Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Florian J. Ocker

Individuen-orientierte Simulationen eingebettet
in High Level Architecture

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian J. Ocker

Individuen-orientierte Simulationen eingebettet in High Level Architecture

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thomas Thiel-Clemen
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 22.08.2011

Florian J. Ocker

Thema der Bachelorarbeit

Individuen-orientierte Simulationen eingebettet in High Level Architecture

Stichworte

Verteilte Simulation, High Level Architecture, Multiagentensysteme

Kurzzusammenfassung

Diese Arbeit beschreibt die Entwicklung einer verteilten Simulation, welche die High Level Architecture als Kommunikations- und Synchronisationskomponente nutzt. Der Simulationsfokus liegt dabei auf verschiedene Formen von Individuen und ihrem Zusammenspiel in gemeinsam genutzten Raum. Zu diesem Zweck wird ein Framework entwickelt, in welches Simulationen eingebettet werden können, um an der verteilten Simulation teilnehmen zu können. Das Framework kapselt dabei die Synchronisationsroutinen und ermöglicht unter Rücksichtnahme der vorgegebenen Interfaces die Entwicklung unterschiedlichster Agenten. Es wird eine exemplarische Simulation entworfen, in dem Schiffe in einem Hafengebiet miteinander interagieren. Die unterschiedlichen Typen von Individuen werden dabei jeweils von einer Simulation umgesetzt. Das Zusammenspiel ergibt sich somit aus der verteilten Gesamtsimulation.

Florian J. Ocker

Title of the paper

Agent-based simulations embedded in High Level Architecture

Keywords

Distributed Simulation, High Level Architecture, Multi-Agent-Systems

Abstract

This thesis describes the development of a prototype that models a distributed simulation supported by the High Level Architecture. The focus of simulation is based on different types of agents that share time and space. For this purpose a framework is modeled, which connects different simulations by encapsulating the synchronization process. Furthermore the framework defines interfaces that have to be implemented by the participating simulations. Anyhow a clear design template is provided within which several cognitive architectures and agent designs can be implemented. Finally a distributed simulation modeling vessels in a harbor is implemented to proof the functionality of the framework. Each simulation is concerning just one type of agent; hence interaction results by linking simulations together.

Inhaltsverzeichnis

1 Einleitung

Simulationen bilden eine definierte Situation und die in ihr vorkommenden Vorgänge ab. So lassen sich mit Hilfe von Computersimulationen komplexe Zusammenhänge in begrenzten Systemen darstellen. Sowohl die Möglichkeit, ein solches Experiment in beliebiger Häufigkeit zu wiederholen, als auch die Gestaltungsfreiheit machen Simulationen zu einem unverzichtbaren Werkzeug in einer Vielzahl von Bereichen, sei es Forschung oder Wirtschaft.

Es gibt vielfältige Möglichkeiten das Problem realitätsnah abzubilden und verwertbare Erkenntnisse aus Simulationen zu extrahieren. Dennoch gibt es Grenzen bezüglich der Machbarkeit im Allgemeinen und den Entwicklungskosten im Speziellen. Mit fortschreitender Entwicklung der Computertechnik und Rechenleistung wachsen die Gestaltungsmöglichkeiten und es können komplexere Problemstellungen untersucht werden.

Bei vielen Simulationen wird die Interaktion verschiedener Entitäten simuliert, welche häufig in der Form eines Agenten auftreten und somit ein unabhängiges Verhalten besitzen. Diese Simulationen sind im Grunde eine Zusammensetzung von vielen Sub-Simulationen. Auf softwaretechnischer Ebene stellt sich daher die Frage, in welcher Form diese Simulationen miteinander kommunizieren und interagieren.

In dieser Ausarbeitung wird der Einsatz von High Level Architecture (HLA) als eine mögliche Form beleuchtet und evaluiert. Dabei werden die Simulationen in die HLA eingebettet, welche den Agenten zeitnah Informationen über die dynamische Umgebung und die Aktionen der teilnehmenden Individuen bereitstellt. Die HLA fungiert also als Kommunikationssystem der Gesamtsimulation.

1.1 Motivation

Szenarien, die das Verhalten und Zusammenspiel von Individuen untersuchen, gewinnen in vielen Bereichen an Bedeutung, so z.B. im Katastrophenschutz. Mögliche Vorgänge können im Vorfeld untersucht und trainiert werden, um im Ernstfall die bestmöglichen Entscheidungen treffen zu können. Das komplexe Verhalten von Individuen in ein Interaktionsverhältnis zu setzen, schafft eine sehr anspruchsvolle Simulationsumgebung. Die gewonnen Erkenntnisse aus der

resultierenden Analyse können sehr wertvoll sein. Dem gegenüber stehen die mit dem Aufwand steigenden Entwicklungskosten. Durch die Möglichkeit, bereits bestehende Simulationen in die Gesamtsimulation einzubinden, lässt sich das Modell mit überschaubarem Aufwand effektiv erweitern. Zudem kann die Simulation durch die HLA auf verschiedene Rechner aufgeteilt werden, wodurch eine höhere Performance erzielt werden kann. Diese ermöglicht es noch komplexere Modelle zu untersuchen. Die HLA scheint ein sehr vielversprechender Ansatz zu sein, der den Ansprüchen an verteilte Individuen-orientierte Simulationen gerecht werden kann.

1.2 Problemstellung

Die Verknüpfung von verschiedenen Simulationen erfordert die Festlegung einer gemeinsamen Schnittstelle, die von allen Simulationen bedient und verstanden werden kann. Die so entstandene Kompatibilität ermöglicht zwar eine Interaktion zwischen den Teilnehmern, jedoch muss ihnen auch noch eine gemeinsame Plattform (in Form einer Landschaft und eines Zeitstrahls) bereitgestellt werden, damit die Agenten in dieser Welt[1] agieren können. Um den Individuen valide Daten zur Verfügung stellen zu können, muss die HLA ein globales Zeitmanagement vollführen und somit den Synchronisationstakt der globalen Simulation vorgeben.

1.3 Zielsetzung

Es soll eine Architektur geschaffen werden, welche es auf einfache Art und Weise ermöglicht, weitere Subsimulationen in Form von Agenten in die bestehende Simulation zu integrieren. In diesem Zusammenhang soll auch evaluiert werden, inwiefern sich die HLA als Kommunikationsschnittstelle eignet. Die an dieser Stelle durchgeführte Machbarkeitsstudie wird durch die Erstellung eines Prototyps begleitet. Die gesammelten Ergebnisse werden durch den Prototyp veranschaulicht und bilden die Grundlage zusammenfassender Aussagen.

1.4 Inhaltlicher Aufbau

Im ersten Kapitel wird die HLA erläutert, welche als methodische Grundlage des Simulationsverbundes dient. In diesem Zusammenhang werden auch die freie Umsetzung *Portico*, welche im Prototyp zum Einsatz kommt, und ihre speziellen Eigenheiten beleuchtet. Zu den Grundlagen zählen weiterhin die verschiedenen Formen der Simulation. So werden spezielle Simulationsansätze beleuchtet und schließlich der Zusammenhang zu den verteilten Simulation hergestellt.

[1] Simulationswelt

Eine umfassende Anforderungsanalyse wird im zweiten Kapitel durchgeführt. Dabei werden verschiedene Problemstellungen untersucht und mögliche Lösungsansätze für eben diese formuliert.

Das dritte Kapitel stellt das Konzept des Prototyps vor, welches ein problemspezifisches Design beinhaltet, das die in der Analyse gewonnen Erkenntnisse berücksichtigt. Zudem wird im dritten Kapitel der Aufbau der einzelnen Agenten beschrieben, die als Bausteine für die Gesamtsimulation dienen und ein exemplarisches Verhalten umsetzten, welches aber im Rahmen der Kommunikationsschnittstelle beliebig komplex sein kann.

Im vierten Kapitel wird die tatsächliche Umsetzung und Implementierung der hier vorgestellten Arbeit behandelt. Die Architektur wird dabei unterstützt von UML-Diagrammen dargestellt. Auch Kommunikationsprotokolle und weitere anwendungsspezifische Entscheidungen werden erläutert.

Schließlich werden die gezogenen Erkenntnisse im fünften Kapitel dargelegt und ausgewertet.

Im letzten Kapitel wird dann ein zusammenfassendes Fazit gezogen. Dieses bewertet die gewählten Methoden und die Umsetzung des Projekts. Schlussendlich werden noch Verbesserungspotentiale aufgezeigt, die als Ausblick für Nachfolgeprojekte dienen können.

2 Methodische Grundlagen

Im folgenden Kapitel werden die Technologien und Methoden dargelegt, welche als Grundlagen des Prototyps dienen. Zunächst wird die Vorgehensweise der Simulation erläutert und ihre steigende Relevanz in der heutigen Welt herausgestellt. Die daraus resultierenden Schlussfolgerungen münden in dem Wunsch nach verteilten Simulationen, welche nicht nur die Simulationen an sich wirtschaftlicher machen, sondern auch ganze neue Möglichkeiten eröffnen. Diese erschöpfen sich nicht nur in größerer Flexibilität im Entwicklungsprozess sondern ermöglichen auch die Umsetzung komplexerer Simulationsmodelle.

2.1 Simulationen

„*Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output*“(Fishwick, 1997).

Diese von *Fishwick* verfasste Definition gibt einen ersten Einblick in die Welt der Simulationen, dessen Begriff in dieser Arbeit immer Computersimulationen meint. Zunächst soll die Zweckdienlichkeit von Simulationen geklärt werden. Laut Definition untersuchen Simulationen konkrete weltliche oder auch theoretische Systeme, um gewisse Informationen zu gewinnen. Es muss diesbezüglich ergänzt werden, dass diese Systeme von dynamischer Natur sind. Es werden also bestimmte Vorgänge oder Zustände untersucht, welche im System auftreten. Die Analyse solcher Systeme ist meist sehr komplex und kann deshalb nur bedingt durch theoretische Ansätze untersucht werden. Simulationen hingegen bieten durch ihre experimentelle Vorgehensweise einen bewährten Ansatz. Die Definition macht deutlich dass es sich bei einem simulierten System nicht zwangsläufig um eine Abbildung der echten Welt handeln muss. Vielmehr werden bestimmte Phänomene untersucht. Die erste Aufgabe der Simulation ist es also die Eigenschaften, welche für die Untersuchung von Relevanz sind, aus dem Originalsystem zu abstrahieren. Der Begriff *Source*

System wird in diesem Zusammenhang gebraucht, um das zu simulierende System, welches echt oder virtuell sein kann, eindeutig von der erschaffenen Welt abzugrenzen (Uhrmacher et al., 2009). Die Charakteristika des *Source Systems* werden dann in einem Modell zusammengefasst. Dieses Modell muss so gestaltet werden, dass Aussagen über die zu lösende Problemstellung getroffen werden können. Anschließend wird das Modell in ausführbare Strukturen umgesetzt. Die Durchführung einer Simulation mit konkreten Parametern nennt man *Simulationsexperiment*. Abschließend werden die Simulationsergebnisse gemäß bestimmter Regeln überprüft. Die darauf aufbauende Interpretation soll Rückschlüsse auf die Problemstellung ermöglichen.

Die Aufgabe und die Erstellung einer Simulation sind hiermit kurz umrissen. Im weiteren Verlauf ist es jedoch von Bedeutung, dass die Begrifflichkeit klar definiert wird. Bestimmte Begriffe sind mit verschiedenen Bedeutungen verknüpft. Beispielsweise wird das Wort *System* sowohl in der Problemstellung als auch in der Umsetzung verwendet, wo jedoch durch den Begriff *Source System* abgegrenzt werden kann. Teilweise wird es sogar für den Begriff Simulation benutzt, welcher ebenfalls überladen ist. Die begriffliche Verwendung des Wortes Simulation ist deshalb schwierig, weil sie viele Bedeutungen umfasst. Werden beispielsweise Agenten simuliert, so kann einerseits jeder Agent an sich als Simulation angesehen werden, andererseits bildet die Gesamtheit der Agenten eine übergeordnete Simulation, die einen größeren Kontext beschreibt. Deshalb ist es im Folgenden notwendig, die Begrifflichkeit Simulation immer in einen Kontext zu setzen und die Bedeutung klar herauszustellen (Spath, 2009).

2.1.1 Discrete Event Simulation

Individuenorientierte Simulationen, später auch als Multi-Agenten-Simulationen bezeichnet, haben besondere Anforderungen bezüglich der Robustheit, Wiederverwendbarkeit und Analysierbarkeit auf der Detailebene. Diese Ansprüche können in den meisten Fällen sinnvoll durch ein diskretes System modelliert werden. Diskrete Simulationen werden dadurch charakterisiert, dass sie eine zählbare Anzahl von Zuständen haben, die mit der Zeit durchlaufen werden. Nach Zeitintervallen, die je nach Simulationstyp bemessen sind, treten Ereignisse (Events) auf, die wiederum den nächsten Systemzustand erzeugen. Weiterhin können durch Ereignisse auch Folgeereignisse erzeugt werden, die dann wiederum zu einem bestimmten Zeitpunkt auftreten. Zu dieser Gruppe von Simulationen zählen auch die zeitgesteuerten Simulation (*Time-Stepped Simulations)*, welche in einer festen Zeittaktung voranschreiten. Diese Simulationsform ist weit verbreitet, da sie häufig zur Simulation von Echtzeitsystemen Anwendung findet.

Im Folgenden soll aber auf ein spezielleres Paradigma eingegangen werden, nämlich die *Discrete Event Simulation* (DES), welches Zustandsänderungen erst

dann ausführt, wenn sie von semantischer Bedeutung sind. Die Simulationszeit kann also Sprünge machen, da sie nicht linear fortschreitet, sondern sprunghaft zum nächsten Ausführungszeitpunkt eines Ereignisses wechselt (Fischer, 1997). Der DES-Ansatz ist demnach um einiges flexibler, da in einem Modell gleichzeitig Phänomene mit stark variierenden Ereigniszyklen untersucht werden können. Das kann am besten verdeutlicht werden, wenn man sich solche Phänomene in einer Time-Stepped Simulation vorstellt. Angenommen man untersucht im gleichen Modell ein Phänomen, in dem Events in einem Abstand von einer Stunde auftreten, und einen zweiten Handlungsstrang, bei dem die Ereignisse sekündlich von Bedeutung sind. In einem Echtzeitsystem wäre es schwierig eine Skalierung zu finden, die es ermöglicht, beide Probleme angemessen und unter Berücksichtigung einer adäquaten Ausführungsdauer zu untersuchen. In einer DES würde die Problematik der Skalierung durch Zeitsprünge zum jeweils nächsten Ereignis wegfallen. Und dennoch ist die DES so gar im Stande die Time-Stepped Simulation zu simulieren. Dies kann durch die Anordnung von Ereignissen in einer linear zeitverketteten Folge erreicht werden. So viel zu den Stärken von der DES, doch wie sind solche Simulationen strukturiert?

Um ein genaueres Verständnis zu erreichen, wird das Grundprinzip der DES nachfolgend kurz umrissen. Das Hauptprinzip besteht darin, eine Kette (Queue) von Ereignissen, die jeweils mit einem Zeitstempel versehen sind, zu verwalten. Der Zeitstempel indiziert den Ausführungszeitpunkt des Ereignisses. Zyklisch wird das Element mit dem kleinsten Zeitstempel zur Ausführung gebracht. Dies kann wiederum neue Ereignisse zur Folge haben, die wieder gemäß ihrem Zeitstempel in die Queue eingeordnet werden. Dadurch, dass Ereignisse prinzipiell immer wieder neue Ereignisse erzeugen können, kann die Simulation unter Umständen endlos laufen. Das Ende der Simulation muss also durch weitere Bedingungen spezifiziert werden. Dies kann durch das Erreichen einer definierten Zeit (*t)* erfolgen oder durch die Ausführung einer bestimmten Anzahl von Ereignissen (*n*). Eine komplexere Möglichkeit ist es, die Simulationsdauer an das Erreichen eines bestimmten Wertes für eine statistisch auswertbare Größe zu koppeln (Uhrmacher et al., 2009).

Es gibt also eine Vielzahl unterschiedlicher Paradigmen für die Umsetzung von individuenorientierten Simulationen. Welche Form gewählt wird, hängt stark von den Zielen ab, die man erreichen will. Es gibt für die beiden hier vorgestellten Ansätze jeweils bekannte Frameworks, auf die später noch weiter eingegangen wird.

Zusammenfassend lässt sich sagen, dass die Bedeutung von Simulationen in der heutigen Zeit nicht mehr bestritten werden kann. Die Vorteile reichen von Zeit- und Kosteneinsparungen bis hin zur Risikominimierung in den zu analysierenden Prozessen. Daher steigt der Bedarf nach immer leistungsfähigeren und vielfältigeren Modellierungs- und Simulationswerkzeugen stetig (Schulz, 2005).

2.1.2 Verteilte Simulationen

Es gibt kein Simulationsparadigma, das alleine alle Anforderungen, die an Simulationssysteme gestellt werden, erfüllen kann (Buchholz, 2009). Die Anforderungen und der Bedarf an Simulationen steigen jedoch stetig. Zudem existiert schon eine Vielzahl komplexer Simulationssysteme. Durch die Verknüpfung verschiedener Simulationen zu einer Verteilten Simulation kann man diesen Anforderungen genügen. Die Simulationen können dadurch nicht nur komplexerer werden, sondern es bestehen auch mehr Möglichkeiten ausgewählte Simulationen wiederzuverwenden. Wiederverwendbarkeit nimmt eine gewichtige Rolle ein, da es nicht nur Kosten, sondern auch Entwicklungszeit einspart. Außerdem können komplizierte, in sich abgeschlossene Simulationen über definierte Schnittstellen erweitert werden, während die innere Abgeschlossenheit unangetastet bleiben kann.

Durch das Aufteilen der Simulation in Module (*Distributed Model*, Klein et al., 1998b) wird die Entwicklung stark vereinfacht. Der erste Ansatz besteht darin, die Submodelle in geographische Gebiete aufzuteilen, für die die jeweiligen Simulationen verantwortlich sind. Der zweite Reduktionsansatz ist die funktionale Aufteilung. Die Subsimulationen haben wenige definierte Aufgaben, die sie umsetzten müssen. Dies erleichtert die Modellfindung und Entwicklung und ermöglicht größere Freiheiten in der Umsetzung. Zudem kann somit schneller auf sich verändernde Anforderungen eingegangen werden, da wiederum die Möglichkeit besteht, einzelne Module wiederzuverwenden. Bei der Entwicklung können sowohl unterschiedliche Programmiersprachen, als auch Plattformen (Betriebssysteme) verwendet werden. Weiterhin ist ein unterschiedliches Zeitverhalten (Time Management) möglich. Außerdem kann die Entwicklung unter Berücksichtigung des gemeinsamen Interfaces weitgehend unabhängig voneinander durchgeführt werden. Das Schlüsselwort in diesem Zusammenhang heißt Interoperabilität und fasst eben diese Freiheit zusammen. Durch den Einsatz verteilter Rechenarchitektur können Simulationen zudem in einer umfangreicheren Infrastruktur eingebettet werden. Die Vernetzung kann sowohl über LAN als auch über Internet erfolgen (Netzwerk Interoperabilität) wodurch, im Vergleich zu einer einzelnen Rechenmaschine, eine viel höhere Kapazitätsgrenze erreicht werden kann (M&S, 2011).

Probleme gibt es vor allem in der Koordination der Gesamtsimulation. Dies betrifft sowohl den Datenaustausch wie auch das globale Zeitmanagement. Es entstehen ein erhöhter Verwaltungsaufwand und Einschränkungen bezüglich des gemeinsamen Interfaces. Dazu können Netzwerkprobleme stoßen, die mit den heutigen Technologien aber weitgehend beherrschbar sind. Weiterhin gibt es bezüglich der Synchronisation unterschiedliche Algorithmen, um die korrekte zeitliche Reihenfolge von Ereignissen zu gewährleisten. An dieser Stelle sollen die zwei Ansätze genannt werden, die von der HLA unterstützt werden. Bei beiden

Ansätzen werden Ereignisse mit einem Zeitstempel versehen und Simulationen, die von diesem Ereignis betroffene sind, werden über dieses Ereignis informiert. Der konservative Ansatz führt Ereignisse nur dann aus, wenn garantiert werden kann, dass keine Ereignisse mit niedrigerem Zeitstempel, also mit früherem Ausführungszeitpunkt, eintreffen. Optimistische Verfahren führen Ereignisse ohne Verzögerung aus, haben aber einen Rollback-Mechanismus mit dem sie die Auswirkung fehlerhafter Ereignisse rückgängig machen können. Damit werden die Ereignisse so lange wieder aufgerollt bis ein konsistenter Zustand erreicht ist. Anschließend wird die Ereignisausführung in korrekter Reihenfolge fortgesetzt (Uhrmacher et al., 2009).

Die Koordination einer Verteilten Simulation kann über unterschiedliche Ansätze gelöst werden. Die *Distributed Interactive Simulation* (IEEE 1278) ist ein verbreiteter Standard zur Steuerung von Simulationssystemen und diente als Basis für die HLA (IEEE 1516), welches der verbesserte, objektorientierte Nachfolgestandard ist.

2.1.3 Multi-Agenten-Simulationen

Multiagentensysteme (MAS) beschreibt eine Form von Simulationen, welche die aktiven Komponenten der zu untersuchenden Systems als Agenten, also intelligent handelnden Einheiten, betrachtet. Diese können dabei jeweils ein beliebig komplexes Verhalten annehmen, welches aus den inneren Zuständen resultiert. Die Besonderheit dieser Systeme liegt in dem Auftreten von emergenten Phänomenen, also die spontane Herausbildung neuer Eigenschaften, basierend auf der Wechselwirkung zwischen den Agenten. Deshalb sind MAS häufig extrem komplex, wodurch die formale Bestimmung der Eigenschaften dieses Systems sich schwierig gestaltet. Die Konsequenz ist ein hoher experimenteller Anteil zur Erstellung des Designs und der Implementierung (Lees et al., 2007). Der Einsatz von Simulationen wird auch in naher Zukunft die einzige arbeitsfähige Methode bleiben, um in umfassender Art und Weise diese Systeme und die in ihr vorkommenden emergenten Phänomene zu untersuchen. Das Modellieren von individuenorientierten Simulationen resultiert immer häufiger in sehr komplexen Modellen. Das führt zu zwei großen Problemen bei der Nutzung bestehender Frameworks: ein Mangel an Skalierbarkeit und ein Mangel an Interoperabilität. Der Einsatz verteilter Simulationen kann als Lösungsmöglichkeit gewählt werden und findet auch in jüngerer Zeit Anwendung (Uhrmacher et al., 2009).

2.2 High Level Architecture

Die *High Level Architecture* (HLA) ist eine vom U.S. Verteidigungsministerium entwickelte Infrastruktur zur Kopplung von Simulationssystem zu einem Simulationsverbund. Die HLA beschreibt eine komponentenbasierte Architektur, für die es inzwischen eine Vielzahl von kommerziellen und freien Implementierungen gibt. Das Konzept ist seit dem Jahr 2000 als internationaler Standard (IEEE 1516) definiert. Eine HLA Simulation nennt man Federation und sie ist nach dem Client-Server-Modell aufgebaut. Ein Federate muss aber nicht zwingend eine Simulation sein. Auch andere Komponenten, wie zum Beispiel Visualisierung, Datensammlung oder eine Schnittstelle zu einem Live-Player, können als Federate fungieren. Den Server bildet die *Run-Time Infrastructure* (RTI), welche die gesamte Simulation koordiniert und über die sämtliche Kommunikation läuft. Die Simulationen (Clients) werden Federates genannt und sie kommunizieren nur über so genannte Botschafter mit der RTI.

Die HLA besteht im Wesentlichen aus folgenden drei Merkmalen: Als Erstes gibt es eine Menge von HLA-Regeln für Federation und Federates, welche die Anforderungen und Zuständigkeiten für eben diese festlegen. Zweitens ist eine Laufzeitschnittstelle (*Run-Time Interface*) definiert, welche von der RTI umgesetzt werden muss. Das dritte Merkmal ist die Definition des *HLA-Objektmodells*, welches die Struktur und Repräsentation von Daten in der HLA beinhaltet.

2.2.1 Botschafter-Pattern

Die HLA ist eine Schichtenarchitektur, welche hierarchisch organisiert ist. Jede Schicht stellt der jeweils nächsten höheren Schicht Dienste zur Verfügung. Die RTI ist für die Federates eine untere Schicht, welche die Funktionalität der RTI vollständig kapselt. Diese können jedoch über das HLA-Interface angesprochen werden. Die Kommunikation zwischen RTI und Federates läuft nach dem Botschafter (Ambassador) Pattern ab, welches die Trennung der Simulationsfunktionalität von der Koordinationsfunktionalität beschreibt. Jegliche Kommunikation findet über die von der RTI bereitgestellten Kommunikationsdienste statt. Die Federates kommunizieren also nur über die RTI miteinander. In der Abbildung 1 werden die Schnittstellen schematisch dargestellt.

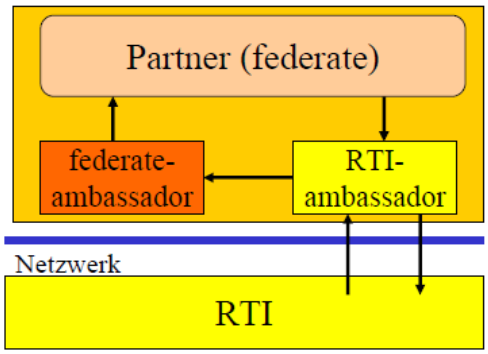

Abbildung 1: Schnittstellen HLA (Buchholz, 2004)

Die RTI stellt jedem Federate eine Kommunikationsschnittstelle zur Verfügung, den *RTI-Ambassador*. Dieser ist im Softwarepaket der RTI enthalten und muss lediglich erzeugt werden. Dieser Botschafter kapselt die Funktionalität der RTI und nimmt alle Anfragen des Partners (Federates) an die RTI entgegen. Zudem erhält das Federate noch eine Kommunikationsschnittstelle den *Federate-Ambassador*, welcher alle Aufrufe der RTI an den Partner entgegen nimmt. Der Federate muss diesen Botschafter selbst implementieren; er ist im RTI-Softwarepaket als Abstrakte Klasse enthalten. Besondere Bedeutung kommen dabei den beiden so genannten *Callbackmethoden* zu, welche Informationen über Ereignisse entgegen nehmen. Die Methode *Reflect Attribute* informiert über Attributänderungen und die Methode *Receive Interaction* über eingehende Interaktionen.

Durch die vollständige Kapselung der RTI und Umsetzung des standardisierten Interfaces ist es möglich, die RTI-technische Weiterentwicklung voranzutreiben oder sogar die RTI auszutauschen ohne den Simulationsverbund anpassen zu müssen.

2.2.2 HLA Regeln

Dieses Regelwerk legt fest, wie die Federates mit der RTI und untereinander über die RTI kommunizieren. Weiterhin enthält es Designprinzipien für das *Object Model Template* und die *HLA Interface Spezifikation*. Das folgende Regelwerk von *Miegel et al.* (Miegel et al., 2003) umfasst knapp die Anforderungen:

Regeln für die Federation

1. Jede Federation muss über ein HLA Federation Object Model verfügen, dass den Spezifikationen des Object Model Template entspricht.

2. Alle Instanzen der im Federate Object Model festgelegten Objekte werden in den Federates repräsentiert, nicht in der RTI.

3. Datenaustausch zwischen den Federates findet nur über die RTI statt.

4. Alle Federates müssen mit der RTI gemäß der *HLA Interface Spezifikation* kommunizieren.

5. Zur Ausführungszeit muss sichergestellt sein, dass Attribute jeweils nur von einem Federate kontrolliert werden.

Regeln für Federates

1. Jeder Federate muss über ein HLA Simulation Object Model verfügen, dass den Spezifikationen des Object Model Template entspricht.

2. Jeder Federate muss die in dem Simulation Object Model spezifizierten Methoden und Attribute im- und exportieren können.

3. Jeder Federate muss in der Lage sein, während der Ausführung Kontrolle über Attribute dynamisch übernehmen bzw. abgeben zu können.

4. Jeder Federate muss in der Lage sein, Bedingungen zu variieren, unter denen sie Attribut-Updates liefern.

5. Die Federates müssen den internen Zeitvorschritt so koordinieren können, dass eine Synchronisation mit der gesamten Simulation stattfinden kann.

2.2.3 Object Model Template

Die HLA ist eine objektorientierte Simulationsarchitektur, welche ihre Klassen und Methoden gemäß dem *Object Model Template* Standard (OMT) abbildet. Das OMT definiert die Vorgaben für das *Federate Object Model* (FOM) und für das *Simulation Object Model* (SOM). Jeder Federate muss ein SOM definieren, welches die Objekte dieser Simulation beschreibt (Möller et al., 2007). Die darin vorkommenden Attribute können öffentlich, mit dynamischem Besitzrecht oder durch fremde Simulationen kontrolliert werden. Das FOM spezifiziert die simulationsübergreifenden Informationen, die innerhalb der Federation ausgetauscht werden können. Dazu gehören Attribute, Objekte, Interaktionen und Assoziationen. Die in diesen Komponenten modellierten Informationen müssen bereits vor der Laufzeit definiert sein und werden deshalb als *non-runtime*

component (Fujimoto et al., 1996) bezeichnet. Die hinreichende Spezifikation von modellierter Information kann simulationsübergreifendes Verständnis garantieren. Jede Simulation kann also, wenn sie die Informationen abbildet, dies nur im Rahmen der Spezifikation tun. Dadurch kann garantiert werden, dass ein Objekt, welches von zwei Simulationen interpretiert wird, jeweils durch exakt die gleichen Werte definiert wird. Somit wird die semantische Unabhängigkeit der Daten gewährleistet, welche als Basis der Interoperabilität dient (Uhrmacher et al., 2009).

In Abgrenzung dazu bezeichnet Fujimoto (Fujimoto et al., 1996), die von der Interface Spezifikation bereitgestellten Dienste, als *runtime components*.

2.2.4 HLA Interface Spezifikation

Die *HLA Interface Spezifikation* umfasst die Dienste (Services), welche die RTI bereitstellen muss. Diese sind im Folgenden aufgelistet und ihre jeweiligen Aufgabengebiete werden anschließend kurz umrissen:

- Federation Management
- Declaration Management
- Object Management
- Data Distribution Management
- Ownership Management
- Time Management.

Das **Federation Management** bietet die Basisfunktionalität an, um Föderationen zu erzeugen und zu zerstören. Weiterhin bietet dieser Service das Beitreten und Austreten von Partnern an und kann die Föderationsausführung anhalten, wiederherstellen, speichern und fortsetzen.

Die Federates können über das **Declaration Management** bekannt geben, welche Information sie bereitstellen und welche sie benötigen. Die Federates können Objektattribute und Interaktionen bekanntgeben (*Publish*) und Interesse für bestimmte Objekte und Interaktionen anmelden (*Subscribe*).

Die Möglichkeit der Erzeugung, des Löschens und der Identifizierung auf Objektebene wird über das **Object Management** bereitgestellt. Weiterhin können Interaktionen und Objektänderungen (Updates) bekannt gegeben und empfangen werden.

Das **Data Distribution Management** spezifiziert, welche Federates über welche Events informiert werden sollen (vgl. Declaration Management). Das RTI benutzt Filter, um Events nur an interessierte Federates weiterzuleiten (Dahlmann et al., 1997). Diese Filtermöglichkeit genügt jedoch bei größeren Simulationen unter Umständen nicht. Wenn zum Beispiel ein Individuum nur Daten von anderen Individuen im Sichtbereich erhalten möchte, so wird das Konzept des Routing Space verwendet. Dabei wird ein multidimensionales Koordinatensystem (durch N-

Punkte) festgelegt, welches das Interessensgebiet des Federates umspannt. Es werden Routing Spaces für Update- und Subscription-Regionen festgelegt, also Bereiche, die für Federates von Bedeutung sind und über die informiert werden soll. Die RTI entdeckt Überlappungen bei der Aktualisierung eines Attributes und informiert alle „in Reichweite“ befindlichen Federates. Die Regionen lassen sich dynamisch zur Laufzeit über den Modify-Region-Service ändern. Bei der Aktualisierung von Attributen werden aus Effizienzgründen nur die Änderungen und nicht alle Attribute des Objekts publiziert. Die Gesamtheit des Objekts kann also nur aus der „Geschichte“ der Aktualisierung erlangt werden.

Jedes Attribut hat zur Laufzeit exakt einen Besitzer, der für die Aktualisierung des Attributwertes verantwortlich ist. Das Besitzrecht wird über einen Dienst verwaltet und kann von den Federates abgefragt werden. Diese können nach Anfrage das Besitzrecht an andere Federates übertragen, die wiederum dann die alleinige Verantwortung übernehmen. Das **Ownership Management** verwaltet diesen Dienst und gibt Änderungen als Mitteilungen bekannt.

Das **Time Management** hat die Aufgabe, den Zeitfortschritt der gesamten Federation zu koordinieren. Die teilnehmenden Federates können dabei unterschiedliche Systeme für ihr lokales Zeitmanagement nutzen. Diese werden über die RTI unter Berücksichtigung anderer Dienste koordiniert, um Informationen in gewissen Zeitrahmen und in korrekter Reihenfolge an die Federates zu übermitteln. Darüber kann sichergestellt werden, dass reproduzierbares kausales Verhalten modelliert wird. Dies wirkt unter anderem dem Entstehen von Anomalien, die beispielsweise durch Verzögerungen im Netzwerk verursacht werden können, entgegen.

2.2.5 HLA Time Management

Das HLA Time Management (HLA-TM) spielt eine außerordentliche Rolle für den Ablauf und die Synchronisation einer verteilten Umgebung, weshalb es einer eingehenderen Betrachtung bedarf. Das Ziel des HLA-TM ist, die Interoperabilität von Simulationen mit unterschiedlichen internen Zeitmanagement Mechanismen zu gewährleisten. Die Unterschiede in den Simulationen variieren dabei von der allgemeinen Abhängigkeit von der Zeit, auch von der Zeit anderer Simulationen, über gewisse Anforderungen (*Constraints*) bezüglich der Synchronisation der teilnehmenden Simulationen bis hin zu einem Ordnungssystem, welches die Nachrichten in eine exakte Reihenfolge bringt.

Dafür fungiert das HLA-TM als zentrale Kontrollinstanz, die mit der Aufgabe betraut ist, den Zeitfortschritt einer Federation zu kontrollieren. Der Zeitfortschritt muss dabei mit anderen Mechanismen koordiniert werden, die für den Datenaustausch verantwortlich sind. Dies ist für zeitsensible Informationen, wie z.B. Attribut-Aktualisierungen oder Interaktionen, notwendig, da von dem zugeordneten Zeitstempel die Gültigkeit dieser Informationen abhängt.

In dem HLA-TM werden mit der Bezeichnung *Zeit* zwei sich unterscheidende Begriffe benannt, die jedoch exakt voneinander abgegrenzt werden müssen. Die (skalierte) Wallclock-Zeit ist die Maßeinheit eines Federates für die globale Echtzeit, welche typischer Weise von einer hardwarebasierten Uhr bezogen wird, auf deren Zeitfortschritt ein Federate keinen Einfluss hat. Durch den Einsatz eines Faktors zum Skalieren, kann der Zeitfortschritt beschleunigt oder verlangsamt werden. Dadurch kann beispielsweise eine Simulation in doppelter Geschwindigkeit der Echtzeit ablaufen. Der zweite Begriff für *Zeit* beschreibt die logische Zeit, die durch ein Federate kontrolliert wird. Die logische Zeit wird benutzt, um koordinierten Zeitfortschritt durchzuführen. Wenn eine Simulation beispielsweise zu einem Zeitpunkt *t* fortschreitet, bedeutet dies, dass alle von der Simulation kontrollierten Entitäten bis zu *t* fortgeschritten sind. Federates nutzen den *Time Advance Service* der RTI um diesen Zustand zu deklarieren. Die aktuelle Zeit eines Federates wird aus dem Minimum von Wallclock und logischer Zeit bestimmt.

Um den Anspruch der Interoperabilität gerecht zu werden, setzt das HLA-TM einen Ansatz namens *Time Management Transparency* um, der den Federates erlaubt, das lokale Zeitmanagement vor anderen Federates versteckt zu halten (Fujimoto, 1998). Federates sind dementsprechend nicht direkt abhängig von den Zeitmanagement-Systemen anderer Federates. Der vereinheitlichte Umgang wird durch verschiedene Services der RTI ermöglicht, die das spezifische Zeitverhalten der Federates unterstützen. Um den Transparency-Ansatz umzusetzen, trifft das HLA-TM folgenden Annahmen bezüglich der Federates:

- Es gibt keinen festgelegten globalen Zeitfortschritt. Zu beliebigen Zeitpunkten können unterschiedliche Federates Zeitfortschritte machen. Dies ist insbesondere für Federates entscheidend, die Zeitfortschritte machen müssen, um bestimmten *Causality Constraints* anderer Federates zu genügen.

- Die Federates und RTI haben Zugriff auf eine externe, synchrone Zeit (Wallclock), die eine gewisse Genauigkeit besitzt. Durch *Clock Synchronization*-Algorithmen kann die Abdriftung der Zeit zwischen den teilnehmenden Systemen eingedämmt werden.

- Jedes Event wird vom auslösenden Federate mit einem Zeitstempel versehen. Der Verwaltungsmechanismus der RTI basiert auf diesen Zeitstempeln. Federates können auch Zeitstempel setzen, die weiter in der Zukunft liegen, als die aktuelle Federate-Zeit. Jedoch dürfen keine Zeitstempel aus „der Vergangenheit“ gesetzt werden.

- Die generierten Zeitstempel müssen nicht in korrekter zeitlicher Reihenfolge generiert werden.

Die von der HLA unterstützten Zeitmanagementsysteme können in zwei Dimensionen charakterisiert werden. Die erste Dimension unterscheidet zwischen Simulationen, die eine feste Abhängigkeit zwischen dem Zeitfortschritt und der Wallclock haben. Diese werden als *constrained* bezeichnet und gehören zur Klasse der (skalierten) Echtzeitsimulationen. Simulationen, die eine solche Abhängigkeit nicht haben, werden als *unconstrained* bezeichnet und bilden die Klasse *der as-fast-as-possible* Simulationen. Die zweite Dimension spezifiziert, ob die Simulation seinen Zeitfortschritt mit anderen Simulationen koordiniert oder ob die lokale Zeit unabhängig von Ereignissen anderer Simulationen fortschreitet. Diesen Vorgang nennt man *Time Regulating* und erfordert gewisse Ordnungsmechanismen, die im nächsten Abschnitt vorgestellt werden.

Federates müssen der RTI bekannt geben, ob sie Time Constrained und Time Regulating sind. Diese Services werden durch die resultierenden Einschränkungen und den Verwaltungsaufwand nur dann in Anspruch genommen, wenn der Federate diese Anforderungen benötigt. Über das Setzen eines Flags[2] werden die Einstellungen bekannt gegeben. Ist das Flag für Time Regulating nicht gesetzt, so werden zeitsensible Nachrichten (*Timestamp Order*, TSO) an diesen Federate von der RTI in zeitunabhängige Nachrichten (*Receive Order*, RO) umgewandelt. Federates, die als *Time Constrained* gekennzeichnet sind, können sowohl TSO als auch RO Nachrichten versenden.

2.2.5.1 Ordnungsmechanismen

Eine zentrale Aufgabe des HLA-TM ist die Sortierung der Nachrichten (*Message Ordering*), welche an die Federates gesendet werden. Dafür wird eine Vielzahl von Diensten angeboten, um den unterschiedlichen Anforderungen der Federates zu genügen. Es werden vier Ordnungsmechanismen angeboten: *Receive Order, Priority Order, Causal Order* und *Timestamp Order*.

Die Receive Order (RO) ist nach dem FIFO-Prinzip[3] aufgebaut und benötigt dementsprechend den geringsten Verwaltungsaufwand. Nachrichten werden in der Reihenfolge der Ankunft an die betroffenen Federates weitergeleitet. Dieser Mechanismus findet Anwendung in Anwendungen, für die minimale Verzögerungszeiten eine größere Rolle spielen als das Einhalten logischer Abhängigkeiten.

Der Priority Order Mechanismus ordnet die einkommenden Nachrichten in einer priorisierten Queue[4] mit dem Zeitstempel als Prioritätsmerkmal. Die Nachrichten mit dem kleinsten Zeitstempel werden als erstes an die Federates ausgeliefert. Somit

[2] Boolescher Wert

[3] First-In-First-Out

[4] Datenstruktur einer Warteschlange

setzt die RTI, nur die lokalen Informationen berücksichtigend, das Ordnungssystem nach Zeitstempeln um. Demzufolge können Nachrichten in der RTI eintreffen, die für ein Federate bestimmt sind, an den schon Nachrichten mit einem niedrigeren Zeitstempel versendet wurden. Darüber hinaus können Nachrichten versendet werden, die für den empfangenden Federate in der Vergangenheit liegen, also mit einem Zeitstempel versehen sind, der kleiner ist als die logische Zeit des Federates. Der Priority Order Mechanismus ist weniger kostenintensiv als Mechanismen, die eine korrekte Zeitreihenfolge garantieren. Jedoch kann er nur Anwendung in fehlertolerante Operationen, wie z.B. Sprachübermittlung, finden.

Bei Benutzung des Timestamp Order (TSO) Mechanismus werden Nachrichten in korrekter zeitlicher Reihenfolge ausgeliefert, unabhängig von ihrem Eintreffen bei der RTI. Um diese Anforderung zu gewährleisten, werden die Nachrichten so lange in der Queue gehalten, bis die RTI garantieren kann, dass keine Nachrichten mit kleinerem Zeitstempel eintreffen können. Außerdem werden keine Nachrichten, die in „der Vergangenheit“ liegen, an die Federates gesendet. Die Federates müssen Zeitfortschritte bei der RTI beantragen, die erst dann genehmigt werden, wenn die RTI garantieren kann, dass keine Nachrichten mit niedrigerem Zeitstempel eintreffen können.

Das *Constrained Ordering* gewährleistet ebenfalls eine konsistente Reihenfolge der Nachrichten, benutzt dabei jedoch bestimmte Abhängigkeiten zwischen den Nachrichten. So muss beispielsweise das Abfeuern einer Waffe vor der resultierenden Zerstörung eines Objekts stattfinden. Empfängt die RTI eine abhängige Nachricht, so wird diese erst ausgeliefert, wenn die Nachricht ausgeliefert wurde, von der die empfangene Nachricht abhängt. Ereignisse, die keinen kausalen Abhängigkeiten unterworfen sind, werden als *Concurrent Events* bezeichnet. Die zugehörigen Nachrichten können ebenfalls geordnet werden (optional). Der zugehörige Service *CATOC* (*Causally And Totally Ordered Communications Supp*ort) garantiert, dass alle Federates diese Nachrichten in derselben Reihenfolge erhalten und somit eine totale Ordnung innerhalb der Concurrent Events vorhanden ist. Mit diesem Mechanismus können definierte Abhängigkeiten eingehalten werden, jedoch werden weder verdeckte Abhängigkeiten berücksichtigt, noch wird auf die mögliche Abhängigkeit von Concurrent Events eingegangen. Folgendes Szenario wird ein solches Problem visualisieren: Angenommen es gibt drei Federates mit jeweils einem Panzer. Der Panzer von Federate A soll auf das erst Ziel schießen, dass in Reichweite gelangt. Der Panzer von Federate B bewegt sich als erstes in dieses Gebiet. Da die Bewegung des Panzers aus C nicht abhängig von der Bewegung der Panzers von B ist, kann die Nachricht, dass der Panzer aus C sich in Reichweite bewegt, vor der entsprechenden Nachricht von B ausgeliefert werden. Somit könnte der Panzer von A fälschlicherweise auf den Panzer von C schießen. Das Constrained Ordering kann also nicht die gleiche Konsistenz liefern wie der Timestamp Ordering Mechanismus. Dennoch bietet dieser Ansatz vergleichsweise geringere

Kommunikationskosten (ohne totale Ordnung) und ist für Systeme, die keine optimistischen Verfahren benutzen, sinnvoll.

2.2.5.2 Time Compensation

Versendete Informationen treffen beim Empfänger, bedingt durch die Netzwerkübertragung, immer mit einer gewissen Verzögerung ein. Die Größe der Verzögerung kann durch Ordnungsmechanismen in der RTI oder Verzögerungen im Netzwerk noch verstärkt werden. Also empfangen Federates konstant Nachrichten, die in *der Vergangenheit* liegen. Es kann eine gewisse Verzögerung akzeptiert werden, da die menschliche Wahrnehmung Ungenauigkeiten im Bereich bis zu 100 Millisekunden nicht erkennen kann (Fujimoto et al., 1996). Weiterhin gibt es Kompensationsansätze (*Time Compensation*) z.B. durch *dead-reckoning models*, die solche Effekte teilweise ausgleichen. Solche Ansätze versuchen beispielsweise die aktuelle Position durch die Bewegungsrichtung und Verzögerungszeit zu extrapolieren.

2.2.5.3 Lookahead

Nehmen verschieden Federates an der Federation teil, die jeweils Time Constrained und Time Regulating Mechanismen benötigen, so kann eine Situation, wie in Abbildung 2 dargestellt, auftreten. In dieser Situation hat Federate D die Möglichkeit, Ereignisse mit dem Zeitstempel 10 zu erzeugen. Deshalb kann die RTI den anderen Federates keinen Zeitfortschritt gewähren, bis Federate D fortschreitet. Dies ist offensichtlich eine ungünstige Situation, da sich die Federates gegenseitig ausbremsen würden. Gelöst werden kann dieses Dilemma durch den Einsatz eines *Lookahead* (von *look ahead*, engl. für vorausschauen). Dies ist ein durch die Federates bestimmter Wert, der aussagt, dass von diesem Federate kein Ereignis bis zu dem Zeitpunkt der aktuellen logischen Zeit plus den Lookahead auftritt. In dem dargestellten Beispiel könnte D mit einem garantierten Lookahead von (z.B.) 5, den anderen Federates versichern, dass bis zur Federation Time 15, kein Ereignis auftritt. Somit könnten die verbliebenen Federates bis zu diesem Zeitpunkt fortschreiten und ihre Ausführung fortsetzen.

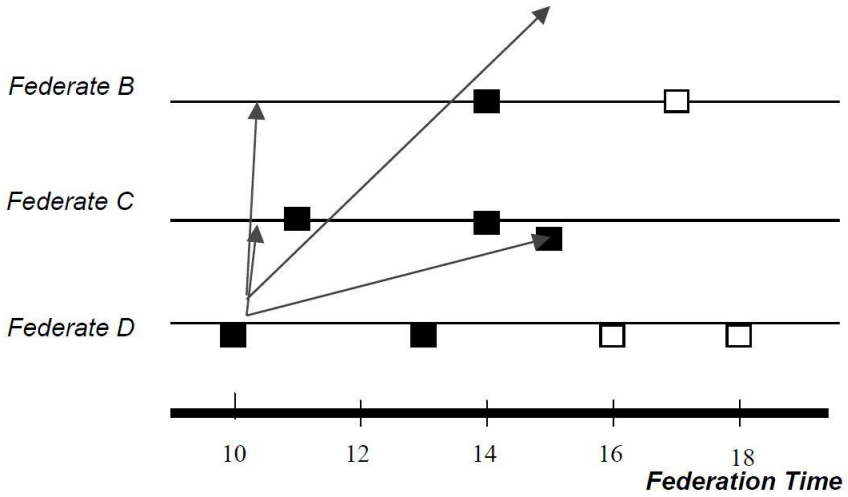

Abbildung 2: Momentaufnahme einer Federation und anstehenden Ereignissen zum Zeitpunkt 10 (Fujimoto 1998)

Ein Federate kann seinen Lookahead zur Laufzeit verändern. Jedoch muss im Falle der Reduzierung der Federate um den Wert der Verminderung in der Zeit fortschreiten. Die Festsetzung des Lookahead hängt stark von den Details der Simulation ab, und kann somit nicht von der RTI bestimmt werden. Folgende Beispiele (Fujimoto, 1998) deuten an, wie ein möglicher sinnvoller Wert bestimmt werden kann:

- Benötigt eine Simulation eine gewisse Zeitspanne, um auf ein Ereignis zu reagieren, so kann diese Zeitspanne als Lookahead gewählt werden. Dies wäre z.B. der Fall bei einem simulierten Panzer, der 500 Millisekunden benötigt, um Aktionen des Operators umzusetzen. Ein sinnvoller Lookahead wäre ebenfalls auf 500 Millisekunden fest zu setzen.

- Physikalische bedingte Zeitfenster, die eine Simulation benötigt, um eine andere zu beeinflussen, können ebenfalls herangezogen werden. Ein Beispiel dafür wären weit entfernte Panzer, die sich beschießen. Die Dauer der Projektile vom Abschuss bis zum Einschlag kann möglicherweise berechnet werden und als Lookahead fungieren.

- Zeittolerante Federates können möglicherweise Ereignisse um einen gewissen Wert in die Zukunft verschieben, wenn diese Verzögerung keinen Einfluss auf die Simulationsausführung hat.

- (Skalierte) Echtzeitsimulationen schreiten in einem regelmäßigen Zeitschritt voran. Die Größe des Lookahead ist dabei meist die Größe des Zeitschritts, da die Simulation nur Ereignisse für kommende Zeitschritte erzeugen kann und nicht für den aktuellen.

Die Festlegung eines Lookahead kann sowohl vom Kontext, als auch durch den Simulationstyp bestimmt werden. Die Größe des Lookahead erfordert von der ausführenden Simulation gewisse Einschränkungen, wirkt sich jedoch positiv auf die Effizienz des HLA-TM aus. Problematisch wird es daher, wenn der Lookahead den Wert Null hat. Es gibt verschiedene Ansätze, um diesen Zustand zu verhindern. So wurde in früheren Versionen der *Zero Lookahead* verboten. In der aktuellen Version werden die Time Management Services mit weiteren Bedingungen verknüpft, um einen solchen Zustand auszugleichen.

2.2.5.4 Time Management Services

Um die spezielle Funktionalität des HLA-TM nutzen zu können, werden zwei Gruppen von Diensten bereitgestellt. Die Gruppe der Transportation Services kapseln die Qualitätsanforderungen an den Datenaustausch und die Dienste zum *Time Advancement* ermöglichen eine zeitliche Synchronisation der Federates.

Die Transportation Services bieten zwei Kategorien an, bezüglich der Anforderungen an die Übertragungsqualität. Dabei wird zwischen Zuverlässigkeit und Nachrichtenreihenfolge unterschieden. Die RTI kann Mechanismen (z.B. *Retransmission*; wiederholtes Senden) benutzen, um die Wahrscheinlichkeit zu erhöhen, dass eine Nachricht den Empfänger erreicht. Durch den erhöhten Verwaltungsaufwand solcher Mechanismen (z.B. das Überprüfen, ob Nachrichten angekommen sind) steigt jedoch die Latenz[5]. Andere Ansätze (z.B. *best effort message delivery*) versuchen die Latenzzeit zu minimieren, was wiederum auf Kosten der Zuverlässigkeit geht. Die zweite Kategorie bilden die bereits diskutierten Ansätze der Ordnungsmechanismen. Auch diese können durch spezifische Dienste angesprochen werden. Ein Federate kann also während eines Simulationsdurchlaufs sowohl ordnungssensible Daten austauschen, als auch Attribute und Interaktionen ohne solche Bedingungen benutzen.

Über die Time Advancement Services können Zeitfortschritte bei der RTI beantragt werden. Durch den koordinierten Zeitfortschritt kann sichergestellt werden, dass ein Federate keine Events mit einem Zeitstempel niedriger als die logische Simulationszeit erhält. Damit die RTI diese Funktionalität gewährleisten kann, muss sie als Kontrollinstanz für Zeitfortschritte fungieren. Somit durchläuft der Zeitfortschritt folgende drei Phasen. Zuerst beantragt ein Federate einen Zeitfortschritt. Anschließend übermittelt die RTI eine Anzahl von Nachrichten

[5] Verzögerungszeit bis ein Ereignis sichtbar wird

(möglicherweise auch keine) an den Federate, in dem der Ambassador über Attributänderungen und Interaktionen informiert wird (siehe 2.2.1). Die RTI vervollständigt diesen Prozess durch die *Time Advance Grant* Prozedur, die dem Federate signalisiert, dass der Zeitfortschritt durchgeführt wurde. Es gibt für ein Federate zwei Mechanismen, um einen Zeitfortschritt zu beantragen, die jeweils ein Zeitmanagementsystem unterstützen:

Für Echtzeitsysteme wird der Dienst namens *Time Advance Request* (TAR) angeboten. Der Federate ruft diesen Dienst auf, um zu einem Zeitpunkt T fortzuschreiten. Die RTI versendet daraufhin alle Nachrichten mit einem Zeitstempel kleiner als T. Wenn die RTI sicherstellen kann, dass keine weiteren Nachrichten mit kleinerem Zeitstempel als T im Umlauf sind oder noch generiert werden, ruft sie die Time Advance Grant Prozedur auf.

Ereignisgetriebene Simulationen haben abweichende Anforderungen und können über den Dienst *Next Event Request* (NER) einen Zeitfortschritt beantragen. Eine solche Simulation kontaktiert diesen Dienst gewöhnlich, sobald alle Aktivitäten bis zur logischen Zeit durchgeführt wurden und ein Zeitfortschritt zu einem Zeitpunkt T angestrebt wird. Die RTI sendet zuerst alle Events, die nicht dem Ordnungsmechanismus unterworfen sind und nach der Receive Order verwaltet werden. Sind keine TSO Nachrichten vorhanden wird der Zeitfortschritt genehmigt. Andernfalls wird die Nachricht mit dem kleinsten Zeitstempel T‘ (mit T‘ kleiner gleich T) ausgeliefert und ein Zeitfortschritt bis T‘ bekannt gegeben.

2.2.5.5 Optimistisches Zeitverhalten

Die bis jetzt vorgestellten Mechanismen stützen sich auf die Annahme, dass die Federates ein konservatives Zeitverhalten umsetzten. Bei diesem Verfahren werden Nachrichten nur verschickt, wenn alle Randbedingungen eingehalten werden können. Es gibt jedoch noch ein andere Variante, das optimistische Verfahren, welches ebenfalls von der HLA unterstützt wird. In diesem Ansatz werden Aktionen unter dem Risiko ausgeführt, dass sie möglicherweise wieder rückgängig gemacht werden müssen, durch so genannte Rollbacks. Einen Rollback durchzuführen bedeutet, den Zustand vor dem fälschlicherweise ausgeführten Ereignis wiederherzustellen und die Prozessausführung unter Berücksichtigung nachträglich eingetroffener Informationen in korrekter Reihenfolge auszuführen. Somit werden Prozesse rekursiv erneut *aufgerollt* bis die Simulation letztlich einen konsistenten Zustand erreicht.

Um eine solche Form der Ausführung unterstützen zu können, müssen weitere Dienste bereitgestellt werden, damit eine optimistische Simulation die Möglichkeit hat, Rollbacks durchzuführen. Zum einen muss es den Federates ermöglicht werden TSO Nachrichten mit niedrigem Zeitstempel zu erhalten bevor die RTI Message Ordering garantieren kann. Hierfür wird der Dienst *Flush Queue Request* bereitgestellt, der alle zwischengespeicherten Nachrichten an den Federate

ausliefert. Zudem muss dem Federate die Möglichkeit eingeräumt werden, eine bereits gesendete Nachricht über ein fälschlicherweise ausgeführtes Ereignis nachträglich zurückzunehmen. Dafür bietet die HLA die *Retract* Funktion an, welche an die Federates weitergeleitet wird, falls die Nachricht schon weiterversendet wurde. Für diesen Fall ist es die Aufgabe der betroffenen Federates diese Nachricht zu löschen oder möglichweise die Auswirkungen der Nachricht zu widerrufen. Schließlich benötigen auch optimistische Federates eine untere Grenze, bis zu der sie Rollbacks durchführen. Dies ist notwendig, um Speicher zu allokieren oder I/O[6] Anfragen zu senden, die nicht zurückgenommen werden können. Durch das Berechnen einer unteren Zeitgrenze wird den konservativen Federates garantiert, dass keine Ereignisse vor diesem Zeitpunkt eintreffen werden. Die Zeitgrenze wird als *Global Virtual Time* (GVT) bezeichnet und in der RTI aus den TSO Nachrichten und weiteren Randbedingungen errechnet. Auch optimistische Federates können damit unter Rücksichtnahme des GVT und ihres Lookahead bestimmte Garantien erfüllen.

2.2.6 Encoding Helpers

Eine verteilte Simulation kann auf den unterschiedlichsten Systemen ablaufen. Diese können verschiedene Prozessoren verwenden, welche unterschiedliche Byte-Größen (32, 64, ... Bit) und unterschiedlichen Byte-Reihenfolge (Big Endian, Little Endian)[7] verwenden.

Um die technische Interoperabilität von der HLA auf verschiedenen Systemen und für verschiedene Programmiersprachen zu gewährleisten, ist es notwendig, die auszutauschenden Daten für alle teilnehmenden Parteien nutzbar zu machen. Die serialisierten Datenformate müssen von allen Teilnehmern bestätigt und umgesetzt werden. Die Alternative wäre, dass jedes System die Daten abhängig vom Sender interpretieren würde. Wenn verschiedene Systeme gleiche oder ähnliche Daten austauschen würden, wäre dieser Ansatz nur suboptimal. Für eine Datenaustausch-Infrastruktur, wie die HLA, wäre ein solcher Ansatz nahezu unmöglich (Möller et al., 2006).

Frühere Interoperabilitätsstandards wie DIS (*Distributed Interactive Simulation*) benutzten ein festgelegtes Austauschmodell. Dies stellte sich aber als große Einschränkung heraus. Deshalb wählt die HLA einen anderen Ansatz und legt die genaue Spezifikation der Typen in der FOM fest. Die Objekte, welche in der FOM

---

[6] Input/Output beschreibt die Interaktionsschnittstelle eines Informationssystems mit externen Geräten

[7] Wenn die Codierung eines Werts mehr Speicherplatz als die kleinste adressierbare Einheit beansprucht, muss die Reihenfolge der Bits im Speicher festgelegt sein. Big Endian speichert den Wert des höchstwertigen Bits zuerst; Little Endian das des niederwertigsten Bits.

definiert sind, werden aus elementaren Datentypen[8] zusammengesetzt. Diese Typen müssen von allen Systemen gleich verstanden werden, ihr Wert muss also der gleiche sein.

Es hat sich herausgestellt, dass in vielen Projekten die Verschlüsselung dieser Datentypen unabhängig voneinander umgesetzt wird. Probleme treten dabei in folgenden Bereichen auf: Das Verschlüsseln und Entschlüsseln ist fehlerhaft (*Coding Errors*), die Spezifikation wird missinterpretiert oder es gibt einen mangelhaften Einblick in das technische Umfeld von Datentypen in Programmiersprache, Compiler, Betriebssystem oder Prozessor.

Es gibt mehrere Ansätze, um Verschlüsselung in der HLA umzusetzen. Der erste und intuitivste Ansatz wäre die eigenständige Verschlüsselung von Objekten in Byte-Arrays. Dies kann aber zu Problemen führen, wenn der Code auf unterschiedlichen Prozessoren ausgeführt wird und wenn Arrays mit variabler Länge codiert werden sollen. Im zweiten Ansatz wird durch ein spezialisiertes Team einen statischen Middleware-Layer entwickelt. Diese „Zwischenschicht“ würde die FOM Anforderungen in den internen Objekten aufnehmen. Eine ähnliche Möglichkeit wäre der Einsatz von Code-generierenden Tools, die basierend auf den unterschiedlichen FOMs den Code erzeugen könnten. Die letzte Möglichkeit ist schließlich der Einsatz von Encoding Helpers, welche die Objekte gemäß Spezifikation in valide Byte-Arrays überführen. Dieser Ansatz wird auch in der neusten HLA-Spezifikation verfolgt und erfordert, bei hoher Fehlerresistenz, wenig technisches Hintergrundwissen. Encoding Helpers benutzen zusätzliche Objekte, die für den Codierungsprozess benötigt werden. Die Verwaltung dieser zusätzlichen Objekte birgt Performanceeinbuße, die jedoch im Vergleich zu den verbleibenden minimalen Codierungsrisiken den Aufwand rechtfertigt. Der Einsatz von Encoding Helpers hat sich als gewinnbringend herausgestellt. Datenfehler aufgrund falscher Codierung können insbesondere im späteren Entwicklungsprozess schwer zu identifizieren und somit kostenintensiv sein.

2.2.7 Portico

Portico ist eine Open-Source HLA Run-Time-Infrastructure Implementierung, die sowohl in Java als auch in C++ verfügbar ist. Lizensiert ist sie unter *der Common Developer and Distribution License* (CDDL). Portico ist in der Version 1.0 verfügbar und setzt den aktuellen HLA Standard 1.3 um.

Es gibt hierbei einige Besonderheiten, die hier kurz aufgeführt werden sollen. Die FOM/SOM Informationen werden lediglich in einem FED-File abgespeichert. Daraus resultierend, müssen alle teilnehmenden Federates sämtliche Informationen des

[8] primitive Datentypen, können nur einen Wert des entsprechenden Wertebereichs annehmen

FED-Files verstehen und können sich nicht bezüglich der RTI konzeptionell voneinander abgrenzen.

Die *SISO* (*Simulation Interoperability Standards Organization*) hat einen HLA API[9] Standard verfasst mit dem Namen *Dynamic Link Compatible* (DLC) API. Portico setzt diesen Standard um, welcher die Spezifikation der IEEE 1516 und IEEE 1.3 API limitiert. Durch die Umsetzung der DLC API erlaubt Portico dem Entwickler dynamisch die Schnittstelle zur HLA zu beziehen – den RTI Ambassador (siehe 2.2.1). Dieser wird über eine Factory-Methode[10] benutzerbezogen generiert und anschließend bereit gestellt.

Die DLC abstrahiert weiterhin bestimmte Details der RTI-Implementierung. Dies betrifft zum Beispiel das Zeitmanagement. Somit ist der zeitrelevante Code Portico-spezifisch. Sollte also die RTI ausgetauscht werden, so müssten alle zeitbezogenen Aufgaben angepasst werden, während die standardisierten Dienste auf gleicher Weise genutzt werden könnten. Bezogen auf die Zeit, müssen also auch spezifische Portico-Typen benutzt werden. Dadurch wird die Kompatibilität zu anderen HLA-Implementierung eingeschränkt. Dieses Problem ist aber nicht DLC-spezifisch, sondern erfolgt aus der aktuellen HLA Spezifikation (PORTICO, 2011).

[9] Application Programming Interface; Programmierschnittstelle

[10] Erzeugungsmuster aus der Softwareentwicklung

3 Anforderungsanalyse

Individuen- oder auch agentenorientierte Simulationen untersuchen das Zusammenspiel von Agenten, die in einer gemeinsamen Umgebung agieren und dabei individuelle oder gemeinsame Ziele verfolgen. Welche Anforderungen an eine geeignete Testumgebung gestellt werden müssen, soll im nachfolgenden Kapitel untersucht werden. Die Schnittstellen der Agenten mit der Umwelt müssen dabei im gleichen Maße evaluiert werden, wie die Kommunikation zwischen den einzelnen Agenten. Diese Interaktion tritt nicht nur innerhalb einer Simulation auf, sondern spielt auch eine Abstraktionsebene höher, in der verteilten Simulation, eine Rolle. Als Kommunikationsmedium (*Enterprise Service Bus*, ESB) zwischen den Simulationen und somit auch den Individuen kommt die HLA zum Einsatz. Durch die Verknüpfung von Simulationen rücken weitere Aspekte in den Vordergrund. So muss grundsätzlich die Erweiterbarkeit bestehender Simulationen geprüft werden, wie auch die Synchronisationsanforderungen, um das System weiterhin validieren zu können. Schließlich werden spezifische Anforderungen an den Prototyp formuliert, welcher exemplarisch die Anforderungen umzusetzen versucht.

3.1 Analyse der Agenten

Die Entwicklung und das Untersuchen dynamischer Testumgebungen für individuenorientierte Simulationen erfordern einen beachtlichen Aufwand. Die Umgebung muss genauso modelliert werden, wie die Interaktionen zwischen Agenten und Umgebung. Viele Simulationssysteme bilden die Agenten im Modell ab, erlauben darüber hinaus jedoch auch noch das Einbinden von Code-Fragmenten oder fremden Modulen in die Simulation (Durfee et al., 1990). Andere Simulationen, wie z.B. das *DIPART* System (Pollack, 1996) importieren den kompletten Agenten. Dieser wird als externe Einheit in den Simulationsablauf integriert und dient als Auslöser von Ereignissen. Auf diese Weise kann Entwicklungszeit gespart werden, die benötigt würde, um den Agenten in die

Programmiersprache der Simulation zu übersetzen. Problematisch wird jedoch die Analyse der Interaktionen und Aktionen mit der Umgebung, da es keine Entsprechung im Modell gibt. Wünschenswert wäre es, die Vorteile beider Ansätze zu vereinen. Das Ziel könnte sein, die externen Agenten, ohne diese signifikant anpassen zu müssen, so weit in das Modell zu integrieren, dass sie wahrnehmbar und kontrollierbar sind und nicht nur als Blackboxes[11] in der Umgebung interagieren. Zu diesem Zweck muss eine abstrakte Struktur, eine Art Platzhalter, für den Agenten modelliert werden, welche man mit dem *tatsächlichen* Agenten verknüpfen könnte. Der Platzhalter hat die Aufgabe, das Verhalten des Agenten zu reflektieren, also sichtbar zu machen, und dazu ein Interface zu definieren, worüber der Agent und die Umgebung verbunden sind In einem solchen System ist der Agent durch die angebotene Anzahl von Services, mithilfe derer er mit der Umgebung interagieren kann, begrenzt, jedoch werden nur wenige Randbedingungen an die Architektur des Agenten gestellt (Uhrmacher et al., 2001).

3.1.1 Interaktionsbedarf der Individuen

Individuen in einer MAS handeln mit- oder gegeneinander, um ein bestimmtes Ziel zu erreichen (Uhrmacher et al., 2009). Sie haben also einen gewissen Aktions- und Interaktionsbedarf. Dieser Aktionsbedarf resultiert aus dem speziellen Verhalten, welches der Agent besitzt. Im Allgemeinen lässt sich das Grundverhalten eines Agenten mit folgender Funktion umschreiben (Thiel-Clemen et al., 2011):

$$\delta_k : S \times T \times I \rightarrow S \times O \qquad \text{with}$$

S : current state of agent *k* (e.g. position, emotional state),

T : current timestamp,

I : sensory input, and

O : sensory output.

Abbildung 3: Funktion - Aktion von Agenten

Individuen bestimmen ihre Aktionen und Reaktionen aus dem aktuellen Zustand und den äußeren Umwelteinflüssen. Diese müssen von den Agenten wahrgenommen werden können. Die Aktionsmöglichkeiten der Individuen umfassen neben der Bewegung auch die Kommunikation mit anderen Agenten. Dem Aktionsbedarf, welcher aus den inneren Zuständen und Wahrnehmung resultiert, kann ein komplexer Prozess der Entscheidungsfindung vorangehen. Dieser hängt stark von der *künstlichen Intelligenz* (KI) des Agenten ab. Es gibt eine Vielzahl von Modellen

[11] Geschlossenes System, welches die inneren Strukturen verbirgt

und Ansätzen unterschiedlicher Agententypen. Sie alle benötigen jedoch die in der Funktion definierten Parameter: Wahrnehmung und Aktionsmöglichkeit. Im Folgenden werden zwei Komponenten des Aktionsbedarfs untersucht, die besonderer Aufmerksamkeit bedürfen. Zum Einen werden die Kommunikationsmöglichkeiten der Individuen beleuchtet und zum Anderen deren Fortbewegung im Raum untersucht. Die Bewegung verursacht eine kontinuierliche Zustandsveränderung der Simulation und hat deshalb für die Synchronisation eine besondere Bedeutung (siehe 3.3). Weitere denkbare Aktionsmöglichkeiten, wie z.B. die Veränderung der Umwelt durch das Setzen von Signalen (Uhrmacher et al., 2009), werden erst einmal außer Acht gelassen.

3.1.2 Wahrnehmung

Die Wahrnehmung von Umwelteinflüsse spielt eine wichtige Rolle bei der Entscheidungsfindung von Individuen. Diese müssen mit Sensoren ausgestattet sein, welche ihnen einen Zugang zu diesen Einflüssen ermöglichen. Die Wahrnehmung muss jedoch auf einen definierten Einflussbereich begrenzt sein, der erst einmal als Sichtbereich bezeichnet wird. So können Hindernisse wie Mauern oder Landschaftserhebungen diesen Bereich einschränken. Darüber hinaus erreicht der Wahrnehmungsbereich nur eine gewisse Ausdehnung, bis zu der ein Individuum die Informationen sinnvoll verarbeiten kann. Zur Vereinfachung wird ein kreisförmiger Bereich um das Individuum definiert, welcher die Grenzen des Sichtbereichs symbolisiert. Der Agent erhält also eine Liste mit aktuellen Ereignissen, die in diesem Sichtbereich auftreten, auf die er anschließend reagieren kann. Für den Prototyp spielen dabei Bewegungsinformationen anderer Individuen die wichtigste Rolle. Diese bestehen aus den Abmaßen der Bewegungsrichtung und der Geschwindigkeit des Objekts. Weiterhin werden verdeckte Informationen beigefügt, die unabhängig vom äußeren Erscheinungsbild sind. Für die Schiffsimulation kann z.B. das priorisierte Vorfahrtsrecht bekannt gegeben werden. Dieses dient als Vereinfachung und unterstützt die Koordination auf See.

Es gibt noch weitere Einflussbereiche, die andere Formen der Wahrnehmung widerspiegeln. So kann ein Kommunikationsbereich festgelegt werden, innerhalb dessen die Individuen miteinander in Verbindung treten können. Eng verwandt dazu ist der Interaktionsbereich, welcher sprachlich abgegrenzt wird, um zu signalisieren, dass innerhalb dieses Bereichs Objekte ausgetauscht werden können. Dieser Bereich ist aus physikalischen Gründen kleiner als der Bereich, in dem verbal kommuniziert werden kann. Aus Einfachheitsgründen werden im Prototyp jedoch alle Bereiche mit dem Sichtbereich zusammengelegt. Die Unterschiede können aber bei anderen Szenarien eine wichtige Rolle spielen.

Schließlich soll noch auf die körperliche Wahrnehmung der Individuen eingegangen werden, im Speziellen die Kollision. Es besteht die Möglichkeit vom direkten Aufeinandertreffen zweier Individuen, die je nach Intensität Folgen für Beteiligte haben kann. Deshalb ist es notwendig, dass ein Kollisionsmelder allen Betroffenen

Auskunft darüber gibt, ob eine Kollision stattgefunden hat. Wünschenswert wäre es auch, die Beteiligten bei einer Kollision in einen vordefinierten Zustand zu versetzen. Dies könnte Einschränkungen der Agenten bis hin zum Verlust der Funktionsfähigkeit mit sich führen. Ein solcher Zustand müsste jedoch im Simulationsmodell abgebildet sein, um für alle teilnehmenden Individuen die gleichen Rahmenbedingungen zu schaffen. Denkbar wäre dann auch eine visuelle Darstellung dieses Folgezustands.

Die Wahrnehmung ist ein existentieller Teil des Agenten und somit auch für externe Agentenformen bedeutend. Diese sollen ebenfalls unterstützt werden, wozu es notwendig ist, eine klare Schnittstelle für die Wahrnehmung anzubieten. Diese sollte dementsprechend unabhängig vom Game-Loop sein und eher nach dem Interrupt-Prinzip[12] funktionieren. Treten Ereignisse auf, so werden die Agenten durch die in dem Interface definierten Methoden informiert.

3.1.3 Kommunikationsmöglichkeiten

Individuen benötigen Kommunikation, um miteinander zu interagieren. Wollen Individuen bestimmte Ziele erreichen und ist dies alleine nicht oder nur durch Mehraufwand erreichbar, so entsteht das Bedürfnis nach Hilfe (Uhrmacher et al., 2009). Kommunikation basiert auf einer Sprache, welche von den Individuen verstanden werden muss. Die Sprache wird in der HLA im FOM definiert. Das Framework soll diese Interaktionsmöglichkeiten in Interfaces kapseln, welche die Agenten umsetzen müssen. Jedes Individuum muss also ein Interface implementieren, in dem es auf alle eingehenden Kommunikationsanfragen reagiert und sei es nur mit einem *Nein*.

Das Framework soll die Agenten so voneinander kapseln, dass diese mit dem Gegenüber immer auf die gleiche Art und Weise kommunizieren können. Die Individuen fremder Simulationen müssen also in die Umgebung eingebunden werden und die notwendige Kommunikation über die RTI soll vollständig gekapselt werden. Mit denen im Sichtbereich wahrgenommenen Agenten kann ein Individuum dann über ein Interface kommunizieren. Die in der SOM enthaltene Semantik muss durch das Interface garantiert werden.

[12] Unterbrechung eines Programm um eine andere, zeitkritische Ausführung zu starten

3.2 Evaluierbarkeit

Die Untersuchung von Simulationen erfolgt meist auf der Basis von gesammelten Daten, z.B. von Zuständen, findet jedoch überwiegend auch visuell statt. Um die Vorgänge darstellen zu können, müssen Beobachterprozesse Zugriff auf bestimmte Teile des Modells haben. Bezüglich MAS kann dieser Vorgang komplex werden, da Agenten zu jeder Zeit der Simulation neu erschaffen werden können und somit, bedingt durch den neuen Zustand, das MAS Modell neu instrumentiert[13] werden muss. Bedingt durch die Fluktuation der Agenten, müsste der Benutzer die zu untersuchenden Individuen durch generische Regeln beschreiben, unter der Bedingung, dass das Verhalten dieser bestimmten Individuen untersucht werden soll. Außerdem können Agenten einen komplexen Zustand annehmen, von dem meist nur ein geringer Teil von Interesse ist. Vorteilhaft wäre die Möglichkeit, nur bestimmte Agenten und ihre spezifischen Subzustände zu untersuchen. Die Folge wäre ein verminderter Speicheraufwand und die Möglichkeit, sich auf bestimmte Phänomene zu fokussieren (Uhrmacher et al., 2009).

Für den Prototyp spielt der Punkt der Evaluierbarkeit eine untergeordnete Rolle, da zum Einen nur eine stark begrenzte Anzahl von Individuen exemplarisch miteinander interagieren. Zum Anderen liegt das Hauptaugenmerk auf dem Zusammenspiel der Agenten innerhalb einer verteilten Simulation. So ist es von Bedeutung, dass von lokalen Agenten erzeugte Ereignisse zeitnah und korrekt in den anderen Simulationen dargestellt werden. Zum Einen können die Simulationsabläufe nebeneinander visuell verglichen werden. Durch die Synchronisationsroutinen muss die Verzögerungszeit so gering sein, dass sie dem menschlichen Betrachter nicht auffällt. Diese Anforderungsqualität ist bereits für viele Simulationsszenarien ausreichend genau. Darüber hinaus können auch noch Zeitmessungen durchgeführt werden, um exaktere Aussagen über das Zeitverhalten treffen zu können. Für gewöhnlich ist eine Verzögerungszeit von weniger als 500 Millisekunden für den Menschen nicht wahrnehmbar (Fujimoto, 1998).

Der Weitere Evaluierungsbedarf hängt stark von dem Szenario ab, welches eine Simulation untersucht. Dieser Bedarf geht über die erforderliche Korrektheit des Frameworks hinaus. Dennoch müssen Möglichkeiten zur Auswertung bestimmter Situationen durch das Framework ermöglicht werden. Häufig wird das Eintreten gewisser Zustände oder Effekte untersucht. Auch bestimmte Aspekte des inneren Zustands eines Agenten können von Interesse sein. Diese Informationen müssen im Modell abgebildet werden und eventuell im Simulationsverlauf aufgezeichnet werden können. Die Repräsentation von Individuen durch Platzhalter kann eine Möglichkeit sein, die Informationen innerhalb des Frameworks zu verwalten und bereitzustellen. Dieser Ansatz würde auch das Einbinden externer Agenten ermöglichen, ohne dabei die Evaluierbarkeit zu vermindern.

[13] Anreichern des Quellcodes mit Zusatzinformationen

3.3 Synchronisationsproblematik

Alle teilnehmenden Individuen müssen mit zeitnahen, validen Informationen versorgt werden, damit die verteilte Simulation dem Systemmodell genügen kann. Dafür ist es notwendig, alle verteilten Systeme auf dem gleichen Stand zu halten. Die HLA bietet verschiedene Synchronisationsansätze (siehe 2.2.5). Welcher Ansatz gewählt wird und welche Anforderungen bezüglich der Fehlertoleranz gemacht werden können, hängt im ersten Schritt von dem Simulationsmodell ab.

Das Framework soll auf der Graphikengine (JME, 2011) aufbauen, welche eine feste Zeittaktung, den Update-Zyklus, vorgibt. Die Wahl des Time-Stepped Paradigmas (siehe 2.1.1) ist für die Visualisierungskomponente unerlässlich, welche dem Framework inhärent ist. Die Engine muss notwendigerweise einen linearen Zeitfortschritt umsetzten, um die Vorgänge flüssig darstellen zu können. Weiterhin sollen die Simulationen ohne großen Aufwand in das Framework eingebunden werden können. Deshalb ist es sinnvoll, die Simulationen und somit auch die Individuen an diesen Zeitfortschritt zu koppeln.

Von Individuen ausgelöste Ereignisse führen zu Zustandsänderungen in der Simulation. Diese müssen zeitnah visualisiert und auch innerhalb der verteilten Simulation in einem gewissen Zeitfenster bekannt gemacht werden. Der Fokus soll dabei erst einmal auf die Bewegung von Individuen gelegt werden. Diese können eine gewisse Bewegungsrichtung und eine Geschwindigkeit annehmen. Wie im vorangegangenen Abschnitt gefordert, sollen die Agenten mit einem Platzhalter verknüpft werden. Dieser fungiert als Repräsentant in der Welt und stellt den Agenten und sein Verhalten visuell dar. Ist ein Agent nun bestrebt eine bestimmte Bewegung auszuführen, so muss diese Aktion an den Platzhalter weitergeleitet werden, welcher dann versucht diesen Befehl durchzuführen. Ein entsprechender Befehl könnte beispielsweise folgende Parameter enthalten:

Move [Richtungsvektor, Geschwindigkeitsgröße].

In diesem Ansatz wird, ähnlich dem Trägheitsprinzip[14], davon ausgegangen, dass der Körper so lange seine Bewegung fortsetzt, bis diese durch interne oder externe Einflüsse verändert wird. Das Individuum kann sich also durch einen Befehl in Bewegung versetzen. Die Durch- und Weiterführung dieses Zustands muss durch die Welt erfolgen. Der notwendige Synchronisationsaufwand eines Bewegungsvorgangs besteht also in der Bekanntgabe der Startposition und den Bewegungsparametern bis zum Bewegungsende.

Inwieweit die tatsächlichen Positionen der Agenten von lokalen zu externen Simulationen divergieren, hängt von verschiedenen Faktoren ab. Ein Faktor kann sein, ob die Aktion erst ausgeführt wird, wenn alle Simulationen darüber in Kenntnis

[14] Ein Körper behält seine gleichförmige Bewegung bei, sofern keine Kraft auf ihn ausgeübt wird.

gesetzt werden, also ob es einen koordinierten Zeitfortschritt innerhalb der Gesamtsimulation gibt. Die Koordination müsste über das HLA-TM erfolgen und die Frameworks müssten ihren Zeitfortschritt bei der RTI beantragen. Der Update-Zyklus der Engine soll aber in den hier festgelegten Anforderungen autonom bleiben, womit von diesem Ansatz abgesehen wird. Wird die Aktion also angestoßen, so wird sie zunächst in der lokalen Simulation ausgeführt, gleichzeitig werden aber auch Informationen als RO Nachrichten über die HLA verschickt. Die Nachrichten werden direkt an die weiteren Federates bekannt gegeben, welche den Inhalt analysieren müssen und anschließend die externe Repräsentation des Agenten in Bewegung setzen. Die Dauer dieses Prozesses ist gleich der Verzögerung, die der Agent im Modell der externen Federates besitzt. Die Verzögerungszeit kann sogar noch höher sein, wenn ein externer Federate erst bei koordiniertem Zeitfortschritt über die neuesten Informationen benachrichtig wird. Es gibt jedoch Ansätze, die den Rückstand kompensieren können. Eine Möglichkeit bildet der dead-reckoning Ansatz, welcher versucht aus Verzögerungszeit und Bewegungsgeschwindigkeit die tatsächliche Position zu extrapolieren. Es ist auch denkbar, die aktuelle Position eines Agenten in einem gewissen Zyklus zu publizieren, um die Simulationen wieder einander anzunähern. Absolute Synchronisation kann jedoch nur über den Einsatz der HLA-TM Ansätze erfolgen.

Für das Framework sollen ein möglichst kostengünstiger Synchronisationsaufwand gewählt werden. Da die verteilte Simulation im Prototyp maximal auf ein lokales Netzwerk ausgeweitet ist, bleibt die Verzögerungszeit bei überschaubarerer Simulationsdauer deutlich unter dem Richtwert von 500ms (Fujimoto, 1998).

Von dem Einsatz weiterer Justiermechanismen für die Positionsbestimmung wird abgesehen. Jedoch müssen die Simulationen den Individuen die gleichen Welt-bezogenen Bedingungen schaffen, als da wären: Zeit und Raum. Zunächst zur Zeit: Da die Simulationen des Frameworks nicht an einen globalen Zeitfortschritt gebunden sind, muss das Framework die Taktung bestimmen. So ist es notwendig, dass beteiligte Frameworks den gleichen Update-Zyklus haben und sich die Simulationen im gleichen Zeitschritt befinden. Dieser Zustand muss vor Beginn der Simulation sichergestellt werden. Wird das gleiche Framework mehrfach verwendet, so wird automatisch die gleiche Zeittaktung verwendet. Nehmen jedoch externe Frameworks oder Simulationen teil, so muss sichergestellt werden, dass diese ebenfalls den gleichen Zeitrhythmus umsetzten. Soll es Federates mit anderen Zeitmodellen als dem Time-Stepped Modell möglich sein an dem Simulationsverbund teilzunehmen, so muss der Zeitfortschritt der Gesamtsimulation über die HLA-TM Mechanismen koordiniert werden.

Die zweite Bedingung, welche die unterschiedlichen Simulationen den Individuen bereitstellen muss, ist eine gemeinsame Umgebung, der Raum.

3.4 Umgebung

Die Umgebung einer Simulation bildet die gemeinsame räumliche Grundlage der Individuen. Es wird ein *Spielfeld* definiert, welches versucht die Gegebenheiten des Source Systems abzubilden. Meist werden nur die für das Szenario wichtigen Strukturen der Geographie in einer Karte abgebildet. Die Karte kann dann durch Geodaten repräsentiert werden. Die verschiedenen Simulationen einer MAS müssen die gleiche Karte verwenden.

Die Geodaten können den Simulationen auf verschiedene Weisen bereitgestellt werden. Wichtig ist nur, dass die Interaktionsgebiete in den Simulationen identisch sind. Geographische Informationen sind (in diesem Ansatz) nicht dynamisch. Deshalb nennt man sie *Sekundäre Informationen* und sie müssen somit schon vor Simulationsbeginn zur Verfügung stehen (Klein et al., 1998b). Ein komplexerer Ansatz könnte die Bereitstellung der Informationen über ein *Geographic Information System*[15] (GIS) lösen. Ein solches System würde den Simulationen dynamisch die benötigten Daten zur Verfügung stellen. Auch Änderungen im Spielfeld wären dann denkbar (Klein et al., 1998a). Es gibt jedoch keine standardisierte Schnittstelle zwischen GIS und Simulationswerkzeugen, sondern nur Ansätze, die auf spezielle Fragestellungen zugeschnitten sind (Strassburger et al., 2005). Von dem Einsatz eines GIS wird in der Anforderungsanalyse abgesehen.

Der geforderte Ansatz betrachtet Geo-Informationen als statische Sekundärdaten, welche jeder Simulation vor Simulationsbeginn vorliegen müssen. Eine Möglichkeit wäre, die Daten als *Height-Map*[16] bekannt zu geben. Ein solches Höhenrelief kann zusammen mit weiteren Informationen, wie z.B. Untergrundbeschaffenheit oder Wasserhöhe, das Spielfeld reproduzierbar beschreiben. Das Framework setzt diese Informationen durch die Grafikengine um. Für fremde Simulationen wäre ein einheitlicher Ansatz wünschenswert. Im ersten Schritt reicht aber die aktuelle Form der Daten aus, damit das Spielfeld simuliert werden kann.

Als weiterer Teil der Umgebung können Objekte verstanden werden, die keine aktiven oder spontanen Aktionen ausführen. Als Beispiel können Ampeln oder Straßenschilder genannt werden. Diese sind in erster Linie Hindernisse für die Individuen und könnten deshalb durch die Geographie modelliert werden, sofern diese eine solche feingranulare Beschreibung erlaubt. Auf den zweiten Blick können solche Objekte jedoch noch einen gewissen Informationsgehalt bereitstellen. Vor allem, wenn sie eine Zustandsänderung bewirken, wie die Ampeln, müssen sie als Objekte modelliert werden. Als Abgrenzung zu den aktiven Individuen haben diese *inaktiven Objekte* keine Bewegungsmöglichkeit und kein *intelligentes Verhalten*. Die Implementierung der inaktiven Objekte hängt von der Simulationsaufteilung ab. Gibt

[15] Informationssystem zur Bereitstellung geographischer Daten

[16] Zweidimensionales skalares Feld, das ein Höhenrelief beschreibt

es beispielsweise verschiedene Simulationen, die bestimmte Individuentypen umsetzten, welche wiederum auf bestimmte inaktive Objekte angewiesen sind, so können die Objekte ebenfalls von diesen jeweiligen Simulation in die Welt integriert werden. Als Beispiel ist eine Autosimulation denkbar, die gewisse Verkehrsregeln über Schilder vorgibt, im Zusammenspiel mit weiteren Simulationen, auf die diese Schilder keinen Einfluss haben. Eine zweite Möglichkeit ist die Integration aller inaktiven Objekte als Bestandteil der Welt in einer Simulation. Diese müssten bei jedem Simulationsexperiment an der verteilten Simulation teilnehmen, um die Umgebung zu ergänzen. Die anderen Simulationen würden jedoch bei einer monolithischen Ausführung möglichweise an Sinnhaftigkeit verlieren. Die Entscheidung hängt dementsprechend von dem Entwicklungsprozess der verteilten Simulation ab. Werden auch externe, bereits bestehende Simulationen eingebunden, so ist der zweite Ansatz ohnehin zu vernachlässigen.

3.5 HLA Erweiterbarkeit von Simulationen

Prinzipiell kann jede bestehende Simulation oder ein jedes Simulations-Werkzeug, das bestimmte Anforderungen erfüllt, über HLA in eine verteilte Simulation eingebunden werden. Um eine bestehende Simulation jedoch in einen Simulationsverbund eingliedern zu können, müssen die HLA-Regeln für Federates umgesetzt werden können. Die Voraussetzung dafür ist ein gewisser Grad von Modularisierung in der zu erweiternden Software. Zudem muss ein gewisser Zugriff auf die Erweiterbarkeit der *Event Handling List* bestehen, welche zukünftige Ereignisse speichert und zu gewissen Zeitpunkten zur Ausführung freigibt. Eingriffe in ein- und ausgehende Events müssen getätigt werden, um diese Informationen global bekanntmachen zu können. Der Zugriff auf die Simulationsobjekte muss gewährleistet sein, damit diese in die Federation integriert werden können. Dies funktioniert nur unter der Voraussetzung, dass die Objekte in sich abgeschlossen und transparent sind.

Eine Möglichkeit, die Simulation und die HLA-Erweiterung miteinander zu verknüpfen, ist ein *Intra-SOM-Layer*. Dieser enthält die publizierten Klassen. Sowohl das HLA Interface als auch die Simulation greifen auf die hier instanziierten Objekte zu und passen ihre Systeme daraufhin an. Bei Veränderung seitens der Simulation werden die entsprechenden Events erzeugt. Bei Veränderungen durch die HLA-Erweiterung muss die Simulation ihre Objekte synchronisieren (Klein et al., 1998a).

3.6 Anforderungen an den Prototyp

Im Zuge dieser Bachelorarbeit wird ein Prototyp erstellt, welcher das analysierte Framework exemplarisch umsetzt. An dieser Stelle werden nochmal die Anforderungen geprüft, die der Prototyp erfüllen soll.

Der Prototyp soll das Framework so generisch wie möglich umsetzten. Die erste Anforderung ist die Kartenfreiheit. Die Simulationen sollen auf beliebigen *Spielflächen* ablaufen. Dazu zählt auch das *Matchup*, also das Simulationsszenario. Die Anzahl der teilnehmenden Individuen und Simulationen soll nicht beschränkt sein. Des Weiteren wird vom Framework gefordert, dass sich die Einbindung von Simulation so einfach wie möglich gestaltet. Es soll lediglich ein Interface für die Simulation und ein Interface für Individuen geben, welches diese implementieren müssen, um an der Gesamtsimulation teilnehmen zu können. Die Kommunikation über die HLA und die tatsächliche Anzahl unterschiedlicher Systeme und Simulationen soll gekapselt sein. Fremde Individuen sollen in das Framework so integriert werden, dass lokale Simulationen über die Welt mit ihnen interagieren können. Ein Individuum hat also keinerlei Kenntnisse darüber, ob es sich in einer verteilten oder monolithischen Simulation befindet. Alle notwendigen Synchronisations- und Kommunikationsroutinen sollen vom Framework erledigt werden. Es bleibt jedoch zu bedenken, dass dieser Prototyp nur exemplarisch andeuten soll, wie eine individuenorientierte Simulation unter dem Einsatz von der HLA aussehen kann. Somit wird nicht die volle Funktionalität der HLA, insbesondere des Time Managements, ausgenutzt. Jedoch kann angedeutet werden, inwiefern der Prototyp weiterentwickelt werden kann, um noch umfassendere Anforderungen zu genügen und um effektiv weitere Simulationen, die außerhalb des Frameworks liegen, einzubeziehen.

3.6.1 Exemplarische Anwendungsfälle

Im Folgenden werden mögliche Situationen für einen Agenten vorgestellt. Die dargestellten Problematiken treten bei jedem beweglichen Agenten auf und es muss den Individuen konzeptionelle Handlungsmöglichkeiten offeriert werden.

1. Ein Agent bewegt sich in eine bestimmte Richtung, nimmt dabei Hindernisse wahr und weicht ihnen aus.

2. Zwei Agenten bewegen sich als Punkte jeweils auf einer linearen (hier als Vereinfachung festgelegt) Kurve in der Welt. Die Kurven kreuzen sich und die Punkte erreichen diese Schnittstelle in einem ähnlichen Zeitfenster, welcher von den Ausmaßen der äußeren Erscheinungsbilder der Individuen

abhängig ist. Der Agent mit der niedrigeren Priorität weicht aus (Vereinfachung[17]).

3. Agent A befindet sich im Aktionsbereich von Agent B. B sendet Informationen an A, welcher, mindestens mit booleschem Ausdruck, auf die Anfrage antwortet.

4. Agent A befindet sich im Aktionsbereich von Agent B. B übergibt ein Objekt an A, welcher entweder das Objekt annimmt, oder es ablehnt.

Auf Grund des Schwerpunktes des Prototyps in verteilten Simulationen sind die Anforderungen bezüglich der Authentizität relativ gering. So werden als weitere Vereinfachungen einige Annahmen getroffen. Die Bewegungsrichtung des Agenten ist gleichzeitig die Ausrichtung seines Körpers und seiner Blickrichtung.

[17] Die Ausweichpflicht wird hier als Vereinfachung nach bestimmten Kriterien festgelegt (vgl. Schifffahrt). Bei Agenten des Typs Fußgänger würde auch der höher priorisierte (durch Geschwindigkeit, Entschlossenheit, Ausstrahlung) seine Bewegungsrichtung geringfügig anpassen.

4 Design

Dieses Kapitel beschreibt die Architektur des Frameworks, welches die Hauptkomponente des Prototyps bildet. Dazu werden konzeptionelle Entscheidung bezüglich des Designs vorgestellt. Besonderes Augenmerk wird auf die Schnittstellen zu den Simulationen und Agenten gelegt. Auch die Anbindung an die RTI wird diskutiert. Dazu werden die Abläufe der Synchronisationsmechanismen beleuchtet.

4.1 Leitbilder

Das Ziel ist die Schaffung eines Frameworks, in dem man einfach und effizient Simulationen zum Verbund hinzufügen kann. Die Simulationen sollen dabei die größtmöglichen Gestaltungsfreiheiten innerhalb des Frameworks haben. Das Framework wiederum kapselt die externen Simulationen und bindet die externen Objekte so in die Welt ein, als würden sie lokal angebunden sein. Lokale Simulationen müssen also nicht unterscheiden, ob sie mit lokalen oder mit externen Objekten interagieren. Die verteilte Simulation wird vollständig dupliziert und im lokalen Framework abgebildet.

Die Konzeptentscheidungen werden mit dem Wunsch der Wiederverwendbarkeit getroffen. Die Vision sieht eine generische Simulationsumgebung vor, in der Landschaft und teilnehmende Objekte beliebig ausgetauscht werden können. Weiterhin können sowohl menschliche als auch KI-getriebene Spieler an der Simulation teilnehmen. Somit kann unterschiedlichsten Anforderungen genügt und eine Vielzahl von Situationen in einem definierten Zeitrahmen abgebildet werden.

Die Anbindung von Simulationen außerhalb des Frameworks an den Verbund soll natürlich möglich sein. Der Verbund soll gemäß der HLA Richtlinien und den Vorgaben des Simulationsverbunds (Raum, Zeit, Szenario) offen für externe

Simulationen sein. Dennoch muss diese Möglichkeit im Prototypentwurf nicht weiter vertieft werden. Die Anbindungsmöglichkeiten (siehe HLA Erweiterbarkeit von Simulationen) und Sinnhaftigkeit (siehe Simulations-Ansatz) stehen außer Frage.

Das formulierte Ziel ist jedoch die Darstellung der konzeptionellen Einsatzmöglichkeit der HLA. Eine sinnvolle Verknüpfung von verteilten Simulationen und MAS soll exemplarisch dargestellt werden. Die Authentizität und damit einhergehende Synchronisationsaufgaben spielen eine untergeordnete Rolle.

4.2 Konzept

Im Prototyp wird ein Framework entwickelt, worüber Simulationen einfach in einen Simulationsverbund integriert werden können. Die gewählte Architektur wird in einem Klassendiagramm in Abbildung 4 abgebildet. Es gibt zwei wichtige Komponenten (HLA und Simulation) innerhalb des Frameworks. Das Interface *ISimulation* muss von jeder teilnehmenden Simulation umgesetzt werden, das Interface *IEntity* von jedem Agenten. Die angebunden Simulationen bilden die Simulations-Komponente. Die zweite Komponente kapselt die Verbindung zum Simulationsverbund und hält somit die Verbindung zur RTI. Das Framework agiert also als ein Federate und hat deshalb nur eine Verbindung zur RTI, kann jedoch beliebig viele Simulationen integrieren.

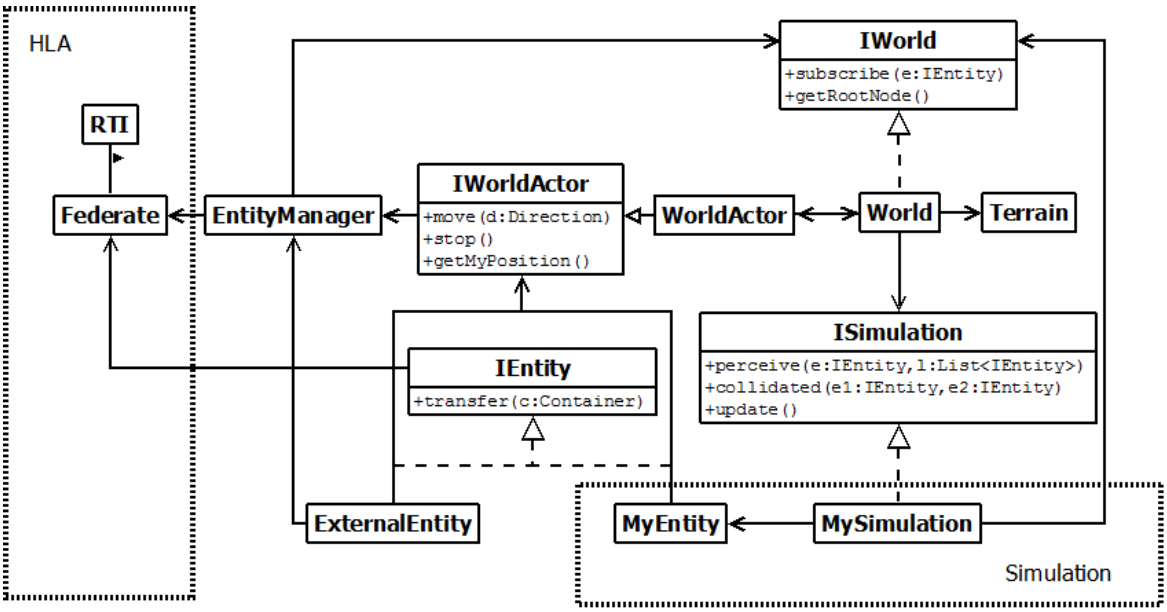

Abbildung 4: Architektur Entwurf

Entitäten sind Agenten, die ein individuelles, gekapseltes Verhalten umsetzen. Auf der Abstraktionsebene der verteilten Simulation werden Entitäten also als Objekte verstanden, welche für andere Simulationen relevant sind. Das betrifft alle Objekte, mit denen potenziell interagiert werden kann. Es können verschiedene Objekttypen im FOM modelliert werden. Die Abbildung 4 zeigt mit der IEntity nur einen generischen Typ, es sind jedoch weitere Subinterfaces denkbar und sinnvoll.

Diese Typen bestimmen das Methodenprotokoll, welches für die Kommunikation zwischen Agenten bereitsteht. Eine Entität kann nur mit anderen Entitäten innerhalb seines Wahrnehmungsbereichs kommunizieren. Dieser wird der Simulation über die Methode *perceive* zur Verfügung gestellt, welche die Informationen dann an die Entität weiterleiten kann. Die Kommunikation ist in diesem Fall begrenzt auf die zur Verfügung stehenden Methoden von IEntity, sonst auch auf die Subinterfaces. Die Auswahl der dadurch ausgelösten Interaktionen wird im Datenmodell (FOM) festgelegt und muss durch das Interface im Framework abgebildet werden. Diese Methoden führen nicht notwendigerweise eine HLA-Interaktion aus, wenn die Kommunikation zwischen zwei lokalen, im Verbund integrierten Entitäten ausgeführt wird. Grundsätzlich sind diese Methoden immer Anfragen und sie lösen potentiell eine Interaktion aus. Die Methoden liefern also immer eine Antwort bezüglich des semantischen Gehalts der Methode. Im Entwurf wird nur eine Methode exemplarisch angeboten. Dieses Interface ist der einzige nicht generische Punkt des Frameworks, da es notwendig ist, die Interaktionen des Simulationsszenarios als Methodenprotokoll anzubieten.

Agenten haben weiterhin die Möglichkeit Aktionen auszuführen, wo bei hier der Schwerpunkt auf die Mobilität gelegt wird. Es gibt einen Stellvertreter für jeden Agenten, der diesen graphisch und physisch in der Welt repräsentiert. Dieser nennt sich *WorldActor* und ist an die Grafikengine gekoppelt. Bei der Registrierung eines Agenten in der World, wird ein solcher Repräsentant erzeugt und an den Agenten gekoppelt. Ein Agent kann diesen Stellvertreter dann steuern, in dem er z.B. eine Bewegungsrichtung beantragt. Dieser führt die Anfrage aus, verändert zyklisch seine Position und informiert über den *EntityManager* die RTI.

Der EntityManager kümmert sich um die Synchronisation aller Entitäten. Zunächst bindet er alle externen Entitäten in die lokale World ein. Hiermit sind alle Agenten gemeint, die nicht lokal sondern über die HLA an dem Simulationsverbund teilnehmen. Als Resultat können die lokalen Simulationen alle externen Agenten wahrnehmen, als würden sie lokal ablaufen. Führen externe Individuen Aktionen aus, so setzt die *ExternalEntity* diese Aktionen lokal um. Auch externe Kommunikationsanfragen werden über die ExternalEntity an die lokalen Entitäten weitergeleitet und anschließend über die HLA beantwortet. Es werden ebenfalls alle Aktionen durch die externen Entitäten in der lokalen Simulation reproduziert. Hiermit wird die gesamte verteilte Simulation in jedem Framework vollständig abgebildet.

Schließlich gibt es noch die Simulationsumgebung, welche durch die Klasse *World* umgesetzt wird. Die *World* ist an die Grafikengine gekoppelt und somit verantwortlich für die Visualisierung. Dazu werden die geographischen Informationen in dem *Terrain* verwaltet und als *Spielfläche* bereitgestellt. Die am Verbund teilnehmenden Agenten werden über die World registriert (*subscribe*). Weiterhin haben Simulationen die Möglichkeit der Darstellung eigener Objekt, welche unabhängig von Simulationsverbund sind. Diese können in die Welt integriert werden und interagieren nur mit den simulationseigenen Individuen. Über die *RootNode*, also dem Szenegraphen der Grafikengine, können beliebige Simulationsobjekte in die Visualisierung eingegliedert werden. Die Simulationen verwalten diese und können ihre Zustände beliebig verändern. Die in der World registrierten Entitäten haben durch ihren graphischen Repräsentanten eine Abbildung in der Welt. Dadurch kann die World Kollisionsberechnungen durchführen, wofür das Konzept der BoundingBoxes[18] genutzt wird.

Die World publiziert einen Update-Zyklus, den die Engine zum Spielfortschritt vorgibt. Die Simulationen können also an den Zyklus gekoppelt werden. Dies ist insbesondere für die Darstellung simulationseigener Objekte notwendig. Sie können auch je nach Agententypus, diesen über den Update-Zyklus einbinden. Auch andere Zeitmodelle sind für die Agenten hier denkbar (siehe Zeitmodelle). Die Simulation dient also im Grunde als Zusammenschluss der zugehörigen Agenten. Darüber hinaus ist es möglich, dass die Simulation noch weitere Aufgaben ausführt, wovon aber im Prototyp abgesehen wird.

4.3 Ökologie

Die Wahrnehmung und Interaktionsmöglichkeiten von Individuen werden durch das Framework bereitgestellt. Dabei werden die auftretenden Ereignisse, wie Wahrnehmung oder Kollision, von der Welt berechnet und über die Simulationen bekannt gegeben, welche diese Informationen gegebenenfalls an die Individuen weiterleiten können. Die Ökologie untersucht die Beziehung zwischen Individuen und ihrer Umgebung.

[18] Bounding Volumes werden in der Computergrafik genutzt, um effiziente Kollisionserkennung zu betreiben. Dazu wird eine Box um das eigentliche Objekt gelegt, welches die ungefähren Ausmaße abbildet und bei Überschneidungen mit anderen BoundingBoxes die Kollisionsdetektion aktiviert.

4.3.1 Kommunikation

Die Kommunikation ist der Kern sozialer Organisation (Uhrmacher et al., 2009). Individuen können jedoch nicht nur miteinander kommunizieren, sondern auch interagieren. Das bedeutet, dass sie auf Aktionen eines anderen Individuums reagieren. Zu diesem Punkt gehört auch der Austausch von Objekten oder Informationsträgern.

Es gibt zwei grundverschiedene Ansätze zur Darstellung von Austauschgütern. In der ersten Möglichkeit werden die Objekte nicht in der HLA sondern nur in den Simulationen abgebildet. Die RTI besitzt deshalb keinerlei Informationen über das Vorhandensein oder den Zustand etwaiger Interaktionsobjekte. Der Austausch findet über den Informationsaustausch durch Interaktionen, die gemäß einem Austauschprotokoll, im FOM festgelegt werden müssen. Das Nachrichtenprotokoll hat drei Stufen. Zuerst wird eine Anfrage an ein Individuum gestellt, welches ein bestimmtes Objekt erhalten soll. Dieses Individuum hat die Möglichkeit mit *OK* oder *NOK* (Not OK) zu antworten. Falls eine positive Reaktion kommt, kann der Sender mit einem finalen *OK/NOK* nach Überprüfung seiner Bedingungen, den Objektversand bestätigen und muss das eigene Objekt dann löschen. Der Empfänger kann das Objekt mit den (in der ersten Methode enthaltenen) Parametern instanziieren.

Der erste Ansatz geht von einem korrekten Verhalten der Interaktionsteilnehmer aus und stellt keine Überwachungsinstanz bereit. Dies wird im zweiten Ansatz durch die RTI umgesetzt. Dabei werden die Austauschobjekte nun als Attribute im FOM festgelegt und für jede Instanziierung mit einem *Handle* (ID, die über die gesamte Federation bekannt ist) versehen. Jedes Attribut kann zur Simulationszeit nur einem Federate gehören. Dies wird durch das Ownership Management sichergestellt, welches auch für den Ablauf der Objektübergabe bestimmte Dienste zur Verfügung stellt. Dieser Service arbeitet aber nur mit Federation-Genauigkeit, was bedeutet, dass innerhalb des Federates ein weiterer Mechanismus dafür sorgen muss, dass das Objekt nur zu dem zugehörigem Individuum gehören kann.

Die vorgestellten Ansätze beschreiben die Möglichkeit, Objekte auf Federate-Ebene auszutauschen. Die Individuen sind aber in einem Framework eingebettet, in dem sie mit anderen Individuen auf lokaler Ebene miteinander interagieren. Die Entitäten (so werden Individuen im Framework bezeichnet) haben die Möglichkeit, Objekte auszutauschen, wenn andere Entitäten in Reichweite sind. Dies läuft über einfache Methodenaufrufe, bei denen das Objekt mit übergeben wird. Der Rückgabewert der Methode ist ein boolescher Ausdruck mit folgender Semantik: *Angenommen* oder *Abgelehnt.* Wenn die interagierende Entität ein externes Individuum ist, dann kommuniziert das Framework den Datenaustausch auf Federate-Ebene und kapselt diesen Vorgang somit vor der Entität.

In allen Austausch-Konzepten hat das *angesprochene* Individuum die Möglichkeit den Datenaustausch abzulehnen. Dies liegt darin begründet, dass ein Individuum

erst einmal die Entscheidungsgewalt hat. Die Gründe für eine ablehnende Antwort können abhängig von den verschiedensten Bedingungen sein, die das Individuum für sich prüfen muss. Im Prototyp sind die beiden zu prüfenden Bedingungen der Platzmangel und die örtliche Entfernung.

Die in Abbildung 5 gezeigte Sequenz veranschaulicht die Kommunikation zwischen Individuen in einem lokalen Framework.

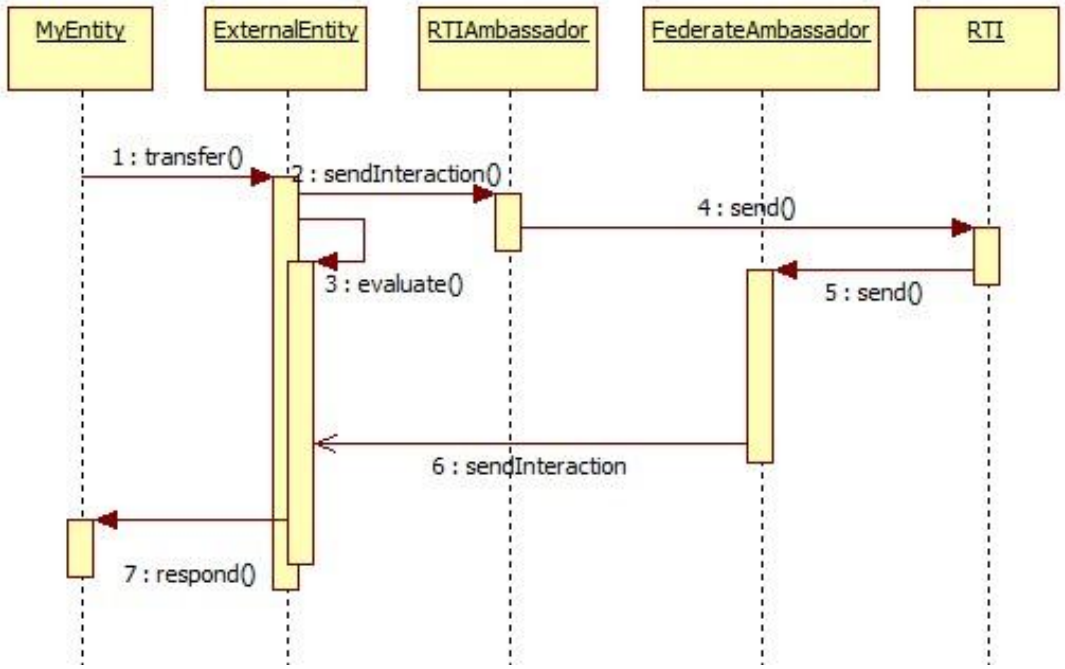

Abbildung 5: Ausführungssequenz bei Kommunikation über die HLA

Ein Agent löst diese Aktion aus. Ist die ExternalEntity im gleichen Framework integriert, so wird die Antwort direkt lokal zurückgeliefert. Ist der Agent jedoch aus einem anderen Framework (bzw. Federate), dann wird die Aktion über die RTI an die externe Simulation und dort an den jeweiligen Agenten geschickt. Der Agent hat dann die Möglichkeit auf die Anfrage zu antworten, bevor sie wieder über die RTI an das anfragende Framework zurückgeschickt wird. Dort nimmt der *FederateAmbassador* die Antwort entgegen, leitet sie an die ExternalEntity weiter, die wiederum auf die Anfrage antwortet. Der lokale Agent benutzt immer die gleiche Kommunikationsschnittstelle, unabhängig davon, welcher Agent sich hinter der ExternalEntity verbirgt. Die Frage, was in dem anderen Framework passiert, kann durch Abbildung 6 beantwortet werden.

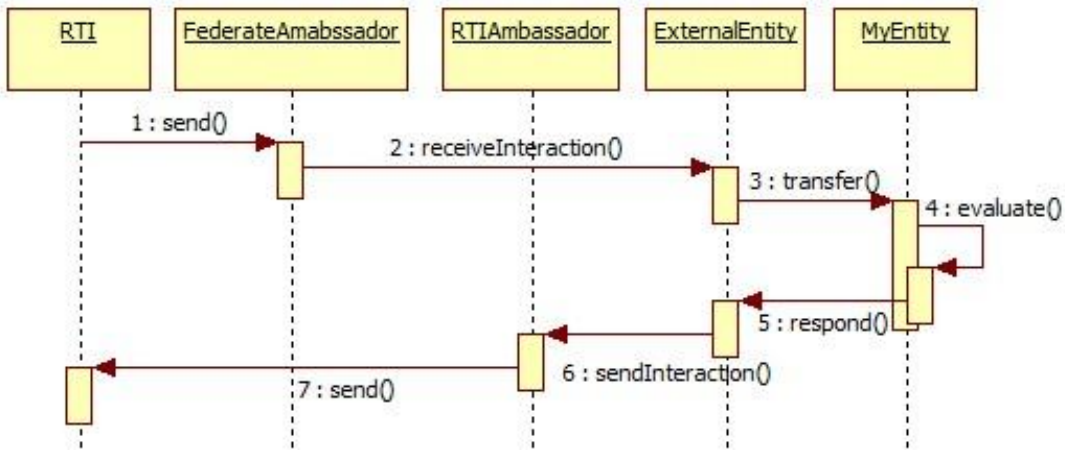

Abbildung 6: Ausführungssequenz Verarbeitung einer externen Anfrage

Eine externe Kommunikationsanfrage wird über den FederateAmbassador an die externe Entität geschickt. Diese repräsentiert den Agenten, welcher die Anfrage gesendet hat, in diesem Framework. Der angefragte Agent beantwortet die gekapselte externe Anfrage auf lokalem Weg. Seine Antwort wird über den *RTIAmbassador* wieder an das anfragende Framework geschickt. Dort geht der Ablauf nach dem Methodenaufruf 5 in Abbildung 5 weiter. Somit schließt sich der Kreis und die Kommunikation ist abgeschlossen.

4.3.2 Bewegung

Durch Individuen ausgeführte Aktionen führen zu einer Veränderung des Simulationszustands. So können Individuen die Umwelt beeinflussen oder auch sich selbst. Die Bewegung spielt dabei eine essentielle Rolle, da sie eine tragende Rolle in den individuenorientierten Simulationen einnimmt. Ein Individuum kann also einen Bewegungsvorgang initiieren. Das Sequenzdiagramm in Abbildung 7 veranschaulicht, welche Vorgänge sich dem anschließen.

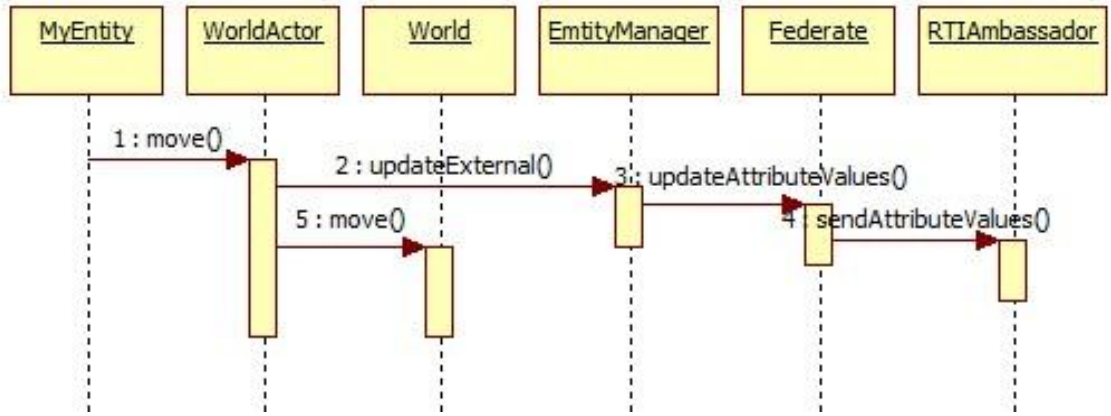

Abbildung 7: Ausführungssequenz bei lokalem Bewegungsvorgang

Auffällig in dieser Abbildung sind die zwei Stränge der Ausführung. Die Bewegungs-Anfrage wird an den Repräsentanten geschickt, welcher diese Aktion lokal in der Simulation ausführt. Parallel dazu wird diese Information über den EntityManager und schließlich dem RTIAmbassador an die RTI geschickt. Diese gibt diese Informationen an die anderen Federates weiter, welche im Falle eines Frameworks folgende Aktionen auslösen.

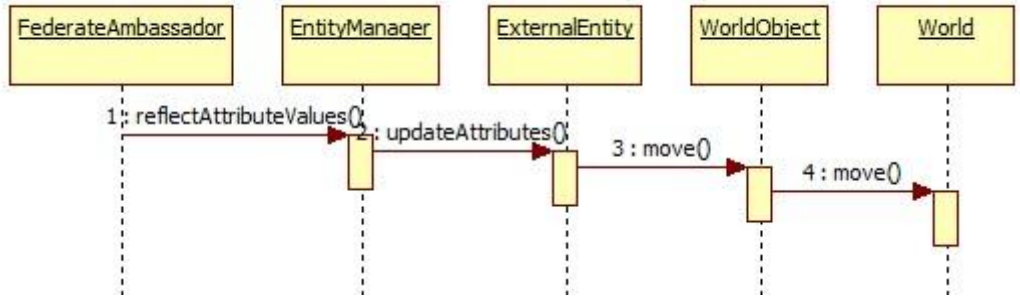

Abbildung 8: Ausführungssequenz bei externem Bewegungsvorgang

Abbildung 8 verdeutlicht den Synchronisationsaufwand bei Bewegungen externer Agenten. Die Informationen werden über den FederateAmbassador entgegengenommen und an die ExternalEntity weitergeleitet. Dieser wiederum setzt die Aktion des externen Agenten in der lokalen Simulationsumgebung um.

4.4 Zeitmodelle

Der vorgestellte Prototyp beinhaltet mit der jMonkeyEngine (JME, 2011) eine Spieleengine, die einen Update-Zyklus vorgibt und diesen an die Simulationsobjekte weitergibt. Alle teilnehmenden Objekte haben also in jedem Zyklus die Möglichkeit Aktionen auszuführen. Dadurch wird der Simulationsfortschritt reguliert und die zeitliche Reihenfolge der Simulation garantiert. Die Agenten können ihre Aktionen innerhalb des Zyklus ausführen, während die Grafikengine die Auswirkungen, wie z.B. den Bewegungsschritt, berechnet.

Der im Prototyp gewählte Ansatz ist möglicherweise der aus Spielesicht intuitivste. Es gibt jedoch noch weitere Möglichkeiten, die Individuen in den Zeitfortschritt zu integrieren. So könnten die Individuen in parallel laufenden Threads ausgelagert sein. In diesem Ansatz wären die Agenten vom eigentlichen Spielfortschritt losgelöst. Threads bieten den Individuen also mehr Freiheit und Unabhängigkeit. Jedes Individuum hat ein zugehöriges Weltobjekt, welches wie die äußere Hülle des Agenten betrachtet werden kann (während das Individuum den Intellekt abbildet). Dieses Weltobjekt hält die äußeren Zustände des Individuums (z.B.: Position, Geschwindigkeit) und ist innerhalb des Frameworks in jedem Fall dem Update-Zyklus unterworfen. So kann die Welt zyklisch die Bewegungen und Koordinationsdaten neu berechnen. Das Individuum kann lediglich diese Informationen abfragen und innerhalb seines Zeitfensters darauf reagieren. Ist ein Agent also als eigenständiger Thread umgesetzt, so kann es zeitunabhängig agieren. Ist es jedoch, wie im oben beschriebenen Ansatz, ebenfalls dem Update-Zyklus unterworfen, so bekommt es in jedem Zyklus, also nur nach Änderung der äußeren Rahmenbedingungen, die Möglichkeit, auf die Umstände zu reagieren. Diese Frequenz ist in der Regel auch ausreichend, da der Agent bei gewöhnlicher Simulationsauslastung ca. alle 50ms sein Zeitfenster bekommt.

Ein dritter Ansatz ist die Umsetzung eines rein ereignisbezogenen Agenten. So müssen dem Agenten alle relevanten äußeren Umwelteinflüsse bekannt gemacht werden. Diese Events, zu denen beispielweise auch die Wahrnehmung gehört, benachrichtigen den Agenten und erlauben ihm, darauf zu reagieren. In einem Simulationsabschnitt in dem der Agent sich weder bewegt noch seine Umwelt entscheidend auf ihn einwirkt, würde der Agent keinerlei Aktionszeiten haben. Nichtsdestotrotz ist diese ein sehr effizienter Ansatz, um das Verhalten eines Agenten zu simulieren. Weiterhin müsste auch bei Veränderungen der inneren Zustände dem Agenten Reaktionszeit zur Verfügung gestellt werden.

Im Prototyp werden die Simulationen an den Update-Zyklus angebunden, welche diesen an die Agenten weitergeben können. Die Simulationen und die Agenten sind jedoch nicht darauf angewiesen, sich diesem Zyklus unterzuordnen. Die Wahl des Zeitmodells liegt somit ganz bei den Simulationen.

4.5 Agenten

Das Framework soll möglichst wenige Einschränkungen für die Agenten vorgeben. Deswegen gilt es zu untersuchen, wie ein Agent möglich generisch beschrieben werden kann, um dennoch an das Framework angebunden werden zu können. Dazu wird die Repräsentation des Agenten in den Stellvertreter ausgelagert (siehe 4.2). Über diesen kann er Aktionen ausführen, die in der Welt abgebildet sind. Will er mit der Umgebung interagieren, so muss die Umgebung auch die Befehle verstehen. Somit ist die Aktionsvielfalt von vornerein begrenzt.

Das Hauptaugenmerk liegt also auf dem Verhalten des Agenten. Welche Umstände führen den Agenten dazu, eine bestimmte Aktion auszuführen? Dieser Vorgang ist letztendlich ein Kreislauf, wie Abbildung 9 darstellt. Der Agent beeinflusst die Umgebung und diese wiederum den Agenten und seine Entscheidungen.

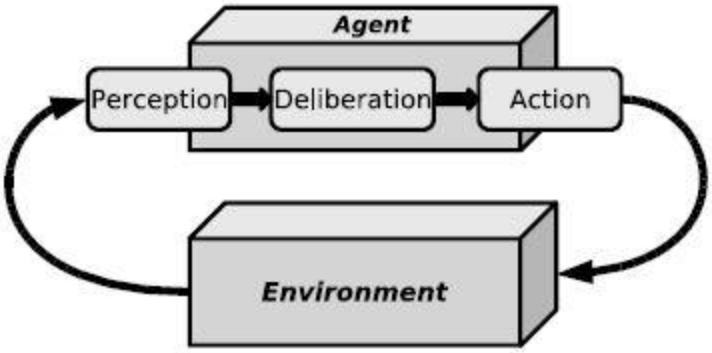

Abbildung 9: Agenten zu Umgebung (Uhrmacher et al., 2009)

Der Agent kann die Umgebung durch Sensoren wahrnehmen. Diese geben ihm Auskunft über den äußeren Zustand. In der Architektur ist dies über die perceive-Methode gelöst, welche dem Agenten Informationen über Ereignisse in seinem Einflussgebiet liefert. Die Wahrnehmung ist also umfassen, da von der weltlichen Fehlerhaftigkeit der Sinne abstrahiert wird und die Sensoren alle Informationen bereitstellen.

Der interessante Punkt beim Agenten ist also die Entscheidungsfindung. Es gibt eine Vielzahl von Agententypen, welche unterschiedliche Mechanismen einsetzen um *Lösungen* zu finden. Ein weit verbreitetes Beispiel bilden die BDI-Agenten (*Belief-Desire-Intensions*), welche drei unterschiedliche Datenstrukturen berücksichtigen: Weltwissen, Ziele und Absichten des Agenten. Ein allgemeinerer Ansatz ist die Entscheidungsfindung auf Basis der Beschreibungslogik. Dies ist eine Form der Wissensrepräsentation, die formal in zwei Teile gegliedert ist. Der erste Teil enthält das terminologische Wissen, also das Wissen über die Konzepte einer Domäne. Der zweite Teil beschreibt die Instanzen und Entitäten dieser Domäne,

also den aktuellen Zustand der modellierten Welt (Spalthoff, 2005). Der Agent kann also über die Beschreibungslogik das Hintergrundwissen über die Welt mit dem aktuellen Zustand der Umgebung kombinieren. Die Beschreibungslogik ist entscheidbar, was bedeutet, dass aus vorhandenem Wissen, neues Wissen generiert werden kann. Der Agent kann somit schlussfolgern und darauf aufbauend seine Entscheidungen treffen.

4.6 Alternativen

In der gewählten Architektur ist das Framework der Federate. Das Framework kann beliebig viele Simulationen integrieren, ist also schon ein lokaler Simulationsverbund. Es besteht von jeder Instanz des Frameworks, und nicht von jeder integrierten Simulation, genau eine Verbindung zur RTI.

Alternative Möglichkeiten könnten sein, dass jede Simulation oder noch feingranularer, jeder Agent (dann als eigenständige Simulation) als Federate fungiert. Bei diesem Konzept wäre der Verwaltungsaufwand über die HLA deutlich höher. Vorteilhaft wäre aber die größere konzeptionelle Freiheit einer jeden Simulation (Agenten), da ein weniger restriktiveres (oder kein) Framework gewählt werden könnte.

4.6.1 Simulations-Ansatz

Ein alternativer Ansatz zum Framework-Konzept könnte die Abbildung jeder eigenständigen Simulation, die an der verteilten Simulation teilnimmt, durch einen Federate sein. Damit einher geht die Notwendigkeit, jede Simulation anzupassen, damit sie HLA fähig ist und der speziellen Anforderungen der verteilten Simulation genügt (siehe HLA Erweiterbarkeit). Dies betrifft vor allem die Punkte, welche vorher durch das Framework umgesetzt wurden. Primär muss die Kompatibilität zum SOM gewährleistet sein. Weiterhin müssen Umgebung und Zeitverhalten mit dem Simulationsverbund abgestimmt werden. Das kann mitunter zu großen Anpassungen bezüglich des Zeitverhaltens führen. Während der Entwicklung der unterschiedlichen Simulation wiederholen sich bestimmte Codeabschnitte, die für die Synchronisationsabläufe mit der HLA notwendig sind. Das resultiert in wiederkehrenden Entwicklungsaufgaben, welche die Entwicklungszeit durch diese Routineaufgaben verlängert. Die Vorteile des Frameworks verschwinden also, damit aber auch seine Restriktionen.

Eigenständige Simulationen können ein Zeitmodell umsetzen, welches den Anforderungen der jeweiligen Simulation am Besten entspricht. Die HLA unterstützt die unterschiedlichsten Modelle, wogegen das Framework das Time-Stepped Paradigma umsetzt. Es kann also ein viel größere Kategorie von Simulationen am

Verbund teilnehmen. Weiterhin haben Simulationen einen gewissen Bedarf bezüglich der Korrektheit des Modells. Das betrifft primär die Synchronisation der beweglichen Objekte, kann jedoch auch Auswirkung auf weitere Bereiche haben. Dieser Authentizitätsbedarf in Bezug auf die Gesamtsimulation kann durch die Nutzung der HLA-TM Dienste befriedigt werden. In das Framework eingebundene Simulationen können deshalb nur den durch das Framework bereitgestellten Korrektheitsgrad erreichen.

Die (räumlich) verteilte Entwicklung von Simulationen ist durch den Simulations-Ansatz flexibler als mit Hilfe eines Frameworks. In einem evolutionären Entwicklungsprozess werden frühere Phasen der Entwicklung möglicherweise zu späteren Zeitpunkten wiederholt. Damit kann man den sich verändernden Anforderungen und Erkenntnissen gerecht werden. Im Fall der verteilten Simulationen bedeutet es, dass die Konsistenz zu den Schnittstellen erfüllt sein muss. Also müssen Änderungen im Datenmodell (SOM, FOM) auch in den Simulationen umgesetzt werden. Im Falle des Frameworks, muss das Framework ebenso diese Veränderungen unterstützen. Nachträgliche Änderungen erfordern also ein neues Framework, worauf sich wiederum die verteilte Entwicklung stützt. Die Loslösung vom Framework ergibt im gewissen Sinne auch eine schnellere und unabhängigere Entwicklungsmöglichkeit der Simulationen unter Rücksichtnahme der Datenmodelle.

Weiterhin soll noch auf den semantischen Vorteil der Anbindung einzelner Simulationen eingegangen werden. Dies ermöglicht spezifisch leistungsstärkere Agenten, da Agenten untereinander (innerhalb der Simulation) kommunizieren können, ohne der HLA-Schnittstelle unterworfen zu sein. Wenn beispielweise eine Familie in einer Fußgängersimulation simuliert wird, so interagieren die Familienmitglieder in besonderer Weise miteinander, im Vergleich zu einzelnen Fußgängern (vgl. Thiel-Clemen et al., 2011). Dabei wird z.B. Rücksicht aufeinander genommen und beispielweise bei Bedarf die Fortbewegung untereinander koordiniert. Dies erfordert einen Kommunikationsaufwand, der nicht notwendigerweise in HLA abgebildet werden muss, sofern die Familienmitglieder alle in der gleichen Simulation agieren.

Zusammenfassend kann gesagt werden, dass der Simulations-Ansatz grundsätzlich mehr Entwicklungsaufwand erfordert. Jedoch sind die Simulationen im Gegenzug nicht den Anforderungen des Frameworks unterworfen. Dies wird, durch die Möglichkeit schon bestehende Simulationen in den Verbund zu integrieren, unterstrichen. Somit können auch Simulationen mit variierenden Eigenschaften integriert werden, unabhängig von ihrem Zeitverhalten, der Programmiersprache und dem System, auf dem sie ablaufen (siehe Verteilte Simulationen).

4.6.2 Agenten-Ansatz

Eine konzeptionelle Überlegung kann es sein, den Schwerpunkt der Simulation auf das Verhalten einzelner Agenten zu legen. Es ist denkbar, die Simulationen in dem Maße zu unterteilen, dass die einzelnen Agenten jeweils als Federate fungieren. Alle weiteren simulationsrelevanten Aufgaben müssen dann, so fern möglich, über die HLA gelöst werden. Die Folge ist ein erhöhter Verwaltungsaufwand der HLA, durch die Verlagerung auf eine höhere Abstraktionsebene. Zu diesen Aufgaben zählen unter anderem die gesamte Wahrnehmung der Agenten, wie auch im Speziellen die Kollisionsdetektion. Ebenfalls wird die komplette Kommunikation zwischen den Agenten dann nur über die HLA übermittelt. Die HLA stellt bestimmte Dienste und Mechanismen bereit, wie z.B. die Routing Spaces, um eine effiziente Filterung der Kommunikation durchzuführen. Damit können die Agenten nur noch mit Informationen versorgt werden, die in ihrem Wahrnehmungsgebiet liegen. Dazu müssen die Agenten sich selbst als Objekt, welches je nach Agententyp in dem SOM definiert ist, registrieren und verwalten.

Zur graphischen Repräsentation des Simulationsvorgangs können, wie bei *Klein et al.* (Klein et al., 1998b), weitere Federates eingebunden werden, die als reine Beobachter fungieren und den Simulationsfortschritt visualisieren.

Ein dem Agenten umgebendes Framework ist sehr schlank, da es bezüglich der Interaktion nicht auf eine World angewiesen ist. Dennoch fallen einige Aufgabengebiete auch dem Agenten zur Last. Da der Austausch von geographischen Daten im bisherigen Ansatz noch außerhalb des HLA-Aufgabenbereichs liegt, müssen die Agenten schon vor Simulationsbeginn mit den geographischen Informationen ausgestattet sein. Alternativ ist die Bereitstellung dieser Informationen über ein GIS denkbar, welches den Agenten nur mit Daten innerhalb ihres Wahrnehmungsbereichs versorgen kann. Die Agenten müssen jedoch verstehen können, wie sie mit diesen Daten umzugehen haben. Konsistenzmechanismen, wie die Schwerkraft, die Geschwindigkeitsbegrenzung und die Undurchlässigkeit von Hindernissen sind ebenfalls durch die Agenten zu gewährleisten. Die HLA hat keine Möglichkeit diese Kriterien zu beeinflussen.

Will sich also ein Agent fortbewegen, so verändert er die Attribute seines Repräsentanten, also des Objekts, welches in der HLA abgebildet ist. Dazu zählt beispielweise die Position des Agenten. Im weiteren Zeitverlauf muss diese Position, während eines Bewegungsvorgangs, aktualisiert werden. Das alleinige Modifikationsrecht besitzt jedoch der Agent. Dieser ist demzufolge für seine Bewegung selbst verantwortlich. Im Simulations-Ansatz wird dieser Vorgang folgendermaßen umgesetzt. Der Agent hat die Möglichkeit eine Bewegung zu beantragen, die Bewegung selbst wird dann jedoch durch die Simulation durchgeführt und auch in der HLA beschrieben. Die Simulation aktualisiert also die Attribute des Objekts.

Der Agenten-Ansatz erfordert also die Aufteilung der Simulationsaufgaben auf Agent und HLA. So sind Agenten nun mehr dafür verantwortlich korrektes Verhalten zu simulieren. Die Auslagerung bestimmter Aufgaben in die HLA erhöht jedoch stark die Wiederverwendbarkeit des Modells. Es ist somit das im höchsten Maße generische System, welches für MAS vorstellbar ist (siehe Leitbilder). Dies kommt auch darin zum Ausdruck, dass es keine Begrenzungen bezüglich der Agententypen gibt. So sind Individuen nicht an ein Zeitmodell gebunden und können auch ohne regelmäßigen Update-Zyklus nur auf bestimmte Ereignisse (Event-orientiert) reagieren. Der gesamte Input wird vorgefiltert durch die HLA bereitgestellt und jegliche Aktion und Kommunikation wird über die HLA Schnittstelle bekannt gegeben.

Dieser Ansatz bietet den Individuen eine große Gestaltungsfreiheit, die sehr spezielle Funktionen und Aufgaben erfüllen können, ähnlich dem Simulations-Ansatz. Es gilt jedoch noch zu evaluieren, wie viele Agenten an der Federation teilnehmen können, um die Kapazitätsgrenze der RTI zu erreichen. Dies erhält besondere Bedeutung vor den vielen Aufgaben, welche die RTI bei diesem Ansatz umsetzen muss.

5 Umsetzung

Dieses Kapitel beschäftigt sich mit den Umsetzungen und den Einzelheiten des vorgestellten Architekturentwurfs. Dabei steht nicht nur die Implementierung des Frameworks im Blickpunkt, sondern auch das Erstellen einer exemplarischen Simulation zur Validierung des Frameworks. Zu diesem Zwecke wird eine verteilte Schiffsimulation entwickelt, die zum einen die Interaktion von beweglichen Individuen durch verschiedene Schiffstypen und zum anderen die Kommunikation zwischen den Agenten abbildet, was in diesem Fall durch den Austausch von Containern umgesetzt wird.

5.1 Szenario

Das im Prototyp umgesetzte Szenario handelt von einer Schiffsimulation, die ein Erzeuger-Verbraucher-System umsetzt. Sie ist in drei unterschiedlichen Simulationen aufgeteilt, welche jeweils einen anderen Individuentyp verwalten. Diese tauschen Container aus und verbrauchen sie gegebenenfalls.

Die erste Simulation beinhaltet Häfen, die an fixen Positionen lokalisiert sind und Container auf und von Schiffen verladen. Häfen können eine beliebige Anzahl von Containern von Schiffen entgegennehmen. Diese werden dann nach einer gewissen Zeitspanne verarbeitet. Dies wird durch die Zuordnung der Hafenfarbe gekennzeichnet. Ankommende Schiffe werden entladen, sobald sie sich im Einzugsgebiet befinden. Anschließend werden die Schiffe mit den in Hafenfarbe markierten Containern neu beladen, bis deren Kapazitätsgrenze erreicht ist. Häfen sind also gleichzeitig Erzeuger und Verbraucher. Sie können jedoch nur Container anderer Hafenfarbe verbrauchen, weshalb sie auf den Transport der Container durch Schiffe angewiesen sind.

Die zweite Simulation besteht also aus Containerschiffen, welche von Hafen zu Hafen pendeln, um dort jeweils Container auszutauschen. Dazu befahren sie feste

Routen und ankern in Dockreichweite. Die Containerschiffe haben die niedrigste Priorität und müssen somit allen anderen mobilen Individuen ausweichen.

Als dritte Simulation fungieren Tanker, welche dieselben Gewässer befahren und nur als bewegliches Bezugssystem für Ausweichmanöver der Containerschiffe dienen.

5.1.1 Container

Die im Szenario auszutauschenden Container dienen als Interaktionsgut der Simulation. Die Container werden als ein ideelles Gut umgesetzt, das nicht in der HLA abgebildet wird. Sie werden jedoch als Objektinstanzen von Schiffen und Häfen gehalten. Bei dem Austausch wird das jeweilige Objekt an den Empfänger übergeben, sofern dieser die Annahme nicht verweigert. Nimmt der Empfänger den Container an, so antwortet er positiv auf die Methode und der Sender löscht die Objektreferenz. Der Austausch findet somit lediglich lokal zwischen lokalen und externen Entitäten statt. Im Hintergrund laufen jedoch die Kommunikationsmechanismen ab (siehe 4.3.1).

Die Container werden damit letztendlich über Interaktionen ausgetauscht. Sie bilden also exemplarisch eine Sprachform zwischen den Agenten ab. Die semantische Bedeutung der Container und der damit verbundenen lokalen Verwaltungsaufwand der Objekte, spielt eine untergeordnete Rolle. Es können auf diese Weise beliebige Informationen ausgetauscht werden, die nicht an eine Objektinstanz gebunden sind. Dennoch verdeutlicht der Containeraustausch exemplarisch einen Ansatz des Austauschs von Interaktionsgütern.

5.2 Realismus

Die Simulation eines zu untersuchenden Systems bedarf immer gewisser Kriterien bezüglich des Abbildungsgrades. So kann von einigen Begebenheiten abstrahiert werden, die keinen Einfluss auf das Simulationsszenario haben. Es müssen jedoch die vorgegebenen Constraints in der Simulation abgebildet werden um den geforderten Authentizitätsgrad zu erreichen.

In dem Szenario der Schiffsimulation soll auf zwei Punkte eingegangen werden, welche die Bewegungsform der Schiffe betrifft. Diese sind in einem echten System, den physikalischen Gesetzten unterworfen. Die Höhenverschiebung durch Wellen wird ignoriert, jedoch soll die Manövrierfähigkeit möglichst wahrheitsgetreu abgebildet werden. So ist die Veränderung der Geschwindigkeit ein dynamischer Prozess, indem das Tempo des Schiffes inkrementell der gewünschten

Geschwindigkeit angepasst wird. Der Prototyp nutzt den Update-Zyklus um die Geschwindigkeit schrittweise anzupassen.

Der zweite Punkt betrifft die Steuerung des Schiffes. Dabei kommt das Prinzip der Trägheit zur Geltung, bei dem ein Bewegungsimpuls in eine bestimmte Richtung ohne Krafteinwirkung fortbesteht. Bei einem Richtungswechsel bleibt dieser Impuls für eine gewisse Zeitdauer vorhanden und kann nur mit der Zeit abgeschwächt und verändert werden.

Die Bewegungsrichtung wird im Prototyp durch einen dreidimensionalen Vektor dargestellt. Wählt der Agent eine andere Richtung aus, so wird der aktuelle Vektor schrittweise an den neuen Vektor angenähert. Dadurch schwimmt das Schiff eine Zeit lang in die initiale Richtung weiter, bevor sie langsam den Kurs wechselt. Dieses Verhalten ist im WorldActor umgesetzt und somit durch das Framework vorgegeben. Die Agenten sind dadurch in das physikalische System eingebettet und können kein Verhalten ausführen, welches nicht den Szenariovorgaben entspricht. Die Umsetzung anderer Typen von Individuen kann ein anderes Bewegungsverhalten erfordern. Das Framework muss also dem Szenario angepasst werden. Dennoch können unterschiedliche Bewegungstypen in einem Szenario umgesetzt werden, in dem das Bewegungsverhalten abhängig vom Agenten ausgewählt wird.

5.3 Ausweichmanöver

Wenn sich Individuen auf begrenzten Raum zur selben Zeit fortbewegen, so besteht die Möglichkeit von Kollisionen. Individuen sind in der Regel darin bestrebt einander auszuweichen, wie man es z.B. bei Fußgängern auf Gehwegen beobachten kann. Dabei ist eine gewisse Priorität von zielgerichteten eiligen Menschen gegenüber schlendernden, eher langsameren Menschen festzustellen. Weitere Einflüsse können dieses Verhalten beeinflussen, was dazu führt, dass auch *höher priorisierte* Teilnehmer ihre Bewegungsrichtung geringfügig anpassen (Huttner, 2011). Dieser Prozesssoll an dieser Stelle nicht weiter ausgeführt werden. Man kann aber zusammenfassen, dass bei auf Kollisionskurs liegenden Individuen, eine gewisse Priorität (im Fußgängerbeispiel vielleicht auch als Dringlichkeit bezeichnet) über das Ausweichverhalten entscheidet. Ein anderes Beispiel liefert der Straßenverkehr in dem das Vorfahrtsrecht (meist) eindeutig vorgegeben ist.

Auch bei der Schiffsimulation soll die Ausweichpflicht über Prioritäten geregelt werden. Dabei wird zur Vereinfachung festgelegt, dass Containerschiffe gegenüber Tankern immer eine geringere Priorität haben. Dabei wurde bisher ein weiteres Kriterium außer Acht gelassen: die unbeweglichen Objekte. Individuen weichen unbeweglichen Objekten in der Regel intuitiv aus. Für den Fall, dass bewegliche Individuen keine Bewegungsgeschwindigkeit aufweisen, werden sie von anderen

Individuen als unbeweglich angesehen. Ein unbewegliches Objekt hat also die höchste Priorität, so auch ein im Hafen ankerndes Containerschiff, dem dann auch ein Tanker ausweichen muss, sofern die Ankerposition auf der Schifffahrtsroute des Tankers liegt.

Als *Ausweichmanöver* wird im ersten Schritt gestoppt und gewartet, bis die Route wieder frei befahrbar ist. Komplexere Ansätze berechnen den Kurs des auf Kollisionskurs befindlichen Objekts und passen dementsprechend Geschwindigkeit und Bewegungsrichtung an. Über Wegfindungsalgorithmen kann ein sinnvoller Kurs bestimmt werden. Die Berechnung muss jedoch dynamisch wiederholt werden, da das Hindernis ebenfalls mobil ist und seine Route verändern kann. Im Prototyp wird der häufig verwendete A*-Algorithmus[19] eingesetzt.

5.4 Zeitmanagement

Das Framework basiert auf einer Grafikumgebung: der *jMonkeyEngine* (JME, 2011), die eine feste Zeittaktung vorgibt, sie ist also zeitregulierend. Um einen einheitlichen (oder zumindest ähnlichen) Zeitfortschritt aller Frameworks zu gewährleisten, muss die Taktfrequenz in allen Systemen identisch sein. Es wird im Prototyp eine lose, ungenaue Zeitgenauigkeit umgesetzt. Die Systeme arbeiten unabhängig voneinander in ihrem Zeitrhythmus (mit globaler Taktfrequenz) und integrieren die Ereignisse der anderen Federates zeitverzögert um die Zeit der Übertragung durch die HLA. Die Verzögerungen haben nur einen minimalen Einfluss auf die Simulationsgenauigkeit, so lange die Übertragungszeit eine bestimmte relative Größe zur Taktfrequenz nicht überschreitet. Durch eine relativ hohe Taktung werden die Bewegungsschritte in relativ kleinen Streckenabschnitten im Vergleich zu der Größe der simulierten Objekte. Auch bei der Wahrnehmung ist die Reichweite (wie in der *echten Welt*) kein exaktes Maß, sondern hängt von vielen unterschiedlichen Faktoren ab. Die im Prototyp vorhandene Ungenauigkeit kann deshalb im Rahmen der Schiffsimulation vernachlässigt werden.

Die in der HLA bereitgestellten Synchronisationsmechanismen führen zu einer erhöhten Genauigkeit der Simulation. Dabei wird die globale Zeit durch die RTI reguliert und die Federates müssen ihren lokalen Zeitfortschritt der globalen Simulationszeit anpassen. Insbesondere bei Einbindung von Simulationen außerhalb des Frameworks besteht die Notwendigkeit der Zeitregulierung, wie sie im Kapitel 2.2.5 beschrieben sind.

[19] Suchalgorithmus, der dazu genutzt werden kann, in einem Graphen die kürzeste Verbindung zwischen zwei Knoten zu finden.

5.5 HLA-basierte Entwicklung

Bei der Erstellung einer HLA-basierten Simulation kann der FEDEP (*Federation Development and Execution Process*) genutzt werden um einen definierten Entwicklungsprozess durchzuführen. Dieser Ansatz ist speziell an die Anforderungen, die bei der verteilten Entwicklung aufkommen, angepasst. Das evolutionäre Vorgehensmodell sieht verschiedene Phasen vor, die in Abbildung 10 dargestellt sind.

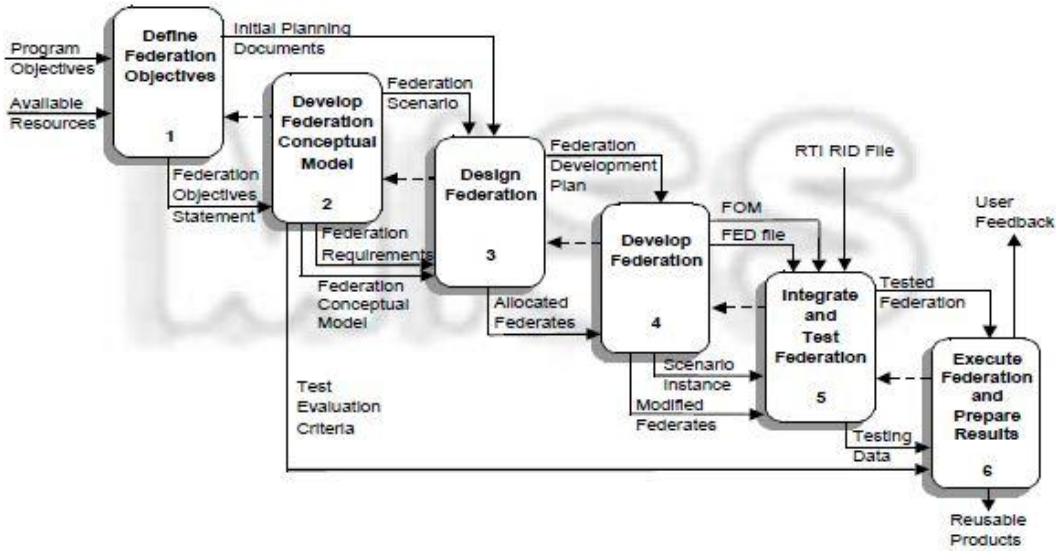

Abbildung 10: Six Step Process der FEDEP (Crosbie et al., 1999)

Zunächst müssen die Bedürfnisse und Entwicklungsziele bestimmt werden. Darauf aufbauend kann im zweiten Schritt das Szenario entwickelt werden. Außerdem muss eine konzeptionelle Analyse durchgeführt werden, die zu den Anforderungen der Federation führen. Im dritten Schritt werden die einzelnen Federates voneinander abgegrenzt, indem ihnen Funktionalität zugeordnet wird. Der vierte Schritt erfordert nun die Definition des FOM. Darauf aufbauend können dann die Federates entwickelt werden. Im fünften Schritt werden die Federates in die Federation integriert, wodurch dann ein Testbedarf entsteht. Im abschließenden Schritt wird die verteilte Simulation dann ausgeführt. Dadurch werden Informationen generiert, die zu Ergebnissen zusammengefasst und interpretiert werden.

Das FEDEP-Modell ist konzipiert für die verteilte Entwicklung in Teams. Dennoch sind die Phasen auch bei der Entwicklung des Prototyps von Bedeutung gewesen. Dadurch, dass der FEDEP ein evolutionäres Vorgehensmodell ist, eignet er sich gut für die experimentelle Entwicklung.

5.5.1 HLA im Framework

Die verschiedensten Ansätze und Möglichkeiten, welche die HLA bietet, wurden bereits ausführlich diskutiert (siehe 2.2). Nun soll ein Einblick in die spezielle Umsetzung gewährt werden.

Die in dem FOM definierten Objekte werden bei der Instanziierung im Framework auch in der RTI registriert. Diese Erzeugt daraufhin das Objekt in der HLA und ordnet diesem einen Schlüssel zu – das Handle. Über dieses Handle kann das Objekt angesprochen werden, um dessen Attribute zu verändern. Unterschieden werden muss dabei zwischen dem *ClassHandle* und dem *ObjectHandle*. Ersteres identifiziert den Objekttyp und das Zweite die Referenz zu der registrierten Objektinstanz. Abbildung 11 zeigt einen Ausschnitt aus der verwendeten FED. Die dargestellte Klasse *Entity* hat eine Menge von Attributen, welche über einen bestimmten Namen identifiziert werden können. Diese Klasse dient im ersten Ansatz zur Abbildung aller Entitäten im Prototyp.

```
12    (objects
13        (class ObjectRoot
14            (attribute privilegeToDelete reliable timestamp)
15            (class Entity
16                (attribute name reliable timestamp TestSpace)
17                (attribute moving reliable timestamp TestSpace)
18                (attribute positionX reliable timestamp TestSpace)
19                (attribute positionY reliable timestamp TestSpace)
20                (attribute positionZ reliable timestamp TestSpace)
21                (attribute directionX reliable timestamp TestSpace)
22                (attribute directionY reliable timestamp TestSpace)
23                (attribute directionZ reliable timestamp TestSpace)
24                (attribute velocity reliable timestamp TestSpace)
25                (attribute size reliable timestamp TestSpace)
26                (attribute type reliable timestamp TestSpace)
27                (attribute viewRange reliable timestamp TestSpace)
28            )
```

Abbildung 11: Ausschnitt aus dem FED der verteilten Schiffsimulation

Werden Informationen an andere Federates publiziert, so muss der Informationsgehalt der Methoden in Nachrichten verpackt werden, welche die RTI verarbeiten kann. Zur Aktualisierung eines Objekts in der HLA muss die Nachricht mit der Objektreferenz, also dem ObjectHandle, versehen werden. Die jeweiligen Attribute können über ein weiteres Handle identifiziert werden, welches durch den Namen und dem ClassHandle bei der RTI erfragt werden kann. Die Werte müssen unter Zuhilfenahme von Encoding Helpers verschlüsselt werden, um der FOM-

Semantik gerecht zu werden (siehe 2.2.6). Die gesamten Informationen werden in einer HLA-spezifischen Map[20] gesammelt und können dann an die RTI verschickt werden. Die Handles werden auch dazu benutzt, eingehende Informationen zu verstehen.

Bei der Versendung von Interaktionen ist der Ablauf ähnlich. Auch Interaktionen sind im FOM spezifiziert und haben somit ein ClassHandle, welches dazu dient, mit Hilfe des Attributnamens, die Attribute zu identifizieren. An Interaktionen kann jedoch auch noch zusätzlich eine beliebige Textnachricht gehängt werden. Die semantische Bedeutung kann ähnlich, wie im Containeransatz, genutzt werden. Dies ist jedoch nicht erforderlich, da Interaktionen ohnehin bestimmte Attribute beinhalten.

Abschließend soll noch der Ablauf der Registrierung von Objekten beleuchtet werden. Die Objekte werden beim Registrierungsprozess an die anderen Federates bekannt gegeben, sofern die Federation einen Zeitschritt fortschreitet, also einen *Tick*[21] beantragen. Dabei wird lediglich Name, Identifikationsnummer der Klasse und des Objekts bekannt gegeben. Es werden jedoch keine Informationen über die Beschaffenheit, Typ oder die Attribute des Objektes preisgegeben. Diese Angaben werden lediglich im FOM bereitgestellt. Die exakte Beschreibung und Interpretation ist Grundvoraussetzung für die Korrektheit der Daten und das Funktionieren der Gesamtsimulation.

---

[20] Assoziative Speicherstruktur

[21] Anstoßen eines Prozesses, der daraufhin (erneut) Bedingungen überprüft

6 Ergebnisse

Der entwickelte Prototyp kann als Testumgebung genutzt werden, um Aussagen über das Framework treffen zu können. Es wird ein Testlauf durchgeführt, bei dem das Szenario der Schiffsimulation untersucht wird. Bei der Durchführung werden gewisse Kriterien untersucht, welche die Lauffähigkeit und Einsatzbereitschaft der HLA verifizieren sollen.

6.1 Testdurchlauf

Das Testszenario sieht die Durchführung des Erzeuger-Verbraucher-Systems vor. Dieses kann nur durch die Transporteure, also den Containerschiffen, funktionieren. Um die Funktionstüchtigkeit des Prototyps zu testen, werden also zwei Frameworks über HLA miteinander verknüpft. Das eine Framework enthält die Simulationen der Häfen und der Tanker. Ein zweites Framework stellt die Containerschiffe bereit. Die Simulationen stellen jeweils zwei Agenten ihres Typus. Die Agenten starten in vordefinierten Positionen und haben feste Routen, die sie befahren.

Zur Veranschaulichung wird in Abbildung 12 eine Momentaufnahme während der Simulation abgebildet. Zu sehen sind zwei Schiffe und ein Hafen. Das äußere Erscheinungsbild der Individuen ist aus Performancegründen vereinfacht. Die Schiffe werden durch Zylinder dargestellt, der Hafen durch einen Würfel. Die Agenten sind von Kugeln umgeben, welche den Wahrnehmungsbereich visualisieren.

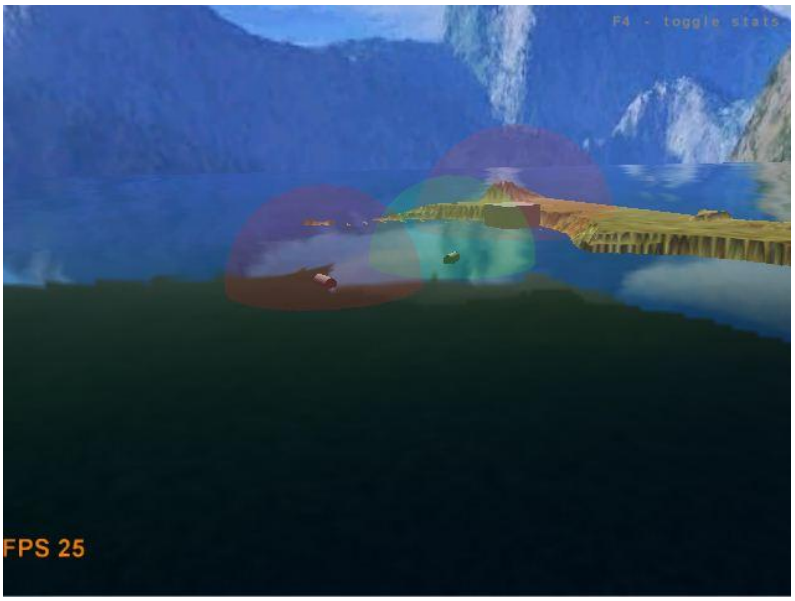

Abbildung 12: Schiffsimulation im Testbetrieb

Die wiederholte Durchführung der Simulation konnte verifizieren, dass die Kommunikation mit wenigen teilnehmenden Agenten funktioniert. Die verteilten Simulationen können so zusammen arbeiten, dass der Simulationsverbund, wie eine monolithische Simulation abläuft. Die Frameworks können sich auch auf verschiedenen Rechnern im Netzwerk befinden, ohne dadurch den Simulationsablauf negativ zu beeinflussen. Die HLA kann die Verbindung aufbauen und ausreichend effiziente Abläufe gewährleisten. Die Visualisierung beider Frameworks zeigt zwei identische Simulation. Positionsmessungen zeigen kaum Abweichung zwischen den lokalen Entitäten und ihrem externen Stellvertreter. Der Prototyp kann also eine sehr einfache Simulation von Individuen, die über die HLA kommunizieren, umsetzen.

Probleme gibt es bei dem Versuch die Last zu erhöhen. Die Steigerung der Individuenanzahl ist im Testlauf nicht durch Portico begrenzt, sondern durch Performanceprobleme, welche durch die Grafikengine ausgelöst werden. Die maximale Teilnehmerzahl, welche Portico unterstützt gilt es in Zukunft zu analysieren, so bald die Grafikoberfläche effizienter gestaltet wird.

Die Testdurchläufe wurden auf einem PC mit Intel® Core™ Duo mit 2GHz, 3GB Ram und einer Mobile Intel 965 Express Grafikkarte, so wie einem Rechner mit AMD® Athlon™ XP 2600+, 1GB Ram und einer Geforce 4600GT Grafikkarte durchgeführt.

6.2 Evaluierung des Frameworks

Das Ziel des Frameworks ist es, den Simulationen die größtmögliche Freiheit innerhalb der Rahmenbedingungen zu gewährleisten. Dennoch sollen die Simulation leicht zu entwickeln und zu integrieren sein. Währende der Entwicklung hat sich herausgestellt, dass es starke Abhängigkeiten zwischen dem Framework und dem ablaufenden Szenario gibt. Das Framework ist also in bestimmten Punkten nicht generisch. Dies betrifft vor allem das Interface der Agenten, welches die Objekte des FOM umsetzen muss. Das Einführen neuer Objekte erfordert also auch eine Anpassung des Frameworks. Diesem Effekt kann durch den Einsatz von vorgegebenen, generischen Modellen, wie z.B. dem RPR FOM (*Realtime Platform-level Reference Federation Object Model*), entgegengewirkt werden. In diesem FOM ist festgelegt, welche Informationen unter den einzelnen Entitäten ausgetauscht werden können. Eine Auswahl davon kann für das jeweilige Szenario dann genutzt werden.

Der zweite nicht generische Punkt im Framework ist die Umsetzung der physikalischen Begebenheiten. Diese muss durch die Welt durchgeführt werden, da sie die Umgebung und Repräsentanten verwaltet. Das Verlagern des Simulationsgebiets in den Weltraum hätte beispielweise Auswirkungen auf die Schwerkraft. Daraufhin müsste das Framework andere Gegebenheiten schaffen. Dies könnte durch die Vorgabe eingeschränkt werden, dass die Simulationen bei erdähnlichen Umgebungen ablaufen müssen. Ein weiterer Punkt, die Bewegungscharakteristik, ist damit jedoch nicht gelöst. So setzt das Framework eine Bewegungsform um, die dem Verhalten von Schiffen ähnlich kommen soll. Nehmen nun Landfahrzeuge, Tiere, Menschen oder Flugobjekte an der Simulation teil, so müssen deren Bewegungsformen ebenfalls umgesetzt werden.

Es zeigt sich also, dass das Framework mit dem Simulationsszenario zusammen entwickelt werden muss. Bestimmte Ansätze und Einschränkungen können diesen Effekt zwar abmildern, jedoch nicht ganz ausschließen. Das Framework ist somit sinnvoll, um die Einsatzmöglichkeit für die HLA im Bereich der individuenorientierten Simulationen zu untersuchen. Die Einsatzmöglichkeiten und Funktionalität der HLA kann es jedoch im großen Maße nicht ausschöpfen.

6.3 Portico Problemfelder

Die High Level Architecture hat sich seit der Standardisierung im Jahr 2000 stark entwickelt und unterstützt eine Vielzahl unterschiedlicher Simulationstypen und Anforderungen. Das Ziel des Prototyps war es lediglich einen Einblick in die HLA zu schaffen und die Konzepte der individuenorientierten Simulationen als verteilte Simulation innerhalb dieser Architektur umzusetzen. Somit wird das Potential, das HLA bietet, nicht voll ausgenutzt und es bleiben bestimmte Problemfelder bestehen,

die während der Entwicklungsphase nicht gelöst werden konnten. Diese können aber für aufbauende Projekte von Bedeutung sein und soll deshalb an dieser Stelle kurz umrissen werden.

Im FED werden die Klassen deklariert, welche in der Simulation erzeugt werden können. So wird im Prototyp die Klasse Entity definiert, welche alle Entitäten repräsentiert (siehe 5.5.1). Wünschenswert wäre jedoch eine Klassenhierarchie. Damit könnten die bewegungsbezogenen Attribute in einer Klasse zusammengefasst werden. Die spezifischen Attribute würden in einer Subklasse ergänzt.

Weiterhin soll es möglich sein, dass Federates sich zu einem beliebigen Zeitpunkt in den Simulationsverbund einklinken. Die Umsetzung dessen ist im Prototyp noch nicht gelungen. Die definierten Leitbilder (siehe 4.1) sehen jedoch eine begrenzte Ausführungsdauer vor. Das beliebige Einklinken von Federates nimmt somit an Bedeutung ab. Die Einbindung von Agenten zu verschiedenen Zeitpunkten ist jedoch gewährleistet und entspricht den Anforderungen an das MAS.

Als letzter Punkt soll noch auf die Nutzung der HLA-TM eingegangen werden. In wie weit Portico diese Funktionalität umsetzt, konnte nicht evaluiert werden da die Anforderungen an das Framework einen anderen Schwerpunkt haben. Dies soll jedoch der erste Ansatzpunkt bei der Weiterentwicklung des Prototyps sein.

7 Zusammenfassung und Ausblick

7.1 Fazit

Es gibt eine Vielzahl bereits implementierter MAS Systeme, die HLA benutzen. Dies zeigt die Relevanz der HLA in den individuenorientierten Simulationen. Der Ansatz ist also vielversprechend. Die Entwicklung eines Frameworks als Grundgerüst von Simulationen ist ebenfalls nicht neu. Auch im Bereich der MAS gibt es einige Beispiele (Uhrmacher et al., 2009). Beim Vergleichen der verschiedenen Ansätze zeigt sich, dass Frameworks, die das Time-Stepped Paradigma umsetzten, den Simulationen ein gewisses Design Template vorgeben, in welchem kognitive Architekturen und verschiedenste Agententypen eingebunden werden können. Diese Struktur schränkt auf der einen Seite sicherlich die Flexibilität der Simulationen ein. Auf der anderen Seite jedoch kann innerhalb des Templates eine Simulationsarchitektur eingebunden werden, indem das Hauptaugenmerk auf die simulationsspezifische Implementierung gelegt wird und wiederkehrende aber MAS-relevante Codefragmente schon im Framework gelöst sind. Vor diesem Hintergrund reiht sich das im Prototyp umgesetzte Framework ein. Simulationen und in ihnen simulierte Individuen haben einen hohen Freiheitsgrad bezüglich ihrer Architektur. Jedoch muss auch festgestellt werden, dass durch die Abhängigkeit zur Spieleengine und dem so vorgegebenen Update-Zyklus Einschränkungen vorhanden sind. Weiterhin wird die graphische Oberfläche durch das Framework vorgegeben. Denkbar wäre es, die Darstellung durch Plug-Ins[22] zu erweitern, eine bessere Alternative könnte die Abkopplung der Visualisierungskomponenten von dem Framework sein. Beide Überlegungen sind jedoch nicht in den aktuellen Prototyp eingeflossen, aber in darauf aufbauenden Projekten überlegenswert.

Auf einen Vergleich zu anderen Ereignisgesteuerten Architekturen wird an dieser Stelle verzichtet, da dies den Rahmen der vorliegenden Arbeit sprengen würde.

[22] Eine Programmerweiterung, die in ein Softwareprodukt „eingeklingt“ werden kann (Wiki1)

Trotz der vielen Vorteile von verteilten Simulationen, wie Wiederverwendbarkeit und die Möglichkeit der Umsetzung sehr komplexer Modelle, müssen an dieser Stelle auch bekannte Probleme aufgezeigt werden, die insbesondere bei Multiagentensystemen auftreten. Durch die parallele Ausführung von Individuen können verschiedene Prozesse gleiche Datenelemente lesen und beschreiben. Ein Mechanismus, der diese Daten effizient zur Verfügung stellt, ist nicht trivial. Wie das im Prototyp vorgestellten Framework verfolgen die meisten Systeme den Ansatz der vollständigen Replikation der Simulationsobjekte und Zustände. Das führt dazu, dass viel Kommunikationsaufwand notwendig ist, um die Systeme zu synchronisieren, selbst wenn es nur an Stellen Änderungen gibt, die wenige Individuen betreffen. Die HLA bietet das Konzept der Routing Spaces an, um diese Problematik einzudämmen. Um diese Funktionalität nutzen zu können, muss jedoch ein erheblicher Teil des Modells in die Abstraktionsstufe der HLA verlagert werden. Ein allgemeinerer Ansatz bildet das *Interest Management* (IM), welches versucht Informationen nur dort verfügbar zu machen, wo sie gebraucht werden. Dieser Ansatz findet besondere Bedeutung in individuenorientierten Simulationen, da die Agenten eine physische Präsenz haben und somit ihr Wahrnehmungsfeld begrenzt ist. Die Umsetzung des IM gestaltet sich jedoch als schwierig. Es werden zwei Klassen von Prozessen eingeführt, welche die Simulation von Ereignissen (*Agent Logical Process*, ALP) und das Bereitstellen von Daten (*Communication Logical Process*, CLP) voneinander trennt. Anfragen bezüglich Daten können über diesen Ansatz effizient gestellt werden, wodurch der Kommunikationsaufwand gesenkt werden kann. Jedoch wird der CLP zum Flaschenhals, da alle Lese- und Schreibeanfragen an ihn gesendet werden. Weiterführende Ansätze teilen den CLP weiter auf und nutzen Heuristiken, um einen effizienten Tree[23] aufzubauen (Uhrmacher et al., 2009). Diese Problematik kann also beliebig komplex werden, der im Prototyp gewählte Ansatz ist jedoch derzeit bewährt, auch wenn der Einsatz von Routing Spaces noch Potential bereitstellt.

In der abschließenden Evaluation des Frameworks muss festgestellt werden, dass das Framework die Kompatibilität innerhalb der Föderation garantiert und den Datenaustausch kapselt. Negativ ist anzumerken, dass die Simulation abhängig von der vorgegeben Plattform sind und zudem eingeschränkt in ihrer graphischen Darstellung.

7.2 Ausblick

Während der Entwicklung des Prototyps wurden mehrere Möglichkeiten für dessen Erweiterung identifiziert, die im Folgenden kurz diskutiert werden sollen. Die Simulationen und Agenten sind im Prototyp an den Update-Zyklus der Spieleengine

[23] Spezieller Graph zur Modellierung einer Monohierarchie

angeschlossen. Höhere Flexibilität würde die Entkopplung dieser Abhängigkeit bieten, in dem die Agenten lediglich über Wahrnehmung- und Interaktionsereignisse informiert werden würde und ansonsten zeitautonom agieren könnten. Auch die Simulationen sind im Framework an das Time-Stepped Zeitmodell gebunden. Die Einführung eines DES könnte die Effizienz des Simulationsverbunds erhöhen und auch vielfältigeren Simulationen die Möglichkeit geben, über das Framework, an der Gesamtsimulation teilzunehmen.

Gewisse Anforderungen, wie z.B. die Wahrnehmung, werden im Prototyp innerhalb des Frameworks gelöst. Die Problematik der vollständigen Replikation aller Simulationsobjekte wurde bereits diskutiert, da sie insbesondere für MAS nicht zwingend notwendig ist und einen hohen Kommunikationsaufwand verursacht. Durch vielfältigere Nutzung der von der HLA bereitgestellten Dienste, könnte man Teile des Modells auf eine andere Abstraktionsebene verlagern und somit für Projekte, die über das entwickelte Framework hinaus gehen, eine Grundlage schaffen, Funktionalität und Erkenntnisse wiederzuverwenden. Exemplarisch soll der Einsatz von Routing Spaces genannt werden, mit Hilfe dessen die vollständige Replikation umgangen werden kann. Dennoch bietet der Prototyp eine erste Grundlage zur Verknüpfung von MAS und HLA und kann für Projekte mit ähnlichem Schwerpunkt als valide Grundlage dienen.

Der Einsatz von Portico war ein Ansatz, um sich den Möglichkeiten der HLA zu nähern. Es gibt jedoch noch weitere, auch freie, Implementierungen von RTIs. Das Austauschen der Implementierung könnte wertvolle Erkenntnisse über die speziellen Vorteile und Nachteile von Portico liefern, welches im Rahmen dieser Arbeit nicht möglich war.

Abschließend soll noch auf die Zukunftsfähigkeit der High Level Architecture eingegangen werden. Die HLA wurde unter dem Standard 1516-2010 im März 2010 als IEEE Standard festgelegt. Diese Technologie wird also nicht nur in vielen Bereichen genutzt, sondern auch noch essentiell weiterentwickelt. Im neuen Standard wurden unter anderem der XML-Support verbessert und die Fehlerresistenz durch einen einheitlichen Einsatz von Encoding Helpers (siehe 2.2.6) gesteigert. Hinzu kam noch die Schnittstelle für *Webservices* (WSDL, Möller et al., 2006), welches nun auch eine Anbindung an das Internet ermöglicht. Das *Defense Modelling and Simulation Office* (DMSO) ist einer der größten Auftraggeber für Simulationsstudien und vergibt so gar nur noch Aufträge für HLA-konforme Computersimulationen (Schumann, 2002). Damit bleibt die High Level Architecture wahrscheinlich auch in naher Zukunft ein sehr flexibler und weit verbreiteter Ansatz, der insbesondere für den Bereich der individuenorientierten Simulationen gewinnbringend eingesetzt werden kann.

8 Abkürzungsverzeichnis

DIS	Distributed Interacitve Simulation
FOM	Federation Object Model
GIS	Geographic Information System
HLA	High Level Architecture
HLA-TM	Time Management in der High Level Architecture
I/O	Input/Output
jME	jMonkeEngine - Grafikengine
KI	Künstliche Intelligenz
MAS	Multi-Agenten-Simulation
NER	Next Event Request
OMT	Object Model Template
RO	Recieve Order
RTI	Run-Time Infrastructure
SOM	Simulation Object Model
TAR	Time Advance Request
TSO	Time Stamp Order

9 Abbildungsverzeichnis

10 Literaturverzeichnis

BUCHHOLZ, P.: *Modellgestützte Analyse und Optimierung Kap. 6 Simulationssoftware*, ls4-www.cs.tu-dortmund.de/download/LehreMaterialien/MAO2011/MAO_6.pdf [Online; Stand 16. August 2011], 2009

CROSBIE, R.; Zenor, J.: *High Level Architecture Module 2 Advanced Topics,* www.ecst.csuchico.edu/~hla/LectureNotes/**FEDEP**1.pdf [Online; Stand 16. August 2011]*,,* California State University, 1999

DAHLMANN, J.; Fujimoto, R.; Weatherly, R.: *The Department of Defense High Level Architecture*. Defense Modeling and Simulation Office, Alexandria; College of Computing, Georgia Institute of Technology, Atlanta; The MITRE Corporation, McLean. Winter Simulation Conference, 1997

DURFEE, E.; Montgomery, T.: *Using MICE to Study Intelligeng Dynamic Coordination.* Department of Elevtrical Engineering and Computer Science, University of Michigan, 1990

FISCHER, J.; *Verteilte Ereignissimulationen*, http://www2.cs.uni-paderborn.de/fachbereich/AG/heiss/lehre/ws97/seminar.html [Online; Stand 16. August 2011],, Universität Paderborn,1997

FISHWICK, P. A.: *Computer simulation: growth through extension*. Transactions of the Society for Computer Simulation International, 14(1):13–23. ISSN 0740-6797, 1997

FUJIMOTO, R.; Weatherly, R.: *HLA Time Management and DIS*, Georgia Institute of Technology, Atlanta, 1996

FUJIMOTO, R.: *Time Management in the High Level Architecture*, Georgia Institute of Technology, Atlanta, 1998

HUTTNER, S.: *Simulation von Bewegungsabläufen*, www.staff.uni-mainz.de/huttner/GeosimulationII/Bewegungsablaeufe4.pdf [Online; Stand 16.

August 2011],, Institut für Informatik, Johannes Gutenberg-Universität Mainz, 2011

JME; jMonkey Engine. http://www.jmonkeyengine.com. [Online; Stand 16. August 2011]

KLEIN, U.; Schulze, T.; Straßburger, S.: *Distributed Traffic Simulation based on the High Level Architecture.* Department of Simulation & Graphics (ISG), Department for Computer Science, Otto-von-Guericke-University Magdeburg, 1998 (a)

KLEIN, U.; Schulze, T.; Straßburger, S.; Menzler, H.-P.: *Traffic Simulation based on the High Level Architecture*. Department of Simulation & Graphics (ISG), Department for Computer Science, Otto-von-Guericke-University Magdeburg; Competence Center Informatik (CCI) GmbH, Meppen, 1998 (b)

LEES, M; Logan B.; Theaodoropoulos, G.: *Distributed Simulation of Agent-Based Systems with HLA.* University of Nottingham; University of Birmingham, 2007

MIEGEL, T.; Holzmeier, R: *High Level Architecture (HLA)*. Institut für Geoinformatik, Universität Münster, 2004

MÖLLER, B.; Gustavson, P; Lutz, B.; Löfstrand, B.: *Making Your BOMs and FOM Modules Play Together;* Pitch Technologies, Linköping, Schweden, 2007

MÖLLER, B.; Karlsson, M; Löfstrand, B.: *Reducing Integration Time and Risk with the HLA Evolved Encoding Helpers;* Pitch Technologies, Linköping, Schweden, 2006

M&S; Modellierung & Simulation. http://wwwmosi.informatik.uni-rostock.de/mosi/researchfield.2006-02-18.6472341180/ [Online; Stand 16.August 2011]

POLLACK, M.: *Planning in Dynamic Environments: The DIPART System.* Department of Computer Science and Intelligen Systems Program, University of Pittsburgh, 1996

PORTICO; The Portico Project http://www.porticoproject.org [Online; Stand 16. August 2011]

SCHULZ, H.-M.: *Verteile Simulationen für Automotive Anwendungen*. InnovationsCampus, Wolfsburg, 2005

SCHUMANN, M.: *HLA-Dienste in verteilten virtuellen Trainingsumgebungen,* http://www.kompetenzzentrum-hla.de/, Frauenhofer IFF, Magdeburg, 2002

SPALTHOFF, A.: *Semantische Technologien: Methoden und Anwendungen*, www.aifb.uni-karlsruhe.de/Lehre/Sommer2005/SemTech/stuff/DL.doc [Online; Stand 16. August 2011], Institut für angewandte Informatik und formale Beschreibungsverfahren, Karlsruher Institut für Technologien, 2005

SPATH, C.: *Simulationen – Begriffsgeschichte, Abgrenzung und Darstellung in der wissenschafts- und technikhistorischen Forschungsliteratur*, Universität Stuttgart, S.40ff, 2009

STRASSBURGER, S; Schumann, M.: *Erfahrungen beim Einsatz der HLA in zivilen Anwendungen.* In: Frauenhofer Institut für Fabrikbetrieb und –automatisierung, Magdeburg, 2005

THIEL-CLEMEN, T.; Köster, G.; Sarstedt, S.: *WALK – Emotion-based pedestrian movement simulation in evacuation scenarios;* Hamburg University of Applied Science; Munich University of Applied Science, 2011

UHRMACHER, A.; Krahmer, M.: *A Conserative, Distributed Approach to Simulationg Multi-Agent Systems,* Department of Computer Science, University Rostock, 2001

UHRMACHER, A.; Weyns, D.: *Multi-Agent Systems. Simulation and Applications,* ISBN: 978-1-42-007023-1; Kapitel 1-3; CRC Press Inc., 2009

WIKI1; WIKIPEDIA: Plug-In. http://de.wikipedia.org/wiki/Plug-in – [Online; Stand 16.August 2011]

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, 22. August 2011

Ort, Datum

Unterschrift