HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Ausbau der agentenbasierten Simulation in Simio

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2016

Autor(en): Ernst Füllemann, Martina Staub
Betreuer: Prof. Dr. Andreas Rinkel
Abgabe: 17. Juni 2016

AUFGABENSTELLUNG

ZIEL DER ARBEIT

Simio ist ein System zur diskreten Ereignissimulation und unterstützt die Modellierung von Prozessabläufen. Die agentenbasierte Modellierung stellt eine neue Modellierungsmethodik dar, die bisher nur ansatzweise in Simio umgesetzt ist.

Ziel der Arbeit ist es, ausgehend von einer umfangreichen Analyse ein Konzept zur agentenbasierten Simulationsmethodik in Simio zu entwerfen und prototypisch umzusetzen. Insbesondere gehören hierzu folgende Schritte:

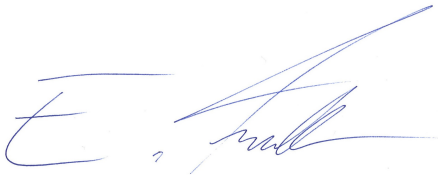
1. Literaturstudie und Analyse agentenbasierter Simulation
2. Analyse von Simio und der Programmierschnittstelle
3. Erstellung eines Prototyps einer agentenbasierten Simulation in Simio
4. Auflistung von Anforderungen an Simio um eine agentenbasierte Simulation zu ermöglichen

EIGENSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, die explizit in der Aufgabestellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe,
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Rapperswil, 17. Juni 2016



Ernst Füllemann

ABSTRACT

Ausgangslage

Simulation ist eine leistungsfähige Methode zur Analyse und Optimierung komplexer Systemen. Die Diskrete Ereignissimulation eignet sich für die Abbildung von Entity-Flows (Einheiten-Fluss). Der agentenbasierte Ansatz wird zur Beschreibung kybernetischer Systeme verwendet. Mit Simio steht ein umfangreiches Simulationstool zur Verfügung, dass die diskrete Ereignis Simulation unterstützt. Es ist eine Erweiterung für Simio zu entwicklen um den agentenbasierten Ansatz zu unterstützen.

Vorgehen/Technologie

Die agentenbasierte Modellierungsmethodik wird analysiert und die Eigenschaften eines Agenten charakterisiert. Simio stellt zur Entwicklung von benutzerdefinierten Erweiterungen eine .NET API zur Verfügung. Die Programmierschnittstelle wird untersucht und eruiert wie einzelne Eigenschaften und Verhalten eines Agenten in Simio umgesetzt werden können.

Ergebnis

In dieser Arbeit werden Konzepte zur agentenbasierten Simulation analysiert. Die Programmierschnittstelle von Simio wird untersucht und ein agentenbasierter Ansatz wird prototypisch umgesetzt. Die Evaluation des Prototypen zeigt welche Anforderungen an Simio gestellt werden um einen agentenbasierten Ansatz zu entwickeln.

MANAGEMENT SUMMARY

AUSGANGSLAGE

Simulation ist eine experimentelle Methode zur Analyse und Optimierung von komplexen Systemen. Dies kann einen Nutzen in den verschiedensten Bereichen erbringen wie zum Beispiel bei Finanzmärkten, Produktion, Ökonomie oder zur Vorbereitung auf Notfall- bzw. Katastrophenszenarien. Eine Simulation ist kosteneffizienter und risikoärmer als das Experimentieren am realen System, falls dies überhaupt Experimente zulässt. Die Simulation ist daher ein sehr wichtiges Instrument für die Entscheidungsfindung, Analyse und Optimierung von Systemen. Durch das Experimentieren mit Simulationsmodellen können fundierte Prognosen und Risikoeinschätzungen erstellt werden.

Simio ist ein umfangreiches Simulationstool. Bis anhin werden die diskrete Ereignissimulation und System dynamics unterstützt. Eine weitere Simulationsmethodik ist die agentenbasierte Simulation, welche zur Beschreibung kybernetischer Systeme verwendet wird. Mittels dieser Simulationsmethodik lassen sich Prozesse intelligenter Objekte simulieren und daraus emergente Verhalten analysieren. Es wurde untersucht ob eine Erweiterung für Simio entwickelt werden kann um einen agentenbasierten Ansatz zu unterstützen.

TECHNOLOGIEN

In Simio können, im Verhältnis zu anderen Programmen, auf einfache Weise eigene Komponenten und Bibliotheken implementiert werden. Es stehen viele Möglichkeiten zur Definition von Prozessen zur Verfügung. Zudem existiert eine .Net API für die Entwicklung eigener Erweiterungen.

Es wurde sich vertieft mit den verschiedenen Simulationsmethodiken und Konzepten auseinandergesetzt. Anhand der Erkenntnisse aus der Studie der agentenbasierten Simulation wurde ein Prototyp für eine Erweiterung umgesetzt. Der Prototyp unterstützt einen Anwendungsfall der agentenbasierten Simulation.

Die Programmierschnittstelle zu Simio wurde analysiert und ein Prototyp für eine agentenbasierte Simulation erstellt. Anhand der Erkenntisse aus der Analyse und der Umsetzung des Prototyps wurden Anforderungen an die Simio API definiert. Die Anforderungen definieren was erfüllt werden muss, damit eine Erweiterung, die das Erstellen von generischen Agenten unterstützt, entwickelt werden kann.

DANKSAGUNG

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mir die Durchführung dieser Arbeit ermöglicht oder erleichtert haben. Besonderer Dank gilt unseren Betreuern Herr Prof. Dr. Rinkel und Frau Olivia Müller für die Unterstützung und angenehme Zusammenarbeit.

Ein weiteres Dankeschön geht an meine Arbeitspartnerin Frau Martina Staub für die gute Zusammenarbeit und das gute Arbeitsklima.

Für die Erstellung dieser Arbeit wurde das *classicthesis* LaTeX-Template benutzt wofür ich mich herzlich bei Prof. Dr.-Ing. André Miede bedanke.

INHALTSVERZEICHNIS

Teil I

TECHNISCHER BERICHT

1 EINLEITUNG

Simulation wird zur Analyse und Optimierung von komplexen Systemen eingesetzt. Oft erlaubt das reale System das Experimentieren nicht, da es zu kostenaufwändig, risikoreich oder nicht möglich wäre. Dies ist der Fall bei der Modellierung von seltenen oder nicht steuerbaren Ereignissen wie zum Beispiel einer Naturkatastrophe.

Durch das Nachbilden des Systems können verschiedene Szenarien erstellt werden. Dadurch lassen sich verschiedene mögliche Lösungsansätze auswerten und vergleichen. Simulation hat nicht den Anspruch so genau wie das reale System zu sein. Die Charakteristiken des realen Systems werden so akkurat wie nötig modelliert. Simulation kann in verschiedenen Bereichen angewendet werden, wie bei der Verbesserung eines Fertigungsprozesses, Analyse einer Supply Chain(Versorgungskette) oder Auswertung eines Katastrophenszenarios.

Es gibt unterschiedliche Simulationsmethodiken. System dynamics wird bei dynamischen Modellen eingesetzt welche aus einer Serie von Beständen und Flüssen bestehen. Die diskrete Ereignissimulation repräsentiert die reale Welt mengenmässig und simuliert ihre Dynamik Ereignis um Ereignis. Die agentenbasierte Simulation wird für kybernetische Systeme verwendet. Der Unterschied zu der agentenbasierten Siulation und der Diskreten Ereignis Simulation besteht darin, dass die Entity-Flows (Flüssen von Einheiten) nicht passiv sind, sondern ein eigenes Verhalten haben. Die agentenbasierte Simulation eignet sich für die Modellierung der Prozesse von intelligenten Objekten sowie das Verhalten von Menschen oder anderen Organismen.

Simio ist eine Simulationsapplikation, die System dynamics und die diskrete Ereignissimulation unterstützt. Die Funktionalität von Simio für die Auswertung und Erstellung von Modellen ist sehr umfangreich. Simio unterstützt den Benutzer mit einer guten Hilfestellung und einem Forum. Im Simio Forum findet ein reger Austausch zwischen Benutzern und Simio-Entwicklern statt. Durch die Registration als Simio Insider lassen sich Anregungen für zukünftige Funktionen diskutieren und eigene Erweiterungen austauschen.

Es wird die technische machbarkeit überprüft ob eine Erweiterung für Simio entwickelt werden kann, die das Erstellen von agentenbasierten Simulationen unterstütz. Benutzerdefinierte Erweiterungen lassen sich durch das .Net API implementieren. Die Simio API ist flüchtig dokumentiert. Im Forum wird der agentenbasierte Ansatz diskutiert, es ist aber noch keine Erweiterung in diesem Bereich entstanden.

2 GRUNDLAGEN

2.1 SIMULATION

Simulation ist eine experimentelle Methode zur Analyse und Optimierung von komplexen Systemen. Dabei werden ein oder mehrere Prozesse eines Systems auf ein Modell abgebildet und imitiert.

Dies kann einen Nutzen in den verschiedensten Bereichen erbringen wie zum Beispiel bei Finanzmärkten, Produktion, Militär, Ökonomie oder zur Vorbereitung auf Notfall- bzw. Katastrophenszenarien. Eine Simulation ist kosteneffizienter und risikoärmer als das Experimentieren am realen System, falls dies überhaupt Experimente zulässt. Die Simulation ist daher ein sehr wichtiges Instrument für die Entscheidungsfindung, Analyse und Optimierung von Systemen. Durch das Experimentieren mit Simulationsmodellen können fundierte Prognosen und Risikoeinschätzungen erstellt werden.[10]

Es gibt unterschiedliche Ansätze für Simulation. Beispiele sind System dynamics Abschnitt 2.2 und die diskrete Ereignissimulation Abschnitt 2.3. Die agentenbasierte Simulation Abschnitt 2.4 ist ein relativ neuer Ansatz. Die Wahl des Ansatzes ist abhängig von den Eigenschaften des zu simulierenden Systems und des Abstraktionsgrades des Problems.

System dynamics eigenen sich vor allem um Trends und Wirkungen zu Simulieren. Einzelne Objekte wie zum Beispiel Menschen, Produkte etc. spielen dabei keine Rolle. Wobei diskrete Ereignissimulation weniger abstrakt ist und ein reales physisches System abbildet wie zum Beispiel eine Fertigungsstrasse. Der agentenbasierte Ansatz liegt am nächsten an der Realität und kann einzelnen Individuen ein eigenes Verhalten geben.[3]

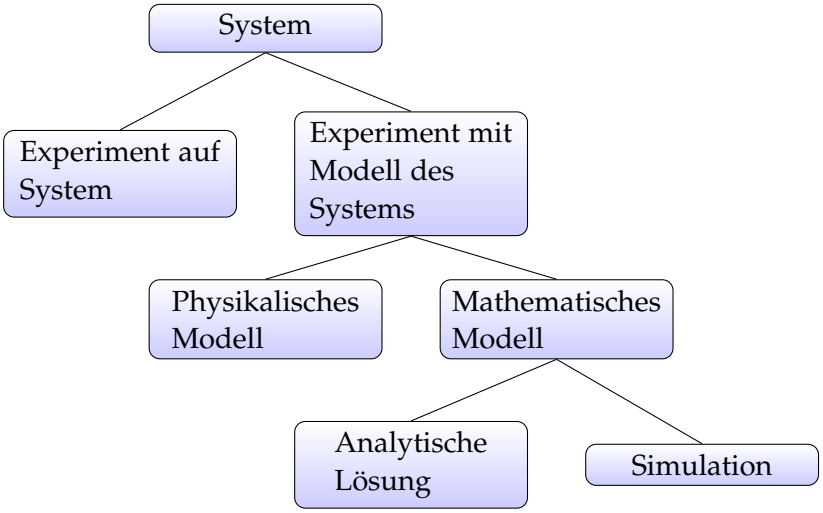


Abbildung 1: Arten ein System zu untersuchen.[4]

2.1.1 *System*

Ein System ist eine Einrichtung oder ein Prozess und besteht aus Systemkomponenten. Diese Komponenten sind miteinander verbunden oder aufeinander bezogen und interagieren miteinander. Systemkomponenten können wiederum selbst Systeme sein. Dabei wird zwischen diskreten und kontinuierlichen Systemen unterschieden. Ein Diskretes System hat eine bestimmte Anzahl definierter Zustände und basiert auf festen Ereigniszeitpunkten. Hingegen bei einem kontinuierlichen System ändern sich die Zustände über der Zeit stetig. Die Änderungen geschehen nicht zu einem bestimmten Zeitpunkt sondern zu jeder Zeit. Als Beispiel kann das Füllen einer Badewanne betrachtet werden. Fortwährend fliesst Wasser in die Wanne. Der Zustand der Badewanne ändert sich laufend. In der Realität sind alle Systeme stetig. Auf einem Rechner simuliert ist alles getaktet–diskret.

2.1.2 *Modell*

Als Modell wird die Abstraktion eines realen oder eines noch nicht bestehenden Systems bezeichnet. Es fasst die relevanten Bereiche zusammen und beinhaltet die Hauptfunktionen und Charakteristiken des realen Systems damit aussagekräftige Experimtente darauf ausgeführt werden können. Je besser ein Modell das System bzw. das zu untersuchende Verhalten des Systems reproduzieren vermag, desto genauere Ergebnisse können untersucht werden. Alles was vom Modell nicht beschrieben wird, wird als Rauschen bezeichnet. Das sogenannte Rauschen kann einen Einfluss auf das System haben, wird aber bewusst nicht berücksichtigt.

2.1.3 *Prozess*

Prozesse können zum Beispiel triviale Abläufe wie die Zubereitung des Abendessens sein oder komplexe wie der Bestellungseingang eines Unternehmens (Abbildung 2). Ein Prozess hat einen Start-, sowie einen Endpunkt und besteht aus Schritten und Entscheidungen. Die einzelnen Schritte können wiederum Subprozesse sein. Ein Prozess kann von anderen Prozessen abhängig sein, oder von einem anderen Prozess gestartet werden. Während des Ablaufes kann dieser auf externe Eingaben angewiesen sein. Ein Prozess kann ein Produkt oder Ergebnis hervorbringen. Störfaktoren können den Ablauf des Prozesses hindern. Der Prozess kann für den Ablauf eine oder mehrere Ressourcen benötigen. Ressourcen sind beispielsweise geforderte Materialien für die Abarbeitung des Prozesses oder Arbeitskräfte für die Ausführung der Arbeitsschritte.

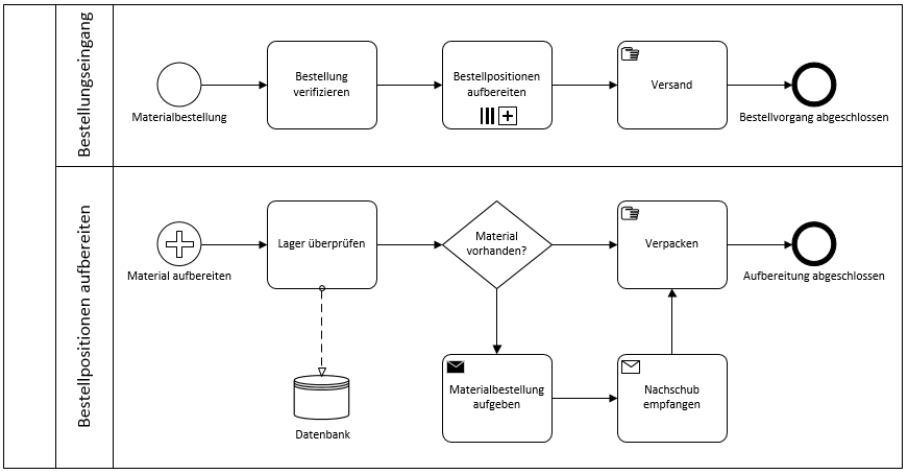


Abbildung 2: BPMN Flussdiagramm eines Prozesses mit parallelen Subprozessen

2.2 SYSTEM DYNAMICS

System dynamics (SD) wird bei kontinuierlichen Systemen eingesetzt und ist ein dynamisches Modell welches aus einer Serie von Beständen und Flüssen besteht. Die Einheiten der Flüsse sind meistens nicht klar definierbar. Es handelt sich dabei um kontinuierliche Einheiten wie zum Beispiel Wasser. Es kann sich dabei aber auch um nicht physische Dinge wie Wissen oder Meinungen handeln. Die Zu- und Abflüsse werden jeweils in Raten beschrieben. Der Zustand ändert sich somit kontinuierlich. SD wird vorallem bei der Analyse von sozioökonomischen Systemen eingesetzt wie zum Beispiel um das Entwicklungspotenzial eines Marktes (Abbildung 3) zu analysieren.

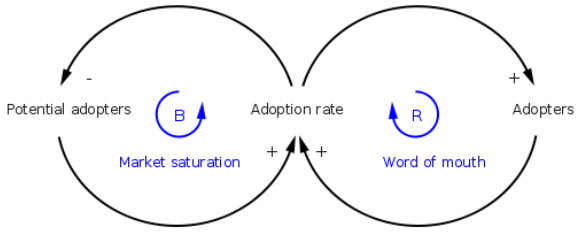


Abbildung 3: Loop Diagramm für die Einführung eines neuen Produktes

Ein Produkt wird eingeführt. Je mehr Menschen dieses Produkt besitzen, desto mehr Menschen gibt es die für das Produkt werben. Mehr Menschen erfahren von dem neuen Produkt und werden zu potenziellen Käufern.

2.3 DISKRETE EREIGNISSIMULATION

Diskrete Ereignissimulation (DES) repräsentiert die reale Welt mengenmässig, simuliert ihre Dynamik Ereignis um Ereignis und generiert detailierte Reporte. DES setzt sich mit Systemen auseinander welche beispielsweise den Fertigungsablauf einer Maschine beinhalten. Die DES besitzt Einheiten welche einen vorgegebenen Prozess durchlaufen. Bei auslösenden Ereignissen können sich die Zustände der Einheiten und der Umgebung verändern.

Anwendung findet diese Art von Simulation bei der Analyse von Geschäfts- und Fertigungsprozessen. [1]

2.4 AGENTENBASIERTE SIMULATION

agentenbasierte Simulation (ABS) wird für die Simulation von komplexen dynamischen Systemen verwendet. Es werden dabei Agenten modelliert, die eigene Entscheidungs- und Handlungsmöglichkeiten haben (Tabelle 1). Die Agenten können mit anderen Agenten und mit ihrer Umwelt interagieren. Dadurch kann das dynamische Verhalten von Schwärmen, Ameisenkolonien, Wettbewerbsmärkten oder Katastrophenszenarien analysiert werden. Durch die Experimente können emergente Verhaltensmuster erkannt werden. Agentenbasierte Modelle können sich durch ihre komplexen Verhaltensweisen sehr nah an reale Systeme annähern.

2.4.1 *Vergleich ABS mit SD*

ABS kann auch in System dynamics angewendet werden und noch bessere Ergebnisse liefern. Voraussetzung dafür ist jedoch, dass das System die nötige dynamische Komplexität mitbringt und eine ABS

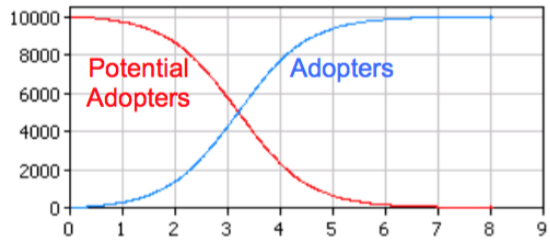


Abbildung 4: SD Simulationsresultat [3]

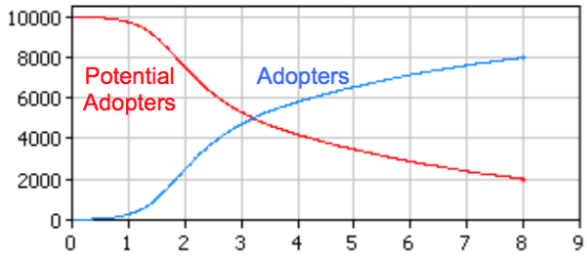


Abbildung 5: ABS Simulationsresultat [3]

rechtfertigt. Beim erwähnten Beispiel aus Abbildung 3 von der Einführung eines neuen Produktes wird bei der SD die mündliche Werbung ein Durchschnittswert genommen.

In der realen Welt ist jeder Mensch einzigartig. Jeder kennt unterschiedlich viele Mitmenschen und macht mehr oder weniger intensiv Werbung. Modelliert man somit jeden Menschen als einen Agenten mit eigenem Wissen und Verhalten werden die Resultate präziser (Vergleich Abbildung 4 und Abbildung 5).

Nicht in jeder Situation bringt eine ABS einen Vorteil. Bei der Simulation von Geldflüssen, unterscheidet sich ein Franken nicht vom anderen. Eine ABS würde keinen Nutzen bringen [3].

2.4.2 *Vergleich ABS mit DES*

DES und ABS haben viel gemeinsam. Die DES arbeitet bereits mit Einheiten welche durch Erweiterung von Autonomität als Agenten angesehen werden können. Die Einheiten des DES sind passiv und haben kein eigenes Verhalten. Mit ABS lässt sich das Verhalten intelligenter Objekte simulieren und somit ein genaueres Abbild des realen Systems. [7] [8]

DES MODELLIERUNGSANSATZ	ABS MODELLIERUNGSANSATZ
Prozessorientiert (top-down Modellierungsansatz); Fokus liegt bei der modellierung des Systems, nicht in den Entitäten	Individuum basiert (bottom-up Modellierungsansatz); Fokus liegt bei den Entitäten und deren Interaktionen
Ein Kontrollfluss (zentralisiert)	Jeder Agent hat seinen eigenen Kontrollfluss (dezentralisiert)
Passive Entitäten; Es geschieht etwas mit den Entitäten während sie sich durch das System bewegen; Intelligenz (Entscheidungsfassung) wird als Teil des Systems modelliert	Aktive Entitäten; Die Entitäten können selbst die Initiative ergreifen eine Aktion auszuführen; Intelligenz wird durch die individuelle Entität repräsentiert
Warteschlangen sind ein Schlüsselelement	Kein Konzept von Warteschlangen
Fluss von Entitäten durch das System; Verhalten wird auf Makroebene modelliert	Kein Konzept von Flussverhalten; Verhalten resultiert aus Entscheidungen welche auf der Mikroebene der einzelnen Agenten beschrieben werden
Eingangsverteilungen basieren meist auf gesammelten/gemessenen (objektiven) Daten	Eingangsverteilungen basieren meist auf Theorien (subjektiv)

Tabelle 1: Attribute welche den Modelltyp definieren[9]

2.4.3 *Definition Agenten*

Allgemein sind Agenten Einheiten, welche über eigene Attribute und Verhalten verfügen. Sie haben eigene Ziele und reagieren autonom auf ihr Umfeld um ihre Ziele zu verfolgen. Eine allgemein eindeutige Definition für Agenten gibt es nicht [11]. Je nach Einsatz und Umfeld eines Agenten, werden seine Fähigkeiten unterschiedlich gewichtet. Keine der Fähigkeiten ist essentiell um einen Agenten auszumachen. Die Kommunikation zwischen zwei Agenten kann in einem Szenario von grosser Bedeutung sein, während sie in anderen Simulationen unerwünscht ist.

2.4.4 *Unterschied zwischen Agenten und Intelligenten Agenten*

Ein einfacher Agent hat für jeden definierten Eingang von Information eine mögliche Aktion als Antwort. Es handelt sich dabei um einen

einfachen wenn-dann-Ablauf. Ein intelligenter Agent zeichnet sich dadurch aus, dass er flexibel autonome Reaktionen auswählen kann um seine Ziele zu erreichen. Seine Fähigkeiten sind in drei Kategorien gegliedert [11].

1. *Reaktivität:* Agenten können Informationen ihres Umfelds aufnehmen, speichern und darauf reagieren um ihre Ziele zu erreichen.
2. *Aktivität:* Sie haben ein zielgerichtetes Verhalten und können auch die Initiative ergreifen um ihre Ziele zu erreichen.
3. *Soziale-Fähigkeit:* Interaktion und Kommunikation mit anderen Agenten und Systembenutzer ist möglich um die Ziele zu erreichen. Die Agenten können unterschiedliche Kommunikationsarten verwenden. z.B. Point-to-Point[1] oder Point-to-Multipoint[2].

2.4.5 *Agententypen*

Durch unterschiedliche Fähigkeiten oder Ausprägungen davon, bilden sich folgende Typen von Agenten.

REAKTIVE AGENTEN: (Simple Reflex Agents) Der Agent verfügt nicht über eigenes Wissen. Er erhält von seinem Umfeld Informationen und reagiert entsprechend darauf.

INTERFACE AGENTEN: Durch Kommunikation mit dem Systembenutzer, werden die Ziele erreicht.

ADAPTIVE AGENTEN: Der Agent kann aufgrund seines eigenen Status und dem Status seiner Umwelt eine möglichst optimale, den Bedingungen angepasste, Entscheidung treffen.

BEOBACHTENDE AGENTEN: (Model Based reflex Agent) Im Gegensatz zum reaktiven Agenten hat dieser Agent ein eigenes Gedächtnis und sammelt Informationen über seine Umwelt. Seine Entscheidungen trifft er basierend auf den gesammelten Informationen. Er weiss wie sich seine Umwelt in der Vergangenheit verhalten hat und kann somit eine möglichst optimale Entscheidung treffen.

LERNENDE AGENTEN: Durch ein Feedback-Element erhält der Agent eine Kritik wie gut er sich verhält. Der Agent kann in einem für ihn unbekannten Umfeld bewegen und Informationen sammeln. Basierend auf seinem aufgebauten Wissen, trifft er seine

1 Point-to-Point oder auch Direktverbingung bezeichnet eine direkte, unmittelbare Verbindung zwischen zwei Punkten.

2 Point-to-Multipoint bezeichnet die Verbindung zwischen einem bestimmten Punkt zu mehreren verschiedenen Punkten wie z.B. Wirelessrouter oder Radiostationen

Entscheidungen. Das Feedback-Element, welches auch der Systembenutzer sein kann, gibt dem Agenten eine Rückmeldung wie gut seine jeweiligen Entscheidungen waren. So lernt der Agent immer bessere Entscheidungen zu treffen.

KOLLABORATIVE AGENTEN: Agenten können durch Zusammenarbeit und durch Kooperation mit anderen Agenten ihre Ziele erreichen. Sie sind autonom und können mit anderen Agenten kommunizieren. Meistens sind sie auch selbstlernend.

KOGNITIVE AGENTEN: (Goal Based, Utility based Agents) Dieser Agent kennt die Folgen seiner Aktionen. Er plant die Schritte, welche ihn zu seinem Ziel führen. Es kann auch Erweiterungen von diesem Agenten geben, bei denen der Agent den Nutzen seiner Ziele ermitteln kann und eine Risikoeinschätzung seiner Ziele durchführen. Durch die Analyse der Nutzen seiner Ziele, kann er entscheiden, welches Ziel das Erstrebenswerteste ist. Er weiss, wie seine Schritte sein Umfeld beeinflussen und ob sich dies positiv oder negativ auf seine weiteren Ziele auswirkt.

2.5 SIMIO

Simio ist eine Simulationssoftware, welche die Modellierung von diskreten und kontinuierlichen Systemen ermöglich. Es baut auf einem objektorientierten Ansatz auf und unterstützt folgende Modellierungsparadigmen: Ereignis-, Prozess-, Objekt- und Agentenbasiert. Simio versucht durch die umfangreiche grafische Oberfläche auch die Modellierung von komplexen Systemen möglichst einfach zu halten und setzt keine Programmierkenntnisse voraus. Trotzdem können in .Net komplexere benutzerdefinierte Erweiterungen programmiert werden.

Simio bietet mit dem Simio Software Discussion Forum[3] ein öffentliches Forum für Austausch und Hilfe an. Durch die kostenlose Anmeldung bei Simio Insider sind weitere Themen im Forum zugänglich, welche nur für Simio Insider zur Verfügung stehen. Jede Person kann Simio Insider werden, es wird lediglich eine Registration benötigt. Die Simio Insider tauschen im Forum ihre selbst entwickelte Erweiterungen aus und geben Anregungen für Verbesserungen an Simio.

2.5.1 *Agentenbasierter Ansatz in Simio*

Simio wirbt damit, ABS zu unterstützen. Wenn man die Tabelle 1 betrachtet, ist zu erkennen, dass zumindes einige ABS Modellierungsansätze implementiert sind. Beispielsweise können Prozessabläufe in Objekten definiert werden (bottom-up, dezentralisiert) und beschränken sich nicht auf die Systemebene. Da zusätzlich DES spezifische

3 Simio Software Discussion Forum: www.simio.com/forums

Elemente wie Warteschlangen implementiert werden, kann Simio als hybrides Simulationstool bezeichnet werden.

In der Objekthierarchie von Simio ist der Agent eine Vorlage der implementierten Elementen Entity und Transporter. Der Agent selbst kann jedoch nicht instanziiert werden (Abbildung 6). Die Objekte in Simio weisen Verhalten von komplexeren reaktiven Agenten und eingeschränkten adaptiven Agenten auf. Sie zeigen jedoch kein intrinsisch autonomes oder intelligentes Verhalten auf und haben auch keine Vorstellung von ihrer Umwelt.

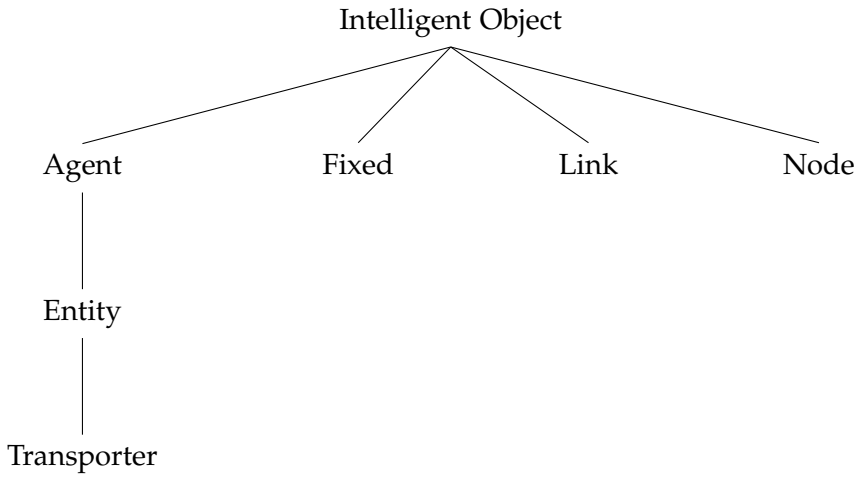


Abbildung 6: Simio Objekthierarchie [5]

2.5.2 *Fähigkeiten von Entity*

1. *Bewegung:* Objekte können sich frei auf einem Pfad bewegen. Ohne Pfad können nur direkte stationäre Ziele angesteuert werden. Pfade können uni- oder bidirektional sein. Die Objekte bewegen sich stets vorwärts. Falls der Pfad eine Breite hat und sich nicht nur auf eine Linie beschränkt, können sich die Entities in Kurven fortbewegen. Die Entities können eine Kollisionserkennung aktiviert haben. Somit wird verhindert, dass sie sich auf dem Pfad kreuzen. Es gibt keine Möglichkeit die Kollisionsdetektion zu konfigurieren (z.B. Distanz).

2. *Aktiviät:* Die Entities haben ein Ziel. Sie wissen nichts über das Modell in dem sie sich bewegen. Sie folgen dem vorgegebenen Pfad und passieren z.B. Servers oder Workstations bis sie zum gewünschten Node gelangen. Die Entities können über Properties und Prozesse verfügen.

3. *Soziales Verhalten:* Die Entities können nicht untereinander kommunizieren.

2.5.3 *Prozesse in Simio*

Simio bietet über ein Benutzerfenster die Möglichkeit Prozessabläufe zu definieren. Prozesse werden zur Modellierungszeit auf einem Modell oder direkt auf einem Objekt(mit ausnahme von Entity und Link) erstellt. Prozesse können einerseits durch einen anderen Prozess mit einem Execute-Step angestossen werden oder indem ein Event als Auslöser referenziert wird. Im Falle eines Prozesses welcher im Modell eines Entities definiert ist besitzt zur Laufzeit jede instanz eine eigene Kopie. Der Prozess besitzt den gleichen Scope wie das zugehörige Objekt und hat somit auch nur zugriff auf Daten welche für das Objekt sichtbar sind. Prozesse werden in Simio hauptsächlich dazu verwendet das Verhalten eines existierenden Objektes anzupassen oder um neue Objektdefinitionen zu definieren beim erzeugen einer Unterklasse eines existierenden Objektes. Ein Prozess besteht aus Steps, Elements und Tokens.[5]

STEP: Steps repräsentieren Aktivitäten in einem Prozessablauf und werden benutzt um Logik zu definieren. Steps sind Zustandslos können aber die Zustände auf Objekten, Tokens, Elements und Entities verändern. Im Prozessfenster können Steps, nach eigenem Ermessen angeordnet und miteinander verbunden werden.

ELEMENT: Ein Element repräsentiert Objekte in einem Prozess welche ihren Zustand über die Zeit verändern. Sie werden in einer Liste gespeichert und von einem oder mehreren Prozessen referenziert.

TOKEN: Token durchlaufen den Prozessfluss und führen die darin enthaltenen Steps aus. Tokens können benutzerdefinierte Zustände enthalten um Informationen von Step zu Step zu transportieren.

2.6 EXISTIERENDE LÖSUNGEN

Es sind sehr viele unterschiedliche Simulationsprogramme für Agentenverhalten auf dem Markt. Viele wurden für einen Anwendungsbereich entwickelt und sind sehr gut darin, z.B. Fussgängerbewegungen zu simulieren. Daher ist es ratsam die Simulationsapplikation nach dem Anwendungszweck auszuwählen. Einige Softwares sind sehr verbreitet und verfügen über Tutorien und Dokumentationen, wobei bei anderen kaum oder keine Dokumentation verfügbar ist.

Cynthia Nikolai und Madey Gregory haben eine Übersicht zusammen gestellt über bekannte agentenbasierte Simulationssoftware. Es wurden dabei 53 Softwares unter den folgenden fünf Gesichtspunkten kategorisiert: Vorausgesetzte Programmiersprache um ein Modell

zu bilden, Betriebssystem, Lizenz, Anwendungsgebiet, Support für den Anwender [6]. Bei vielen Software werden Programmierkenntnisse vorausgesetzt um ein Modell zu erstellen. Ein wesentlicher Vorteil von Simio ist, dass alles in der Benutzeroberfläche getätigt werden kann. Simio stellt jedoch trotzdem eine Programmierschnittstelle für eigene Erweiterungen zur Verfügung.

Es ist schwierig die verschiedenen Applikationen für agentenbasierte Simulation zu vergleichen. Viele sind für Spezialgebiete entwickelt worden und verfolgen daher andere Ansprüche. Der Katalog im Anhang Anhang A kategorisiert die Applikationen nach ihrem primären Anwendungszweck.

Wir finden Anylogic von den aufgeführten Applikationen erwähnenswert, da es von seiner Funktionalität und Bedienung eine grosse Ähnlichkeit mit Simio hat, aber bereits agentenbasierte Simulation unterstützt.

2.6.1 *Vergleich zwischen Anylogic und Simio*

Wie Simio unterstütz Anylogic die Eventbasierte Simulation und System dynamics sowie zusätzlich die agentenbasierte Simulation. In der Bedienung für Eventbasierte Simulationen sind die beiden Applikationen sehr ähnlich. Anylogic verfügt wie Simio über eine sehr gute und intuitive Benutzeroberfläche und setzt für simple Simulationen beinahe keine Programmierkenntnisse voraus. Mit wenigen Klicks kann eine Simulation erstellt werden (Abbildung 7). Da sich bei Anylogic die Agenten frei bewegen, können sehr einfach Dichtediagramme (Abbildung 8) erstellt werden um die Verhaltensmuster zu analysieren.

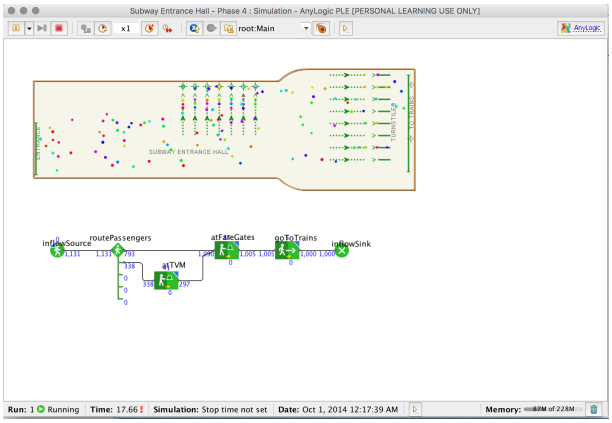


Abbildung 7: Beispiel-Simulation in Anylogic

Bei Anylogic können komplexere Eigenschaften direkt im Property in Java implementiert werden und bietet daher gegenüber Simio grösseren Spielraum.

Interessante Eigenschaften von Anylogic gegenüber Simio sind [2]

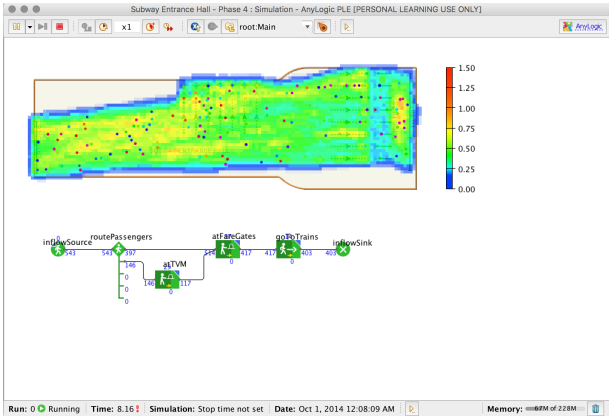


Abbildung 8: Dichtediagramm der Beispiel-Simulation in Anylogic

- Agenten bewegen sich frei (nicht an Pfad gebunden)
- Erkennung von Nachbar-Agenten
- Messaging zwischen einzelnen Agenten und von Agenten zu Agenten-Gruppen
- Zugriff von Agent auf Variablen und Parametern von anderen Agenten
- Agenten nehmen ihre Umwelt war und reagieren darauf wie z.B. Agenten könne nicht durch Wände gehen

3 KONZEPTE

In agentenbasierter Modellierung steht das Verhalten einzelner Agenten im Mittelpunkt. Die Auswirkungen des Zusammenspiels mehreren evtl. verschiedenen Agenten ist Zentrum der Untersuchung von Prognosen oder Vermutungen und wird simuliert. Aufgrund der in Abschnitt 2.4 beschriebenen Agententypen und unter berücksichtigung der Funktionsweise von Simio wurden die folgenden Konzepte ausgearbeitet:

- Kommunikation
- Eigenschaften
- Wahrnehmung
- Wissen

Dies geschah zum einen durch Brainstorming im Team sowie auch im Dialog mit Herrn Prof. Dr. Andreas Rinkel und seiner Assistentin Frau Olivia Müller.

3.1 KOMMUNIKATION

Kommunikation als Konzept beschreibt welche Möglichkeiten zur Handlung einem Agenten zur Verfügung stehen. Dabei ist die Kommunikation als Informationsaustausch zu verstehen und kann somit je nach Implementation als physische Handlung, als Wortwechsel oder als beliebiges anderes Verhalten dargestellt werden. Kommunikationsarten sollen zur Modellierzeit bestimmt, aber auch zur Laufzeit gelernt bzw. verlernt werden können. Kommunikation kann eine Voraussetzung sein für die Aneignung von Wissen (Abschnitt 3.4) oder die Veränderungen von Eigenschaften (Abschnitt 3.2). Umgekehrt können aber auch Eigenschaften und Wissen benötigt werden um Kommunikationsarten zu erlernen. Kommunikation muss gegebenenfalls parallel ausgeführt werden können.

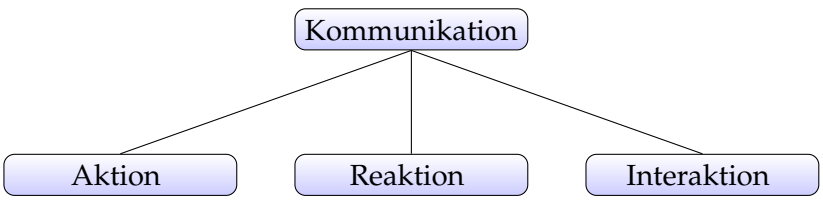


Abbildung 9: Kommunikation als Konzept für Agenten

3.1.1 *Aktion und Reaktion*

Reaktionen sind eine Untergruppe der Kommunikation. Eine Reaktion beschreibt eine Handlung eines Agenten die auf die Veränderung des Zustandes von Ausser- bzw. Innerhalb oder durch Wahrnehmung (siehe Abschnitt 3.3) erfolgt. Ein Agent kann eventuell nicht erkennnen ob ein Ereignis durch einen anderen Agenten, den Systembenutzer oder per Zufall ausgelöst wird. Aus diesem Grund wird für die Wahrnehmung zwischen Aktion und Reaktion unterschieden. In einem Agenten jedoch laufen nur Reaktionen ab. Aktionen und Reaktionen können unter Umständen zu einer Interaktion (Unterabschnitt 3.1.2) führen und Bestandteil einer solchen sein, jedoch ist dies nicht die ursprüngliche Intention.

3.1.2 *Interaktion*

Eine Interaktion beschreibt eine Abfolge von Aktionen und/oder Reaktionen. Im Gegensatz zu Aktion und Reaktion muss jedoch im Vorhinein klar sein mit welchen Akteuren interagiert wird. An einer Interaktion müssen mindestens Zwei Akteure beteiligt sein. Das Konzept der Interaktion ist dazu angedacht, verteilte Algorithmen bzw. Protokolle umzusetzen. Im Rahmen der Interaktion sollen bekannte Konzepte wie Point to Point, Point to Multipoint, synchroner und asynchroner sowie verbindungsorienterter und verbindungsloser Informationsaustausch implementiert werden können.

3.2 EIGENSCHAFTEN

Eigenschaften bezeichnen die verschiedenen Zustände, die ein Agent annehmen kann. Dieses Konzept ist abgeleitet von Datenfeldern aus der Programmierung und kann von Simio übernommen werden. Zusätzlich sollen Eigenschaften zur Laufzeit hinzugefügt oder entfernt werden können.

3.3 WAHRNEHMUNG

Die Wahrnehmung bildet einen oder mehrere Filter ab über welche entschieden werden kann welche Informationen aus der Umwelt aufgenommen und welche verworfen werden. Die Arten der Wahrnehmung lassen sich in drei Unterkategorien aufteilen (Abbildung 10). Jede Art von Wahrnehmung verfügt über einen Kanal und einen Bezugsbereich. Der Kanal dient zur Implementation von Unterschiedlichen Wahrnehmungn wie z.B. Akkustische Signale, Infrarot, Ultraschall und deren Unterscheidung bzw. Abhörung. Der Bezugsbereich kann auch von einem Medium verändert oder geblockt werden (Luft, Wasser, Blei usw.).

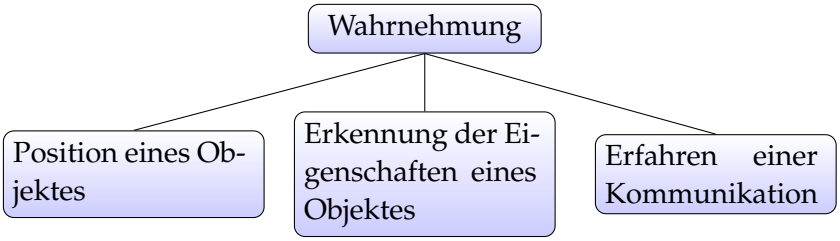


Abbildung 10: Arten der Wahnehmung

3.3.1 *Begrenzung der Wahrnehmung*

POSITION EINES OBJEKTES: Über einen definierten Kanal können Agenten andere Objekte und deren Position wahrnehmen. Unter Umständen lassen sich andere Objekte oder Agenten nicht lokalisieren. Z.B. wenn das Objekt zu klein ist um es wahrzunehmen oder nicht über den nötigen Kanal verfügt wird. Über diese Art von Wahrnehmung kann auch eine Kollisionserkennung implementiert werden.

ERKENNUNG DER EIGENSCHAFTEN EINES OBJEKTES: Lässt ein Objekt oder ein Agent dies zu, kann die Position wahrgenommen werden und Eigenschaften gelesen werden. Dies kann z.B. die Dimension eines Objektes sein.

ERFAHREN EINER KOMMUNIKATION: Kommunizieren zwei Agenten miteinander kann dies evtl. von einem dritten Agenten wahrgenommen werden. Je nach Spezifikation des Filters der Wahrnehmung kann eine Kommunikation wahrgenommen werden, ohne dass es zwingend eine eigene Reaktion zur Folge hat. Eine weitere Möglichkeit ist, dass eine Kommunikation wahrgenommen wird ohne dass die beteiligten Objekte wahrgenommen wurden.

3.3.2 *Verarbeitung der Wahrnehmung*

Die Informationen die ein Objekt von seiner Umwelt erhält, können eine Reaktion hervorrufen oder im Wissen des Objektes gespeichert werden. Da nicht alle Informationen erfasst werden können, verändert sich das Wissen des Objektes im Laufe der Zeit (Abschnitt 3.4). Das Vergessen kann abhängig von der Art der Wahrnehmung sein.

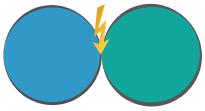
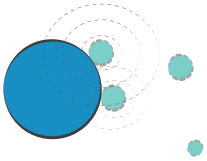
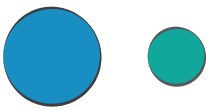
Wahrnehmungsart	Beschreibung
Physischer Kontakt 	Objekte können sich berühren respektive kollidieren und sich durch den Kontakt wahrnehmen. Bei blinden Menschen oder von Sensoren gesteuerten Objekten eine grundlegende Art um die Umgebung wahrzunehmen.
Erfahren einer Kommunikation 	Objekte können auch nur eine Kommunikation mit einem anderen Objekt auslösen. Objekte müssen sich nicht zwingend sehen um Kommunikation wahrzunehmen. Ein Ebola-Erreger kann nicht gesehen werden und trotzdem eine Auswirkung haben. Die Auswirkung, respektive die Fähigkeit des Erregers, wird erkannt.
Erkennen einer Gestalt 	Für die Wahrnehmung eines Objektes, reicht es aus, wenn diese sich in einem Wahrnehmungsradius befinden. Dadurch nimmt ein Objekt seine Umwelt war.

Tabelle 2: Arten der Wahrnehmung

3.4 WISSEN

Das Konzept des Wissens beschreibt die Kenntnisse von Objekten, Abläufen und Sachverhalten. Wissen ist in dieser Arbeit als Wertungsneutral einzustufen. Somit kann Wissen auch Überzeugungen, Bedürfnisse oder falsche Tatsachen beinhalten. Wissen kann bereits zur Modellierungszeit zur Verfügung stehen oder zur Laufzeit angeeignet, verloren oder modifiziert werden. Um Wissen zur Laufzeit anzueignen, zu verlernen oder zu modifizieren, können Eigenschaften oder Kommunikation vorausgesetzt werden. Somit kann Wissen auch eine Voraussetzung zur Aneignung von Kommunikation oder zur Veränderung von Eigenschaften sein. Wissen soll in verschiedenen Komplexitäten vorliegen und modifizierbar sein. So kann z.B. eine Kenntnis vorhanden sein oder nicht (Boolean), die Präferenz für ein Produkt einen numerischen Wert annehmen (Integer, Float) oder die Ausführung einer Kommunikation verbessert oder verschlechter werden (Objekte).

Ein Bestandteil des Konzeptes Wissen ist das Vergessen. Das Vergessen soll für jedes Wissen konfigurierbar sein. Beispielsweise kann Wissen als Funktion über der Zeit reduziert werden. Je nach Komplexität des Wissens wird konfiguriert wie der Vorgang des Vergessens abläuft. Es können z.B. die älteste oder unwichtigsten Informationen zuerst vergessen werden.

Teil II

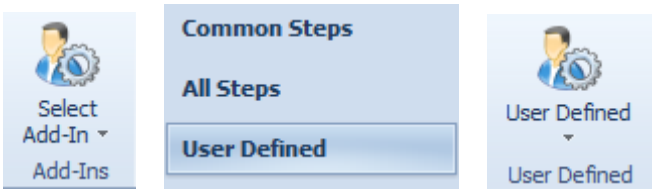
UMSETZUNG

4 ANALYSE DER SIMIO API

Das .NET Application Programming Interface (API) *Simio API Reference Guide.chm* steht jedermann zur Benutzung frei und kann im Simio Installationsverzeichnis gefunden werden. Das Dokument gibt eine Übersicht welche Daten dem Benutzer zur Verfügung stehen. Die Dokumentation ist jedoch minimal und Zusammenhänge sind nur in den wenigen Beispielen welche im Simio Insiders Forum und im *UserExtensions* Verzeichnis von Simio beschrieben sind verfügbar.

4.1 GRUNDLEGENDE ERWEITERUNGEN

In Simio sind bereits Schaltflächen für Erweiterungen im Graphical User Interface (GUI) sichtbar. Diese sind für *Add-In* im Project Home Reiter, *User Defined Element* im Definitionsfenster und *User Defined Step* im Prozessfenster definiert Abbildung 11.



(a) Add-In (b) Step (c) Element

Abbildung 11: GUI Zugriff auf benutzerdefinierte Erweiterungen

Der Methodenrumpf der *Execute* Methode des Interfaces *IDesignAddIn* wird ausgeführt wenn auf das Add-In Ribbon geklickt wird. Dies ist die einzige Möglichkeit die zur Verfügung steht um selbstgeschriebenen Code aktiv auszuführen. Beispielsweise können durch Windows Forms oder Windows Presentation Foundation (WPF) Daten repräsentiert bzw. durch den Benutzer angefordert oder manipuliert werden. Der Methode wird ein *IDesignContext* Objekt mitgeliefert über welches auf das aktive Model zugegriffen werden kann. Wird ein Benutzerdefinierter Step oder ein Element erstellt, hat Simio die volle Kontrolle und der Code wird zu gegebener Zeit ausgeführt.

Für einen Teil der Objekte in Simio stehen Interfaces zur Verfügung die die Handhabung unterstützen wie z.B. *IElementObject*, *IEvent* und *IFacility*. Ist ein solches Objekt durch den Benutzer dirket konfigurierbar steht zudem ein entsprechendes Definitionsinterface zur Verfügung wie *IElementDefinitionInterface*. Dieses ermöglicht die Anpassung von Eingabefeldern in Simio (siehe Listing 1 und Abbil-

dung 12). Für die Hauptelemente der Simulation wie Source, Server, Sink und Entity stehen jedoch keine Interfaces zur Verfügung.

Listing 1: Beispiel eines Definitionsinterfaces

```
1  public string Name => "Custom Walking";
2  public void DefineSchema(IPropertyDefinitions propertyDefinitions
       )
3  {
4      var updateTimeInterval = propertyDefinitions.
           AddExpressionProperty("UpdateTimeInterval", "0.5")
5      as IExpressionPropertyDefinition;
6      updateTimeInterval.UnitType = SimioUnitType.Time;
7      (updateTimeInterval.DefaultUnit as ITimeUnit).Time = TimeUnit
           .Seconds;
8      updateTimeInterval.Required = true;
9      updateTimeInterval.DisplayName = "Update Time Interval";
10 }
```

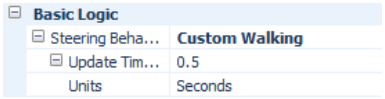


Abbildung 12: Anzeige der im Definitionsinterface definierten Werte und Properties

4.2 ERSTELLEN EINES NEUEN OBJEKTES

Das *Facility* Objekt welches durch das aktive Model bezogen werden kann stellt eine Methode zur Objekterstellung zur verfügung. Diese Methode ist die einzige Möglichkeit Objekte zur Modellierzeit in der Facility zu erstellen und in Simio verfügbar zu machen. Diese Methode beschränkt sich jedoch auf die bereits in Simio existierenden Objekte wie z.B. Source, Server oder Sink und verhindert somit die benutzung selbst Implementierter Klassen. Durch einen String-Parameter wird der Methode mitgeteilt was für ein Objekt erstellt werden soll. Hier kann im gegensatz zu Simio auch ein Agent erstellt werden. Dieser besitzt jedoch sogut wie keine Funktionalität da er wie in Abbildung 6 erwähnt eine Vorlage für die Objekte Entity und Transporter darstellt.

Listing 2: Erstellung von Objekten

```
1  //Folgende Zeile erstellt ein Source Objekt in der Facility
2  var source = intelligentObjects.CreateObject("Source", new
       FacilityLocation(0, 0, 1)) as
3  IIntelligentObject;
4  //Folgende Zeile resultiertin einer NULL-Referenz (TestAgent
       implementiert IIntelligentObject)
5  var agent = intelligentObjects.CreateObject("Agent", new
       FacilityLocation(0, 0, 1)) as TestAgent;
```

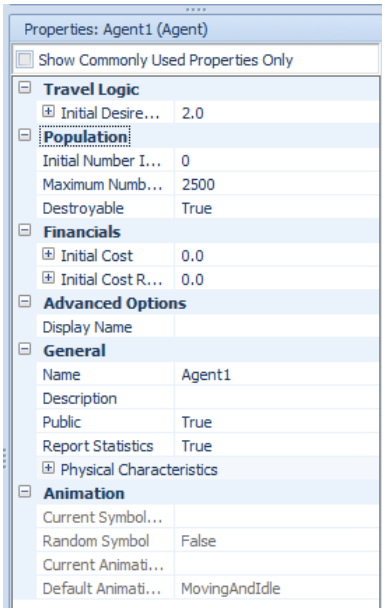


Abbildung 13: Properties des Agenten aus Abbildung 6

4.3 EINFLUSS AUF DIE BENUTZEROBERFLÄCHE

Der einzige Ansatz um Einfluss auf die Benutzeroberfläche zu nehmen wurde im *IDesignContex* Objekt mit der Methode *ExecuteUICommand* gefunden. Diese Methode kann laut API dazu benutzt werden Befehle durch die Benuzteroberfläche zu emulieren. Die konkrete Funktionsweise ist jedoch nicht beschrieben und konnte auch nicht durch Trial und Error während dieser Arbeit evaluiert werden. Das Einbetten von benutzerspezifischen Ansichten in Simio wird nicht unterstützt.

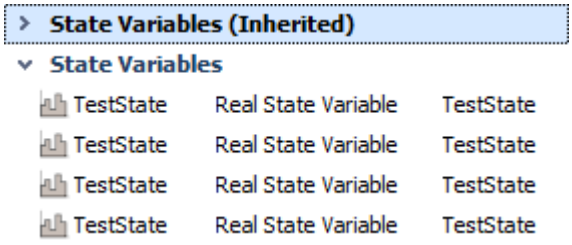
4.4 MANIPULATION VON DATEN

Das definieren von Daten wie z.B. States und Properties kann zur Modellierungszeit über das *IDesignContext* Objekt vorgenommen werden. Das Erstellen von States durch Code ist jedoch mit Vorsicht vorzunehmen. Durch Experimientieren haben wir festgestellt, dass es möglich ist, mehrere identische States zu erstellen (siehe Abbildung 14). Dieses Verhalten ist in Simio nicht nachzubilden und könnte ein Bug der API sein. Zur Laufzeit können jedoch keine Definitionen von solchen Daten mehr hinzugefügt bzw. gelöscht werden was die Implementation eines Lernenden Agenten ausschliesst. Die States können zur Laufzeit angepasst werden. Diese sind jedoch auch in Simio schon zur Laufzeit manipulierbar. Somit stehen zwei möglichkeiten zur Veränderung von States durch Code zur Verfügung. Einerseits können States im entsprechenden Kontext Objekt über den Namen abgerufen und manipuliert werden. Andererseits können Simio-

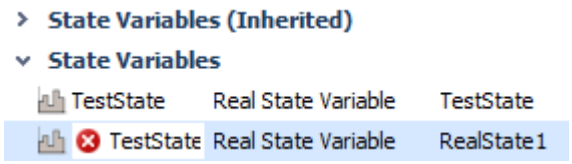
Events gefeuert werden welche in Simio abgegriffen werden können und in einem Simio Prozess die Manipulation des States durch einen *Assign*-Step auslösen. Letztere wird jedoch in dieser Arbeit bevozugt, da nicht Funktionalität die von Simio bereits unterstützt wird auch noch im Code implementiert werden soll.

Listing 3: Hinzufügen eines States zum aktiven Model

```
1 _context.ActiveModel.StateDefinitions.AddRealState("TestState");
```



(a) Durch Benutzercode erstellte States



(b) Durch Simio erstellte States

Abbildung 14: Identische States in Simio

5 PROTOTYP

Als Prototyp wird ein Agent in Simio umgesetzt. Die Agenten haben einen Einkaufsladen als Ziel. Sie wählen den Weg zu ihrem gewünschten Ziel. Falls sie auf ihrem Weg zum Ziel eine Werbung der Konkurrenz sehen, ändern sie ihre Meinung und ändern das Ziel auf den Laden der Konkurenz.

Folgende Aspekte eines Agenten werden realisiert:

1. *Bewegung:* Die Agenten bewegen sich frei und sind nicht an einen Pfad gebunden. Sie erkennen andere Agenten und Objekte und weichen diesen aus.

2. *Kommunikation:* Das Austauschen von Informationen ist auf eine bestimmte Distanz möglich.

3. *Reaktivität und Aktivität:* Anhand der erhaltenen Informationen können die Agenten ihr Ziel ändern.

5.1 TUTORIAL PROTOTYP

Der Prototyp ist ein eigenes Steering Behavior in einem Travel-Step. Für die Ausführung des Prototyps muss das entwickelte Add-In gestartet werden. (Abbildung 15).

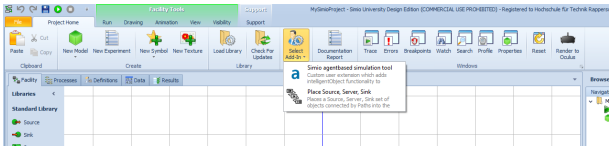


Abbildung 15: Starten des Add-In in Simio

Der Travel-Step mit dem eigenen Steering Behavior kann auch ohne die Ausführung des Add-In ausgewählt werden. Jedoch wird beim Start der Simulation eine Fehlermeldung angezeigt.

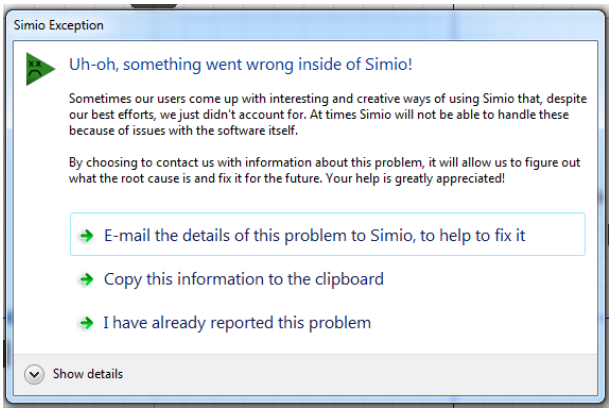


Abbildung 16: Fehlermeldung wenn vergessen wurde das Add-In zu starten

Beim Öffnen des Add-In wird ein Fenster angezeigt (Abbildung 17). Beim Knopf Add kann ein Obstacle (Hindernis) hinzugefügt werden. In Simio können Grafiken erstellt und Bilder eingefügt werden um eine möglichst reale Umgebung zu erschaffen. Da von der API nicht darauf zugegriffen werden kann, konnte keine Erkennung der Agenten für ihre Umgebung implementiert werden. Das Add-In ermöglicht durch das Einfügen eines Hindernis ein Objekt hinzuzufügen, welches Agenten erkennen und darauf reagieren können.

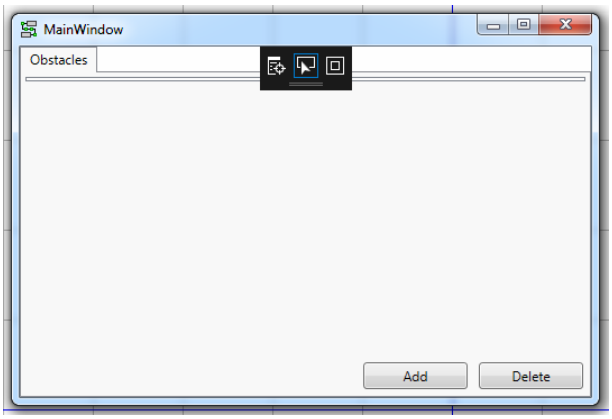


Abbildung 17: Fenster zum einfügen eines Hindernisses

Für die Ausführung des Prototypen muss minimal eine Source, zwei Sinks, ein Entity Model und ein weiteres IntelligentObject, das als Werbung dient, vorhanden sein (Abbildung 18). Das eingefügte Hindernis ist vom Typ IntelligentObject, da von Agenten nur IntelligentObcets erkannt werden.

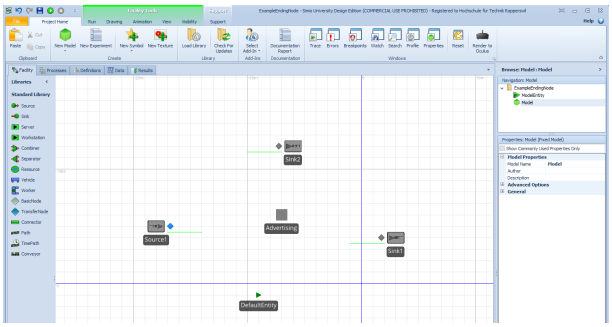


Abbildung 18: Aufbau des Prototyps

Das eingefügte Hindernis wird während der Simulation noch nicht angezeigt. Dem Hindernis wird ein Symbol hinzugefügt und unter dem Reiter Visibilty das Feld *Entity Instances* angewählt.

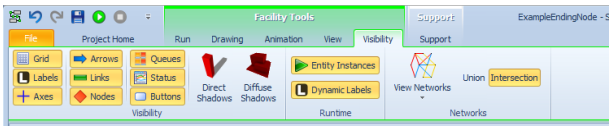


Abbildung 19: Der Visibility Reiter

Der Source wird ein Prozess beim Exit hinzugefügt (Abbildung 20). Der Prozess besteht aus einem *Transfer*-Step. Der *Transfer*-Step bewegt das Entity vom aktuellen Node in den Freespace (freier Raum). Beim folgenden *Travel*-Step kann in der Detailansicht (Abbildung 21) als Steering Behavior (Lenkverhalten) Freespace Walking to Destination anegeben werden. Mit den folgenden Properties kann die Distanz angegeben werden in welcher ein Agent auf einen anderen Agenten reagiert und auszuweichen beginnt. Das Property Reaction Distance legt die Distanz in welcher ein Agent auf ein Objekt reagiert fest.

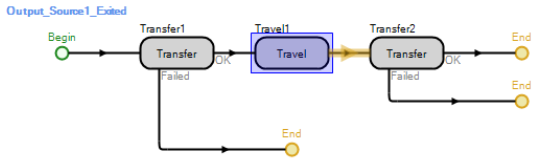


Abbildung 20: Der Prozess im Model

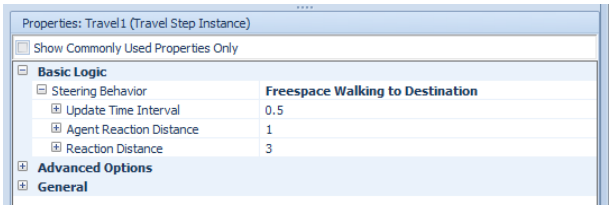


Abbildung 21: Detailansicht des Travel-Steps

Der letzte *Transfer*-Step beendet die Bewegung der Entity, wenn diese ihr Ziel erreicht hat.

Dem Entity Model wird der End-Node und Kommunikations-Node mit zwei States angegeben. Der Kommunikations-Node ist im Prototyp die Werbung.

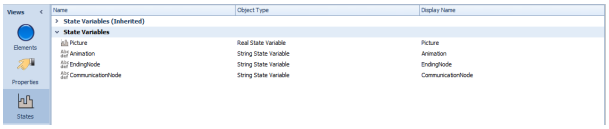


Abbildung 22: Ansicht der States des Entity Models

Beim Entity Model wird ein Prozess mit einem *Assign*-Step hinzugefügt. Der Prozess wird durch das Event Information Update gefeuert. Dieses Event wird aufgerufen, wenn der Agent das Plakat sieht und seine Meinung ändert. Der Agent läuft nicht mehr zu seinem ursprünglichen End-Node. Der *Assign*-Step fügt dem *EndNode*-State des Agenten den Namen des neuen End-Nodes hinzu (Abbildung 24).

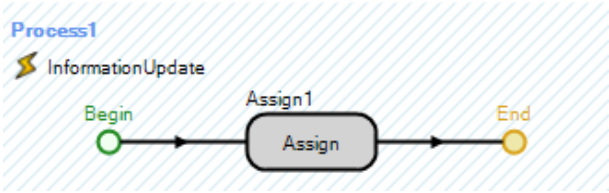


Abbildung 23: Prozess des Entities mit Assign-Step

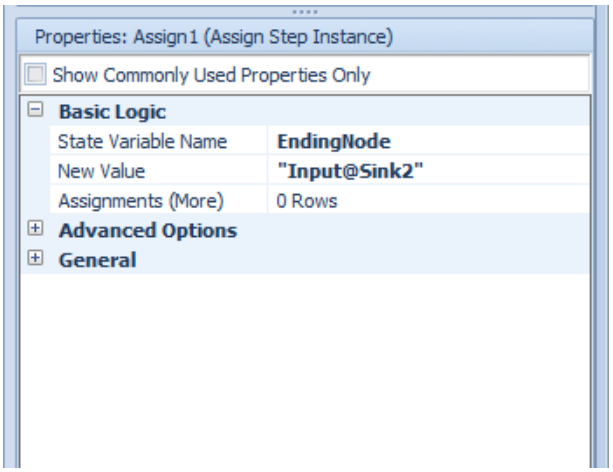


Abbildung 24: Detailansicht des Assign Step

Die Simulation wird gestartet (Abbildung 25). Die Entities bewegen sich auf ihren im State Endknoten definiertes Ziel zu. In dieser Anwendung auf die Sink1. Falls sie die Werbung passieren, ändern sie ihr Ziel und steuern auf die Sink2 zu. Die Objekte können während der Laufzeit verschoben werden. Dabei lässt sich beobachten, wie die Agenten selbstständig erkennen, dass sich die Position ihres Ziels verändert hat. Anhand des veränderten Wissens wird das Bewegungsverhalten angepasst.

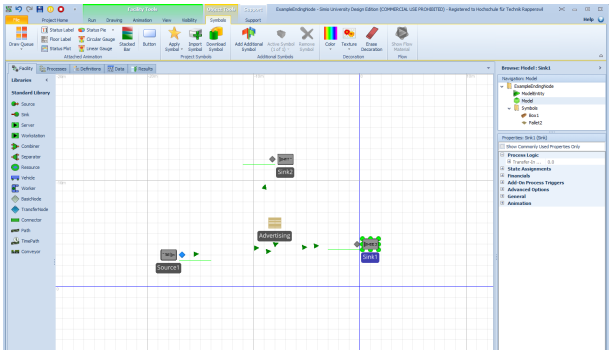


Abbildung 25: Der laufende Prototyp

5.2 TUTORIAL ERSTELLUNG STEERING BEHAVIORS

Die Simio API bietet einen *Travel*-Step. Im *Travel*-Step lässt sich ein Steering Behavior auswählen (Abbildung 21). Die API bietet ein Interface *ITravelSteeringBehavior* an, mit welchem ein eigenes Bewegungsverhalten implementieren werden kann.

Die wichtigsten Klassen zur Umsetzung einer eigenen Bewegung sind:

ITravelSteeringBehaviorContext

ITravelSteeringBehavior

TravelSteeringMovement

Abbildung 26: API des Steering Behavior

Das Steering Behavior Interface wird implementiert (Abbildung 26). In der Steer()-Methode wird das Bewegungsverhalten implementiert. Innerhalb der Steer-Methode wird der aktuelle Ort des Agenten ausgelesen. Die Überprüfung ob der Agent auf einen anderen Agent oder Objekt reagieren muss, findet statt. Der neue Bewegungsvektor wird berechnet. Innerhalb der *Steer*-Methode wird ein neues *TravelSteeringMovement* mit dem neuen Bewegungsvektor erstellt.

```
1 context.SetPreferredMovement(new TravelSteeringMovement()
2 {
3         Direction = direction,
4         Velocity = entity.velocity,
5         OrientInTheSameDirection = true,
6         Acceleration = acceleration,
7         Acceleration = accelerationVector,
8         AccelerationDuration = time
9 });
```

Listing 4: TravelSteeringMovement

Beim Prototyp wurde die Beschleunigung (Acceleration) nicht betrachtet. Dieses *TravelSteeringMovement* beschreibt den neuen Bewegungsvektor.
Das *TravelSteeringMovement* wird dem Kontext *ITravelSteeringBehaviorContext* mit der Methode *SetPreferredMovement(TravelSteeringMovement)* übergeben. Die *Steer*-Methode wird dem Kalender des Kontexts als Callback mit einem Zeitintervall übergeben.

```
1 context.Calendar.ScheduleEvent(context.Calendar.TimeNow +
      timeInterval, null, _ => Steer(context));
```

Listing 5: ITravelSteeringBehavior Kalender

```
context.SetPreferredMovement(new TravelSteeringMovement()
{
        Direction = direction,
        Velocity = entity.velocity,
        OrientInTheSameDirection = true,
        Acceleration = acceleration,
        Acceleration = accelerationVector,
        AccelerationDuration = time
});
```

Listing 6: TravelSteeringMovement

Der Kontext implementiert das Interface *ITravelSteeringBehaviorContext* und ist der Ausführungsrahmen des Steering Behaviors.

Simio API Reference Guide

ITravelSteeringBehaviorContext Interface

API Reference ▸ SimioAPI.Extensions ▸ **ITravelSteeringBehaviorContext**

An interface used when the engine calls SimioAPI.ITravelSteeringBehavior.Initialize.

Members

All Members | Methods | Properties

☑ Public ☑ Protected ☑ Instance ☑ Static ☑ Declared ☑ Inherited

Icon	Member	Description
	Calendar	The object used to manipulate the simulation calendar (Inherited from IExecutionContext.)
	EntityData	The runtime data for the Entity
	ExecutionInformation	The object used to report runtime information (Inherited from IExecutionContext.)
	GlobalData	An object reference which is passed to all steering rules originating from the same step.
	MovementFinished()	Called by the steering behavior when movement has been finished. If a token is waiting at a Travel step it won't be released until this point. Calling this method tell Simio the steering behavior is complete.
	Random	The object used to query the random number generator (Inherited from IExecutionContext.)
	SetPreferredMovement (TravelSteeringMovement)	Assigns the preferred movement of the entity in terms of a directional velocity vector in the units meters per hour. The Simio engine may decide not to actually ultimately assign this movement based on other factors. Calling this method simply indicates what the steering behavior prefers the movement to be.
	WithCandidate (IIntelligentObjectRuntimeData)	Creates a new execution context with the provided object assigned as the Candidate object. (Inherited from IExecutionContext.)

Abbildung 27: API des Steering Behavior Context

Der Kalender plant die Methode nach jedem Intervall in der Simulationszeit aufzurufen, solange der Agent das Steering Behavior ausführt. Im Prototypen ist das Intervall bei Verwendung der Standardkonfiguration auf 0.5 Sekunden gesetzt. Dadurch wird alle 0.5 Sekunden einen neuen Bewegungvektor für den Agenten gesetzt. Das Intervall kann bei der Detailansicht vom *Travel*-Step angepasst werden (Abbildung 21).

5.3 UMSETZUNG DES BEWEGUNGSVERHALTEN

Die Beschleunigung wurde bei diesem Prototyp nicht beachtet. Falls auf keine anderen Objekte oder Agenten reagiert werden muss, wird der Normalvektor zwischen dem Agenten und seinem Ziel berrechnet. Der Richtungsvektor zusammen mit der Geschwindigkeit werden dem *TravelSteeringMovement* übergeben, welches dann den Bewegungsvektor berechnet.

```
1 FacilityDirection directionVector = (endingNode.Location - entity
      .Location).ToUnitLength();
```

Listing 7: TravelSteeringMovement

5.4 UMSETZUNG DER ERKENNUNG

In den Properties des Steering Behavious werden die Distanzen zur Erkennung von anderen Objekten und Agenten in Meter angegeben (Abbildung 21).
Der Agent besitzt zwei Reaktionsvektoren (Abbildung 28). Der erste Vektor ist so lang wie im Property definiert wurde, der zweite ist halb so lang.

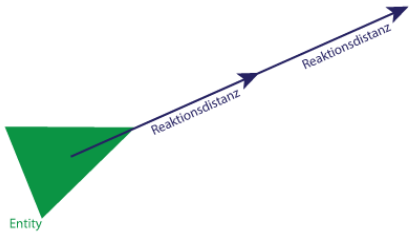


Abbildung 28: Reaktionsvektoren eines Agenten

Der kleine Reaktionsvektor dient dazu Objekte zu erkennen wenn der grosse Vektor bereits darüber hinaus geht (Abbildung 29). Der Reaktionsvektor hat drei Variablen X, Y, Z. Die Y Variable wird nicht beachtet, da sich die Bewegung in der Ebene abspielt. In Simio ist Y die Variable für die Dimension Höhe. Bei der Erkennung wird überprüft ob der X und Z Wert des Vektors innerhalb der Fläche des Objektes liegt.
Im Prototyp ist das Invervall für den Berechnungsvektor auf 0.5 Sekunden eingestellt. Vom Moment an, wenn der Agent seine Source verlässt, wird alle 0.5 Sekunden der neue Bewegungsvektor berechnet. Die Berechnung des Bewegungsvektors geschieht bei jedem Agenten zu einem anderen Zeitpunkt. Daher ist es möglich, dass obwohl vor der Bewegung überprüft wurde ob der Weg blockiert ist, ein anderer Agent sich dazwischen bewegt hat.

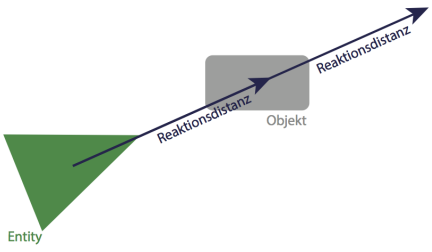


Abbildung 29: Erkennung durch Reaktionsvektoren

Der Agent kann auf Objekte, die in seiner Bewegungsrichtung sind, reagieren. Das Spektrum kann erweitert werden indem dem Agenten weitere Reaktionsvektoren in andere Richtungen hinzugefügt werden.

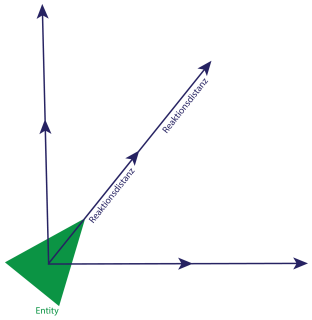


Abbildung 30: Mehrere Reaktionsvektoren

Dem Agenten können verschiedene Vektoren angegeben werden. Im Prototyp besitzt der Agent Reaktionsvektoren für Agenten und Reaktionsvektoren für Objekte seiner Umwelt. Es können unterschiedliche Distanzen eingegeben werden.
Der Agent reagiert auf einen anderen Angenten wenn er einen Meter von ihm entfernt ist. Bei Objekten seiner Umwelt ist die Distanz drei Meter. Die Unterscheidung ergibt ein natürlicheres Verhalten der Agenten. In der realen Welt nehmen wir Häuser etc. von Weitem wahr und wählen unseren Weg entsprechend.

5.5 UMSETZUNG DES AUSWEICHEN

Überschneidet ein Vektor ein anderes Objekt, wird die Abwendung berechnet (Abbildung 31). Die Abwendung kann auch durch einen fixen Wert multipliziert werden um eine stärkere Lenkung zu erzielen. Überschneidet der kürzere Vektor ein anderes Objekt ist es sinnvoll eine stärkere Abweichung einzuleiten.

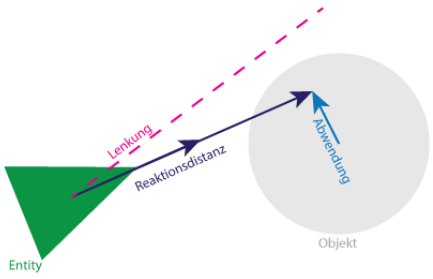


Abbildung 31: Ausweichen eines Objektes

Falls der Agent einen Bewegungsvektor in der exakten Richtung wie das kollidierende Objekt hat, würde der Abwendungsvektor in die Gegenrichtung zeigen. Da sich der Agent wieder auf sein Ziel aus-

richtet, wählt er wieder dieselbe Richtung. Somit blockiert der Vektor vor dem Objekt. In diesem Falle wird ein perpendikulärer Vektor zur Berechnung der Abwendung zur Hilfe genommen.

5.6 UMSETZUNG DER REAKTION

Die Reaktion auf die Werbung wird durch States und Events umgesetzt. In Simio wird dem Entity Model zwei States mit den Namen CommunicationNode und EndingNode hinzugefügt (Abbildung 22). Der CommunicationNode hat einen String mit dem Namen des Nodes mit welchem der Agent kommunizieren kann.
Das Entity Model hat einen Prozess mit einem *Assign*-Step, welcher beim *Trigger*-Event InformationUpdate ausgeführt wird. Der *Assign*-Step fügt dem State EndingNode einen neuen Wert hinzu. Falls der Erkennungsvektor den Kommunikationspunkt detektiert, wird das Event InformationUpdate gefeuert.

```
1 if(Intersect(perceptionVector, CommunicationNode)
2 {
3         InformationUpdate.fire();
4 }
```

Listing 8: Feuern des InformatonUpdate Events

5.7 UMSETZUNG HINDERNIS

Das Einfügen eines Hindernisses wurde umgesetzt da ein Agent sonst nicht auf seine Umwelt reagieren kann. Somit kann ein Agent sich frei bewegen und sich seinen Weg zu seinem Ziel bahnen. Es lassen sich aus der API Objekte wie zum Beispiel Source, Sink etc. instanzieren jedoch nicht neue eigene Objekte erstellen. Die Obstacle sind von der Klasse Agent von Simio. Sie haben jedoch keine Funktionalität.

5.8 AUSBAU DES PROTOTYPS

Das Ziel des Ausbaus ist, die erfolgreich umgesetzten Funktionen des Prototyps auf die Konzepte Wahrnehmung und Kommunikation aus Kapitel 3 als Objekte zu abzubilden. Dadurch wird der Prototyp zu einem Konstrukt welches offen für die Implementation von neuem Verhalten ist und für zukünftige Projekte benutzt werden kann.
Die Weiterentwicklung des Prototyps wird separat geführt und ist im CustomBehavior Ordner der AgentAddIn Solution zu finden. Der jeweils nicht verwendete Prototyp muss vom Projekt entfernt werden wenn ein Beispiel ausgeführt wird.

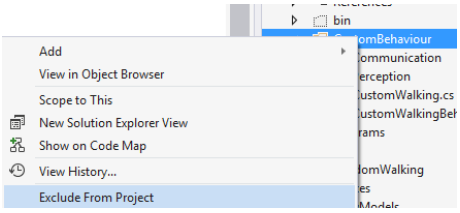


Abbildung 32: Prototyp aus Projekt entfernen

Die Klasse *CustomWalking* welche das Interface *ITravelSteeringBehavior* implementiert, hält je eine Liste mit *Perception*(Wahrnehmung) und *Communication*(Kommunikation) Objekten. *Perception* und *Communication* sind beides abstrakte Klassen welche spezifisches Verhalten den abgeleiteten Klassen überlassen. Die *Perception* Objekte dienen der bildung von spezifischen wahrgenommenen Gruppen von Objekten und die *Communication* Objekte spezifizieren Reaktionen auf diese Gruppen. *Perception* Klassen besitzen Eventhandler über die sich *Communication* Objekte registrieren können.

```
1 var perceptionOne = new IntelligentObjectPerception(thisAgent, 5,
      "Server");
2 var communicationOne = new FireEventCommunication(thisAgent, "
    InformationUpdate",
3 Constants.FireOwnEvent);
4 perceptionOne.IntelligentObjectPercieved += communicationOne.
    OnPercieved;
```

5.8.1 *Wahrnehmung*

Die abstrakte *Perception* Klasse definiert Felder *peceptionDistance* und *Subject* wobei das *Subject* ein String mit dem Namen der Wahrgenommenen Objekten ist. In dieser Arbeit wurden zwei abgeleitete Klassen von *Perception* implementiert. Eine für die Wahrnehmung von IntelligentObjects und eine für die Wahrnehmung von Agenten(Entities). Diese Unterscheidung muss gemacht werden, da IntelligentObjects nur über den Kontext *IDesignContext* und Agenten nur über den Kontext *ITravelSteeringBehaviorContext* verfügbar sind.
Weitere mögliche Implementationen von Wahrnehmungsklassen wären z.B. Wahrnehmung von Prozessen oder der Gestalt eines Objektes.

5.8.2 *Kommunikation*

Die von *Communication* abgeleitete Klasse *CollicionAvoidance* kapselt die Kollisionsvermeidung aus dem bestehenden Prototypen.

5.8.2.1 *Events Feuern*

FireEventsCommunication ist eine abgeleitete Klasse von *Communication* und dient der Auslösung von Events. Als Parameter benötigt *FireEventsCommunication* einen String zur Identifierung des auszulösenden Events sowie einen Integer als Option. Über die Option wird entschieden ob der auszulösende Event auf sich selbst ausgelöst werden soll oder auf allen wahrgenommenen Objekten. Diese Optionen sind jedoch noch ausbaufähig. So könnten beispielsweise ein Event beim nächstgelegenen Objekt oder einer anderen Kondition ausgelöst werden. Durch die Auslösung eines Events wird die weitere Verarbeitung an Simio übergeben. In Simio könnne Prozesse durch den Event angestossen werden. In den Prozessen können durch *Assign*-Steps States verändert werden was es hinfällig macht eine *Communication* Klasse zu implementieren welche States manipuliert.

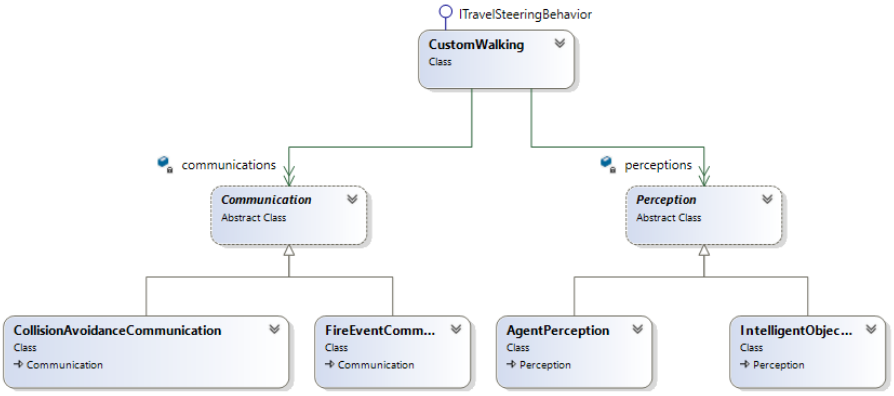


Abbildung 33: Klassendiagramm des ausgebauten Prototyps

6 ANFODERUNGSKATALOG

Anforderung	Begründung
Kontext für Agenten	Ein Agent muss seine Umwelt wahrnehmen können. Deshalb benötigt er Zugriff auf alle Objekte die er wahnehmen könnte. In unseren Prototypen ist der Agent auf die Sicht auf Agenten die im selben *Travel*-Step ihr Steeringbehaviour erhalten haben beschränkt. Nur mit statischem *IDesignContext* wird Zugriff auf die Objekte in der Facility ermöglicht.
Hinzufügen und entfernen von States, Properties, usw. zur Laufzeit	Im Rahmen dieser Arbeit nicht benötigt aber notwendig für die Implementierung von lernenden Agenten.
Zugriff auf das Simio GUI	Wird benötigt für eine saubere Integration. Ansichten können nur in separaten Fenstern erstellt werden. Erstellung von Add-In's durch Implementierung des *IDesignAddIn* hat für die agentenbasierte Simulation statische Daten zur Folge.

7 SCHLUSSFOLGERUNG

7.1 ERGEBNIS

Ergebnis der Arbeit ist ein Funktionsfähiger Prototyp der Ansätze der agentenbasierten Simulation umsetzt sowie ein Anforderungskatalog für Simio basierend unserer Analyse der Simio API. Der Prototyp ist erweiterbar aufgebaut und für weitere Projekte eine gute Grundlage. Zur Demonstration des Prototypen wurde ein Simio Beispielprojekt erstellt.

7.2 FAZIT

Das Ziel der Arbeit wurde erreicht. Der Prototyp ist aber noch stark ausbaufähig, selbst ohne die erfüllten Anforderungen aus dem Katalog. Neueinsteiger die sich weiter mit der API auseinandersetzen wollen soll diese Arbeit eine Hilfe sein.

7.3 AUSBLICK

7.3.1 *Ausbau der Kommunikation*

Das Konzept Kommunikation wurde in dieser Arbeit ermöglicht. Weitere Implementierungsmöglichkeiten können sich als herausfordernd erweisen wegen dem Übergang zwischen Simio und Codeumgebung.

7.3.2 *Ausbau der Wahrnehmung*

Die Wahrnehmung von Objekten ist im Prototyp implementiert. Eine mögliche Erweiterung besteht darin, globale Effekte oder Veränderungen der Umgebung wahrzunehmen.

7.3.3 *Konzept Wissen*

Das Konzept Wissen konnte aus Komplexitätsgründen nicht weiter verfolgt werden. Dieses Konzept kann jedoch von sich aus eine weitere Projektarbeit mit grösserem Aufwand rechtfertigen.

Teil III

ANHANG

PERSÖNLICHER BERICHT

Die Arbeit „Ausbau der agentenbasierten Simulation in Simio“ war sehr vielseitig und Lehrreich. Die Erarbeitung von Konzepten zu agentenbasierter Simulation war äusserst spannend und Anspruchsvoll, da zu dem Thema viele verschiedene Ansichten existieren. Aus meiner Sicht war es deshalb auch sehr entlastend regelmässig Diskussionen mit unseren Betreuern zu führen.
Die Analyse der Simio API fand ich sehr harzig, da diese kaum dokumentiert war und somit sehr viel Energie und Zeit für kaum sichtbare Resultate verschlang. Viele Ansätze konnten nicht verfolgt werden oder führten zu Sackgassen. Als wir jedoch die Einstiegspunkte für unsere Konzepte in der Simio API ermittelt hatten konnte es endlich mit der Entwicklung in .NET voranschreiten und machte wieder mehr Spass. Ich konnte mich gut in die Entwicklungsumgebung einarbeiten und Entwicklungserfahrung sammeln. Gerne hätte ich jedoch in dieser Phase noch weitergearbeitet aber die Arbeit war leider zu Ende als ich so richtig in fahrt kam.
Ich bin mit den erreichten Ergebnissen der Arbeit zufrieden. Der Prototyp zeigt auf was erreichbar ist und der erstellte Anforderungskatalog erörtert die übrigen nötigen Anpassungen um den Prototyp komplett komplett zu integrieren. Ich konnte mir ein breites Wissen über Simulation aneignen und hoffe dies in Zukunft wieder einmal anwenden zu können. Auch die Zusammenarbeit zwischen Studien- und Bachelorarbeit funktionierte aus meiner Sicht wieder sehr gut und hat zu keinerlei Komplikationen geführt.

A

KATALOG EXISTIERENDER LÖSUNGEN

A.1 ALLGEMEINE AGENTENBASIERTE APPLIKATIONEN

Anwendung	Produkt
Allgemeine agentenbasierte Anwendung	AnyLogic
	Ascape
	DOMAR
	JAS
	MAGSY
	Mason
	SeSAm
	SimPack
	Swarm
Multi-agent Systeme mit agentenbasierter Simulationsschicht	Madkit
Allgemeine agentenbasierte Anwendung für parallele Applikationen	DeX

Tabelle 3: Software für allgemeine agentenbasierte Anwendung [6]

A.2 APPLIKATIONEN FÜR VERTEILTE SIMULATIONEN

Anwendung	Produkt
Allgemeine Anwendung für verteilte Simulationen	AnyLogic
	DOMAR
	Jade
	Madkit
	ADK
	Cougaar
	SimAGent
	oRIS

Tabelle 4: Software für verteilte Simulation [6]

A.3 APPLIKATIONEN MIT PÄDAGOGISCHEM FOKUS

Anwendung	Produkt
Allgemein für Bildung in Simulation	MIMOSE
	NetLogo
	StarLogo
	TNG
	VSEit
Aritificial Intelligence	Breve
Für Studenten um das Verhalten von dezentralisierten Systemen zu modellieren	StarLogo
	TNG
Implementierung von Software Agenten	Brahms
	SimAgent
Lernverhalten	SOAR
Computer Simulation	FAMOJA
	Matlab
	oRIS
	SeSAm
	SimPack
Soziale Wissenschaften, Mathematik und Naturwissenschaften	AgentSheets
	StarLogo
Objektorientierte Prinzipien	jECHO
Programmiertechniken für neue Anwender im Bereich Simulation	Matlab
	NetLogo
	StarLogo
	Sugarscape
Wissenschaftliche und technische Mathematik, Datenanalyse, Exploration und Visualisierung	Matlab

Tabelle 5: Software mit pädagogischem Fokus [6]

A.4 APPLIKATIONEN FÜR MULTI-AGENT SYSTEME

Anwendung	Produkt
Allgemeine Anwendung für vMulti-agent Systeme	AgnetBuilder
	oRIS
	MAGSY
Verteilte Multi-agnt Systeme	ADK
	Cougaar
	Jade
Komplexe Umwelt	SimAgent
Organisationsprozesse	Brahms

Tabelle 6: Software spezialisiert für Multi-Agent Systeme [6]

A.5 APPLIKATIONEN FÜR ARTIFICIAL INTELLIGENCE

Anwendung	Produkt
Allgemeine Anwendung für Artificial Intelligence	Breve
	iGen
	SOAR
Machine Learning und logisches Denken	ABLE
	Zeus
Soziale Wissenschaften	Omonia
3D Simulationen	Breve
Menschenähnliche intelligente Agenten	SimAgent

Tabelle 7: Software für Artificial Intelligence [6]

A.6 APPLIKATIONEN FÜR SOZIALE WISSENSCHAFTEN

Anwendung	Produkt
Allgemeine Anwendung für Soziale Wissenschaften	AgentSheets
	LSD
	FAMOJA
	MAML
	MAS-SOC
	MIMOSE
	NetLogo
	Repast
	SimBioSys
	StarLogo
	Sugarscape
	VSeit
Hilft Anfängern beim erstellen von Modellen	NetLogo
Soziale Systeme	Moduleco

Tabelle 8: Software für Soziale Wissenschaften [6]

A.7 APPLIKATIONEN FÜR NATURWISSENSCHAFTEN

Anwendung	Produkt
Allgemeine Anwendung für Naturwissenschaften	Matlab
	NetLogo
	OpenStarLogo
	StarLogo

Tabelle 9: Software für Naturwissenschaften [6]

A.8 APPLIKATIONEN FÜR SEHR SPEZIFISCHE ANWENDUNGEN

Anwendung	Produkt
Angewandte Simulation/Elektronisches CAD	Jade's Sim++
Biologie	SimBiosSys
Zelluläre Automaten	JCA-Sim
Ökonomie/Auktionsmechanimen	JASA
Ökologische Modellierung	Echo
	jEcho
	SME
Unternehmen	jES
Evolutionäre Algorithmen	ECJ
Menschliche Leistungen Modellieren	iGen
Natürliche Ressourcen Management	Cormas
Politische Phänomene	PS-I
Regel-Maschine und Scripting Umgebung	JESS
	Zeus
Organisatorische Prozesse simulieren	Brahms
Base24 Applikationen Testen	SimPlusPlus
Städte Simulationen	OBEUS

Tabelle 10: Software für sehr spezifische Anwendungsgebiete [6]

GLOSSAR

Boolean	Variable die nur Zwei Zustände annehmen kann. 23
bottom-up	Eine Vorgehensweise bei der Problemlösung. Zuerst werden Teilprobleme gelöst, mit deren Hilfe grössere, darüber liegende Probleme in Angriff genommen werden. Die einzelnen Teillösungen werden von unten nach oben zusammengesetzt, bis das Gesamtproblem gelöst ist. 12, 15
Element	Datentyp welcher exklusiv für Prozesse in Simio bestimmt ist. 16
emulieren	Funktionen eines Computers/Programmes auf einem anderen nachbilden. 29
Entity	Teil eines Objektmodells in Simio welcher eigenes Verhalten hat. 15, 16, 28, 32–35, 40, 41, 65, 72
Event	Ereignis. In Simio: Objekt welches ein Ereignis in Simio repräsentiert das ausgelöst(gefeuert) werden kann. Prozesse können auf das feuern eines Events in gang gesetzt werden. 16, 30, 34, 40, 42, 68
Facility	Simio Laufzeitumgebung eines Modells. Die Graphische repräsentation einer Simulation befindet sich in Simio in der Facility Ansicht. 28, 43
Float	Datentyp der Fliesskommazahlen speichert. 23
Integer	Datentyp der ganzzahlige Werte speichert. 23, 42
Link	Verbingdungselement in Simio. 16
Node	Knotenpunkt zwischen Links in Simio. 16, 33, 34, 40
perpendikulärer	senkrecht, lotrecht. 40
Property	Eigenschaft. In Simio: Eigenschaften eines Simio Objektes welche zur Modellierungszeit defniert und festgelegt werden. Diese können zur Laufzeit nicht mehr verändert werden. 16, 28, 29, 33, 38, 43, 58, 65

Scope	Rahmen in welchem Objekte, Daten und Funktionen sichtbar sind. In Simio haben auch Prozessen einen Scope. 16
Server	Repräsentiert eine Resource mit Kapazität mit optionalen Input- und Outputbuffern in Simio. Besitzt andere zusätzliche Parameter als die Workstation. 16, 28
Sink	Die Sink zerstört Entities und kann Statistiken führen. 28, 32, 35, 40
Source	Ein Objekt welches die Erstellung von Entities durch eine spezifizierte Rate oder ein spezifiziertes Ankunftsmuster in Simio erlaubt. 28, 32, 33, 38, 40
State	Zustand. In Simio: Zustände eines Simio Objektes welche zur Modellierungszeit definiert werden. States können zur Laufzeit verändert werden. 29, 30, 34, 35, 40, 42, 43, 65
Step	Aktivität in einem Prozess in Simio. Wird benötigt um Logik zu definieren. 16, 17, 27, 30, 31, 33–35, 37, 40, 42, 43, 65
String	Datentyp zur Repräsentation von Zeichenketten. 28, 40–42
Token	Objekt welches für den Ablauf eines Prozesses in Simio benötigt wird. Führt Steps in Prozessen aus. 16, 17
top-down	Eine Vorgehensweise bei welcher der Entwurf mit abstrahierten Objekten beginnt die konkretisiert werden. 12
Transporter	Objekt welches eine Bewegliche Resource darstellt in Simio. 15, 28
Windows Forms	Programmierschnittstelle zur Erstellung graphischer Benutzeroberflächen. 27
Workstation	Repräsentiert eine Resource mit Kapazität mit optionalen Input- und Outputbuffern in Simio. Besitzt andere zusätzliche Parameter als der Server. 16

LITERATURVERZEICHNIS

[1] Eduard Babulak und Ming Wang. *Discrete Event Simulation*. http://cdn.intechopen.com/pdfs/11536.pdf. [Online; letzter Zugriff 20.03.2016].

[2] Andrei Borshchev. *The Big Book of Simulation Modeling: Multimethod Modeling with AnyLogic 6*. AnyLogic North America, 2013.

[3] Andrei Borshchev und Alexei Filippov. *From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools*. http://link.springer.com/chapter/10.1007/3-540-45982-0_1. [Online; letzter Zugriff 13.04.2016]. 2004.

[4] George Fishman. *Discrete-Event Simulation*. Springer-Verlag New York, 2001.

[5] Simio LLC. *Simio Reference Guide*. Download on www.simio.com/downloads/public/academic/SimioReferenceGuide.zip. [Online; letzter Zugriff 15.03.2016]. 2006 - 2012.

[6] Cynthia Nikolai und Gregory Madey. "Tools of the Trade: A Survey of Various Agent Based Modeling Platforms". In: *Journal of Artificial Societies and Social Simulation* 12.2 (2009), S. 2. ISSN: 1460-7425. URL: http://jasss.soc.surrey.ac.uk/12/2/2.html.

[7] Dirk Pawlaszczyk. *Skalierbare agentenbasierte Simulation*. http://link.springer.com/article/10.1007/s13218-010-0031-5/fulltext.html. [Online; letzter Zugriff 15.03.2016]. 2009.

[8] Wolf-Ulrich Raffel. *Agentenbasierte Simulation als Verfeinerung der Diskreten-Ereignis-Simulation*. http://www.wuraffel.de/wissenschaft/publikationen/Dissertation.pdf. [Online; letzter Zugriff 15.03.2016].

[9] PO Siebers, CM Macal, J Garnett, D Buxton und M Pidd. "Discrete-Event Simulation is Dead, Long Live Agent-Based Simulation!" In: *Journal of Smulation, 4(3)* (2010).

[10] Roger D. Smith. "Simulation Article". In: *Encyclopedia of Computer Science* (1988).

[11] Michael Wooldridge. *Intelligent Agents: The Key Concepts*. Springer-Verlag Berlin Heidelberg, 2002.

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

ACRONYMS

ABS agentenbasierte Simulation

API Application Programming Interface

BPMN Business Process Model and Notation

DES Diskrete Ereignissimulation

GUI Graphical User Interface

SD System dynamics

WPF Windows Presentation Foundation