Hosser, 2004

[Gabaglio03] V. Gabaglio, “GPS/INS Integration for Pedestrian Navigation“, Geodätisch-geophysikalische Arbeiten in der Schweiz, Band 64, Zürich, 2003, ISBN 3-908440-07-6

[Gamma96] E. Gamma, R. Helm, „Entwurfsmuster: Elemente wieder verwendbarer objektorientierter Software“, 1. Auflage, Addison Wesley Verlag, München, 1996, ISBN 3-8273-1862-9

[Giesecke04] S. Giesecke, M. Stier, and S. Grumbein. „Pfadsuche in Architekturmodellen und Stereoprojection“, Softwarepraktikum, Universität Stuttgart, 2004

[Gipps85] P. Gipps, B. Marksjo, „A Micro-Simulation Model for Pedestrian Flows“, Mathematics and Computers in Simulation, 1985

[Gipps87] P.. Gipps, „Simulation of pedestrian traffic in buildings“, Band 35, Institut für Verkehrswesen, Universität Karlsruhe, Karlsruhe, 1987

[Gregor06] S. Gregor, „Entwicklung einer Hardwareplattform für die Ermittlung von Positionsdaten innerhalb von Gebäuden“ (Bachelorarbeit), Haw-Hamburg, 2006

[Hanisch03] A. Hanisch, J. Tolujew, K. Richter, T. Schulze, “Online Simulation of Pedestrian Flow in Public Buildings”, Winter Simulation Conference, 2003

[Helbing90] D. Helbing, „Physikalische Modellierung des dynamischen Verhaltens von Fußgängern“, Diplomarbeit, Georg-August Universität, Göttingen, 1990

[Helbing96] D. Helbing, „Stochastische Methoden, nichtlineare Dynamik und quantative Modelle sozialer Prozesse“, Doktorarbeit, Universität Stuttgart, 1992, ISBN 978-3861115472

[Henein04] C. Henein, T. White, „Agent-based modelling of forces in crowds.”, Lecture Notes in Computer Science, Band 3415, Springer, 2004.

[Huber99] M. Huber, “A BDI-Theoretic Mobile Agent Architecture”, Proceedings of the Third Annual Conference on Autonomous Agents, ACM Press, 1999

[Jennings95] N. Jennings, M. J. Woolridge, “Intelligent Agents: Theory and Practice”, Knowledge Engineering Review 10 (2). Cambridge University Press, USA, 1995

[Jurgeit03] J.Florian, “Rauminformationssystem mit PostgreSQL und SVG”, Diplomarbeit, Universität Innsbruck, 2003

[Kahlbrandt98] B. Kahlbrandt, „Software-Engineering“, Springer-Verlag, Berlin, Heidelberg, 1998, ISBN 3-540-63309-x

[Klügl03] F. Klügl, R. Herrler, “From Simulated to Real Environments: How to use SeSAm for software development. Accepted at the First German Conference on Multiagent System Technologies”, 2003.

[Koychev07] M. Koychev, „Projektbericht“, Haw-Hamburg, 2007

[Kutak07] E. Kutak, „Projektbericht“, Haw-Hamburg, 2007

[Laugier05] C. Laugier, S. Petti, “Steps Towards Safe Navigation in Open and Dynamic Environments”, ICRA-2005 Workshop on Cooperative Robotics, Barcelona, 2005

[Lee05] S. M. Lee, U. Ravinder, “Developing an agent model of human performance in air traffic control operations using apex cognitive architecture”, NASA Amnes Research Center, Report, Moffet Field, CA, USA, 2005

[Linstaedt06] S. Linstaedt, “Integration von Agentenplattformen in Middleware – am Beispiel von Jadex und Java EE”, Diplomarbeit, Universität Hamburg, 2006

[Little07] R. Little, T. A. Birkland, “Socio-Technological Systems Integration to Support Tsunami Warning and Evacuation”, 40th Hawaii International Conference on system Sciences, 2007, IEEE 1530-1605/07

[Lovas94] G. Lovas, “Modeling and Simulation of Pedestrian Traffic Flow” Transportation Research, 1994

[May89] A. May, “Traffic Flow Fundamentals”, Prentice Hall, New Jersey, USA, 1989, ISBN 978-01-3926-072-8

[May03] A. May, T. Ross, "Pedestrian navigation aids: information requirementsand design implications", Ubiquit Comput, Springer-Verlag, London, 2003

[Meng05] L. Meng, Alexander Zipf, „Map-based Mobile Services: Theories, Methods and Implementations”, Springer-Verlag, Berlin, New York, 2005, ISBN 3-540-23055-6

[Mengistu07] D. Mengistu , “Multi-Agent Based Simulations in the Grid Environment”, Printfabriken, Karlskrona, 2007 ISBN 978-91-7295-118-1

[Napitupulu07] J. Napitupulu, „Projektbericht“, Haw-Hamburg, 2007

[Narasimhan07] S. Narasimhan, „Simulation and Optimized Scheduling of Pedestrian Traffic“, Doktorarbeit, Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart, 2007

[Noy01] N. Noy, D. McGuinness, “Ontology Development 101: A Guide to Creating Your First Ontology”, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, 2001.

[Odell01] J. Odell, H. Parunak, "Representing Agent Interaction Protocols in UML" In: Proceedings of the First International Workshop on Agent-oriented Software Engineering (AOSE2000). Springer-Verlag, Berlin, 2001.

[Okazaki93] S. Okazaki, S. Matsushita, „A study of simulation model for pedestrian movement with evacuation and queuing“, Conf. Engineering for Crowd Safety, 1993

[Pokahr07] A. Pokahr, L. Braubach, „Jadex – user guide“, Version . 0.96, Universität Hamburg, 2007

[Schumann07] A. Schumann, „Projektbericht“, Haw-Hamburg, 2007

[Schwinger05] W. Schwinger, Ch. Grün, “Context-awareness in Mobile Tourism Guides – A Comprehensive Survey”, Johannes Kepler Universität Linz, Technische Universität Wien, 2005

[Singh03] M. Singh, “Distributed Enactment of Multiagent Workflows: Temporal Logic for Web Service Composition,” Conf. Autonomous Agents and Multiagent Systems, ACM Press, 2003

[Vidal04] J. Vidal, P. Buhler,"Multiagent Systems with Workflows", IEEE, Februar 2004

[Wood01] M. Wood, S. DeLoach, "An Overview of the Multiagent System Engineering Methodology”, In: Proceedings of the First International Workshop on Agent-oriented Software Engineering. Springer-Verlag, Berlin, 2001

[Wooldridge00] M. Wooldridge, N. Jennings, "The Gaia Methodology for Agent-Oriented Analysis and Design”, In: Journal of Autonomous Agents and Multi-Agent Systems, 3(3):285–312. 2000

[Zipf04] A. Zipf, M. Jöst, "Implementing Adaptive Mobile GI Services based on Ontologies Examples from pedestrian navigation support", GIScience 2006 – the Fourth International Conference on Geographic Information Science, Münster, 2006

[cordis.europa.eu] Das sechste Rahmenprogramm der Europäischen Union, http://cordis.europa.eu/fp6/dc/index.cfm , Stand: November 2007

[fipa.org] Foundation for Intelligent Physical Agents, http://www.fipa.org/index.html , Stand: November 2007

[fseg.gre.ac.uk] http://fseg.gre.ac.uk/exodus/index.html, Stand: Januar 2008

[ist-highway.org] HIGHWAY – Intelligente Fahrzeug-Navigation, http://www.ist-highway.org , Stand: November 2007

[stanford.edu] Ontology-Editor, http://protege.stanford.edu/overview/protege-owl.html Stand: April 2008

[simsesam.de] Shell for Simulated Agent Systems, http://www.simsesam.de/, Stand: November 2007

[uni-stuttgart.de] Raumsuche mit FIFI-Infoterminal, http://infoterminals.informatik.uni-stuttgart.de:8090/FIFI-1.2/, Stand: Januar 2008

[wearitatwork.com] WEARIT@WORK - Der integrierte mobile Assistent, http://www.wearitatwork.com , Stand: November 2007

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24.06.2008

Ort, Datum Unterschrift: Milen Koychev