Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at
TU UB WIEN Universitätsbibliothek
The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

DIPLOMARBEIT

Analyse der Struktur und Funktionalität eines Virtual Overlay Multi-agent System bei einem Influenza-Modell.

Ausgeführt am Institut für

Analysis und Scientific Computing

der Technischen Universität Wien
unter der Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr. techn. Felix Breitenecker

durch

Julian Rudolf Ruths

Vorgartenstraße 145/6/31
1020 Wien

9. Mai 2016
Datum

Unterschrift

Erklärung zur Verfassung der Arbeit

Julian Rudolf Ruths, Vorgartenstraße 145/6/31, 1020 Wien, Österreich

"Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Ort, Datum Unterschrift

Danksagung

Zunächst möchte ich Prof. Felix Breitenecker für die Betreuung der Diplomarbeit danken, ebenso Florian Miksch und Philipp Pichler für die vielen konstruktiven Gedanken und das intensive Fachsimpeln. Mein Dank geht auch an alle Mitglieder der „Agent Task Force“. Unsere Treffen waren wirklich eine Bereicherung.

An meine Mathehomies – Andi, Basti, Claire, Flo, Iri, Janne und Stefanopolus: Danke! Die viele Zusammenarbeit im Studium und die vielen lustigen Abende und Momente abseits der Universität haben mir wohl am meisten geholfen, einen kühlen Kopf zu bewaren.

Danke an Daniela, Marion, Mathias und Willi. Mit euch Musik zu machen, hat mir unwahrscheinlich viel Freude und Energie gegeben!

Georg, Nik und Sebastian: Was soll ich sagen? Ihr seids halt einfach meine Burschen. Punkt.

Michaela: Ohne dich hätte sowieso nichts funktioniert. Danke, dass du immer an meiner Seite bist. Ich bin dir für jeden Moment dankbar, den du mit mir verbringst – und für jeden falsch gesetzten Beistrich, (!) auf den du mich aufmerksam gemacht hast.

Mein größter Dank gilt meinen Eltern Andrea und Rudolf und meinen Schwestern Alexandra und Johanna. Ihr seid mir immer ohne zu zögern zur Seite gestanden und habt mir so vieles ermöglicht. Ich glaube, ihr wisst, wie viel ihr mir bedeutet. Ich danke euch von ganzem Herzen.

Kurzfassung

Motivation: Die Modellbildung versucht mit Hilfe einer vereinfachten Darstellung der Realität – dem Modell – Problematiken oder Prozesse der Wirklichkeit innerhalb dieses Modells darzustellen bzw. zu lösen. Eben diese Vereinfachungen machen es allerdings notwendig, zu evaluieren, ob die richtigen Annahmen und Abstraktionen für den darzustellenden Prozess getroffen wurden. Die Validierung soll dabei helfen, zu entscheiden, ob das Modell die zugrundeliegende Problematik ausreichend gut löst.

Agentenbasierte Modelle sind sehr flexibel und dank steigender Rechenleistung auch bei praktisch relevanten Problemen anwendbar, aber die Validierung solcher Modelle steht bis dato noch vor vielen Schwierigkeiten. Die komplexe Struktur von agentenbasierten Modellen und der starke Einfluss des Verhaltens einzelner Agenten auf das resultierende Verhalten des Modells macht die Anwendung klassischer Validierungsmethoden nur bedingt möglich. Spezielle Methoden für agentenbasierte Modelle wurden bereits entwickelt, aber bislang kaum eingesetzt und dokumentiert. Bei dem in dieser Arbeit beschriebenen VOMAS-Ansatz (Virtual Overlay Multi-agent System) handelt es sich um eine dieser neuen Validierungsmethoden.

Methoden: Basierend auf Literaturrecherchen wurden die Hauptprobleme der Validierung von agentenbasierten Modellen isoliert und in Frage kommende Validierungsmethoden identifiziert und beschrieben. Für den speziellen Fall der Validierung mittels eines VOMAS wurden ebenfalls Methoden aufgelistet, deren Anwendung durch das VOMAS ermöglicht bzw. erleichtert werden. Im Hinblick darauf, welche Eigenschaften ein VOMAS haben und was es dem Anwender ermöglichen soll, wurden alle Komponenten detailliert analysiert und eine passende Gesamtstruktur entwickelt.

Die Struktur wurde unter folgenden Annahmen an das Aussehen und Eigenheiten eines VOMAS entwickelt: Ein VOMAS ist hierarchisch angeordnet, mit einem Manageragent – dem VO-Manager – an der Spitze. Dieser delegiert den größten Teil der Informationssammlung an die ihm unterstellten VO-Agenten. Beim Zugriff des VOMAS auf die Simulation muss unterschieden werden, ob auf Simulationsagenten oder auf globale Variablen des Modells zugegriffen wird. Der für die Speicherung gesammelter Information zuständige VOMAS-Agent – der Logger-Agent – ist beinahe unabhängig vom Ablauf der Simulation oder den Tätigkeiten der anderen VOMAS-Agenten. Er sollte deshalb meist parallel zu den restlichen Abläufen arbeiten. Ist eine Speicherung der Informationen in Form eines anderen Mediums gewünscht, sollte er außerdem ohne viel Aufwand gegen andere Typen von Logger-Agenten ausgetauscht werden können. Weitere Komponenten des VOMAS sollten je nach Bedürfnissen ebenfalls austauschbar sein um eine effektive und flexible Datenaggregation zu ermöglichen.

Eine existierende Influenza Simulation wurde mit dem beschriebenen theoretischen

Ansatz ausgestattet und auf praktische Anwendbarkeit getestet.

Ergebnisse: Die Bausteine eines VOMAS wurden detailliert untersucht und ein Vorschlag für ein plattform- und beinahe anwendungsunabhängiges VOMAS-Design entwickelt. Die Formulierung der anwendungsabhängigen Komponenten wurde in den allgemeinen Modellierungskreislauf eingebettet, um die parallele Entwicklung von VOMAS und Modell zu erleichtern.

Das VOMAS-Design wurde für das Influenzamodell mit der objektorientierten Programmiersprache Java implementiert und ist in der Lage, eine Vielzahl neuer Daten zu sammeln, aus denen detaillierte Informationen zum Verhalten der Agenten und des Modells an sich gewonnen werden können. Mit Hilfe des VOMAS können unter anderem erstmals beide Seiten einer Infektion (aus Sicht der Person, die eine andere infiziert, und aus Sicht jener, die infiziert wird) protokolliert werden und diese Infektionen mit den Orten, an denen sie stattgefunden haben, in Verbindung gebracht werden. Die Ergebnisse unterstreichen, dass das Influenza-Modell weiter überarbeitet werden muss. So werden beispielsweise bei einer vernünftigen Parametrisierung des Modells 84% der Kleinkinder, 82% der Schüler und 74% der Arbeitstätigen über den Lauf einer Simulation infiziert.

Conclusio: Im Zuge dieser Arbeit wurden die Aufgabenbereiche der einzelnen Komponenten eines VOMAS detailliert beschrieben und eine Struktur vorgestellt, welche die Erledigung dieser Aufgaben effizient ermöglicht. Das entwickelte Design war im Anwendungsbeispiel des Influenza-Modells gut einsetzbar und konnte eine Vielzahl aufschlussreicher Informationen zum Verhalten der Agenten sammeln. Zusätzlich ist das Design flexibel in den meisten objektorientierten Systemen implementierbar. Dies unterstreicht das Potential des VOMAS-Ansatzes.

Abstract

Motivation: Modeling and simulation use simplified representations of real world systems – so called models – to find solutions to real problems. The simplifications and model assumptions chosen to create a model need to be examined however, to make sure the right assumptions were chosen in regards to the problem the model tries to solve. The process of validation helps to decide whether a model is able to solve a given problem in a satisfying way.

Agent-based models are becoming increasingly popular in the field of modeling and simulation. They are able to simulate very complex behavior and thanks to the ongoing increase of computing power, they can be applied to real world problems. In spite of their intuitiveness, they are very hard to validate. This is mostly due to their complex structure and emergent behavior. Therefore, new validation methods targeted at agent-based models were developed. A new approach for validating agent-based models uses a virtual overlay multi-agent system (VOMAS). However, so far very few cases about the usage of VOMAS in a model have been documented.

Methods: Based on literature regarding agent-based modeling and validation, this thesis sets out to identify the main problems which occur when faced with the task of validating an agent-based model. Validation methods, which can still be used for agent-based models were listed, as well as those methods that the use of a VOMAS make either possible or at least easier to apply. Keeping in mind the initial concept of a VOMAS, each component of the multi-agent system was analyzed in detail and a basic structure was developed.

Some of the basic ideas and abilities a VOMAS should have are (1) a hierarchal structure with a manager agent at the top delegating tasks to its subordinate agents; (2) when gathering information, one has to distinguish between accessing an individual simulation agent or a global attribute of the simulation itself; (3) the agent tasked with logging certain information should be able to work (a) independently from and (b) parallel to the other VOMAS agents as well as the simulation itself; (4) the way the information is saved for later use should also be up to the user; (5) other components should be made interchangeable to allow for a flexible and effective gathering of information on the model.

To test the usage of the developed VOMAS design, it was implemented for an influenza framework.

Results: The parts of a VOMAS were described in detail and a platform-independent design of a VOMAS was developed. The proposed design is, in most parts, also independent of the actual application. The remaining steps which need to be taken when developing a VOMAS for a specific model, were imbedded in the general modeling

cycle.

The VOMAS design was implemented for the influenza model using the object-oriented programming language Java. VOMAS was able to collect vast amounts of new data, out of which detailed information regarding the behavior of the agents and the model itself can be extracted. For the first time information about infectious contacts between persons could be gathered and the places where those contacts took place could be taken into consideration. The results proved the need for further validation of the model since a simulation run using a very plausible set of parameters showed that 84% of infants, 82% of pupils and 74% of workers were infected with the influenza virus.

Conclusion: Within this thesis the components of a VOMAS were described in detail and a design for a VOMAS was developed. Applied to the influenza model, the design proved to be very flexible and effective and was able to gather useful information which can be used to validate the model. In addition, the developed design can be implemented in most object-oriented modeling environments. Therefore, the VOMAS approach shows great potential for future applications.

Inhaltsverzeichnis

Inhaltsverzeichnis vii

1 Einleitung 1
1.1 Modellbildung und Simulation . . . 1
1.2 Agentenbasierte Modelle . . . 2

2 Validierung 7
2.1 Der Modellierungskreislauf . . . 8
2.2 Validierung von agentenbasierten Modellen . . . 12
2.2.1 Probleme bei der Validierung von agentenbasierten Modellen . . . 12
2.2.2 Validierungsmethoden für agentenbasierte Modelle . . . 14
2.2.3 Spezielle Validierungsprozesse für agentenbasierte Modelle . . . 17

3 Virtual Overlay Multi-agent System 22
3.1 Ursprünge und Grundstruktur von VOMAS . . . 23
3.2 Validierung mit einem VOMAS . . . 26
3.3 Analyse der VOMAS-Struktur und der einzelnen Komponenten . . . 28
3.3.1 Schnittstellen zwischen einem VOMAS und dem Simulationsmodell . 29
3.3.2 Analyse des VOMAS-Design . . . 33
3.4 Konzepte der Parallelität bei einem VOMAS . . . 57
3.5 VOMAS im Modellierungskreislauf . . . 57
3.5.1 Dokumentation eines VOMAS . . . 61

4 Praktische Umsetzung des VOMAS-Designs am Beispiel eines Influenza-Modells 65
4.1 Das Influenzamodell . . . 65
4.1.1 Modellbeschreibung . . . 66
4.1.2 Probleme bei der Validierung . . . 69
4.2 Umsetzung des VOMAS-Designs für das Influenzamodell . . . 71
4.2.1 Struktur des VOMAS . . . 71
4.2.2 Datenbankdesign . . . 80
4.2.3 Notwendige Änderungen am Modell . . . 84
4.2.4 Erkenntnisse und Herausforderungen bei der Implementierung . . . 85
4.3 Ergebnisse . . . 87

5 Zusammenfassung **92**

6 Diskussion **94**

Abbildungsverzeichnis **96**

Tabellenverzeichnis **97**

Literaturverzeichnis **99**

KAPITEL 1

Einleitung

Das zentrale Thema dieser Arbeit ist die **Validierung** von **agentenbasierten Modellen**. Es muss davon ausgegangen werden, dass diese Begriffe nicht jedem Leser geläufig sind. Während der *Validierung* im Laufe dieser Arbeit viel Aufmerksamkeit gewidmet wird, handelt es sich bei den Begriffen *Modell* und *agentenbasiert* um grundlegende Konzepte die vor dem Lesen weiterer Kapitel verstanden werden müssen. Die folgenden zwei Abschnitte bieten daher eine kurze Einführung in die Prinzipien der Modellbildung (Abschnitt 1.1) sowie eine kurze Beschreibung eines speziellen Modellierungsansatzes – dem agentenbasierten Modellieren (Abschnitt 1.2).

1.1 Modellbildung und Simulation

Lösungen zu einem postulierten Problem können einerseits über theoretische Überlegungen, andererseits durch Experimente gefunden werden. Die Simulation stellt einen Zwischenschritt zwischen diesen beiden Wegen dar. Abbildung 1.1 stellt die Lösungsfindung durch die Zuhilfenahme eines Modells dar.

Ausgehend von der Wirklichkeit – einem realen System – wird ein Modell erstellt. Innerhalb des Rahmens dieses Modells können nun über rein theoretische Überlegungen oder durch Computermodelle Lösungen zu Problemen innerhalb des Modells gefunden werden. Diese Lösungen können dann für das reale System adaptiert und interpretiert werden.

Ein Modell ist eine abstrahierte Repräsentantion der Wirklichkeit – sozusagen ein vereinfachtes Abbild der Realität. Der Begriff des Models kann sehr unterschiedlich verstanden werden. In der Architektur bezeichnet ein Modell eines Hauses beispielsweise eine Miniaturnachbildung dieses Gebäudes aus Pappkarton oder anderem Material. Im Sinne der weiteren Untersuchungen dieser Arbeit bezeichnet der Begriff Modell, ein Simulationsmodell.

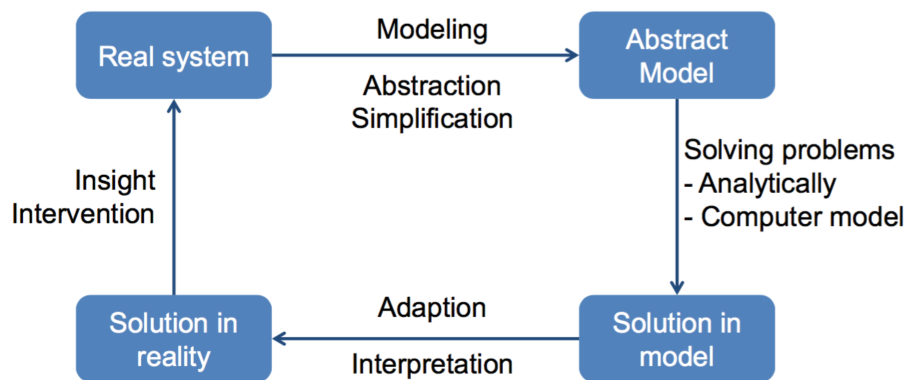

Abbildung 1.1: Problemlösung mit Hilfe von Modellen

Für den Begriff der *Simulation* findet sich folgende Definition von Shannon aus [Sha98]: „*Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system and its underlying causes or of evaluating various designs of an artificial system or strategies for the operation of the system.*“ Den Begriff *System* definiert Shannon weiters folgendermaßen: „*By a system we mean a group or collection of interrelated elements that cooperate to accomplish some stated objective.*“

Die in der oben angeführten Definition genannten Modelle können von unterschiedlichster Art sein. Man unterscheidet hier zwischen dem jeweiligen Modellierungsansatz aus dem die Modelle entstanden sind. Die folgende Liste beinhaltet einige der gängigsten Modellierungstechniken:

- Differentialgleichungen
- System Dynamics
- Zelluläre Automaten
- Markov-Modelle
- Discrete Event Simulation
- Agentenbasierte Modelle

1.2 Agentenbasierte Modelle

Die agentenbasierte Modellierung ist ein in den letzten Jahren immer beliebter werdender Modellierungsansatz, der in verschiedensten Bereichen Anwendung findet. Hierbei werden

Individuen (auch *Agenten* genannt – aus dem Lateinischen *agere* = *handeln* bzw. *agieren*) gewisse Eigenschaften zugesprochen und weiters Regeln definiert, wie sich diese Eigenschaften auf das Verhalten der Agenten auswirken. Das Verhalten des Systems, in dem sich die Agenten befinden, ergibt sich dann aus dem Zusammenwirken dieser Agenten. Dadurch können aus wenigen simplen Regeln komplexe Dynamiken resultieren, die weder vorhersagbar noch beabsichtigt waren. Als bekanntestes Beispiel für diesen Effekt der *Emergenz* gilt die Modellierung von Schwarmverhalten bei Fischen, Vögeln oder Insekten. 1987 erstellte Craig Reynolds in [Rey87] ein einfaches Modell, in dem jeder Agent drei simplen Regeln folgt (siehe Abbildung 1.2):

- Jeder Agent bewegt sich zum Mittel der Positionen seiner Nachbaragenten.
- Jeder Agent vermeidet Zusammenstöße mit Nachbaragenten (z.B. über einen Mindestabstand).
- Jeder Agent bewegt sich in Richtung Mittel der Flugrichtungen seiner Nachbaragenten.

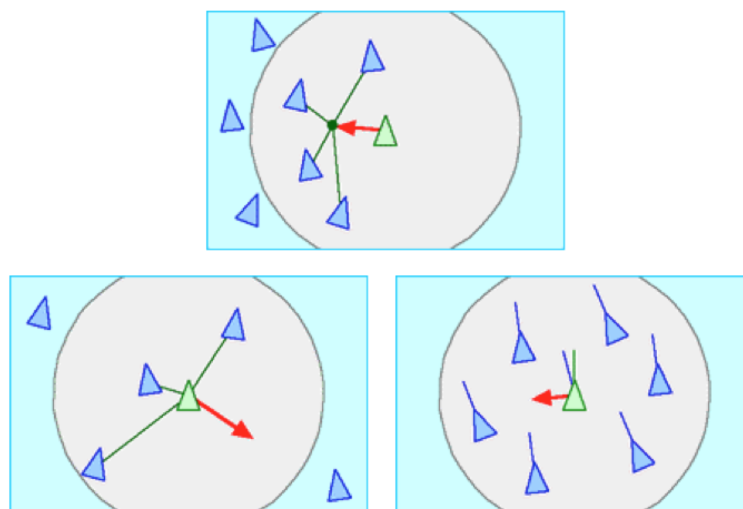

Abbildung 1.2: Einfache Regeln für ein Schwarmverhalten aus [Rey87]

Aus diesen drei Regeln und dem Zusammenwirken der Agenten können bereits realistische Simulationen von Schwarmverhalten entstehen.

Im Gegensatz zu Optimierungsprozessen, welche die Findung eines optimalen Ablaufes als Ziel haben, ist ein agentenbasiertes Modell meist eher beschreibender Natur. Das heißt, dass aus einem agentenbasierten Ansatz, mittels möglichst plausiblem Verhalten der Agenten, ein realitätsnahes Verhalten des Systems resultieren soll.

Agentenbasierte Modelle kommen bereits in vielen verschiedenen Bereichen zum Einsatz, wobei die Gründe für die steigende Popularität vielfältig sind: Komplexe Systeme sind schwer als Ganzes zu erfassen und Zusammenhänge können nur selten von außen erkannt werden. Zu Grunde liegende Daten und Informationen werden immer detaillierter und besser geordnet, was einer *bottom up* Modellierung entgegenkommt. Der wichtigste Fortschritt ist allerdings die enorm wachsende Rechenleistung, welche komplexe agentenbasierte Modelle erst ermöglicht.

Der Agent steht im Mittelpunkt der agentenbasierten Modellierung, aber was einen Agenten ausmacht, also welche Eigenschaften ihn definieren, ist bislang noch nicht einheitlich erarbeitet. Eine vielfach verwendete Definition eines Agenten geht aber auf die Winter Simulation Conference (2005 & 2006) zurück und ist aus [MN06] entnommen:

Ein Agent ist

- **autonom/selbsthandelnd**, interagiert also selbständig mit seiner Umgebung und anderen Agenten. Zur Verfügung stehende Aktionen werden durch verschiedenste, dem Agenten mitgeteilte, Informationen ausgelöst.
- **abgeschlossen/identifizierbar**, hat also klar erkennbare Grenzen. Es muss eindeutig ersichtlich sein, ob etwas Teil des Agenten ist oder nicht. In diesem Sinne muss es auch möglich sein, mehrere Agenten voneinander unterscheiden zu können.
- gänzlich definiert über seine **Zustände**, welche mit der Zeit variieren können, aber nicht müssen. Je größer die Menge der möglichen Zustände eines Agenten ist, desto vielfältiger sind die daraus resultierenden Handlungen.
- **sozial**. Er kann bzw. muss mit anderen Agenten kommunizieren. Informationsaustausch jeglicher Art zwischen Agenten beeinflusst deren Zustände und somit auch resultierende Handlungen.

Weitere Eigenschaften sind oftmals anwendungsessentiell:

Ein Agent kann

- **lernfähig/adaptierend** sein. Er kann sein Verhalten aufgrund von vorhergegangenen Informationen und Handlungen ändern.
- **zielorientiert** agieren. Dies erlaubt es einem Agenten, seine Handlungen in Beziehung zum Erreichen seiner Ziele zu stellen und somit womöglich sein Verhalten in Zukunft anzupassen.
- **heterogen** in seinen Zuständen zu anderen Agenten sein. Es ist nicht notwendig, jedem Agenten dieselben Eigenschaften zuzuschreiben. Oftmals variieren Agenten innerhalb eines Modells stark voneinander.

Ein Agent besteht aus Attributen und Regeln. Attribute können statisch (Name, grundlegende Eigenschaft, ...) oder dynamisch (Gedächtnis, Ressourcen, Ziel, ...) sein. Die Regeln geben vor, wann und unter welchen Umständen bestimmte Aktionen ausgeführt werden oder wie gewisse Attribute aktualisiert werden sollen (siehe Abbildung 1.3).

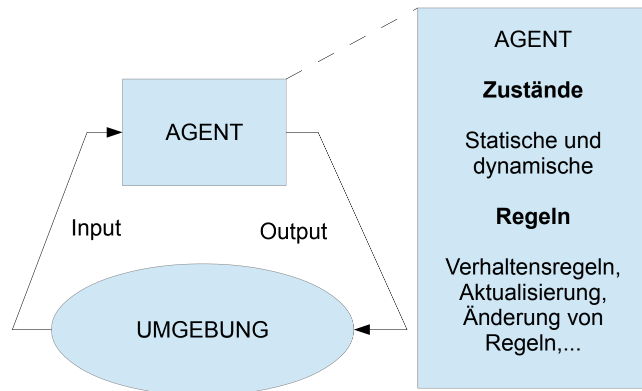

Abbildung 1.3: Agenten und ihre Umwelt

Agenten interagieren nicht nur miteinander, sondern auch mit der Umgebung in der sie „leben". Diese Umgebung – manchmal auch Topologie genannt – bestimmt außerdem in welcher Art und Weise die Agenten in Verbindung stehen. In Abbildung 1.4 sind ein paar Beispiele möglicher Topologien abgebildet.

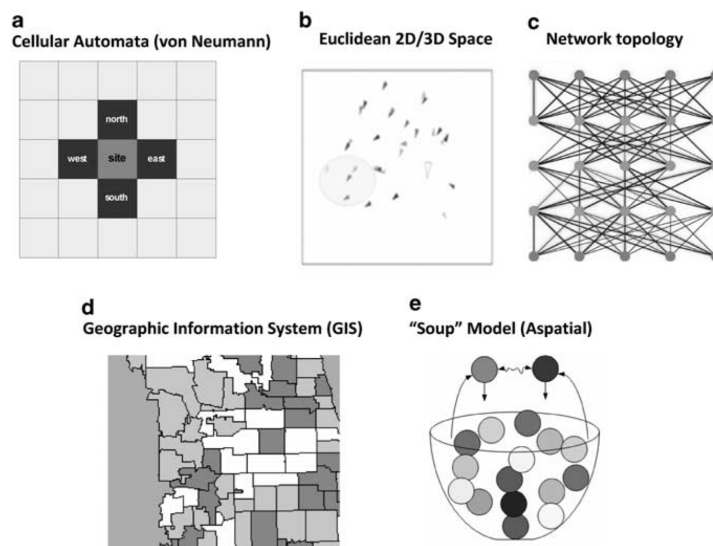

Abbildung 1.4: Topologien für die Interaktion zwischen Agenten aus [MN10]

Relativ simpel, aber sehr unflexibel ist eine Zellstruktur (a). Hier ist je nach Wunsch eine Zelle mit einer bestimmten Anzahl und Anordnung seiner Nachbarzellen verbunden. In dieser Topologie können Agenten von Zelle zu Zelle wandern. 2 bzw. 3-dimensionale euklidische Räume (b) und Netzwerke (c), sowohl statische als auch dynamische, eignen sich in bestimmten Fällen. Manchmal ist geografische „Nähe“ auch gänzlich unwichtig. Im Falle des *soup* Modells (e), werden zufällig zwei Agenten ausgewählt um miteinander zu kommunizieren und danach wieder in den Topf zurückgeworfen.

KAPITEL 2

Validierung

Modelle sind immer eine Abstraktion der Wirklichkeit und können deshalb auch nie „formal korrekt" sein. Verschiedene Abstraktionen führen zu unterschiedlichen und mitunter zahlreichen Modellen. Da die Wirklichkeit nie vollkommen in einem Modell abgebildet werden kann, muss das Ziel der Modellbildung und Simulation jenes sein, ein Modell zu finden oder zu konstruieren, das die zugrundeliegende Problematik hinreichend gut löst. Der Prozess der Validierung ist hierbei ein wichtiges Hilfsmittel.

Die Validierung beschäftigt sich mit der Beantwortung der Frage:

„Wird das *richtige* Modell entwickelt?"

Ein anderer wichtiger Begriff, der im Zusammenhang mit der Validierung erwähnt werden muss, ist jener der Verifikation:

„Wird das Modell *richtig* entwickelt?"

Die Validierung soll also dabei helfen ein Modell zu erstellen, dass zugrundeliegende Fragestellungen gut lösen kann. Im Zuge dessen beschäftigt sich die Validierung somit auch mit der Findung von Methoden, die es ermöglichen sollen, zu entscheiden ob das Modell die Fragestellungen schlussendlich auch lösen kann. Ist ein Modell valide genug, können gefundene Erkenntnisse oder Lösungen auf das reale System übertragen werden. Die Validierung eines Modells kann allerdings immer nur im Rahmen der Modellgrenzen passieren. Die Abstraktion, also die nicht detailgetreue Abbildung der Wirklichkeit, muss trotz erfolgreicher Validierung immer im Hinterkopf behalten werden.

Validierung findet nicht zu einem bestimmten Zeitpunkt in der Modellentwicklung statt. Vielmehr muss sie eine omnipräsente Rolle einnehmen. Validierung sollte so oft und so regelmäßig wie möglich durchgeführt werden. Nur so ist garantiert, dass das Modell stetigen Kontrollen unterzogen wird. Die Behebung von Fehlern, die sehr früh in der Modellentwicklung entstanden sind, aber erst in fortgeschritteneren Stadien zu Tage kommen, ist mit viel

Aufwand verbunden und wirft den Modellprozess unter Umständen weit zurück. Ein sorgfältiger Validierungsprozess kann dabei helfen, solche Fehler möglichst früh zu erkennen und deren Fortpflanzung zu vermeiden. Validierung sollte nicht als Hindernis in der Modellentwicklung gesehen werden, sondern als *sicheres* Vorantreiben des Modellierungsprozesses.

Die Strukturierung dieses Kapitel ist wie folgt: In Abschnitt 2.1 wird eine Darstellung des Modellierungskreislaufs präsentiert, die sich auf [Bal94] und [Sar10] beruft. Zum besseren Verständnis hilft eine zusammenfassende Auflistung der einzelnen Phasen. Um den Übergang zwischen zwei Phasen kontrolliert geschehen zu lassen, stehen Validierungsprozesse zur Verfügung welche ebenfalls beschrieben werden. In Abschnitt 2.2.1 wird erarbeitet, vor welchen Herausforderungen die Validierung von agentenbasierten Modellen steht. Konkrete Methoden, die auf agentenbasierte Modelle angewandt werden können, werden in Abschnitt 2.2.2 und Abschnitt 2.2.3 präsentiert.

2.1 Der Modellierungskreislauf

Um Validierung im Kontext der Modellbildung und Simulation – und im finalen Aspekt bezüglich agentenbasierter Modelle – zu verstehen, muss zunächst der allgemeine Prozess des „Modellentwerfens“ verstanden werden. Ein Modell folgt in seiner Entstehung im Regelfall keiner geradlinigen Struktur. Annahmen müssen getroffen werden, welche sich als falsch oder ungenau herausstellen können und die Modellentwicklung „zurückwerfen“. Ein Modell durchläuft in seiner Entstehung gewisse Stadien, angefangen bei der Fomulierung der Modellfrage, über die Implementierung bis hin zur Ausführung von Experimenten – man spricht von einem Modellierungskreislauf. In der Literatur finden sich zahlreiche Arbeiten zu diesem Thema. Der folgende Überblick basiert auf [Bal94] und [Sar10].

Der Modellierungskreislauf kann grob in 4 Teilbereiche – **Problem Entity**, **Conceptual Model**, **Computerized Model** und **Decision Support** – unterteilt werden, welche im Folgenden näher beschrieben werden. Diese Teilbereiche können wiederum in weitere Stadien unterteilt werden. Die Beschreibung des Prozesses als *Kreislauf* rührt daher, dass nach dem **Computerized Model** immer wieder zur **Problem Entity Phase** zurückgegangen wird. Dies wird solange wiederholt, bis das Modell in allen Belangen zufriedenstellend ist. Erst dann wird es zum **Decision Support** weitergeleitet. Dies ermöglicht einerseits eine „sichere“ iterative Modellbildung – beginnend bei einem Minimalmodell, das stetig erweitert wird – und unterwirft das Modell andererseits regelmäßigen Qualitätskontrollen. In [PP15] werden der Modellierungskreislauf und die einzelnen Stadien anhand eines Beispiels aus der Archäologie – Truppenbewegungen des römischen Heeres zwischen zwei Städten – anschaulich vorgestellt und liefern einen guten Einblick in die praktische Umsetzung dieses Grundprinzips der Modellbildung.

Problem Entity Phase: Ein bisweilen vages oder unstrukturiertes Problem wird an einen Wissenschaftler kommuniziert (**Communicated Problem**). Dieses Problem wird nun so formuliert, dass es einen wissenschaftlichen Zugang erlaubt, zum Beispiel über

die Ableitung von Forschungsfragen (**Formulated Problem**). Verschiedene Lösungsmethoden werden in Betracht gezogen und jene ausgewählt, die für das Problem am besten erscheint. Hier sollte allerdings die Präferenz des Modellierers keine Rolle spielen (**Proposed Solution Technique**). Abschließend wird das zugrundeliegende System auf besondere Eigenschaften und Merkmale hin untersucht. Es werden Abhängigkeiten im System ausfindig gemacht und festgehalten, welche Informationen das System benötigt, um zu funktionieren (**System Knowledge and Objectives**).

Conceptual Model Phase: Ein Modellierungsexperte entwirft in Gedanken ein grobes Konzept des Modells (**Conceptual Model**). Anschließend wird das Modell in schriftlicher Form festgehalten, sodass es an andere Personen kommuniziert werden kann. Die Möglichkeiten hierfür reichen von mathematischen Formeln, über Flow Charts bis hin zu Pseudocodes (**Communicative Model**).

Computerized Model: Das **Communicative Model** wird in ein ausführbares Programm übersetzt (**Programmed Model**). Damit können nun Experimente – welche davor erdacht und konzipiert wurden (**Experimental Model**) – durchgeführt werden, deren Ergebnisse (**Simulation Results**) dabei helfen sollen, die Forschungsfragen zu beantworten.

Integrated Decision Support: Die Ergebnisse werden analysiert und interpretiert bevor sie Entscheidungensträgern vorgelegt werden.

Im Laufe des Modellierungsprozess können Fehler oder falsche Annahmen offensichtlich werden, die eine Rückstufung des Modells in ein früheres Stadium zur Folge haben. Dies ist besonders dann ärgerlich, wenn sich ein Fehler relativ früh unbemerkt eingeschlichen hat und erst in weit fortgeschrittenen Modellierungsstadien bemerkbar macht. Um dies zu vermeiden gibt es Konzepte, mit Hilfe derer überprüft werden kann, ob der Wechsel von einer Phase in die nächste „ohne Probleme“ abgelaufen ist. Sollten im Zuge dieses Prozesses Fehler aufgedeckt werden, muss das Modell die jeweilige Phase erneut durchlaufen und nochmals validiert werden. Die Zusammenhänge zwischen den Validierungsprozessen und dem Modellierungskreislauf sind in Abbildung 2.1 grafisch dargestellt.

Conceptual Model Validation: Alle Theorien und Annahmen, die vom realen System auf das Modell übertragen wurden, werden nochmals auf ihre Richtigkeit überprüft. Hierfür stehen eine Reihe von mathematischen Methoden (Linearität, Unabhängigkeit, ...) und statistische Methoden (Verteilung von Daten, Maxima, ...) zur Verfügung.

Computerized Model Verification: In diesem Schritt wird überprüft, ob die Umsetzung des **Communicative Model** in das **Programmed Model** fehlerfrei durchgeführt wurde. Je nach Programmiersprache oder Simulationsumgebung stehen hierfür verschiedene Werkzeuge aus dem Bereich der Informatik und Softwareentwicklung zur Verfügung.

Operational Validation: In dieser Phase findet der Großteil der Validierungsarbeit statt. Es wird überprüft, ob der Modelloutput im Hinblick auf die Simulationsfragen präzise genug erscheint. Prinzipiell laufen alle Unternehmungen in dieser Phase darauf hinaus, dass Ergebnisse aus dem Modell mit Informationen aus dem realen System verglichen werden. Dies kann subjektiv (Grafiken) oder objektiv (statistische Tests) erfolgen. Voraussetzung hierfür ist allerdings, dass sich das System ausreichend überwachen lässt, also dass die notwendigen Daten zum Vergleich zur Verfügung stehen. Dies ist gerade bei agentenbasierten Modellen nicht immer der Fall (siehe Abschnitt 2.2.1). Ist ein Vergleich mit dem realen System nicht möglich, kann das Modell auch mit anderen – bereits validierten – Modellen verglichen werden (siehe Abschnitt 2.2.2 – **Comparison to Other Models**).

Data Validation: Alle Daten, die für die Entwicklung des Modells verwendet wurden und für die Ausführung von Experimenten benötigt werden, werden auf Korrektheit, Genauigkeit und Anwendbarkeit überprüft. Da aus diesen oftmals Annahmen an das reale System abgeleitet werden, welche dann die Basis für das **Conceptual Model** bilden, kann ein unaufmerksamer Umgang mit Daten weitreichende Konsequenzen haben.

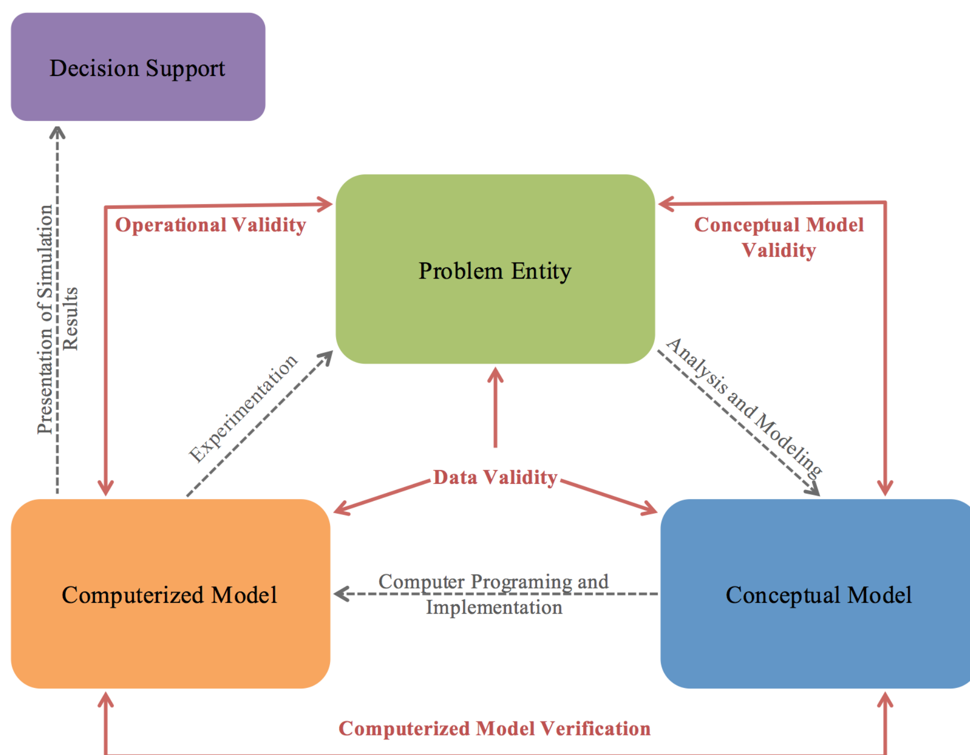

Abbildung 2.1: Darstellung des Modellierungskreislauf nach [PP15]

Es sei angemerkt, dass der Ursprung eines Fehlers weiter zurückreichen kann als angenommen. So kann ein Fehler der im Zuge der **Operational Validation** Phase gefunden wurde, durch Implementierungsfehler, oder aber beispielsweise auch im **Conceptual Model** zustande gekommen sein. Für weitere Methoden, die auf die einzelnen Unterpunkte der Phasen im Modellierungskreislauf angewandt werden können, sei auf [Bal94] verwiesen.

Generell wird empfohlen, bereits in der **Formulated Problem** Phase festzulegen, welche Validierungsmethoden mit welcher Genauigkeit angewendet werden sollen. Eine etwas andere Herangehensweise schlagen Senge und Forrester in [SF80] vor. Im Zentrum steht ein Fragenkatalog, dessen Beantwortung dabei helfen soll, fehlerhaftes Verhalten und fehlerhafte Annahmen zu finden, um das Vertrauen in die **Modellstruktur** und das **Modellverhalten** zu stärken. Einige Beispiele aus dem Fragenkatalog sind hier aufgelistet:

Modellstruktur

- Strukturelle Übereinstimmung: Ist die Modellstruktur konsistent mit dem relevanten Teil des zugrundeliegenden Systems?
- Isolierung: Ensteht das zu untersuchende Problem ausschließlich aus Aspekten die im Modell enthalten sind? Ist also sichergestellt, dass keine relevanten Informationen ausgelassen wurden?

Modellverhalten

- Anormales Verhalten: Werden gewisse Annahmen des Modells „ignoriert“, resultiert das in ungewöhnlichem Verhalten?
- Übertragbarkeit: Kann das Modell das Verhalten von „ähnlichen“ Systemen beschreiben?
- Extreme Belastung: Verhält sich das Modell unter „extremen“ Inputdaten entsprechend vernünftig?
- Statistische Übereinstimmung: Hat der Modelloutput ähnliche statistische Eigenschaften wie der Output des realen Systems?

Andere Aspekte des Modells, wie zum Beispiel die **Implementierung** oder die **Anpassungsfähigkeit**, können durch Beantwortung weiterer Fragen auf Fehler untersucht werden.

Es gibt in der Literatur zahlreiche Beschreibungen verschiedener Validierungsmethoden- und Techniken (siehe z.B.: [Bal94], [Sar10], [Tro04] oder [Klü08]). Einige davon – jene die im Bereich der agentenbasierten Modelle Einsatz finden – werden im Laufe dieser Arbeit genauer präsentiert. So vielfältig die unterschiedlichen Herangehensweisen zur Validierung von Modellen zwar sein können, haben sie doch alle einen gemeinsamen Konsens: Validierung sollte immer sorgfältig und stetig über den Modellierungskreislauf hinweg durchgeführt werden.

Abschließend sei angemerkt, dass Validierung wiederum unter den Mantel der „Reproduzierbarkeit“ fällt. Popper und Pichler geben in [PP15] einen guten Überblick darüber, welche Eigenschaften ein Modell besitzen muss, damit sichergestellt ist, dass es von Dritten reproduziert werden kann. Dies ist die beste Absicherung auf Richtigkeit eines Modells, da es von möglichst vielen verschiedenen Personen eingesehen und auf unterschiedlichste Weise überprüft bzw. nachgestellt und selbst wieder validiert werden kann.

2.2 Validierung von agentenbasierten Modellen

Wie bereits im Zuge des Modellierungskreislauf angeführt wurde, kann man den Begriff der Validität weiter verfeinern (**Conceptual Validity**, usw.). Klügl, die besonders im Feld der Validierung von agentenbasierten Modelle tätig ist, schlägt eine Unterteilung in zwei generelle Kategorien vor.
Die erste unterscheidet die verschiedenen Methoden der Validierung:

- **Face Validity:** Das Modell hat *face validity*, falls es einer *face validation* (in der Literatur auch manchmal als *plausibility checking* bezeichnet) unterzogen wurde. Unter *face validation* fallen alle jene Methoden, die auf menschlicher Intelligenz und Intuition basieren. Das Modellverhalten und die Ergebnisse sollen also plausibel erscheinen.

- **Empirical Validity:** Daten, die ein Modell liefert, werden verschiedenen statistischen Methoden unterzogen und mit Daten aus dem Referenzsystem verglichen. Bei dem Referenzsystem kann es sich um die Wirklichkeit handeln, aber auch um ein anderes Modell, das bereits ausreichend valide ist (siehe **Comparison to Other Models** in Abschnitt 2.2.2)

Die zweite Kategorie beschäftigt sich damit, welcher Aspekt des Modells untersucht wird:

- **Behavioral Validity:** Hierfür wird der Einfluss der Modellparameter und der für das Modell relevante Daten auf das Modellverhalten und die resultierenden Ergebnisse analysiert. Beispielsweise kann untersucht werden, ob das Modell in der Lage ist, bekannte Ergebnisse zu reproduzieren oder Systemzustände vorherzusagen.

- **Structural Validity:** Es wird untersucht, wie die einzelnen Modellteile miteinander in Verbindung stehen. Ebenso wird die Abhängigkeit der Variablen voneinander geprüft bzw. – im Hinblick auf agentenbasierte Modelle – werden die Entscheidungsprozesse der Agenten analysiert.

2.2.1 Probleme bei der Validierung von agentenbasierten Modellen

Agentenbasierte Modelle haben oft ein sehr komplexes Verhalten, das unter Umständen, nicht zur Gänze untersucht bzw. beobachtet werden kann. Validierungsmethoden, die auf mathematischen Formalismen fundieren (siehe Abschnitt 2.2.2 – **formale Methoden**), kommen daher selten oder nur mit Restriktionen in Frage. Selbst wenn es möglich ist, einzelne

Teile eines Modells mit formalen Methoden zu validieren, impliziert dies nicht, dass das Modell als Ganzes valide ist (*„Das Ganze ist mehr als die Summe seiner Teile“*[1]).

Klügl beschreibt in [Klü08] einige grundlegende Probleme, die bei der Validierung von agentenbasierten Modellen auftreten können. Diese Probleme können sowohl in der Natur der Validierungsmethoden liegen, als auch im Konzept des agentenbasierten Ansatz an sich.

Brauchbare Vergleichswerte: Statistische und empirische Methoden sind sehr hilfreich bei der Validierung von Modellen. Allerdings müssen hierfür Daten gesammelt werden, die das Referenzsystem qualitativ ausreichend gut beschreiben können. Informationen zum Gesamtverhalten der Agenten lassen sich in der Regel leicht erheben (z.B. die Anzahl an erkrankten Agenten, oder die Menge an Geld die über einen bestimmten Zeitraum ausgegeben wird). Auf dem Mikro-Level erweist sich dies allerdings oft als sehr schwierig. So kann das Verhalten von Agenten oft schwer in einer Form erfasst werden, die statistische Untersuchungen ermöglicht (Wie entscheidet ein Kranker, ob er zur Arbeit geht oder zu Hause bleibt? Wie plant der Besucher eines Einkaufszentrum seine Einkaufswege?). Ist dies in vereinzelten Fällen dennoch möglich, bleibt das Problem, dass eine einzelne Person schwer als statistischer Repräsentant herangezogen werden kann.

Verfügbarkeit von Daten: Abgesehen davon, dass nicht nur das Gesamtverhalten des Systems validiert werden muss, sondern auch die einzelnen Teilmodelle bis hin zum individuellen Verhalten der Agenten, ist eine statistische Validierung oftmals aufgrund fehlender Daten nicht möglich (selbst wenn sie theoretisch fassbar wären). Sei es aufgrund der Tatsache, dass die Daten an sich schon zu komplex sind, oder dass das Sammeln der Daten schlicht und einfach nicht möglich, (Datenschutz) bzw. zu kostenintensiv ist (Einzelbefragungen, Umfragen, Testszenarien).

Zu viele Parameter: Agentenbasierte Modelle sind in der Lage, sehr komplexes Verhalten zu simulieren. Das ist vor allem deshalb möglich, weil Agenten – theoretisch – beliebig komplex und vielfältig agieren können. Je komplexer das System, desto mehr Parameter müssen definiert werden um es akkurat beschreiben zu können. Eine *Überparametrisierung* kann aber unter Umständen dazu führen, dass im Zuge einer Kalibrierung und Optimierung beliebige Ergebnisse reproduziert werden können. Falsifizierung – das Widerlegen von Hypothesen – ist in solch einem Fall nicht möglich. Dasselbe Phänomen kann eintreten, wenn nicht ausreichend aussagekräftige Daten vorhanden sind, wie im EOS Projekt [DP95] gezeigt wurde.

Starke Abhängigkeit vom einzelnen Verhalten: Agentenbasierte Modelle sind in der Regel stark nichtlinear und kleine Änderungen der Parameter oder des Einzelverhaltens von Agenten können das Gesamtverhalten des Modellsystems stark beeinflussen. Diese geringfügigen Änderungen müssen aber nicht immer Parameteränderungen sein. Beispielsweise kann eine zufällig gewählte Updatereihenfolge der Agenten zu unerwartetem, oder gar chaotischem Verhalten des Modellsystems führen.

[1] Aristoteles.

All diese Problemzonen geben Anlass zur Frage, warum nicht besser mit anderen Modellierungsmethoden gearbeitet werden sollte, deren Validierung einfacher bzw. erfahrener ist? Die Antwort ist simpel: andere Methoden sind schlicht und einfach nicht in der Lage, Phänomene aus Bereichen, in denen agentenbasierte Modelle zum Einsatz kommen, auch nur ansatzweise zufriedenstellend zu beschreiben. Es müssen also neue Validierungsmethoden gefunden und bereits bekannte ausreichend getestet werden, um den agentenbasierten Modellierungsansatz attraktiver zu machen.

2.2.2 Validierungsmethoden für agentenbasierte Modelle

Es gibt zahlreiche allgemeine Validierungsmethoden, die mehr oder weniger effektiv auf agentenbasierte Modelle angewandt werden können. Balci unterteilt in [Bal94] Validierungs- und Verifizierungsmethoden in 6 Gruppen und kategorisiert und beschreibt über 40 verschiedene Techniken. Die folgende Auflistung zeigt jene Methoden, die für agentenbasierte Modelle angemessen erscheinen.

Informelle Techniken: Dazu zählen Methoden, die auf dem subjektiven Beurteilen des Modells aufbauen. Mathematische Formalismen werden hier nicht angewandt, was allerdings nicht heißt, dass diese Methoden weniger durchdacht und strukturiert sind. Informelle Methoden lassen sich prinzipiell bei jeder Art von Modellierung anwenden – also auch bei einem agentenbasierten Ansatz:

- **Audit:** Eine einzelne Person untersucht die Simulationsstudie auf korrekt angewandte Praktiken, Standards und Richtlinien. Audits werden während des gesamten Modellierungskreislaufes periodisch durchgeführt.
- **Inspections, Reviews und Walkthroughs:** Bei jeder dieser Methoden wird eine Gruppe von Personen damit beauftragt, mögliche Fehler im Modell zu finden und zu dokumentieren. Die Methoden unterscheiden sich vorwiegend durch die Zusammensetzung der Teams und der Aufgaben der einzelnen Teammitglieder.
- **Face Validation:** Personen, die mit dem betrachteten System vertraut sind, sowie Projektmitarbeiter und potentielle Benutzer des Modells versuchen mit Hilfe von gesundem Menschenverstand und Intuition festzustellen, ob Modellverhalten und Ergebnisse „sinnvoll“ erscheinen. Diese Methode ist im Bereich der agentenbasierten Modellierung sehr vielseitig anwendbar und in gewissen Modellen ein sehr mächtiges Werkzeug für die Validierung (siehe **Immersive Face Validation**).
- **Turing Test:** Daten des Modells und Daten, die aus dem realen System extrahiert wurden, werden einem Experten vorgelegt. Kann dieser nicht zwischen Modelldaten und wirklichen Daten unterscheiden, sagt man, dass das Modell den *Turing Test* bestanden hat.
- **Desk Checking:** Eine Technik, die vor allem am Anfang der Simulationsstudie eingesetzt werden sollte, um Fortpflanzung von methodischen Fehlern möglichst

von Beginn an zu vermeiden. Die Arbeit der einzelnen Beteiligten wird – idealerweise von einer außenstehenden Person – auf Korrektheit, Konsistenz, Vollständigkeit und Unmissverständlichkeit untersucht.

Statische Techniken: Das **Programmed Model** wird Zeile für Zeile theoretisch (in Gedanken) ausgeführt und auf seine Richtigkeit geprüft. Da es sich bei der Implementierung eines agentenbasierten Modells meist um eine umfangreiche Programmierung handelt, sind Techniken wie **Consistency Checking** (Einhaltung von Programmierstandards) und **Data Flow Analysis** (Löschung von nicht benutzten oder nicht deklarierten Variablen) gängige und nützliche Methoden.

Dynamische Techniken: Das Modell wird ausgeführt und das Modellverhalten unter unterschiedlichen Aspekten analysiert und dokumentiert. Hierfür muss unter Umständen zusätzlicher Quellcode an verschiedenen Stellen des programmierten Modells eingefügt werden, um notwendige Informationen extrahieren zu können.

- **Bottom Up Testing:** Jedes Teilmodell – angefangen beim „untersten“ Level – wird nach seiner Fertigstellung auf Korrektheit getestet. Nach dem Zusammenfügen einzelner Teilmodelle zu einem größeren Modul wird ein Integrationstest durchgeführt.
- **Black Box Testing:** Von Interesse ist die Genauigkeit der Transformation von Input in Output. Wie diese Transformation aussieht, ist hier egal (deshalb der Begriff *black box*).
- **Predictive Validation:** Die Qualität von Vorhersagen des Modells wird untersucht. Das Verhalten des zu modellierenden Systems unter bestimmten Einflüssen wird dokumentiert. Das Simulationsmodell wird dann unter denselben Voraussetzungen gestartet und die Ergebnisse werden mit denen der Wirklichkeit verglichen.
- **Statistical Techniques:** Diese Methode ist nur anwendbar, wenn das zu untersuchende System ausreichend genau beobachtbar ist – also genügend aussagekräftige Daten sowohl aus dem System, als auch im weiteren Verlauf aus dem Modell herausgelesen werden können. Wie bereits angemerkt, ist das bei agentenbasierten Modellen oftmals gar nicht möglich oder mit hohem Aufwand und Kosten verbunden.
- **Visualisation:** Sowohl internes Verhalten (Veränderungen innerhalb des Systems) als auch externes Verhalten (die Ergebnisse) des Modells, werden während der Laufzeit grafisch aufbereitet. Die Visualisierung muss immer im Zusammenhang mit einem Test auf Plausibilität der Daten passieren.
- **Sensitivity Analysis:** Alle Parameter werden auf ihren Einfluss auf das Modellverhalten und die Ergebnisse untersucht (qualitativ und quantitativ). Einflussreiche Parameter müssen dann möglichst genau bestimmt werden.

Sargent listet in [Sar10] noch weitere Methoden auf, die in die Kategorie der dynamischen Techniken fallen:

- **Animation:** Eine Animation während der Laufzeit erlaubt Einblicke in bestimmte Prozesse des Modells (siehe **Immersive Face Validation**).
- **Comparison to Other Models** oder **Model Alignment:** Die Ergebnisse des zu validierenden Modells werden mit den Ergebnissen von bereits validierten Modellen verglichen. Das müssen nicht unbedingt andere *agentenbasierte* Modelle sein. Einfache Modelle können zum Beispiel mit analytischen Lösungen (falls diese existieren) verglichen werden. Diese Methode basiert auf der Annahme, dass die Eigenschaft eines Modells, *valide* zu sein, weitergeben werden kann (also eine transitive Eigenschaft ist): Wird ein System S valide genug durch ein Modell M dargestellt und produziert ein anderes Modell N ausreichend genaue Ergebnisse wie Modell M, so ist N ebenso ein valides Modell, um S zu beschreiben.
- **Event Validity:** Das Eintreten von relevanten Ereignissen (events) wird sowohl qualitativ als auch quantitativ mit dem realen System verglichen. In einem Waldbrandmodell wären beispielsweise die Anzahl (quantitativ) der Brandherde und deren Größe (qualitativ) von Interesse.
- **Extreme Condition Tests:** Es wird überprüft, ob sich das zu validierende Modell in jedem auch noch so unwahrscheinlichen Zustand „vernünftig" verhält. Beispielsweise kann ein Produktionsunternehmen mit leerem Lagerbestand keine Güter produzieren.
- **Traces:** Einzelne Instanzen (entities) werden durch das Modell hindurch verfolgt, um die Richtigkeit der internen Logik des Modells zu überprüfen.

Formale Techniken: Mathematische Beweise werden zur Überprüfung des Modells verwendet. Falls Methoden aus dieser Kategorie anwendbar sind, ermöglichen sie als einzige eine konkrete Antwort auch die Frage *„Ist das Modell richtig?"*. Allerdings sind *„formale Beweise auf Richtigkeit nicht anwendbar auf annähernd komplexe Simulationsmodelle."*[2]

Zwangsbedingte Techniken (Überprüfung auf Einhaltung gewisser Annahmen über den gesamten Simulationslauf – **Assertion Checking**) und **Symbolische Techniken** (Ursache-Wirkung Untersuchungen, Durchlaufen logischer Pfade mittels speziell generierten Testdaten und Inputwerten – **Cause-Effect Graphing**) wurden von Balci in [Bal94] noch als seperate Kategorien genannt, in [Bal97] aber in die oben genannten Kategorien eingearbeitet.

[2] [Bal94] p.152.

2.2.3 Spezielle Validierungsprozesse für agentenbasierte Modelle

Einen Schritt in Richtung strukturierter Validierungsprozesse, dezidiert entworfen für den agentenbasierten Modellierungsansatz, machen Klügl in [Klü08], Klügl und Louloudi in [LK12] und Niazi et al. in [NHK09]. Die Erkenntnisse der ersten zwei Arbeiten werden in diesem Kapitel beschrieben. Die Arbeit von Niazit et al. ist Grundlage dieser Diplomarbeit und zentrales Thema in den nachfolgenden Kapiteln.

Klügl schlägt in [Klü08] ein neues Konzept der Valdierung vor, welches in Abbildung 2.2 grafisch dargestellt ist.[3] Vorrausetzung hierfür ist, dass bereits ein ausführbares Modell existiert, welches brauchbaren Output produzieren und das Verhalten des Systems in geeigneter Art und Weise visualisieren kann (z.B. mittels **Animation**). Obwohl der vorgeschlagene Validierungsprozess erst zu diesem Zeitpunkt beginnt, heißt das natürlich nicht, dass davor nicht bereits andere Validierungsversuche unternommen werden können. Vielmehr sollte das unbedingt bereits passiert sein – im Hinblick auf den Modellierungskreislauf aus Abschnitt 2.1 muss also zumindest das **Conceptual Model** validiert sein.

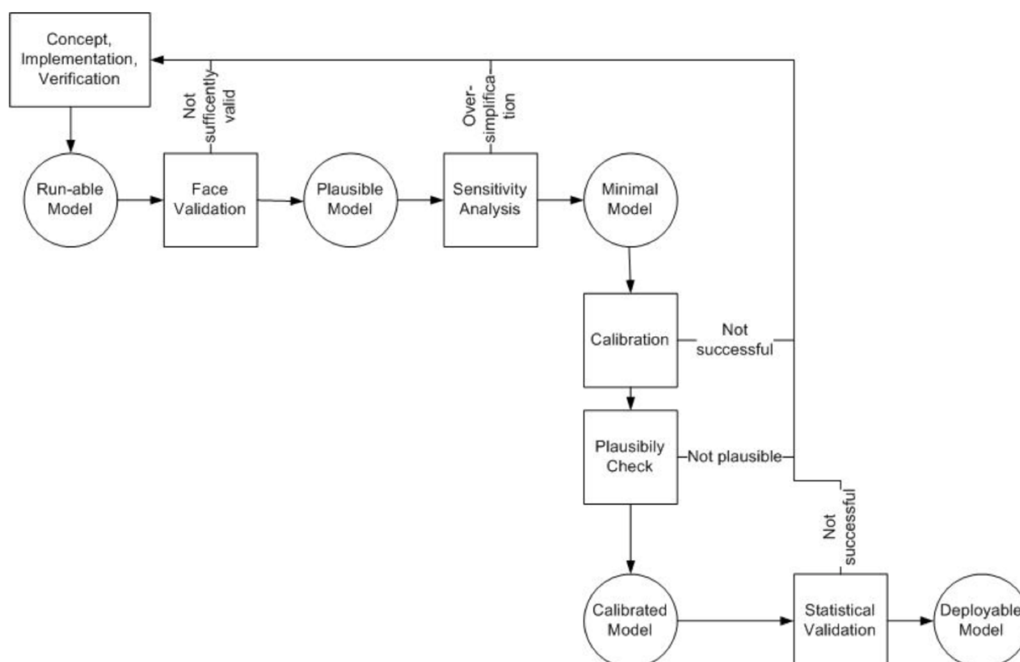

Abbildung 2.2: Validierungsmethode aus [Klü08]

[3]Rund umrahmte Elemente beschreiben den Zustand in dem sich das Modell befindet, rechteckig umrahmte Elemente beschreiben den Prozess, der das Modell von einem Zustand in den nächsten überführt. Ist einer dieser Prozesse nicht erfolgreich, wird der Zustand des Modells in den Ursprungszustand zurückgesetzt, erforderliche Änderungen müssen unternommen werden und der Prozess startet erneut.

Anfangs wird das ausführbare – und bis zu diesem Schritt positiv validierte – Modell einer **Face Validation** unterzogen, welche in drei Abschnitte unterteilt ist: **Animation Assessment**, **Output Assessment** und **Immersive Assessment**. Diese Unterschritte setzen die Anwesenheit eines Experten des zu modellierenden Sachverhalts voraus, der bei der Face Validation hilft:

- **Animation Assessment:** Ein Experte versucht anhand von Animationen, die mit Informationen aus dem Modell erstellt werden, das generelle Verhalten des Modells und der einzelnen Teile zu untersuchen, Fehlverhalten festzustellen und diese zu dokumentieren. Auch eine Animation auf „Agentenlevel" sollte hier möglich sein, um das Verhalten einzelner Agenten nachzuvollziehen.
- **Output Assessment:** Ein Experte untersucht die Modellergebnisse. Hierbei kann beispielsweise untersucht werden, ob die Ergebnisse konsistent mit vorhergegangenen Erkenntnissen sind oder ob Messwerte in vernünftiger Relation zueinander stehen.
- **Immersive Assessment:** Ein Experte betrachtet die Simulationsumgebung durch die Augen eines einzelnen Agenten, um das konkrete Verhalten des Agenten mitzuverfolgen. Mit welcher Methodik die Validierung in diesem Bereich ablaufen sollte, ist Thema einer weiteren Arbeit von Klügl in Kooperation mit Louloudi [LK12], auf welche in diesem Kapitel noch eingegangen wird.

Diese drei Verfahren können unterschiedlich kostenintensiv sein. Es sollte mit jenem Verfahren begonnen werden, das am billigsten ist und möglichst schnell aus einem großen Pool an Hypothesen jene verwerfen kann, die nicht plausibel sind.

Nach einer erfolgreichen Face Validation, gilt das Modell als plausibel. Auf das **Plausible Model** werden nun Sensitivitätsanalysen angewandt. Alle Parameter werden auf ihren Einfluss auf das Modell untersucht. Falls die Änderung eines Parameters nicht in einer Änderung des Modellverhaltens resultiert, kann der Parameter und die von ihm abhängigen Teile des Modells gelöscht werden[4]. Dieser Schritt verhindert das in Abschnitt 2.2.1 angesprochene Problem der *Überparametrisierung*. Die noch übrigen Parameter des nun minimalen Modells (**Minimal Model**) müssen kalibriert[5] und ein abschließender Plausabilitätstest durchgeführt werden. Dieser besteht im Prinzip aus einer weiteren Face Validation, wobei hier nicht allzu genau ins Detail gegangen werden muss. Wurde der Plausibilitätstest bestanden, ist das Resultat ein **Calibrated Model**, welches allerdings noch nicht „einsatzfähig" ist, da es nur auf konkrete Datensätze kalibriert wurde. Eine erfolgreiche statistische Validierung mit anderen Datensätzen (also verschieden von jenen, die bei der Kalibrierung werdet wurden) führt zu einem anwendbaren Modell (**Deployable Model**).

Wie bereits in Abschnitt 2.2.1 angemerkt wurde, ist die Validierung von agentenbasierten Modellen mit Hilfe von statistischen und empirischen Methoden aufgrund von nichtvorhandenen bzw. nicht aggregierbaren Daten selten möglich. Im Hinblick auf die in Abschnitt 2.2

[4] Klügl und Louloudi verweisen auf [STG12].
[5] Klügl und Louloudi verweisen auf [KL00] und [ZKP00].

präsentierte Unterteilung der Validierungsmethoden, muss also mit Methoden der **Face Validation** gearbeitet werden. Klügl und Louloudi entwickeln in [LK12] einen Prozess, der es dem Benutzer ermöglicht, die Simulation durch die Augen eines Agenten zu erleben. Der Benutzer hat dann die Möglichkeit, als Beobachter (passiv) oder als tatsächlicher Agent (aktiv) an der Simulation teilzunehmen, um mögliche Fehler im Verhalten der Agenten aufzudecken und zu dokumentieren – **Immersive Face Validation**. Vorraussetzung für diese Methode ist eine funktionierende Übersetzung der Simulation in eine möglichst detaillierte virtuelle Umgebung. Im besten Fall sollte ein möglichst unabhängiges Modul konzipiert werden, das über vordefinierte Schnittstellen mit dem eigentlichen Modell kommunizieren kann. Damit wäre sichergestellt, dass das Hauptaugenmerk auf der Modellierung der Agenten liegt und nicht zu viel Zeit in aufwändige Visualisierungsarbeit investiert werden muss.

Immersive Face Validation ist inspiriert von einer bereits gängigen Strategie, das Verhalten von Agenten zu programmieren. Im Participatory Ansatz [GH06] beispielsweise übernehmen Experten, Entscheidungsträger oder Laien die Rollen verschiedener Agenten in der Simulation (ermöglicht durch eine *Virtual Reality (VR)* Umgebung) und interagieren mit der Umwelt und anderen Agenten. Diese Interaktionen werden aufgezeichnet und später analysiert, um ein realistischeres Verhalten der Agenten zu extrahieren und implementieren zu können. **Immersive Face Validation** macht sich diese Strategie zu Nutze und liefert eine Validierungsmethode, die sich grob in zwei Schritte unterteilen lässt: **Validation Setup** und die **Immersive Face Validation**[6].

Validation Setup: In der Regel wird eine detaillierte Visualisierung, im Vergleich zu einem „normalen“ Modelldurchlauf, wesentlich mehr Rechenzeit in Anspruch nehmen. Das VR-Modul sollte also beliebig an- und abgeschaltet werden können. Im Falle der Verwendung werden potentielle Benutzer über die grobe Funktionsweise des Modells aufgeklärt und ihre Rolle in der Validierung wird definiert.

Immersive Face Validation: Das Modul soll zwei Arten der Teilname an der Simulation ermöglichen:

- **Beobachter:** Der Benutzer wird als unabhängiger Dritter in die Simulationsumgebung eingeschleust und hat dadurch die Möglichkeit, den Ablauf und die Agenten genauer zu beobachten.
- **Agent:** Der Benutzer schlüpft in die Rolle eines Agenten. Er hat die Möglichkeit, das Verhalten des Agenten zu steuern und kann mit anderen Agenten interagieren.

Gegebenenfalls hat der Benutzer die Möglichkeit, die Simulation anzuhalten, um Kommentare bezüglich Fehlverhalten oder sonstiger Auffälligkeiten zu hinterlassen. Nach Ablauf eines Simulationsdurchgangs werden die Eindrücke des Benutzers besprochen und etwaige Änderungen am Modell vorgeschlagen.

[6] Der präsentierte Ablauf (siehe Abbildung 2.3) beschreibt im Prinzip den Schritt des **Immersive Assessment**.

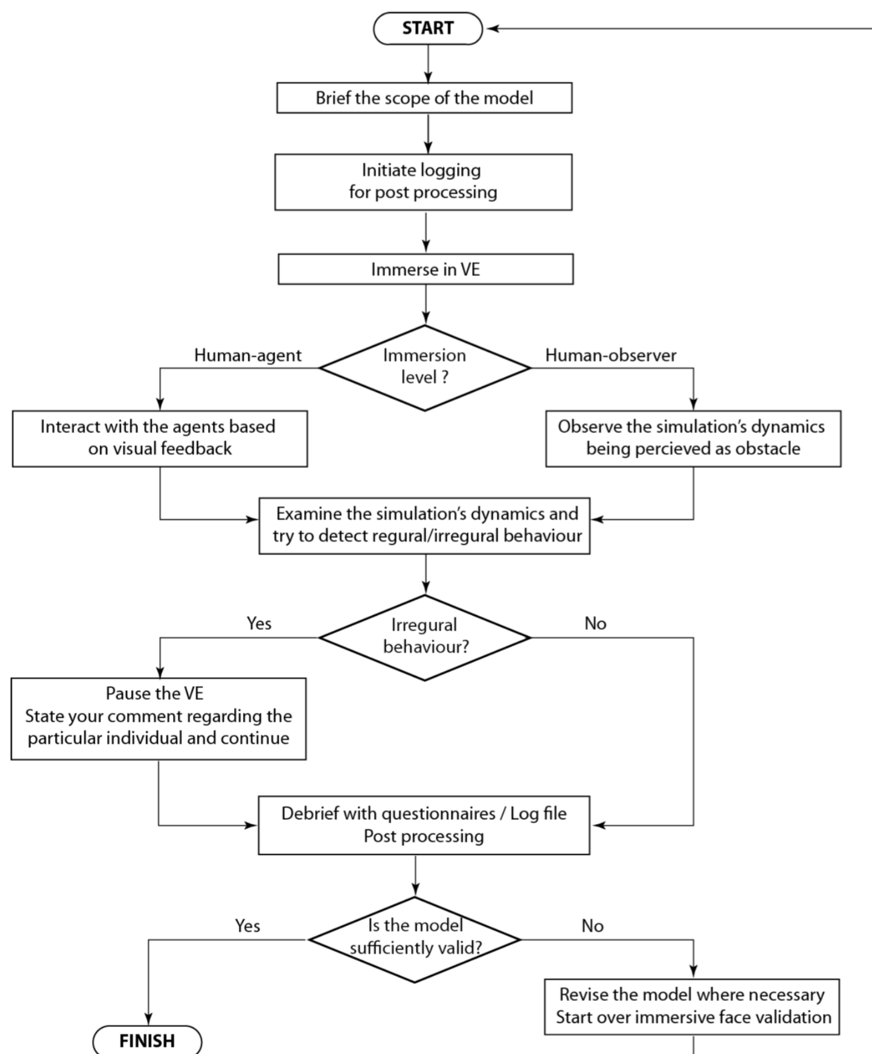

Abbildung 2.3: Entscheidungsdiagramm aus [LK12]

Offensichtlich ist diese Methode nicht bei jedem agentenbasierten Modell anwendbar. Es sei darauf hingewiesen, dass Klügl und Louloudi vor allem Modelle im Bereich der *Pedestrian Simulation* und der *Crowd Simulation* nennen, welche sich für diesen Ansatz gut eignen. Ganz allgemein kann gesagt werden, dass diese Methode für Modelle geeignet ist, in denen

- ein Abstandsbegriff eingeführt werden kann, der außerdem von zentraler Wichtigkeit für die Simulation ist.
- ein visuelles Erfassen der Umgebung das Verhalten der Agenten beeinflusst.

Immersive Face Validation, in gewissen Anwendungsbereichen sicher ein vielversprechendes Hilfsmittel für die Validierung, steht allerdings noch vor einigen essentiellen Problemen. So muss zunächst noch ein VR-Modul entwickelt werden, dass den geforderten Ansprüchen entspricht. Außerdem ist, wie schon erwähnt, das „normale“ Modell im Regelfall schneller als die VR-Umgebung. Soll ein Benutzer als Agent in die Simulation geschleust werden, um mit anderen Agenten zu interagieren, muss diese Information zurück in das Modell geschickt werden. Dieser zeitverzögerte Input muss dementsprechend berücksichtigt werden.

KAPITEL 3

Virtual Overlay Multi-agent System

Niazi et al. präsentieren in [NHK09] eine *„neuartige Technik, [...] die bei jeglicher Art von agentenbasierten Modellen angewandt werden kann“*[1]. Diese neuartige Validierungsmethode verwendet ein Virtual Overlay Multi-agent System – VOMAS – ein Multiagentensystem, das auf einer zweiten Ebene „über“ der eigentlichen Simulation agiert und Informationen sammelt. Bei einem VOMAS handelt es sich grob gesagt um ein System von Beobachtern, Buchhaltern und Aufsehern, welches bei der Validierung des Modells hilft. So können Attribute von Agenten, die von speziellem Interesse sind, auf Änderungen hin beobachtet werden oder Verletzungen von Modellannahmen aufgezeigt und relevante Informationen dazu gespeichert werden.

Den Autoren zufolge hat VOMAS seinen Ursprung vor allem in Konzepten des Software Engineering und baut auf dem Companion Modelling Ansatz auf [B^{+}03]. Modellierungsexperten entwickeln das Modell – und im Zuge dessen auch das VOMAS – in enger Zusammenarbeit mit Experten des zu modellierenden Referenzsystems – den Subject Matter Experts (SME). Der SME gibt vor allem bei der Frage, nach welchen Gesichtspunkten validiert werden sollte, hilfreiche Hinweise. Demzufolge ist ein VOMAS weniger eine Validierungsmethode, sondern sollte als Teil des Modellierungskreislaufs gesehen werden (siehe Abschnitt 3.5), der eine Validierung des Modells erleichtert. Ein VOMAS sollte in diesem Sinne immer vom Beginn des Modellierungsprozesses von Grund auf neu mitentwickelt werden[2].

In Abschnitt 3.1 folgt eine erste grobe Übersicht der VOMAS Struktur, damit der Zusammenhang und die Aufgaben der Komponenten zumindest oberflächlich vorstellbar sind. Abschnitt 3.2 gibt eine Auflistung der möglichen Anwendungsgebiete, bevor in Abschnitt 3.3 eine detaillierte Analyse eines entwickelten VOMAS-Designs präsentiert wird. Abschnitt 3.4 ist ein kurzer Diskurs in Richtung möglicher Parallelität in einem VOMAS. Abschließend wird

[1] [NHK09] p.1.

[2] Diese Entwicklung kann allerdings oftmals nach einem vorgefertigten Schema verlaufen und es können gewisse allgemeine Grundstrukturen einmal entworfen und je nach Bedarf wiederverwendet werden.

in Abschnitt 3.5 die Entwicklung eines VOMAS in den allgemeinen Modellierungskreislauf eingebettet und ein Vorschlag für die Dokumentation eines VOMAS gemacht.

3.1 Ursprünge und Grundstruktur von VOMAS

Der VOMAS-Ansatz hat seine Ursprünge in verschiedenen Bereichen, vor allem aber bedient er sich Konzepten des Software Engineering [Nia11]:

Software Engineering Team: Wird ein Gruppe von Personen mit der Entwicklung einer neuen Software beauftragt, werden den Mitgliedern im Regelfall verschiedene Rollen zugewiesen, wie zum Beispiel *Manager, Softwareexperte, Quality Assurance Engineer* oder *End User*. Die Zusammenarbeit dieser Personen garantiert eine fehlerfreiere Entwicklung des Produktes. Der VOMAS Ansatz verlangt, dass nicht nur Modellierungsexperten (*Simulation Specialist* – SS), sondern auch Experten des zu simulierenden Systems (*Subject Matter Experts* – SME) am Modellierungsprozess beteiligt sind. Der SME hat im Regelfall wenig bis keine Erfahrung in der Modellbildung und Simulation, kann aber bei der Entstehung des Conceptual Models wichtige Informationen liefern und beim Design der Experimente helfen. Vor allem spielt er bei der Validierung des Modells eine wichtige Rolle. Der SME weiß oft besser als der SS, welche Kriterien für die Validierung wichtig sind, nach welchen Merkmalen Ausschau gehalten werden muss und er kann Daten und Modellverhalten verlässlicher auf Plausibilität prüfen. Durch eine enge Zusammenarbeit profitieren beide Seiten von neu erworbenem Wissen. So wird der SME im Laufe der Entwicklung mit Konzepten der Modellbildung vertraut und kann womöglich auch in technischen Belangen nützliches Feedback geben.

Design by Contract: Es wird überprüft, ob vordefinierte Abläufe – Invarianten – auch wirklich eingehalten werden. Für das weitere Verständnis müssen zunächst die Begriffe *pre-* und *post-condition* erläutert werden. Unter einer pre-condition versteht man einen Systemzustand, der angenommen werden muss, bevor eine bestimmte Änderung im Modell auftreten kann. Diese Änderung resultiert wiederum in einem neuen Systemzustand – der post-condition. Niazi definiert eine Invarianz bei agentenbasierten Modellen nun folgendermaßen: *„If a set of pre-conditions C_{pre} holds in an ABM M, then M is guaranteed to undergo a state change which will result in the post-conditions C_{post}."*[3] Invarianzen definieren also Änderungen in der Simulation, die nach einer festen Vorstellung abzulaufen haben – nur unter bestimmten Bedingungen, dann aber mit mit Sicherheit zu erwartenden Folgen. Um zu überprüfen, ob für das Modell definierte Invarianzen eingehalten werden, können VOMAS Agenten die Simulation regelmäßig auf die zugehörigen pre-condidtions untersuchen und den Verlauf der Simulation dahingehend verfolgen, ob die geforderten post-conditions tatsächlich eintreten. Ebenso kann man ein Verhalten definieren, das von keinem Interesse für den SME ist: *„If pre-conditions C_{pre} do not hold in an ABM M, then M is not guaranteed to undergo any*

[3] [Nia11] p.200.

particular state change of interest to the SME.“[4] Unerwünschtes oder falsches Modellverhalten kann auf diese Weise noch während der Simulation aufgedeckt werden.

Die folgende Einführung in die Struktur eines VOMAS ist lediglich ein grober Überblick und soll dabei helfen, detailliertere Untersuchungen, die im weiteren Verlauf der Arbeit präsentiert werden, im Hinblick auf das große Ganze zu verstehen.

Abbildung 3.1 zeigt die zwei Level der Simulation: Die Simulationsebene, auf der die eigentliche Simulation stattfindet und die VOMAS-Ebene. Agenten der VOMAS-Ebene, im Folgenden VO-Agenten genannt, können Agenten der Simulationsebene, im Folgenden Sim-Agenten genannt, beobachten und gegebenenfalls wichtige Informationen zu deren Verhalten oder Fehlverhalten speichern. Information darf allerdings immer nur in eine Richtung fließen – von den Sim-Agenten zu den VO-Agenten. Das VOMAS darf den Verlauf der Simulation nicht beeinflussen.

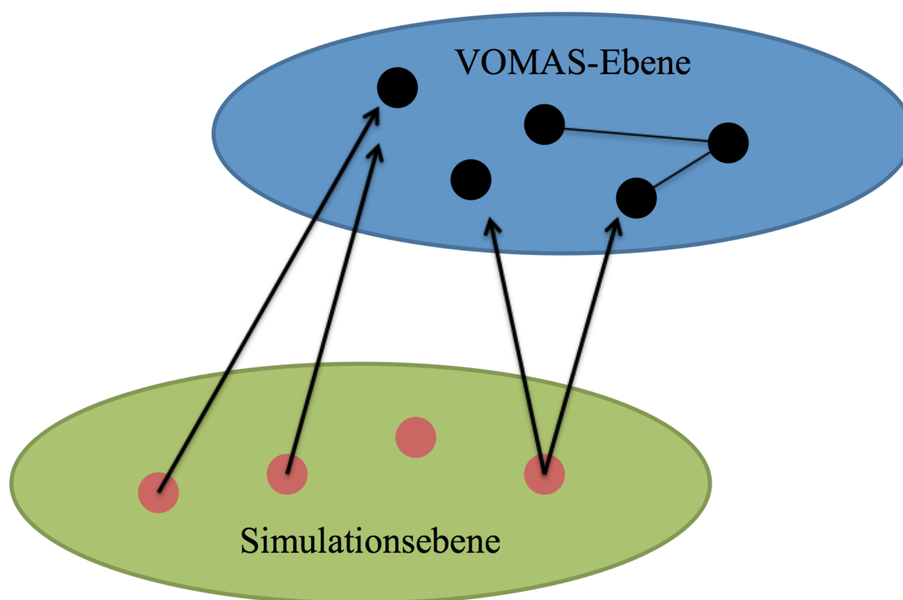

Abbildung 3.1: Darstellung der VOMAS- und der Simulationsebene nach [NHK09]

Das Virtual Overlay Multi-agent System besteht aus verschiedenen Komponenten (Agenten). Die einzelnen Agenten führen hierbei unterschiedliche Aufgaben aus. Einige sammeln wichtige Informationen zum Verhalten der Simulationsagenten, andere stellen sicher, dass keine Modellannahmen verletzt werden. Soll gesammelte Information dauerhaft gespeichert werden, wird sie an einen Agenten weitergeleitet, dessen einzige Aufgabe eben dieses ist. Die einzelnen Aufgaben werden innerhalb einer festen Hierarchie weiterdelegiert, an deren Spitze ein Manageragent steht. Im Folgenden werden die Bezeichnungen aller Agenten

[4] [Nia11] p.201.

aufgelistet und deren Aufgaben kurz beschrieben. Eine detaillierte Beschreibung wird in Abschnitt 3.3 geliefert.

- **VO-Manager:** Der VO-Manager ist die zentrale Schaltstelle. Er regelt die Interaktionen der anderen Agenten im VOMAS und hat Zugriff auf globale Attribute der Simulation.
- **Console-Agent:** Informationen oder Nachrichten, die in Echtzeit ausgegeben werden sollen, werden vom Console-Agent für den Benutzer sichtbar gemacht.
- **VO-Agent:** Der VO-Agent steht als einziger mit den Simulationsagenten in Verbindung. VO-Agenten können eine reine Beobachterrolle einnehmen, aber auch in die eigentliche Simulationsebene eingeschleust werden (siehe Abschnitt 3.3.1). Die VO-Agenten können **Watch-Agenten** und **Constraint-Agenten** besitzen.
- **Watch-Agent:** Soll ein bestimmtes Attribut der Simulationsagenten beobachtet werden, wird ein Watch-Agent mit dieser Aufgabe betraut. Relevante Änderungen des Attributs können außerdem gespeichert werden.
- **Constraint-Agent:** Verletzungen von Modellannahmen werden von Constraint-Agenten aufgezeichnet. Wird eine solche Verletzung entdeckt, kann das dem Benutzer über den Console-Agent mitgeteilt werden. Hat ein Weiterlaufen der Simulation nach Aufdeckung des Fehlers keine Relevanz mehr, kann die Simulation auch gestoppt werden.
- **Logger-Agent:** Informationen, die für die Validierung des Modells interessant sind, werden dem Logger-Agent in der Form von **Log Entries** geschickt. Er speichert diese dann in geeigneter Form ab.
- **Log Entry:** Wichtige Informationen werden in Form eines Log Entry an den Logger-Agent weitergeleitet.
- **Sim-Agent:** Der Sim-Agent repräsentiert die Klasse der Simulationsagenten der eigentlichen Simulation.

Zum besseren Verständnis der folgenden Abschnitte ist der Zusammenhang der verschiedenen Komponenten eines VOMAS in Form eines Klassendiagramms in Abbildung 3.2 dargestellt.

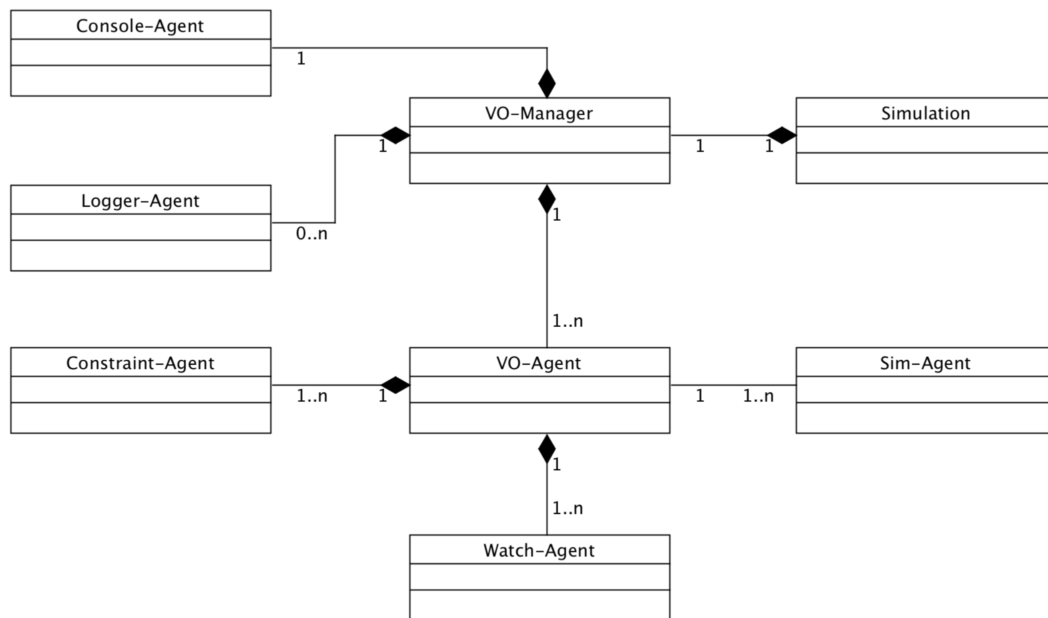

Abbildung 3.2: Klassendiagramm des VOMAS-Design

3.2 Validierung mit einem VOMAS

Naylor und Finger empfehlen, ein Modell in drei Schritten zu validieren [NF67]:

- Entwickle ein Modell mit hohem Grad an *face validity* (siehe Abschnitt 2.2 – **Face Validity**).
- Überprüfe, ob die Modellannahmen „richtig“ gewählt wurden (siehe Abschnitt 2.1 – **Conceptual Model Validation**) und stelle sicher, dass diese im Modell nicht verletzt werden.
- Vergleiche die Input-Output Transformationen mit jenen des realen Systems (siehe Abschnitt 2.2 – **Behavioral Validity**).

Wie bereits erwähnt, ist ein VOMAS ein System aus Beobachtern, Aufsehern und Buchhaltern. Diese Rollenaufteilung lässt sich sehr gut auf die gerade erwähnten Schritte übertragen. Die Beobachterrolle ermöglicht einen hohen Grad an face validity (mittels Watch-Agenten). Die Aufseherrolle verhindert Verletzungen von Modellannahmen und deckt unerwünschtes Modellverhalten auf (mittels Constraint-Agenten). Die Buchhalterrolle ermöglicht eine sehr flexible Datenaggregierung und damit auch eine gute Input-Output-Analyse (mittels Logger-Agent). Jede dieser Eigenschaften, für sich alleine genommen, erleichtert bereits den Validierungsprozess. Die Kombination dieser drei allerdings macht ein VOMAS zu einem sehr flexiblen und vielseitig anwendbaren Validierungswerkzeug. Ob und mit welcher Gewichtung diese Rollen zum Einsatz kommen, ist von Modell zu Modell verschieden. Im

Zuge dieser Arbeit wurde beispielsweise ein VOMAS-Design entwickelt dessen Fokus vor allem auf der Beobachter- und Buchhalterrolle lag, es sich also im Prinzip um ein äußerst flexibles Protokollmodul handelt.

Ein von Beginn an mitentwickeltes VOMAS ermöglicht eine effiziente Anwendung vieler der in Kapitel 2 angeführten Validierungsmethoden:

- **Turing Test, Black Box Testing, Predictive Validation, Statistical Techniques, Traces, Model Alignment:**

  Diese Methoden haben als Voraussetzung, dass ausreichend Daten, sowohl vom realen System als auch vom Modell, verfügbar sind bzw. gesammelt werden können. Die Watch-Agenten können die erforderliche Information sammeln und an den Logger-Agent weiterleiten. Der Logger-Agent erlaubt die Abspeicherung der Daten in einer Form, die sich für die nachfolgenden Untersuchungen am effizientesten erweist. Da der Logger-Agent die Information in Form von Log Entries erhält und damit praktisch unabhängig vom Rest des VOMAS entwickelt werden kann, kann eine Variation von Logger-Agenten erstellt werden, die, je nach Anwendung, Informationen in unterschiedlicher Form und in unterschiedlichen Plattformen abspeichern können.

- **Animation, Visualisation, Immersive Face Validation:**

  Die Anwendung dieser Methoden mit Hilfe eines VOMAS hängt stark davon ab, wie das VOMAS auf die Simulation zugreifen kann. Erlauben die Schnittstellen zwischen VO-Agenten und Sim-Agenten, sowie VO-Manager und der Simulation, eine Visualisierung der Simulationsumgebung, können verschiedene Abläufe und Simulationsagenten grafisch verfolgt werden. Auf die genannten Schnittstellen wird in Abschnitt 3.3.1 im Detail eingegangen.

- **Sensitivity Analysis, Event Validity:**

  Watch-Agenten ermöglichen es, bestimmte Informationen zum Simulationsmodell unter Beobachtung zu haben. Wann Alarm geschlagen werden soll, oder ab welchem Schwellenwert beispielsweise ein numerisches Attribut von Interesse ist, kann dem Watch-Agent je nach Bedarf mitgeteilt werden.

- **Assertion Checking, Extreme Condition Tests:**

  Diese Methoden versuchen falsches Modellverhalten aufzudecken. Dies ist genau die Aufgabe der Constraint-Agenten in einem VOMAS. Constraint-Agenten können vor allem bei einer anfänglichen Parametrisierung des Modells hilfreiche Dienste leisten. Bei einer Parametrisierung wird durch Variation der Parameter ein vernünftiger Bereich bestimmt, in dem sich der Parameter bewegen darf.[5] Die Einschränkung der Parameter auf einen gewissen Bereich kann zum Beispiel daher rühren, dass unter- oder oberhalb gewisser Schwellenwerte unerwünschtes oder irrelevantes Verhalten

[5]Im Zuge einer Kalibrierung können Parameter dann genauer bestimmt werden, um bekannte Daten zu reproduzieren.

beobachtet werden kann. Constraint-Agenten können solch unerwünschtes Verhalten oder unerwünschte Zustände erkennen und die Simulation stoppen, um Zeit zu sparen. Die Simulation kann dann mit einem neuen Parameterwert gestartet und erneut untersucht werden. Im selben Sinn kann '"falsches" Verhalten der Agenten erkannt werden. Das hilft beim Adjustieren der Verhaltensregeln der Agenten.

- **Bottom Up Testing:**

  Ein VOMAS kann natürlich auch nur für Teilmodelle entwickelt bzw. nur auf einzelne Teilmodelle angewandt werden.

In Abschnitt 2.1 wurde angemerkt, dass für viele dynamische Methoden zusätzlicher Quellcode hinzugefügt werden muss, um relevante Daten sammeln zu können. Dies ist bei einem korrekt mitentwickelten VOMAS nicht notwendig, da bereits sehr früh definiert wird, welche Informationen für die Watch-Agenten verfügbar gemacht werden müssen und entsprechende Schnittstellen implementiert sein sollten (siehe Abschnitt 3.5).

3.3 Analyse der VOMAS-Struktur und der einzelnen Komponenten

Niazi et al. geben in [NHK09] nur eine grobe Beschreibung der Aufgaben und der Struktur eines VOMAS wieder. Im Zuge dieser Arbeit wurde ein Vorschlag für eine plattformübergreifende Struktur entwickelt, analysiert und getestet. In der vorgeschlagenen Struktur greift ein VOMAS an genau zwei Stellen auf das Modell zu – über den VO-Manager und die VO-Agenten. Was bei diesen Zugriffen beachtet werden muss wird im ersten Unterabschnitt diskutiert. Abschnitt 3.3.2 ist eine detaillierte Analyse der einzelnen Komponenten des entwickelten VOMAS-Design.

Die Implementierung von agentenbasierten Modellen hat oft einen starken Programmieranteil. Diesbezüglich haben objektorientierte Programmiersprachen eine Struktur, die für agentenbasierte Modelle in natürlicher Weise in Fragen kommen. Zusätzlich stehen verschiedenste Dokumentationstools zur Verügung. Das bekannteste davon ist UML[6]. Im Zuge der folgenden Abschnitte dieses Kapitels, werden Klassen- und Sequenzdiagramme verwendet, um die Komponenten und internen Abläufe des entwickelten VOMAS-Design zu beschreiben. Konkrete Methoden einzelner Klassen werden in folgender Weise beschrieben:

`methodName(paramter)`[7]

Beschreibung: Informationen zu den Aufgaben der Methode. Ist die Aufgabe der Methode ersichtlich (get-Methoden), wird dieser Unterpunkt nicht angeführt.
Paramter: Informationen über die Parameter, die der Methode übergeben werden müssen. Handelt es sich um eine parameterlose Methode, wird dieser Unterpunkt

[6] Unified Modelling Language.
[7] *Kursiv* geschriebene Methoden können erst in der endgültigen Klasse implementiert werden.

nicht angeführt.
Rückgabewert: Informationen zum Rückgabewert der Methode. Hat die Methode keinen Rückgabewert, wird dieser Unterpunkt nicht angeführt.
Anmerkungen: Weitere Informationen zur Methode können hier angeführt sein.

Einige Methoden tragen den Begriff *startup* in der Bezeichnung. Dabei handelt es sich meist um Informationen zu den Sim-Agenten oder der Simulation, die sich während eines Simulationslaufes nicht mehr ändern können und daher nur einmal eingefordert werden müssen.
VO-Agenten sammlen zu allen Simulationsagenten startup Daten. Watch-Agenten hingegen sammeln nur Informationen zu Sim-Agenten, die gewissen Kriterien entsprechen (beispielsweise nur rote Autos, oder nur männliche Personen). Das erwähnte Parameter File `parameterFile` ist ein Dokument in dem alle Informationen, die das VOMAS betreffen, abgespeichert sind. Hierunter fallen jedenfalls die Typen der Watch- und Constraint-Agenten, die zur Anwendung kommen sollen und Informationen darüber, welche Sim-Agenten konkret beobachtet werden müssen. Ein Beispiel für solch ein Dokument kann in Abschnitt 4.2.1 eingesehen werden.

3.3.1 Schnittstellen zwischen einem VOMAS und dem Simulationsmodell

Ein VOMAS muss mit dem eigentlichen Modell gekoppelt werden, um auf Informationen zugreifen zu können. Dies wird über Schnittstellen ermöglicht, die möglichst früh in der Modellentwicklung definiert werden sollten. Hier muss vor allem beachtet werden, dass ein VOMAS die Simulation nicht beeinflussen darf. Welche Aspekte bei der Definition dieser Schnittstellen beachtet werden müssen, werden in diesem Abschnitt untersucht.

Für die Validierung eines Modells mit einem VOMAS sind einerseits Informationen zu den Simulationsagenten als auch Daten zum gesamten Modellverhalten von Interesse. Die Informationsbeschaffung funktioniert über Schnittstellen (1) zwischen den VO-Agenten und den Sim-Agenten sowie (2) zwischen dem VO-Manager und der Simulation. Zusätzlich müssen Mechanismen implementiert werden, die das VOMAS starten und beenden können. Sauber definierte Schnittstellen erlauben in bestimmten Fällen auch einen plattformübergreifenden Zugriff auf die Sim-Agenten und die Simulation (beispielsweise über einen Server).

1. VO-Agenten und Sim-Agenten: Für die Beobachtung der konkreten Simulationsagenten sind die VO-Agenten verantwortlich. Hier spielt die Nähe des VO-Agenten zu den Sim-Agenten eine Rolle. Wie die Nachbarschaft eines Agenten definiert ist, hängt von der Anwendung des Modells ab. Die VO-Agenten müssen in jedem Fall eine Verbindung zu den einzelnen Sim-Agenten besitzen und auf deren Attribute zugreifen können. In bestimmten Fällen müssen die VO-Agenten auch in der Lage sein, auf die Nachbarschaft dieses Agenten zuzugreifen. Der Begriff der Nachbarschaft kann weiter unterteilt werden in:

- Proximity Based: Hier spielt der tatsächliche Abstand – im Sinne der euklidischen Metrik – der Agenten für die Validierung eine Rolle. Zum Beispiel können

VO-Agenten an wichtigen Stellen in der Simulationswelt platziert werden, um Aufgaben auszuführen. In einem Räuber-Beute Modell könnten sie beispielsweise an einer Wasserquelle platziert werden, um vorbeiziehende Tiere zu zählen oder in einem Verkehrsmodell Autos an einer Ampel beobachten.

- Link Based: In Wirtschaftsmodellen spielt beispielsweise der Begriff der Distanz unter Umständen keine Rolle. Vielmehr lassen sich Prozesse in der Wirtschaft über Netzwerke darstellen. Firma A beliefert Firma B und Firma C, diese stehen also in Verbindung, egal ob Firma A im selben Gebäude, oder viele Kilometer weit entfernt ist. VO-Agenten können in diesem Fall in das Netzwerk eingeschleust werden und Transaktionen beobachten.

In jedem dieser Fälle muss sichergestellt werden, dass die VO-Agenten die Simulation und speziell das Verhalten der Sim-Agenten in keiner Weise beeinflussen.

Um den VO-Agenten Zugriff auf Informationen zu den Simulationsagenten zu ermöglichen, müssen die Simulationsagenten das Interface ***I_SimAgent*** implementieren (siehe Abbildung 3.3). Dieses Interface fungiert als Markierung der Simulationsagenten und macht sie „sichtbar“ für die VO-Agenten des VOMAS. Zu Beginn der Simulation werden diese Markierungen an die VO-Agenten verteilt, wodurch sowohl VO-, Watch- und Constraint-Agenten die im Interface beschriebenen Methoden aufrufen können.

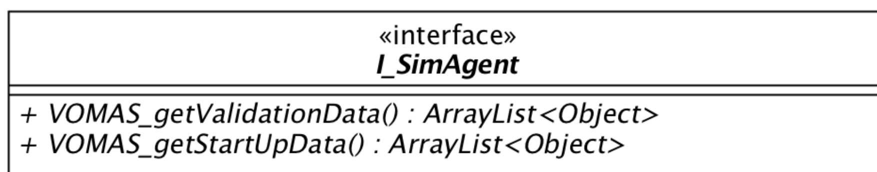

Abbildung 3.3: Zugriff auf die Sim-Agenten durch die VO-,Watch- und Constraint-Agenten

VOMAS_getStartUpData()

Beschreibung: Ermöglicht den Zugriff auf validierungsrelevante Daten zu Beginn der Simulation.
Rückgabewert: Um welche Informationen es sich handelt, ist von Modell zu Modell verschieden.
Anmerkungen: Die Methode steht repräsentativ für die Informationsbeschaffung durch die VO-, Watch- und Constraint-Agenten des VOMAS. Dabei kann es sich auch um mehrere Methoden handeln.

`VOMAS_getValidationData()`

Beschreibung: Ermöglicht den Zugriff auf validierungsrelevante Daten.
Rückgabewert: Um welche Informationen es sich handelt, ist von Modell zu Modell verschieden.
Anmerkungen: Die Methode steht repräsentativ für die Informationsbeschaffung durch die VO-, Watch- und Constraint-Agenten des VOMAS. Dabei kann es sich auch um mehrere Methoden handeln.

2. VO-Manager und Simulation: Oftmals sollen globale Attribute und generelle oder kumulative Informationen zum System gesammelt werden. Das ist eine der Aufgaben des VO-Managers. Muss die Simulation von Seiten des VOMAS aus bestimmten Gründen vorzeitig abgebrochen werden, kann der VO-Manager dies ebenfalls veranlassen. Dies wird ermöglicht, indem die Simulation das Interface ***I_Simulation*** implementiert (siehe Abbildung 3.5). Die Simulation hingegen hat die Möglichkeit, einen Informationssammlungslauf durch das VOMAS auszulösen (im Folgenden als „starten des VOMAS" bezeichnet) oder das VOMAS gänzlich zu beenden – der VO-Manager implementiert hierfür das Interface ***I_Vomas*** (siehe Abbildung 3.4). Das starten des VOMAS ist hier zu unterscheiden von der Initialisierung des VOMAS. Die Initialisierung geschieht nur einmal, zu Beginn des Simulationslaufes. Das starten des VOMAS resultiert in einer einmaligen Informationssammlung. Wie oft ein VOMAS gestartet werden sollte, hängt von der Taktung des eigentlichen Modells ab. In einem agentenbasierten Modell müssen die Agenten ihre Attribute regelmäßig aktualisieren, denn nur so geschieht Veränderung. Diese Updates repräsentieren meist eine abgelaufene Zeitspanne. Beispielsweise können die Ereignisse innerhalb eines Tages den Zustand und das Verhalten eines Agenten für den nächsten Tag beeinflussen. Wie „fein" diese Taktung gewählt wird, hängt von der konkreten Anwendung ab. Ein VOMAS sollte demnach immer dann gestartet werden, nachdem Änderungen im Modell und bei den Agenten *passieren hätten können*. Dadurch kann keine Information unbemerkt bleiben. Genauso wie ein VOMAS nicht zu selten auf die Simulation zugreifen sollte, darf es auch nicht im falschen Moment Informationen sammeln – beispielsweise, wenn noch nicht alle Agenten aktualisiert wurden. Der VO-Manager muss benachrichtig werden, wann es „sicher" ist, auf das Modell zuzugreifen – beispielsweise immer nachdem eine Iteration des Modells beendet wurde und bevor eine weitere gestartet wird.

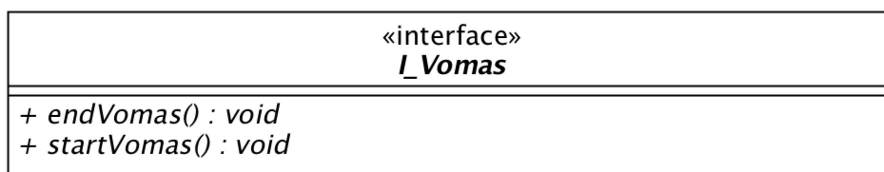

Abbildung 3.4: Zugriff auf das VOMAS durch die Simulation

endVomas()

Beschreibung: Beendet das VOMAS. Sollten Teile des VOMAS noch aktiv sein, werden diese sicher heruntergefahren.
Anmerkungen: Diese Methode wird nur einmal aufgerufen.

startVomas()[8]

Beschreibung: Startet das VOMAS.
Anmerkungen: Der Aufruf dieser Methode passiert mehrere Male während eines Simulationslaufes[9].

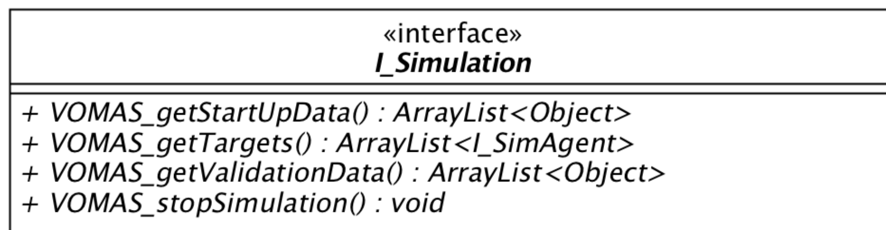

Abbildung 3.5: Zugriff auf die Simulation durch den VO-Manager

VOMAS_getStartUpData()

Beschreibung: Ermöglicht den Zugriff auf validierungsrelevante Daten zu Beginn der Simulation.
Rückgabewert: Um welche Informationen es sich handelt, ist von Modell zu Modell verschieden.
Anmerkungen: Die Methode steht repräsentativ für die Informationsbeschaffung durch den VO-Manager. Dabei kann es sich auch um mehrere Methoden handeln.

VOMAS_getTargets()

Beschreibung: Stellt Verknüpfungen zu allen Simulationsagenten zur Verfügung.
Rückgabewert: Eine Liste aller Sim-Agenten.
Anmerkungen: Es können auch Verweise auf die Simulationsagenten sein. Beispielsweise kann auf die Simulationsagenten über einen Server zugegriffen werden. Damit kann plattformunabhängig gearbeitet werden. Das ist nur bei der Linked-based Version möglich.

[8]Der interne Ablauf dieser Methode ist in Abbildung 3.9 dargestellt.
[9]Abbildung 3.24 zeigt die VOMAS-internen Abläufe nach Aufruf dieser Methode.

VOMAS_getValidationData()

Beschreibung: Ermöglicht den Zugriff auf validierungsrelevante Daten.
Rückgabewert: Um welche Informationen es sich handelt, ist von Modell zu Modell verschieden.
Anmerkungen: Die Methode steht repräsentativ für die Informationsbeschaffung durch den VO-Manager. Dabei kann es sich um mehrere Methoden handeln.

VOMAS_stopSimulation()

Beschreibung: Beendet den Simulationslauf vorzeitig.
Anmerkungen: In der Simulation wurden Verletzungen von Zwangsbedingungen gefunden und ein Weiterlaufen des Simulationslaufes hat keinen Mehrwert für die Validierung. Die Simulation wird beendet um Zeit zu sparen.

3.3.2 Analyse des VOMAS-Design

Vorraussetzung für die Anwendung des entwickelten VOMAS-Design ist, dass die Verantwortlichkeit der VO-Agenten für die Sim-Agenten zu Beginn der Simulation verteilt und im Laufe der Simulation nicht mehr gewechselt wird. Ein VO-Agent ist also innerhalb eines Simulationslaufes immer nur für eine feste Menge an Simulationsagenten zuständig. Außerdem nehmen die VO-Agenten eine reine Beobachterrolle ein. Sie werden also nicht in der eigentlichen Simulation platziert (siehe Kapitel 6 – **Diskussion**).

Im Folgenden werden die Aufgabenbereiche der jeweiligen VOMAS-Agenten beschrieben und deren Attribute und Methoden angeführt. Die Analyse jeder VOMAS Komponente wird nach einer Beschreibung der grundsätzlichen Aufgaben, unterteilt in (1) Interface, (2) abstrakte Klasse und (3) finale Klasse.

1. Interface: Im Interface werden jene Methoden aufgelistet, die von außen aufgerufen werden müssen. Die Kommunikation der Agenten des VOMAS untereinander funktioniert demnach ausschließlich über die Methoden der implementierten Interfaces.

2. abstrakte Klasse: Die abstrakte Klasse implementiert immer das zum jeweiligen Agenten zugehörige Interface – muss also alle Methoden des Interface übernehmen. Zusätzlich zu diesen Methoden, werden alle notwendigen Attribute und internen Methoden aufgelistet, um die Funktionstüchtigkeit des VOMAS zu garantieren. Einige dieser Methoden können in einer abstrakten Klasse bereits implementiert werden. Der Aufbau der einzelnen abstrakten Klassen erhebt allerdings keinen Anspruch auf Vollständigkeit. Die Struktur kann nach Bedarf erweitert werden, erlaubt aber im präsentierten Zustand sicherlich ein Mindestmaß an Flexibilität für den Anwender.

3. finale Klasse: Die finale Klasse wird von der abstrakten Klasse abgeleitet und muss alle geerbten abstrakten Methoden implementieren. Da das Aussehen dieser Methoden von der tatsächlichen Anwendung abhängt, kann in einer theoretischen Analyse

nicht viel Information dazu geliefert werden. Deshalb wird dieser Unterpunkt nicht bei jeder Komponente aufgelistet.

Zum besseren Verständnis wird empfohlen, zunächst nur die Interfaces der einzelnen Klassen zu studieren, um sich mit dem groben Konstrukt vertraut zu machen. Sequenzdiagramme helfen dabei, die Zugriffe der VOMAS-Agenten untereinander und im weiteren die Abläufe der internen Methoden besser zu verstehen.

- **VO-Manager**

  Die Rolle des VO-Managers in Verbindung mit der Simulation wurde bereits im vorigen Abschnitt diskutiert. Innerhalb des VOMAS ist der VO-Manager die zentrale Schaltstelle. Er stellt den VO-, Watch- und Constraint-Agenten den Logger- und den Console-Agenten zur Verfügung und hat alleinigen Zugriff auf das Parameter File. Der VO-Manager erteilt den VO-Agenten – und damit auch den Watch- und Constraint-Agenten – den Befehl die Simulationsagenten zu überwachen.

  Interface (siehe Abbildung 3.6):

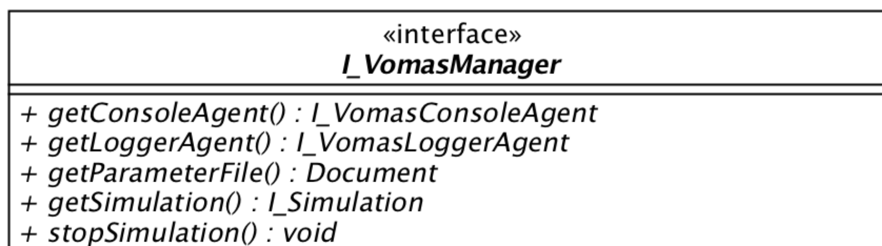

Abbildung 3.6: Zugriff auf den VO-Manager

`getConsoleAgent()`

Rückgabewert: Der Console-Agent des VOMAS.
Anmerkungen: Da nur der VO-Manager eine Verbindung zum Console-Agenten besitzt, kann nur über den VO-Manager auf diesen zugegriffen werden. Die Methode wird von weiteren VOMAS-Agenten implementiert, dort aber nicht noch einmal extra beschrieben.

`getLoggerAgent()`

Rückgabewert: Der Logger-Agent des VOMAS.
Anmerkungen: Da nur der VO-Manager eine Verbindung zum Logger-Agenten besitzt, kann nur über den VO-Manager auf diesen zugegriffen werden. Die Methode wird von weiteren VOMAS-Agenten implementiert, dort aber nicht noch

einmal extra beschrieben.

`getParameterFile()`

Rückgabewert: Das Parameter File des VOMAS.
Anmerkungen: Da nur der VO-Manager eine Verbindung zum Parameter File besitzt, kann nur über den VO-Manager auf dieses zugegriffen werden. Die Methode wird von weiteren VOMAS-Agenten implementiert, dort aber nicht noch einmal extra beschrieben.

`getSimulation()`

Rückgabewert: Eine Verknüpfung zur Simulation.
Anmerkungen: Da nur der VO-Manager eine Verbindung zur Simulation besitzt, kann nur über den VO-Manager auf diese zugegriffen werden.

`stopSimulation()`[10]

Beschreibung: Der VO-Manager gibt der Simulation Bescheid, dass eine Zwangsbedingung verletzt wurde und der Simulationslauf gestoppt werden sollte. Danach beendet er das VOMAS.
Anmerkungen: Sollen Daten zum Grund des Abbruchs gespeichert werden, können diese im Zuge der Methode an den Logger-Agenten weitergeleitet werden.

[10] Der interne Ablauf dieser Methode ist in Abbildung 3.9 dargestellt.

Abstrakte Klasse (siehe Abbildung 3.7):

VOMAS_anyManager

– agents: ArrayList<I_VomasAgent>
– consoleAgent: I_VomasConsole
– loggerAgent: I_VomasLogger
– parameterFile: Document
– simulation: I_Simulation

+ VOMAS_anyManager(simulation: I_Simulation)
+ endVomas() : void
+ getConsoleAgent() : I_VomasConsoleAgent
+ getLoggerAgent() : I_VomasLoggerAgent
+ getParameterFile() : Document
+ getSimulation() : I_Simulation
+ startVomas() : void
+ stopSimulation() : void
– *afterGatheringData() : void*
– *beforeGatheringData() : void*
– *createConsoleAgent() : void*
– *createLoggerAgent() : void*
– *distributeAgents() : void*
– *evaluateData() : void*
– *evaluateStartUpData() : void*
– gatherData() : void
– gatherStartUpData() : void
– getTargets() : ArrayList<I_SimAgent>
– *initialTasks() : void*
– initiateVOMAS() : void
– sendAgentsToGatherData() : void
– sendAgentsToGatherStartUpData() : void
– shutDownLoggerAgent() : void
– shutDownSimulation() : void

Abbildung 3.7: Attribute und Methoden der VOMAS_anyManager Klasse

– **Attribute:**

`agents`**:** Die Liste der VO-Agenten, die dem VO-Manager unterstehen.
`consoleAgent`**:** Der Console-Agent des VOMAS.
`loggerAgent`**:** Der Logger-Agent des VOMAS. Hat das VOMAS mehrere Logger-Agenten, müssen mehrere Variablen erstellt werden und die *getLoggerAgent*-Methode angepasst werden.
`parameterFile`**:** Ein Dokument, in dem Informationen zum VOMAS gespeichert sind. Hierunter fallen jedenfalls die Typen der Watch- und Constraint-Agenten, die zur Anwendung kommen sollen und Informationen darüber, welche Sim-Agenten konkret beobachtet werden müssen.
`simulation`**:** Verknüpfung zur Simulation.

- **Methoden:**

  `VOMAS_anyManager(simulation)` [11,12]

  Beschreibung: Der Konstruktor der VO-Manager Klasse.
  Paramter: Die Simulation erstellt den VO-Manager und übergibt sich selber als Parameter.

  `afterGatheringData()`

  Beschreibung: Aufgaben, die nach dem Sammeln von Daten erledigt werden müssen.
  Anmerkungen: Hierunter kann zum Beispiel das Speichern der Log Entries durch den Logger-Agenten fallen.

  `beforeGatheringData()`

  Beschreibung: Aufgaben, die vor dem Sammeln von Daten erledigt werden müssen.

  `createConsoleAgent()`

  Beschreibung: Erstellt den Console-Agent.

  `createLoggerAgent()`

  Beschreibung: Erstellt den Logger-Agent.

  `distributeAgents()`

  Beschreibung: Erstellt die VO-Agenten und teilt diesen einen oder mehrere Sim-Agenten zu, die dann von ihnen zu überwachen sind.
  Anmerkungen: Die Verteilung der Sim-Agenten kann je nach Anwendung variieren.

  `evaluateData()`

  Beschreibung: Der VO-Manager sammelt Informationen zur Simulation und leitet diese unter Umständen an den Logger-Agenten weiter.

  `evaluateStartUpData()`

  Beschreibung: Der VO-Manager sammelt Informationen zu Beginn der Simulation und leitet diese unter Umständen an den Logger-Agenten weiter.

[11] Der interne Ablauf dieser Methode ist in Abbildung 3.8 dargestellt.
[12] Abbildung 3.23 zeigt die VOMAS-internen Abläufe nach Aufruf dieser Methode.

`gatherData()`

Beschreibung: Beauftragt sowohl den VO-Manager, als auch die VO-Agenten mit dem Sammeln von Informationen.

`gatherStartUpData()`

Beschreibung: Beauftragt sowohl den VO-Manager, als auch die VO-Agenten mit dem Sammeln der *startup*-Informationen.

`getTargets()`

Beschreibung: Fragt bei der Simulation um die zu überwachenden Sim-Agenten an und speichert diese in `targets` ab.
Rückgabewert: Eine Liste aller Sim-Agenten.

`initialTasks()`

Beschreibung: Aufgaben, die gleich nach der Initialisierung erledigt werden müssen.
Anmerkungen: Hierunter fällt zum Beispiel das Importieren des Parameter Files.

`initiateVOMAS()`

Beschreibung: Beauftragt den VO-Manager damit, den Logger-Agenten, den Console-Agenten und die VO-Agenten zu erzeugen.

`sendAgentsToGatherStartUpData()`

Beschreibung: Beauftragt die VO-Agenten mit dem Sammeln der *startup*-Informationen.
Anmerkungen: Eine ähnliche Methode wird von den VO-Agenten für ihre Watch- und Constraint-Agenten implementiert, dort aber nicht noch einmal extra beschrieben.

`sendAgentsToGatherData()`

Beschreibung: Beauftragt die VO-Agenten mit dem Sammeln von Informationen.

`shutDownLoggerAgent()`

Beschreibung: Der VO-Manager schickt dem Logger-Agenten die Nachricht, dass er seine Arbeit sauber beenden soll.

`shutDownSimulation()`

Beschreibung: Der VO-Manager schickt der Simulation die Nachricht, dass der aktuelle Simulationslauf abgebrochen werden sollte.

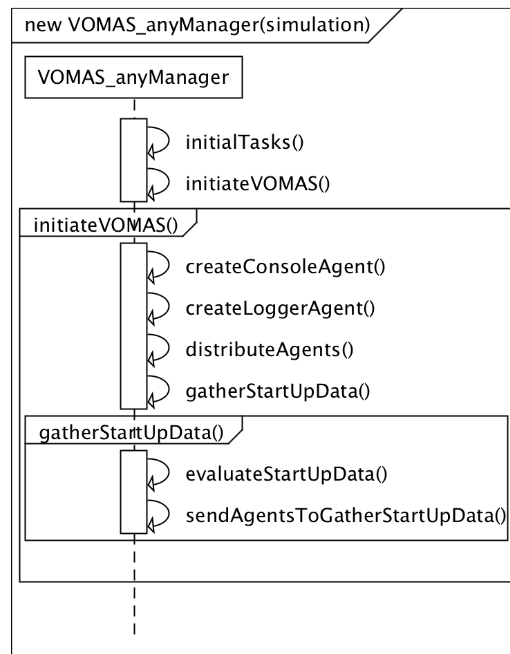

Abbildung 3.8: Interner Ablauf des Konstruktors der VOMAS_anyManager Klasse

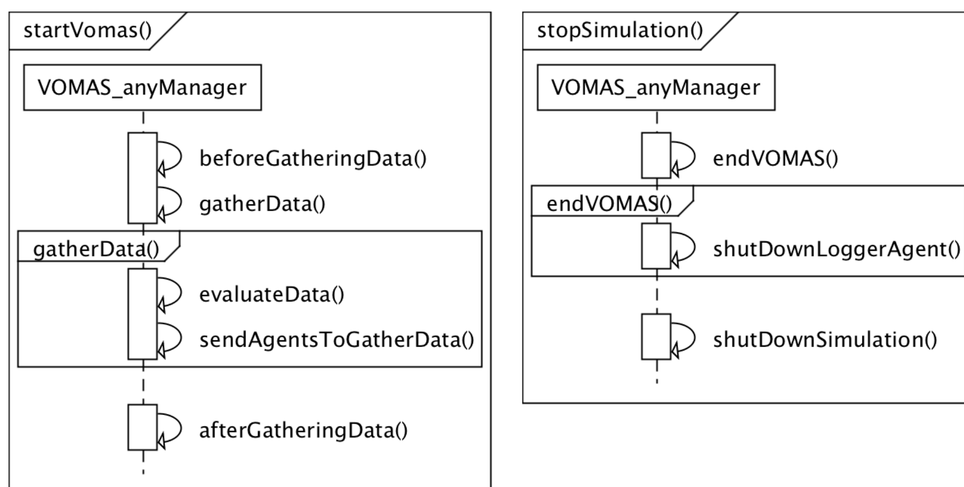

Abbildung 3.9: Interner Ablauf der Methoden *startVomas* und *stopSimulation*

- **Logger-Agent**

Der Logger-Agent speichert Informationen dauerhaft, damit diese nach einem Simulationslauf analysiert werden können. Wie der Logger-Agent die Informationen speichert, kann je nach Modell und Bedarf variieren. Konkrete Informationen zum Speicherort sollten im Parameter File gespeichert sein. Der Logger-Agent kann in den meisten Fällen außerdem auch parallel zum Rest des VOMAS laufen.

Interface (siehe Abbildung 3.10):

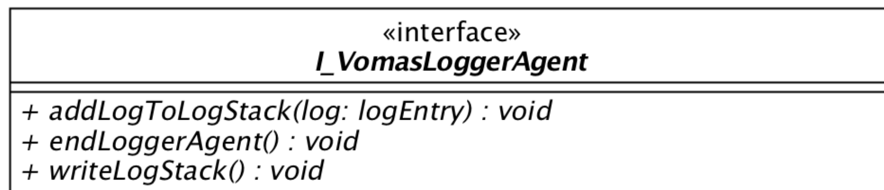

Abbildung 3.10: Zugriff auf den Logger-Agent

`addLogToLogStack(log)`

Beschreibung: Fügt einen Log Entry der Menge an Log Entries hinzu, die vom Logger-Agenten gespeichert werden müssen.
Paramter: Der Log Entry enthält Information, die für die Validierung des Modells relevant sind.
Anmerkungen: Je nach Anwendung können auch mehrere Log Stacks existieren, denen jeweils andere Typen von Log Entries zugeteilt werden.

`endLoggerAgent()`

Beschreibung: Arbeitet der Logger-Agent parallel zum Rest des Programmes, wird der Logger-thread mit dieser Methode sauber beendet. Dazu zählt unter anderem, dass der Log Stack ein letztes Mal abgearbeitet wird.

`writeLogStack()`

Beschreibung: Alle Log Entries, die dem Logger-Agent gesendet wurden, werden gespeichert.
Anmerkungen: Existieren mehrere Log Stacks, werden mittels dieser Methode alle Log Entries in allen Log Stacks gespeichert.

Abstrakte Klasse (siehe Abbildung 3.11): Bis auf den Konstruktor ist es nicht sinnvoll, weitere Methoden auf einer abstrakten Ebene zu implementieren, da jede seiner

Methoden von der endgültigen Anwendung abhängt.

VOMAS_anyLoggerAgent
– manager: I_VomasManager
+ VOMAS_anyLoggerAgent(manager: I_VomasManager) : void *+ addLogToLogStack(log: logEntry) : void* *+ endLoggerAgent() : void* *+ writeLogStack() : void*

Abbildung 3.11: Attribute und Methoden der VOMAS_anyLoggerAgent Klasse

- **Attribute:**

  `manager`: Der VO-Manager des VOMAS.

- **Methoden:**

  `VOMAS_anyLoggerAgent(manager)`

  Beschreibung: Der Konstruktor der Logger-Agent Klasse.
  Parameter: Der VO-Manager übergibt sich selber als Parameter `manager`.
  Anmerkungen: Der Logger-Agent hat nur Zugriff auf den VO-Manager, weil er Zugang zum Parameter File benötigt.

Finale Klasse: Für das Influenzamodell wurde ein Logger-Agent implementiert, der verschiedenste Daten in einer Datenbank speichert. Diese Version des Logger-Agenten wird in Abschnitt 4.2.1 detailliert beschrieben. %newpage

- **Console-Agent**

Der Console-Agent kann dem Benutzer noch während eines Simulationslaufes Nachrichten anzeigen.

Interface (siehe Abbildung 3.12):

«interface» ***I_VomasConsoleAgent***
+ displayMessage(message: String) : void

Abbildung 3.12: Zugriff auf den Console-Agent

`displayMessage(message)`

Beschreibung: Zeigt dem Benutzer eine Nachricht an.
Paramter: Die Nachricht, die angezeigt werden soll.
Anmerkungen: Um Nachrichten von der Simulation und dem VOMAS zu unterscheiden, sollten diese dementsprechend gekennzeichnet werden.

Abstrakte Klasse: Den Console-Agenten auf einer abstrakten Ebene zu definieren ist nicht sinnvoll, da er keine anwendungsabhängigen Methoden besitzt.

Finale Klasse: Eine Möglichkeit anzuzeigen, dass die ausgegebene Nachricht über den Console-Agenten läuft und mit dem VOMAS in Verbindung steht, ist vor die Nachricht den String `VOMAS:` zu setzen.

- **VO-Agent**

Der VO-Agent ist verantwortlich für die Informationsbeschaffung auf der Simulationsagentenebene. Gewisse Informationen kann der VO-Agent selber evaluieren, andere Informationen kann er von Watch- oder Constraint-Agenten sammeln und überprüfen lassen. Falls notwendig können diese Informationen an den Logger-Agenten weitergeleitet werden.

Interface (siehe Abbildung 3.13):

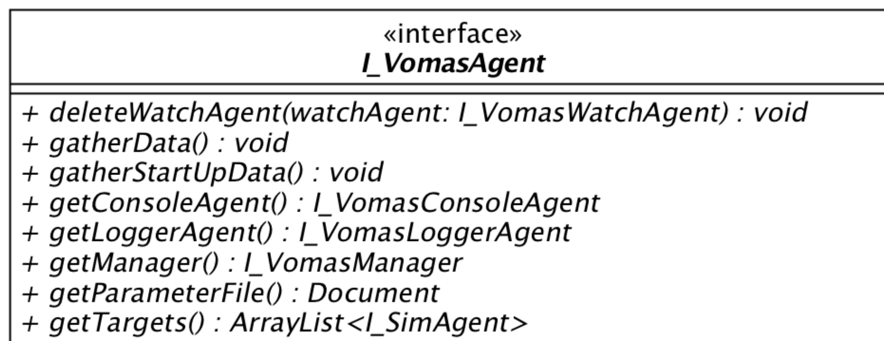

Abbildung 3.13: Zugriff auf den VO-Agent

`deleteWatchAgent(watchAgent)`

Beschreibung: Unter Umständen muss ein Watch-Agent den ihm zugeteilten Sim-Agenten nicht mehr beobachten, da sein weiteres Verhalten für die Validierung ab diesem Zeitpunkt nicht mehr von Relevanz ist.
Paramter: Der zu löschende Watch-Agent.

Anmerkungen: Der Watch-Agent schreibt sich selbst auf die „Abschussliste“, wird aber erst gelöscht, nachdem er seine Arbeit beendet hat.

`gatherData()`[13]

Beschreibung: Beauftragt den VO-Agenten damit, seinen Watch- und Constraint-Agenten den Befehl zu geben, ihre Sim-Agenten zu überwachen und zu überprüfen.

`gatherStartUpData()`[14]

Beschreibung: Beauftragt sowohl den VO-Agenten, als auch seine Watch- und Constraint-Agenten mit dem Sammeln und Prüfen der *startup*-Informationen.

`getTargets()`

Beschreibung: Liefert die Sim-Agenten die von diesem VO-Agenten überwacht werden sollen.
Rückgabewert: Eine Liste von Sim-Agenten.
Anmerkungen: Es ist auch möglich, dass ein VO-Agent nur einem Sim-Agenten zugeteilt ist.

Abstrakte Klasse (siehe Abbildung 3.14):

– **Attribute:**
`manager`: Der VO-Manager des VOMAS.
`targets`: Eine Liste von Sim-Agenten, die dem VO-Agenten zugeteilt sind.
`watchAgents`: Eine Liste von verschiedenen Watch-Agenten, die die `targets` beobachten sollen.
`constraintAgents`: Eine Liste von Constraint-Agenten, die die `targets` überprüfen sollen.
`deletedWatchAgents`: Eine Liste von Watch-Agenten, die nicht mehr zum Einsatz kommen müssen.

– **Methoden:**

`VOMAS_anyAgent(manager, targets)`[15]

Beschreibung: Der Konstruktor der VO-Agent Klasse. Im Zuge dieser Methode werden die Watch-Agenten und die Constraint-Agenten erstellt.

[13] Der interne Ablauf dieser Methode ist in Abbildung 3.16 dargestellt.
[14] Der interne Ablauf dieser Methode ist in Abbildung 3.16 dargestellt.
[15] Der interne Ablauf dieser Methode ist in Abbildung 3.15 dargestellt.

```
VOMAS_anyAgent
- constraintAgents: ArrayList<I_VomasConstraintAgent>
- deletedWatchAgents: ArrayList<I_VomasWatchAgent>
- manager: I_VomasManager
- targets: ArrayList<I_SimAgent>
- watchAgents: ArrayList<I_VomasWatchAgent>
+ VOMAS_anyAgent(manager: I_VomasManager,
                 targets: ArrayList<I_SimAgent>)
+ deleteWatchAgent(watch: I_VomasWatch) : void
+ gatherData() : void
+ gatherStartUpData() : void
+ getConsoleAgent() : I_VomasConsole
+ getLoggerAgent() : I_VomasLogger
+ getManager() : I_VomasManager
+ getParameterFile() : Document
+ getTargets() : ArrayList<I_SimAgent>
- evaluateStartUpData() : void
- evaluateStartUpData(target: I_SimAgent) : void
- extractConstraintAgentNames(parameterFile: Document) : String[]
- extractWatchAgentLoggingValues(parameterFile: Document)
                                 : boolean[]
- extractWatchAgentNames(parameterFile: Document) : String[]
- initiateConstraintAgents(constraintAgentNames: String[]) : void
- initiateWatchAgents(watchAgentNames: String[],
                      watchAgentLoggingValues: boolean[]) : void
- sendConstraintAgentsToCheckData() : void
- sendConstraintAgentsToCheckStartUpData() : void
- sendWatchAgentsToGatherData() : void
- sendWatchAgentsToGatherStartUpData() : void
- updateWatchAgentList() : void
```

Abbildung 3.14: Attribute und Methoden der VOMAS_anyAgent Klasse

Paramter: Der VO-Manager übergibt sich selber als Paramter `manager`. `targets` sind jene Sim-Agenten, die diesem VO-Agenten zugeteilt sind.

`evaluateStartUpData()`

Beschreibung: Der VO-Agent sammelt zu Beginn der Simulation Informationen über die ihm zugteilten Sim-Agenten `targets` und leitet diese an den Logger-Agenten weiter.

`evaluateStartUpData(target)`

Beschreibung: Der VO-Agent sammelt zu Beginn der Simulation Informationen über den Sim-Agent und leitet diese an den Logger-Agenten weiter.

`extractConstraintAgentNames(parameterFile)`

Beschreibung: Entnimmt aus dem Parameter File des VOMAS, welche Typen von Constraint-Agenten in diesem Simulationslauf zum Einsatz kommen sollen.
Paramter: siehe VO-Manager Attribut `parameterFile`
Rückgabewert: Gibt einen Array von Strings mit den Klassennamen der Constraint-Agenten zurück.
Anmerkungen: Der User kann vor jedem Simulationslauf entscheiden, welche Constraint-Agenten zum Einsatz kommen sollen. Es muss deshalb jedes Mal bei der Initialisierung nachgefragt werden, welche das sind.

`extractWatchAgentLoggingValues(parameterFile)`

Beschreibung: Entnimmt aus dem Parameter File des VOMAS, ob bestimmte Watch-Agenten ihre Beobachtungen immer an den Logger-Agenten weiterleiten sollen, oder das nur unter bestimmten Vorraussetzungen tun sollen.
Paramter: siehe Vo-Manager Attribut `parameterFile`
Rückgabewert: Gibt einen Array von Booleans zurück.
Anmerkungen: Der User kann vor jedem Simulationslauf entscheiden, welche Watch-Agenten ständig mitschreiben sollen, und welche nicht. Es muss deshalb jedes Mal bei der Initialisierung nachgefragt werden, welche Variation vom Benutzer gewünscht wurde.

`extractWatchAgentNames(parameterFile)`

Beschreibung: Entnimmt aus dem Parameter File, welche Typen von Watch-Agenten in diesem Simulationslauf zum Einsatz kommen sollen.
Paramter: siehe VO-Manager Attribut `parameterFile`.
Rückgabewert: Gibt einen Array von Strings mit den Klassennamen der Watch-Agenten zurück.
Anmerkungen: Der User kann vor jedem Simulationslauf entscheiden, welche Watch-Agenten zum Einsatz kommen sollen. Es muss deshalb jedes Mal bei der Initialisierung nachgefragt werden, welche das sind.

`initiateConstraintAgents(constraintAgentNames)`

Beschreibung: Erstellt die vom Benutzer gewünschten Constraint-Agenten.
Paramter: Ein String Array mit den Klassennamen der gewünschten Constraint-Agenten.
Anmerkungen: Da eine Verletzung von Zwangsbedingungen nur gespeichert werden kann, wenn diese tatsächlich auftritt, gibt es bei den Constraint-Agenten keinen `continuousLoggingValue`.

`initiateWatchAgents(watchAgentNames, watchAgentLoggingValues)`

Beschreibung: Erstellt die vom Benutzer gewünschten Watch-Agenten.
Paramter: Ein String Array mit den Klassennamen der gewünschten Watch-Agenten und ein boolean Array mit der Information, ob immer mitgeschrieben werden soll.

`updateWatchAgentList()`

Beschreibung: Aktualisiert die Liste der Watch-Agenten `watchAgents`.
Anmerkungen: Die Informationen hierfür werden der Liste `deletedWatchAgents` entnommen.

Finale Klasse: Steht der Dokumenttyp des Parameter File fest – und damit auch, wie auf das Dokument zugegriffen wird – können die Methoden *extractConstraintAgentNames*, *extractWatchAgentNames* und *extractWatchAgentLoggingValues* bereits in der abstrakten Klasse implementiert werden. Damit muss in der finalen Klasse nur noch die Methode *evaluateStartUpData* formuliert werden.

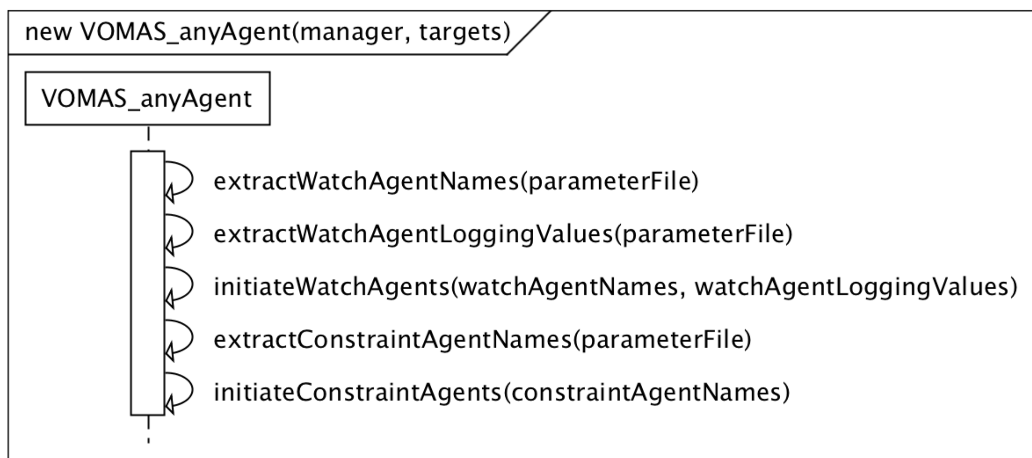

Abbildung 3.15: Interner Ablauf des Konstruktors der VOMAS_anyAgent Klasse

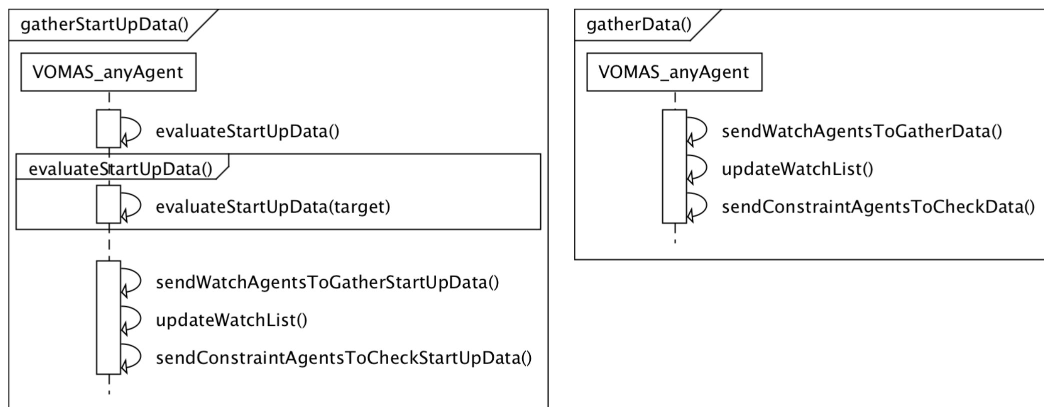

Abbildung 3.16: Interner Ablauf der Methoden *gatherStartUpData* und *gatherData*

- **Watch-Agent**

Der Watch-Agent beobachtet die Simulationsagenten auf interessante Veränderungen hin. Welche Attribute für ihn von Interesse sind, hängt von seinem Typ ab. Ein VOMAS kann also verschiedene Watch-Agent Typen besitzen, die die Simulationsagenten auf verschiedenste Veränderungen hin beobachten. Ein Watch-Agent kann Informationen entweder immer stur an einen Logger-Agenten weiterleiten, oder zuerst selber entscheiden, ob die Information validierungsrelevant ist. Ist der Watch-Agent der Meinung, dass das weitere Verhalten eines Sim-Agenten nicht mehr von Interesse ist, kann er diesen Agenten aus seiner Beobachtungsliste streichen.

Interface (siehe Abbildung 3.17):

Abbildung 3.17: Zugriff auf den Watch-Agent

`gatherData()`[16]

Beschreibung: Der Watch-Agent sammelt für jeden der zu beobachtenden Sim-Agenten die für seine Tätigkeit relevanten Informationen.

[16] Der interne Ablauf dieser Methode ist in Abbildung 3.19 dargestellt.

`gatherStartUpData()`[17]

Beschreibung: Beauftragt den Watch-Agenten damit, für alle Sim-Agenten Informationen, die zu Beginn der Simulation relevant sind, zu sammeln.

Abstrakte Klasse (siehe Abbildung 3.18):

VOMAS_anyWatchAgent
– continuousLogging: boolean – agent: I_VomasAgent – deletedTargets: ArrayList<I_SimAgent>
+ VOMAS_anyWatchAgent(loggingValue: boolean, agent: I_VomasAgent) : void + gatherData() : void + gatherStartUpData() : void – deleteTarget(target: I_SimAgent) : void – evaluateData() : void *– evaluateData(target: I_SimAgent) : boolean* *– isValidTarget(target: I_SimAgent) : boolean* – gatherStartUpData(target: I_SimAgent) : void – generateLoggingData() : void *– generateLoggingData(target: I_SimAgent) : void* *– generateStartUpData(target: I_SimAgent) : void* – getConsoleAgent() : I_VomasConsole – getLoggerAgent() : I_VomasLogger – getManager() : I_VomasManager – getParameterFile() : Document – getTargets() : ArrayList<I_SimAgent>

Abbildung 3.18: Attribute und Methoden der VOMAS_anyWatchAgent Klasse

– **Attribute:**

`continuousLogging`**:** Gibt an, ob Informationen immer an den Logger-Agenten weitergeleitet werden sollen oder nur unter bestimmten Umständen.
`agent`**:** Der VO-Agent, dem dieser Watch-Agent unterstellt ist.
`deletedTargets`**:** Eine Liste von Sim-Agenten, die von diesem Watch-Agenten nicht mehr beobachtet werden müssen.

– **Methoden:**

`VOMAS_anyWatchAgent(loggingValue, agent)`

Beschreibung: Der Konstruktor der Watch-Agent Klasse.
Paramter: Der VO-Agent übergibt sich selber als Paramter `agent`. `loggingValue`

[17] Der interne Ablauf dieser Methode ist in Abbildung 3.19 dargestellt.

wird auf die Variable `continuousLogging` gespeichert.

`deleteTarget(target)`

Beschreibung: Fügt einen Sim-Agenten der Liste von Agenten hinzu, die nicht mehr beobachtet werden müssen (`deletedTargets`).
Paramter: Der zu löschende Sim-Agent.

`evaluateData()`

Beschreibung: Überprüft, für alle Sim-Agenten, ob validierungsrelevante Informationen gesammelt werden sollten.

`evaluateData(target)`

Beschreibung: Überprüft, ob von einem Sim-Agenten validierungsrelevante Informationen gesammelt werden sollten.
Paramter: Der zu beobachtende Sim-Agent.
Rückgabewert: `true`, falls validierungsrelevante Informationen gesammelt werden sollen.
Anmerkungen: Oft geben bereits einzelne Attribute Auskunft darüber, ob interessante Veränderungen bei einem Sim-Agenten passiert sind.

`isValidTarget(target)`

Beschreibung: Überprüft, ob der Sim-Agent in diesem Simulationslauf überhaupt von Interesse für den Benutzer ist. Die Informationen dazu sind im Parameter File gespeichert.
Paramter: Der in Frage kommende Sim-Agent.
Rückgabewert: `true`, falls der Sim-Agent beobachtet werden soll.
Anmerkungen: Soll der Sim-Agent nicht weiter beobachtet werden, wird er mit der internen Methode *deleteTarget* gelöscht werden.

`gatherStartUpData(target)`

Beschreibung: Überprüft, für einen Sim-Agenten, ob dieser beobachtet werden soll. Falls das der Fall ist werden validierungsrelevante Informationen gesammelt.

`generateLoggingData()`

Beschreibung: Sammelt zu allen Sim-Agenten validierungsrelevante Informationen und leitet diese an den Logger-Agenten.
Paramter: Der zu beobachtende Sim-Agent.

generateLoggingData(target)

Beschreibung: Sammelt von einem Sim-Agenten validierungsrelevante Information und sendet diese an den Logger-Agenten weiter.
Paramter: Der zu beobachtende Sim-Agent.

generateStartUpData(target)

Beschreibung: Sammelt von einem Sim-Agenten validierungsrelevante Informationen zu Beginn der Simulation und leitet diese an den Logger-Agenten.
Paramter: Der zu beobachtende Sim-Agent.

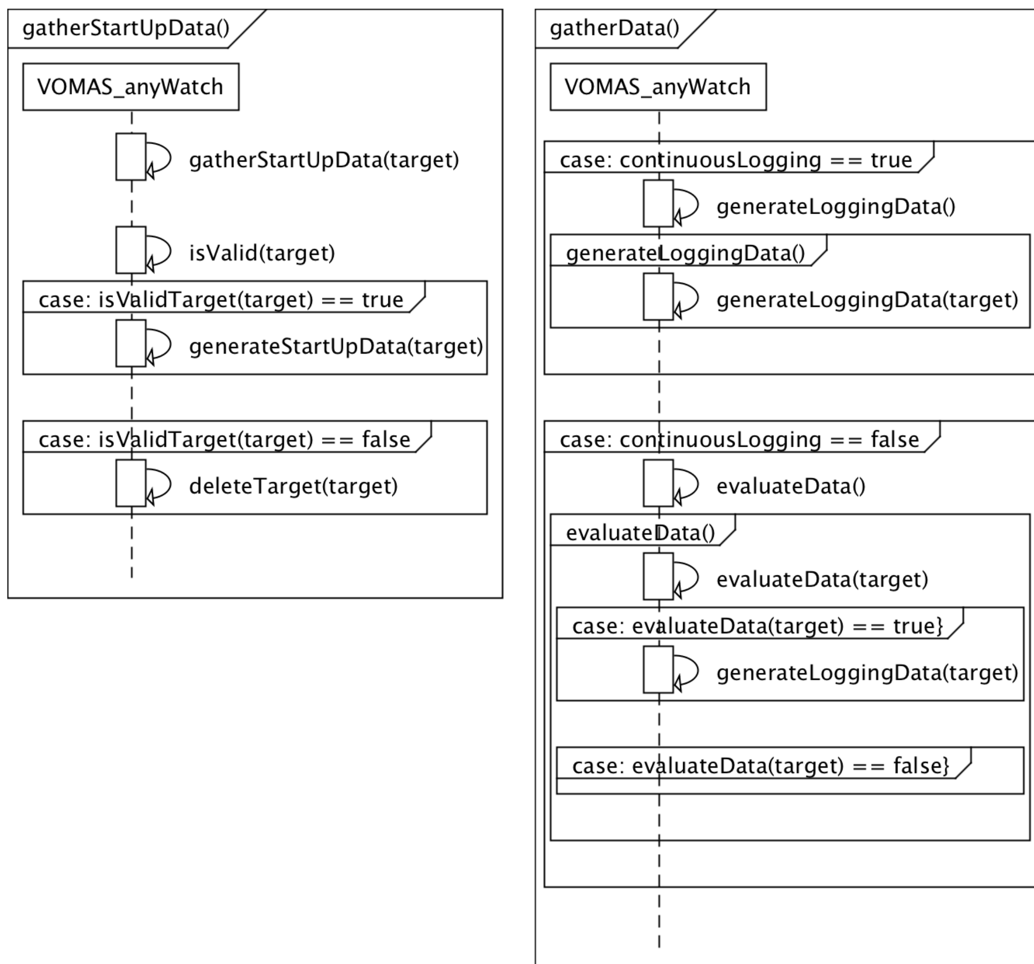

Abbildung 3.19: Interner Ablauf der Methoden *gatherStartUpData* und *gatherData*

- **Constraint-Agent**

Constraint-Agenten überprüfen das Verhalten und die Attribute der Simulationsagenten auf Verletzungen von Zwangsbedingungen hin. Wie bei den Watch-Agenten auch, erlaubt ein VOMAS die Anwendung verschiedener Typen von Constraint-Agenten. Prinzipiell sollte für jede Zwangsbedingung ein eigener Constraint-Agent entworfen werden. Stößt ein Constraint-Agent auf die Verletzung einer solchen Zwangsbedingung, teilt er das dem Benutzer noch während des Simulationslaufes mit. Außerdem hat er die Möglichkeit, diverse Informationen zu der Verletzung an den Logger-Agenten weiterzuleiten und – falls notwendig – den VO-Manager anzuweisen, die Simulation vorzeitig abzubrechen.

Interface (siehe Abbildung 3.20):

Abbildung 3.20: Zugriff auf den Constraint-Agent

`checkData()`[18]

Beschreibung: Der Constraint-Agent sammelt für jeden der zu beobachtenden Sim-Agenten die für ihn relevanten Informationen und entscheidet, ob Zwangsbedingungen verletzt wurden.

`checkStartUpData()`[19]

Beschreibung: Der Constraint-Agent sammelt für jeden der zu beobachtenden Sim-Agenten die für ihn relevanten Informationen und entscheidet, ob bereits zu Beginn der Simulation Zwangsbedingungen verletzt sind.

Abstrakte Klasse (siehe Abbildung 3.21):

- **Attribute:**
  `agent`: Der VO-Agent, dem der Constraint-Agent unterstellt ist.

[18] Der interne Ablauf dieser Methode ist in Abbildung 3.22 dargestellt.
[19] Der interne Ablauf dieser Methode ist in Abbildung 3.22 dargestellt.

VOMAS_anyConstraintAgent
– agent: I_VomasAgent
+ VOMAS_anyConstraintAgent(agent: I_VomasAgent) : void + checkData() : void + checkStartUpData() : void – checkData(target: I_SimAgent) : void – checkStartUpData(target: I_SimAgent) : void *– evaluateData(target: I_SimAgent) : boolean* *– evaluateStartUpData(target: I_SimAgent) : boolean* – getConsoleAgent() : I_VomasConsoleAgent – getLoggerAgent() : I_VomasLoggerAgent – getManager() : I_VomasManager – getTargets() : ArrayList<I_SimAgent> *– violationDetected(target: I_SimAgent) : void* *– startUpViolationDetected(target: I_SimAgent) : void*

Abbildung 3.21: Attribute und Methoden der VOMAS_anyConstraintAgent Klasse

– **Methoden:**

`VOMAS_anyConstraintAgent(agent)`

Beschreibung: Der Konstruktor der Constraint-Agent Klasse.
Paramter: Der VO-Agent übergibt sich selber als Parameter `agent`.

`checkData(target)`

Beschreibung: Überprüft die Daten eines Simulationsagenten auf Verletzungen von Zwangsbedingungen.
Paramter: Der zu überprüfende Sim-Agent.

`checkStartUpData(target)`

Beschreibung: Überprüft die Daten eines Simulationsagenten auf Verletzungen von Zwangsbedingungen zu Beginn der Simulation.
Paramter: Der zu überprüfende Sim-Agent.

`evaluateData(target)`

Beschreibung: Überprüft die Daten eines Simulationsagenten auf Verletzungen von Zwangsbedingungen.
Paramter: Der zu überprüfende Sim-Agent.
Rückgabewert: `true`, falls Verletzungen von Zwangsbedingungen gefunden wurden.

`evaluateStartUpData(target)`

Beschreibung: Überprüft, ob bereits zu Beginn der Simulation durch einen Simulationsagenten Zwangsbedingungen verletzt wurden.
Paramter: Der zu überprüfende Sim-Agent.
Rückgabewert: `true`, falls Verletzungen von Zwangsbedingungen gefunden wurden.

`violationDetected(target)`

Beschreibung: Gibt dem Benutzer Bescheid, dass eine Verletzung von Zwangsbedingungen gefunden wurde und leitet gegebenenfalls Informationen zu dieser Verletzung an den Logger-Agenten weiter. Falls gewünscht, kann er dem VO-Manager mitteilen, dass der Simulationslauf abgebrochen werden soll.
Paramter: Der Sim-Agent, bei dem eine Verletzung von Zwangsbedingungen gefunden wurde.

`startUpViolationDetected(target)`

Beschreibung: Gibt dem Benutzer Bescheid, dass eine Verletzung von Zwangsbedingungen zu Beginn der Simulation gefunden wurde und leitet gegebenenfalls Informationen zu dieser Verletzung an den Logger-Agenten weiter. Falls gewünscht, kann er dem VO-Manager mitteilen, dass der Simulationslauf abgebrochen werden soll.
Paramter: Der Sim-Agent, bei dem eine Verletzung von Zwangsbedingungen gefunden wurde.

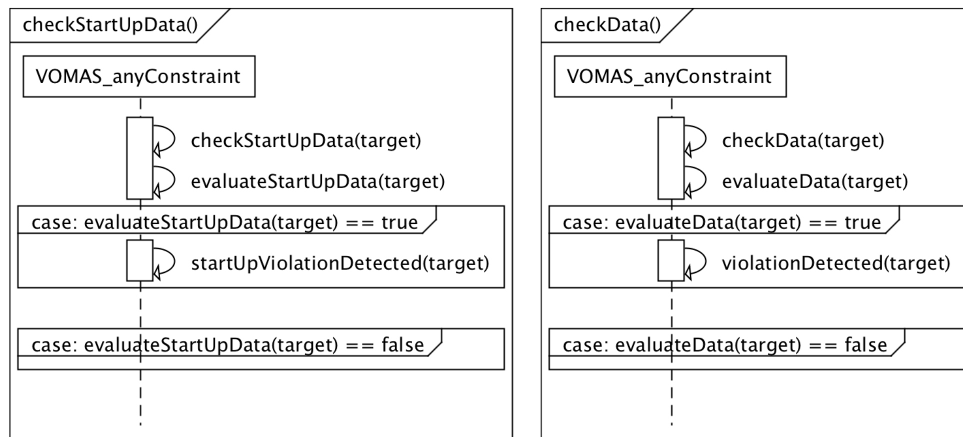

Abbildung 3.22: Interner Ablauf der Methoden *checkStartUpData* und *checkData*

- **Abschließende Anmerkungen**

  Das entwickelte Design wirkt auf den ersten Blick womöglich etwas umständlich. Bei einigen Methoden muss beispielsweise nur mittels einer Schleife eine Menge von Agenten durchlaufen und eine weitere Methode aufgerufen werden. Innerhalb einer Simulationsumgebung kann damit allerdings ein funktionierendes Grundgerüst implementiert werden. Dieses kann dann, bis auf die modellabhängigen Methoden, für andere Modelle ähnlicher Struktur, erneut verwendet werden.

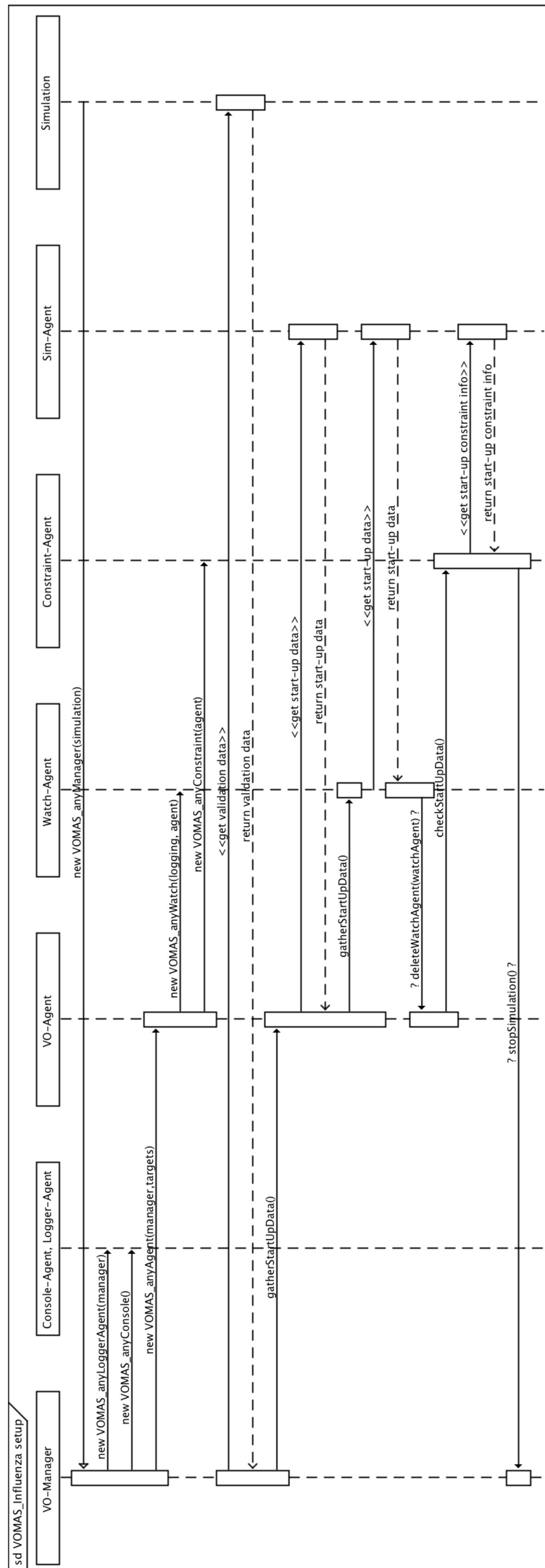

Abbildung 3.23: Sequenzdiagramm der Initialisierung eines VOMAS

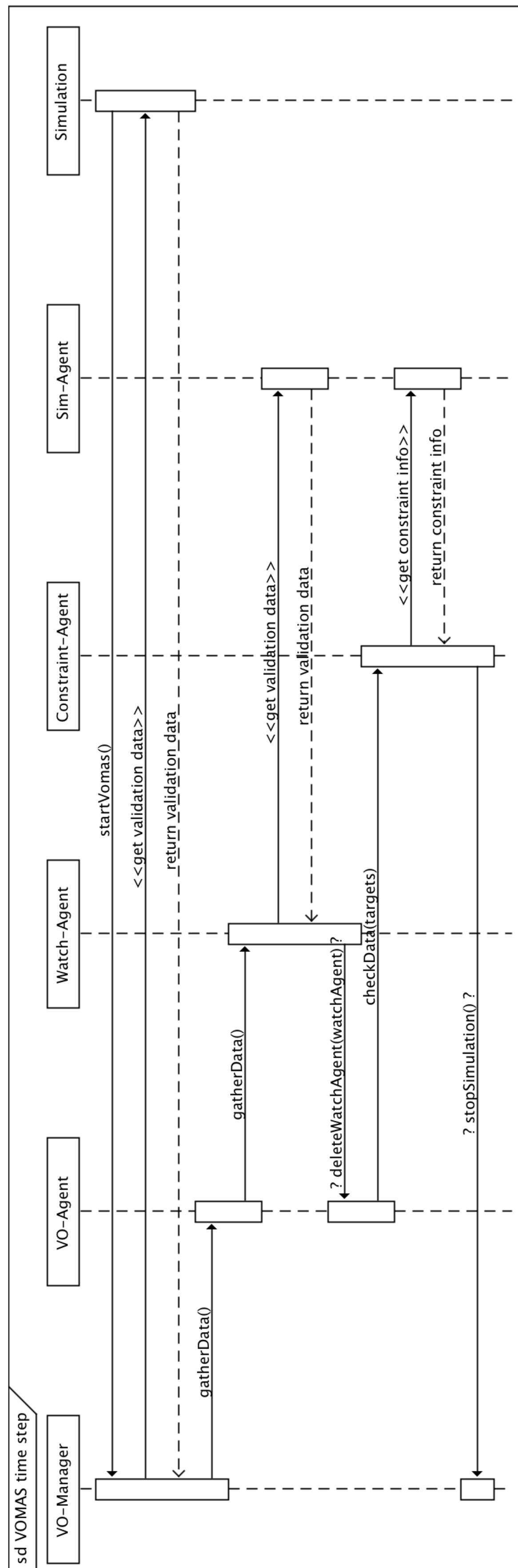

Abbildung 3.24: Sequenzdiagramm der Informationssammlung eines VOMAS

3.4 Konzepte der Parallelität bei einem VOMAS

Wenn man von Parallelität in einem VOMAS spricht, muss unterschieden werden, ob (1) die VO-, Watch- und Constraint-Agenten innerhalb des VOMAS, sobald sie dazu aufgerufen werden, parallel ihren Aufgaben nachgehen, oder (2) das VOMAS für sich genommen parallel zur Simulation läuft. Ersteres ist prinzipiell immer möglich, falls es die Modellierungssoftware erlaubt. Dies kann unter Umständen viel Zeit sparen. Die notwendigen Änderungen, die ein gleichzeitiges Arbeiten der Agenten ermöglichen, sind dank der erarbeiteten Struktur nicht aufwändig. Da sie aber stark von der verwendeten Modellierungssoftware abhängen, werden sie hier nicht weiter angeführt.

Die Frage, ob ein VOMAS parallel zur Simulation laufen kann, muss mit Vorsicht betrachtet werden. Ein VOMAS würde dann einmal initialisiert und der Informationssammlungsprozess auch nur einmal gestartet werden. Den VO-Agenten müsste also auf anderen Wegen mitgeteilt werden, wann sie die ihnen zugeteilten Sim-Agenten zu überprüfen haben. Eine Möglichkeit wäre, den Sim-Agenten zu erlauben, auf Änderungen in ihrem Zustand aufmerksam zu machen. Hierfür müssten sie nicht wissen, um welche Änderungen es sich handelt, sondern nur, dass irgendeine Änderung passiert ist. Ein VO-Agent könnte dann auf diesen Sim-Agenten zugreifen und alle notwendigen Informationen zu der gemeldeten Änderung einholen – Watch-Agenten sammeln Informationen, Constraint-Agenten prüfen auf verbotenes Verhalten. Auf den ersten Blick scheint der Vorteil hierbei zu sein, dass nicht ständig – also nach jeder Iteration des Modells – alle Simulationsagenten, die beobachtet werden sollen, tatsächlich auf Änderungen überprüft werden müssen. Dies kann unter Umständen eine enorme Zeitersparnis bedeuten. Fraglich ist bei diesem Ansatz allerdings, ob die strikt geforderte Trennung zwischen VO-Agenten und Simulationsagenten damit verletzt werden würde. Bei genauerer Betrachtung ergibt sich noch ein weiteres Problem: der Sim-Agent muss entweder (a) wissen, welche Änderungen für den Benutzer von Interesse wären, oder (b) bei jeder Änderung irgendeiner seiner Attribute Alarm schlagen. Letzteres wird vermutlich bei vielen Modellen zu einer großen Anzahl unnötiger Zugriffe der VO-Agenten auf die Sim-Agenten führen – noch mehr sogar als bei einem Zugriff auf alle Sim-Agenten nach jeder Iteration. Ersteres hingegen verlangt, dass der Sim-Agent immer weiß, welche Änderungen oder Informationen für den Benutzer von Interesse sind. Hier muss die Frage gestellt werden, ob das im Sinne der agentenbasierten Modellbildung ist. Agenten sollten einfachen Regeln folgen und nicht weniger, aber auch nicht mehr über ihre Zustände wissen, als für ihr Verhalten notwendig ist.

3.5 VOMAS im Modellierungskreislauf

Während der Entwicklung eines VOMAS stellen sich ganz generelle Fragen, die beantwortet werden müssen, bevor an eine Implementierung gedacht werden kann, wie zum Beispiel:

- Was sind potentielle *watch-values*? Wie oft sollen diese beobachtet werden?
- Wie soll der generierte Output aussehen?

- Auf welche Informationen soll bzw. kann zugriffen werden?
- In welcher Form sollen Informationen gespeichert werden?
- Gibt es *constraints*, also Beschränkungen, die nicht verletzt werden dürfen?
- Wie sehen die Schnittstellen zwischen VO-Agenten und Sim-Agenten aus?

Da ein VOMAS von Beginn des Modellierungsprozesses an parallel mitentwickelt werden sollte, wäre es wünschenswert, die Beantwortung dieser Fragen und damit das Design des VOMAS in den Modellierungskreislauf einbetten zu können. Im Folgenden werden deshalb die einzelnen Phasen des Kreislaufs nochmals unter Berücksichtigung der Entwicklung eines VOMAS betrachtet. Da das Design eines VOMAS stark von dem zu validierenden Modell abhängt, ist es schwierig und womöglich auch nicht ratsam, ein strenges Designverfahren zu entwickeln. Es ist allerdings die Auffassung des Autors, dass ein VOMAS eine anwendungsübergreifende Kernstruktur besitzt. Ein Vorschlag für eine methodische Ausformulierung derselbigen ist im Folgenden beschrieben.[20]

Mit der Formulierung der Forschungsfragen sollten auch bereits erste Überlegungen zur Validierung gemacht werden. Sargent empfiehlt in [Sar10], dass das Modellentwicklungsteam mit den Auftraggebern und Anwendern zu Beginn des Modellierungsprozesses besprechen sollte, welche Art der Validierung vorgenommen werden soll, wie „genau“ die Validierung sein muss und welche Validierungsmethoden auf jeden Fall zum Einsatz kommen sollen. Wir gehen im weiteren Verlauf davon aus, dass ein agentenbasiertes Modell entwickelt werden soll und die Entwicklung eines VOMAS die Anwendung einiger dieser geforderten Methoden ermöglicht bzw. erleichtert. Mit „Beginn des Modellierungsprozesses“ ist hier die **Problem Entity Phase** gemeint. Da der Modellierungskreislauf zyklisch passiert, muss jedes Mal, wenn eine Phase erneut durchschritten wird, natürlich ebenso das VOMAS gegebenenfalls adaptiert oder geändert werden.

Problem Entity Phase: In dieser Phase wird die Grundstruktur des VOMAS entwickelt und generelle Informationen für das VOMAS abgeleitet.

- **Communicated Problem:** In diesem Schritt müssen noch keine Designschritte für ein VOMAS gemacht werden.
- **Formulated Problem:**

  Watch-Agent: Mit der Formulierung der Forschungsfragen muss auch geklärt werden, welche Informationen notwendig sind, um diese zu beantworten. Hieraus lassen sich erste *watch-values* ableiten. Ob diese ständig beobachtet werden müssen, oder erst ab einem gewissen Wert von Interesse sind, muss noch nicht festgelegt werden (die Möglichkeit, diese Option zu nutzen,

[20] Die Nomenklatur aus Abschnitt 2.1 wird übernommen. Jede Phase wird wieder in ihre Unterphasen aufgefächert. Zu jeder Unterphase werden jene Komponenten eines VOMAS genannt, die in diesem Schritt weiterentwickelt werden können.

sollte aber im Hinterkopf behalten und mitentwickelt werden). Es geht lediglich darum, zu identifizieren, welche Informationen wichtig sein könnten. Dies legt auch fest, welche Informationen über die Schnittstelle zwischen VO-Agent und Sim-Agent verfügbar gemacht werden müssen.

Logger-Agent: In welcher Form sollen Informationen zum Modell zur Verfügung stehen und sollen Informationen überhaupt gespeichert werden? Einfachere Modelle verlangen möglicherweise nur nach einer Face Validation im Sinne einer visuellen Darstellung des Modellverlaufes. Womöglich sind keinerlei Daten vorhanden, um Vergleiche mit dem realen System anzustellen, oder es ist nur das grobe Modellverhalten von Interesse. Hier reicht eine einfache Ausgabe von Informationen über den Console-Agent aus, der auf vermeintliche Verletzungen von Modellannahmen aufmerksam macht. Sollen allerdings Informationen für spätere Analysen dauerhaft gespeichert werden, wird ein Logger-Agent verwendet.

- **Proposed Solution Technique:** Es wird davon ausgegangen, dass ein agentenbasierter Ansatz gewählt wurde. Für andere Modellierungstechniken ist VOMAS nicht geeignet.
- **System Knowledge and Objectives:**

Constraint-Agent: Mit Hilfe von Detailwissen über das System und dem Ableiten von Modellannahmen werden gleichzeitig auch Zustände definiert, die nicht eintreten sollen. Daraus können *constraints* abgeleitet werden. Welche Informationen zur Überprüfung der constraints benötigt werden, werden identifziert und der Menge an Informationen, die über die Sim-Agent Schnittstelle zur Verfügung stehen muss, hinzugefügt.

Conceptual Model Phase: Es werden vor allem Fragen zum Zugriff des VOMAS auf die Simulation beantwortet (siehe Abschnitt 3.3.1) und der Logger-Agent wird designed.

- **Conceptual Model:**

VO-Agenten: Mit der Entwicklung eines konzeptuellen Modells muss nun überlegt werden, in welcher Verbindung die VO-Agenten mit den Sim-Agenten stehen. Die VO-Agenten dürfen die Sim-Agenten nicht beeinflussen, da dies eine Änderung des Simulationsverhalten zur Folge haben könnte. Die VO-Agenten müssen für die Sim-Agenten also in jedem Sinne unsichtbar sein. Dies muss auch dann der Fall sein, wenn notwendige Informationen nur dadurch gesammelt werden können, dass VO-Agenten tatsächlich an der Simulation teilnehmen – wenn beispielsweise die gewünschte Information nur aus der Simulationsumgebung extrahiert werden kann und kein Attribut der Sim-Agenten an sich ist.

VO-Manager: Die Schnittstellen zwischen dem VOMAS und der Simulation werden definiert und es wird erarbeitet, wie oft das VOMAS innerhalb eines Simulationslaufes zum Einsatz kommen soll – wie oft VOMAS-Agenten also ihren Aufgaben nachgehen sollen.

Log Entry: Zu diesem Zeitpunkt sollte festgelegt werden, in welcher Form Information zur Verfügung steht und wie diese von den Watch-Agenten dem Logger-Agenten weitergeleitet werden soll. Die Beantwortung dieser Fragen definiert implizit das Design der Log Entries.

Logger-Agent: Es wird festgelegt, in welchem Medium der Output gespeichert werden soll. Handelt es sich bei der zu speichernden Information um relativ simple Daten (Anzahl der infizierten Agenten in einem Epidemiemodell, Länge des Ampelrückstaus in einem Verkehrsmodell), kann diese beispielsweise in einer Excel-Datei gespeichert werden. Müssen große Mengen an komplexen Daten gesammelt werden, empfiehlt es sich, die Vorteile einer Datenbank zu nutzen. Das Design dieser Datenbank sollte, bis auf einzelne konkrete Datentypen, in diesem Schritt zur Gänze möglich sein.

- **Communicative Model:** Neben dem eigentlichen Modell, sollte auch das VOMAS sauber ausformuliert werden (siehe Abschnitt 3.5.1). Alle constraints und watch-values sollten festgehalten und nochmal überprüft werden. Nachträgliche Änderungen am Modell, da womöglich weitere Daten für die Validierung notwendig sind, können sich sonst als äußerst umständlich erweisen. Das Design des Logger-Agenten sollte abgesegnet werden, ebenso ein mögliches Datenbankdesign.

Computerized Model Phase: Technische Details werden besprochen und das VOMAS schlussendlich parallel zum Modell entwickelt.

- **Programmed Model:** Bei der Wahl des Simuliationswerkzeugs – also ob eine Programmiersprache oder eine Simulationssoftware gewählt wird – sollten sowohl das Modell als auch das VOMAS berücksichtig werden. Wird eine Datenbank verwendet, muss Rücksicht darauf genommen werden, dass das Simulationswerkzeug mit dieser kompatibel ist. Falls VO-Agenten an der Simulation teilnehmen sollen, können diese tatsächlich „unsichtbar“ für die Sim-Agenten sein? Kann ich die gewünschte Taktung realisieren? Muss ein Sinn von Parallelität implementiert werden? Das VOMAS und das Modell müssen in gleicher Weise berücksichtigt werden. Wurde ein Simulationswerkzeug gewählt, das die Implementierung des Modells erleichtert, aber keine zufriedenstellende Validierung zulässt, hat das Modell keine praktische Relevanz.
- **Experimental Model:** In diesem Schritt kann die Flexibilität eines VOMAS zur Gänze ausgenutzt werden. Vor jedem Experiment oder einzelnen Simulationslauf, kann nun entschieden werden, welche Attribute beobachtet werden sollen, ob diese ständig mitgeschrieben werden müssen oder erst ab einem bestimmten Zeitpunkt von Interesse sind und welche Simulationsagenten beobachtet werden sollen. All diese Spezifikationen werden in einem Paramter File gespeichert, auf welches die VOMAS-Agenten Zugriff haben. Diesbezüglich kann eine grafische Benutzeroberfläche die Handhabung, besonders für Modellfremde, vereinfachen.

- **Simulation Results:** Jegliche Information, die vom Logger-Agent gespeichert wurde, kann nun analysiert werden.

Kombiniert man die Informationen aus diesem Abschnitt mit jenen aus Abschnitt 3.2, gelangt man zu folgender Erkenntnis: ein VOMAS wird hauptsächlich in der **Problem Entity Phase** und der **Conceptual Model Phase** entworfen und ist vor allem für die Feststellung der **Operational Validity** hilfreich.

3.5.1 Dokumentation eines VOMAS

Ebenso wie die Entwicklung und die endgültige Struktur des Modells dokumentiert werden sollte, so sollte auch die Struktur des VOMAS in vernünftiger Form festgehalten werden. Besondere Aufmerksamkeit sollte den *watch-values*, *constraints*, *startup-values* und *selektiven Attributen der Simulationsagenten* gelten. Im Folgenden werden diese Begriffe näher erklärt und ein Vorschlag für deren Dokumentation präsentiert. Die entwickelten „Formulare“ können im Laufe der Modellentwicklung in verschiedenen Phasen des Modellierungskreislaufes ausgefüllt werden und sind außerdem bei der finalen Implementierung des VOMAS hilfreich.

Zunächst werden zwei Begriffe erklärt:

- **Simulationsattribute:** Im Kontext des restlichen Abschnitts, sind damit Informationen gemeint, die die Simulation im Großen betreffen. Diese Informationen sind den Simulationsagenten nicht bewusst und sie haben demnach auch keinen Zugriff darauf.
- **Agentenattribute:** Im Kontext des restlichen Abschnitts, sind damit nicht nur agenteneigenen Attribute gemeint, sondern auch Informationen auf die der Simulationsagent außerdem noch Zugriff hat.

Watch-values: Gibt es Änderungen im Modell oder bei den Simulationsagenten die zu Zuständen führen, die validierungsrelevant sind, werden diese zu *watch-values* erklärt und einzeln in folgender Form dokumentiert (siehe Tabelle 3.1):

A Eine Beschreibung des Zustandes, der validierungsrelevant ist.

B Eine Liste von Attributen, die beobachtet werden müssen. Zusätzlich werden deren Datentyp festgehalten und notiert ob diese Attribute über die Simulationsagenten oder die Simulation erreichbar sind.

C Wie kann festgestellt werden, dass es zum gefragten Zustand gekommen ist?

D Welche Informationen sollen an den Logger-Agenten gesendet werden und sind diese über die Simulationsagenten oder die Simulation erreichbar?

E Was ist die minimale „Zeit“ die vergehen muss, bis es zum Eintreten des Zusatndes kommen kann.

F Ist es irgendwann nicht mehr notwendig, auf das Eintreten des Zustandes hin zu beobachten?

	watch-value		
A	Beschreibung des watch-value		
B	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
C	Wann ist von Interesse?		
D	Beschreibung der logging Attribute		
	Attribut	Datentyp	Zugriff
E	Minimaler Zeitschritt		
F	Löschung des Agenten		

Tabelle 3.1: Watch-value Formular

Aus **B** und **D** werden die Schnittstelleninformationen erweitert. Ist eines der Attribute ein Agentenattribut, wird ein Watch-Agent erstellt. Notwendige Attribute der Simulation werden dann über die *getManager*-Methode erfragt. Handelt es sich nur um Simulationsattribute, wird die Aufgabe vom VO-Manager übernommen.

Wird ein Watch-Agent erstellt, ergibt sich aus **D** seine Methode *generateLoggingData(target)*.

Wird ein Watch-Agent erstellt, ergibt sich aus **C** die Methode *evaluateData(target)*. Übernimmt die Aufgabe der VO-Manager, wird damit seine Methode *evaluateData* erweitert.

Der minimale **E**-Wert aller watch-values und constraints ergibt die minimal notwendige Taktung des VOMAS. Seltener darf das VOMAS nicht zum Einsatz kommen.

Wird ein Watch-Agent erstellt, ergibt sich aus **F**, ob die Methode *deleteTarget* verwendet werden kann.

Constraints: Gibt es Gründe für einen möglichen Abbruch der Simulation, wird ein constraint definiert. Dabei kann es sich um eine Verletzung von Zwangsbedingungen handeln, oder um einen Modellzustand, ab dem der weitere Verlauf nicht mehr validierungsrelevant ist. Constraints werden in folgender Form dokumentiert (siehe Tabelle 3.2):

A Der Grund warum die Simulation abgebrochen werden könnte.

B Eine Liste von Attributen, die beobachtet werden müssen. Zusätzlich werden deren Datentyp festgehalten und notiert ob diese Attribute über die Simulationsagenten oder die Simulation erreichbar sind.

C Wie kann festgestellt werden, dass es zu dem unerwünschten Modellverhalten gekommen ist?

D Welche Informationen sollen an den Logger-Agenten gesendet werden und sind diese über die Simulationsagenten oder die Simulation erreichbar?

E Was ist die minimale „Zeit" die vergehen muss, bis es zum Auftreten des Grund kommen kann.

F Welche Nachricht soll dem Benutzer angezeigt werden?

G Soll die Simulation gestoppt werden?

Aus **B** und **D** werden die Schnittstelleninformationen erweitert. Ist eines der Attribute ein Agentenattribut, wird ein Constraint-Agent erstellt. Notwendige Attribute der Simulation werden dann über die *getManager*-Methode erfragt. Handelt es sich nur um Simulationsattribute, wird die Aufgabe vom VO-Manager übernommen.

Wird ein Constraint-Agent erstellt, ergibt sich aus **C** die Methode *evaluateData(target)*. Übernimmt die Aufgabe der VO-Manager, wird damit seine Methode *evaluateData* erweitert.

Der minimale **E**-Wert aller watch-values und constraints ergibt die minimal notwendige Taktung. Seltener darf das VOMAS also nicht zum Einsatz kommen.

Wird ein Constraint-Agent erstellt ergibt sich aus **D**, **F** und **G** die Methode *violationDetected(target)*.

Startup-Informationen: Gibt es validierungsrelevante Informationen zu den Simulationsagenten oder der Simulation, die sich während eines Simulationslaufes nicht mehr ändern können werden diese als *startup*-Informationen bezeichnet. Diese Informationen müssen nur einmal eingefordert werden sind in folgender Form zu dokumentieren:

A Eine Liste von Informationen die gesammelt werden müssen und ob diese über die Simulationsagenten oder die Simulation erreichbar sind.

Aus **A** werden die Schnittstelleninformationen erweitert und die *evaluateStartUpData(target)*-Methoden des VO-Agenten und die *evaluateStartUpData(target)*-Methode des VO-Managers definiert.

	constraint		
A	Beschreibung des constraint		
B	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
C	Anzeichen für den constraint		
D	Beschreibung der logging Attribute		
	Attribut	Datentyp	Zugriff
E	Minimaler Zeitschritt		
F	Nachricht an den Benutzer		
F	Abbruch der Simulation. Wenn ja, warum?		

Tabelle 3.2: Constraint Formular

Selektive Attribute: Gibt es Eigenschaften der Simulationsagenten, die darüber entscheiden können, ob ihr Verhalten für den SME von Interesse ist, werden diese als *selektive* Informationen bezeichnet und in einer einfachen Liste notiert.

In Kapitel 4 wird diese Dokumentationsmethode angewandt und veranschaulicht.

KAPITEL 4

Praktische Umsetzung des VOMAS-Designs am Beispiel eines Influenza-Modells

Bisher gab es wenig Berichte über die tatsächliche Anwendung eines VOMAS. Muaz et al. verwenden in [NSHK10] den VOMAS-Ansatz für die Validierung eines Waldbrandmodells, geben aber wenig Aufschluss über den tatsächlichen Entwicklungsprozess beim Design des VOMAS. Es müssen also weitere Erfahrungen über die Validierung mit VOMAS gesammelt und getestet werden, um zu evaluieren, für welche Modelle sich ein VOMAS als nützlich erweisen könnte. Mit diesem Kapitel wird ein erster Schritt in diese Richtung getan.

In Abschnitt 4.1 wird das Influenzamodell, an dem der VOMAS-Ansatz getestet wurde, beschrieben. Es sei hier nochmals daran erinnert, dass ein VOMAS gemeinsam mit dem Modell entwickelt und nicht nachträglich implementiert werden sollte. Abschnitt 4.2 befasst sich mit der Umsetzung des VOMAS-Designs aus Kapitel 3 in die Praxis. Es wird soweit wie möglich versucht, nach dem Muster von Abschnitt 3.5 und Abschnitt 3.5.1 vorzugehen – auch wenn mit einem bereits fertig implementierten Modell gearbeitet wird. Abschließend werden in Abschnitt 4.3 einige Ergebnisse präsentiert, die aus Informationen entstanden sind, deren Erhebung erst durch das VOMAS ermöglicht wurden.

4.1 Das Influenzamodell

Die Influenza, auch „echte Grippe“ oder Virusgrippe genannt, tritt in unseren Breitengraden jährlich für ein paar Wochen in den Wintermonaten auf. Unter Umständen kann es zu einem bevölkerungsweiten Ausbruch der Krankheit kommen – man spricht dann von einer Epidemie, welche eine große Belastung für das Gesundheitswesen darstellt und hohe Sozialkosten verursachen kann. Impfungen oder gesundheitsaufklärende Maßnahmen – z.B.

bessere Hygiene in den Wintermonaten – können dazu beitragen, einen solchen Ausbruch zu verhindern.

4.1.1 Modellbeschreibung

Die folgende Modellbeschreibung liefert nur einen sehr groben Überblick über das Influenzamodell, der für das Verständnis der kommenden Abschnitte aber ausreicht. Detaillierte Informationen sind in [Pic13] und [FM12] zu finden.

Beim betrachteten Influenzamodell handelt es sich um ein modulares agentenbasiertes Modell, das aus folgenden 4 Teilmodulen besteht: Bevölkerungsmodul, Kontaktmodul, Krankheitsmodul und Protokollmodul. Ein Zeitschritt im Simulationslauf entspricht dem Vergehen eines Tages. In jedem Zeitschritt führt eine Person dieselben Aktionen durch.

Bevölkerungsmodul: Das Bevölkerungsmodul ist für die Erstellung der Bevölkerung zuständig. Jede Person im Modell hat ein Alter, ein Geschlecht und gehört einem Typ von Person an (Kleinkind, Schulkind, Erwachsener erwerbstätig, Erwachsener arbeitslos und Pensionist). Desweiteren besitzt jede Person einen Gesundheitszustand und ein Immumsystem, das vom Krankheitsmodul gesteuert wird (siehe Abbildung 4.1).

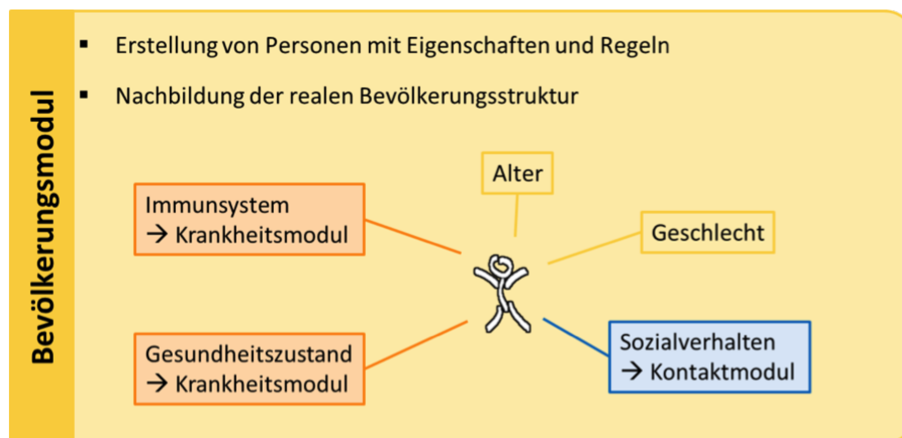

Abbildung 4.1: Bevölkerungsmodul aus [FM12]

Kontaktmodul: Das Kontaktmodul ist verantwortlich für die Erstellung von Orten und die Verwaltung der Kontakte an eben diesen. In dem Modell werden 4 Ortstypen unterschieden:

- Schule
- Arbeitsstätte
- Haushalt
- Freizeit

Mit Ausnahme des Ortstyps Freizeit gibt es von jedem dieser Typen mehrere Elemente. Einer Person werden bei der Initialisierung je nach Personentyp Orte zugewiesen, die sie bis zum Ende der Simulation täglich besucht (siehe Abbildung 4.2). Außnahme hierfür ist, wenn die Person aus Krankheitsgründen zu Hause bleibt. Besucht eine Person einen ihr zugewiesenen Ort, kommt sie dort mit anderen Personen in Kontakt. Je nach Ortstyp kommt der Kontakt mit einer anderen Person **zufällig** (Schule, Arbeitsstätte), **altersbasiert** (Freizeit) oder **total**[1] (Haushalt) zustande. In Kontakt treten immer Paare von Personen. Ist eine von beiden infektiös, kann es zu einer Krankheitsübertragung kommen. Die Wahrscheinlichkeit dafür und der weitere Verlauf einer möglichen Ansteckung wird im Krankheitsmodul geregelt. Außerdem können Orte mittels eines Faktors (*intensityFactor*) eine mögliche Krankheitsübertragung fördern.

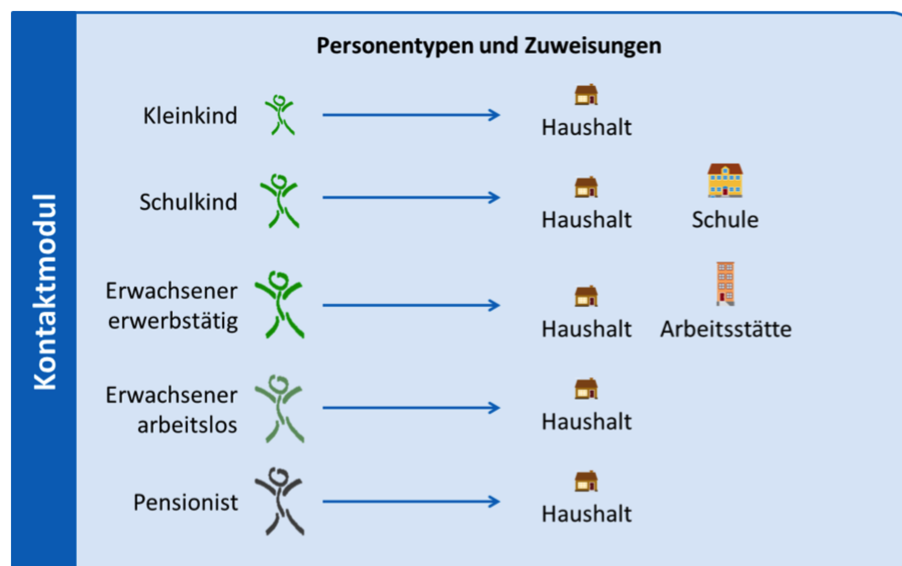

Abbildung 4.2: Kontaktmodul aus [FM12]

Krankheitsmodul: Diese Modul regelt alle Gesundheitszustände, steuert den Krankheitsverlauf, den Infektionsprozess und errechnet, ob es nach einem Kontakt zu einer Infektion kommt. Folgende Attribute zur Gesundheit einer Person werden hier gespeichert:

- infektiös
- geimpft
- natürlich immun
- milde Symptome
- schwere Symptome
- keine Symptome

[1] Jede Person an diesem Ort hat mit jeder anderen Person Kontakt.

- bleibt zu Hause
- latent infiziert

Der Verlauf der Krankheit ist in Abbildung 4.3 dargestellt. Die Simulation wird mit einer voreingestellten Anzahl an geimpften, natürlich immunen sowie bereits infizierten Personen gestartet.

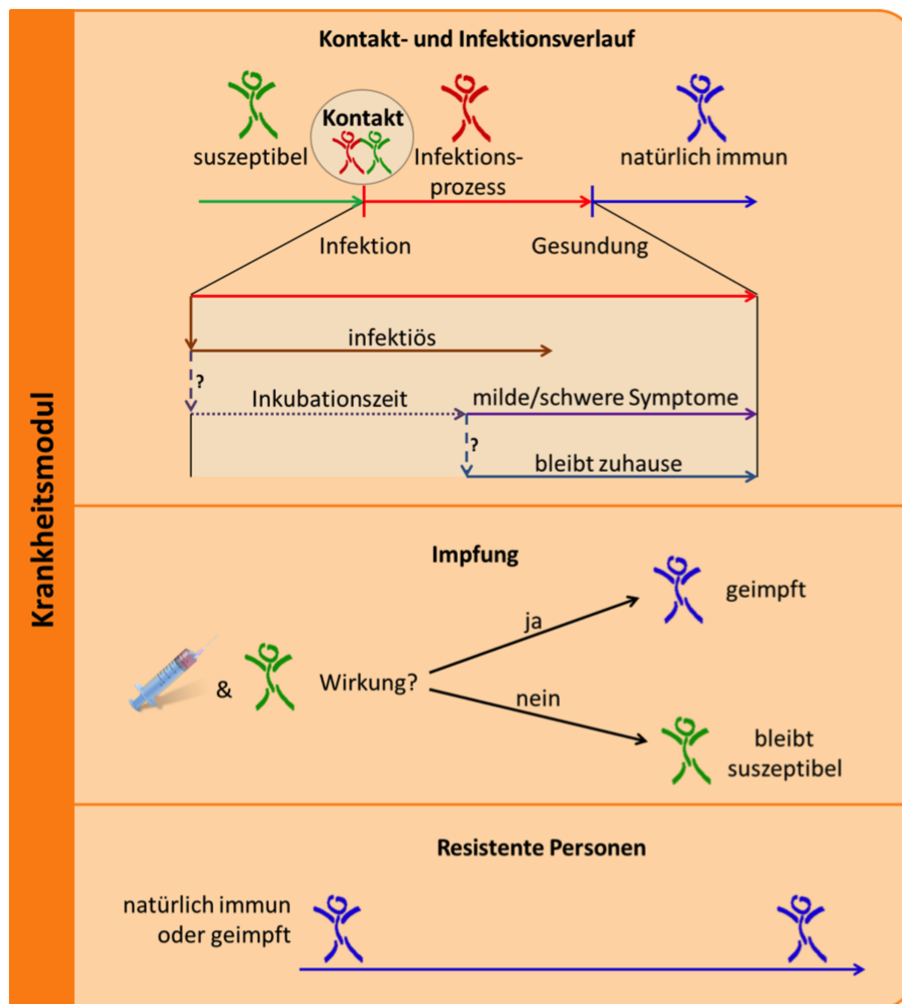

Abbildung 4.3: Krankheitsmodul aus [FM12]

Protokollmodul: Das Protokollmodul wird in Abschnitt 4.2.2 separat erwähnt und wird durch das implementierte VOMAS ersetzt.

4.1.2 Probleme bei der Validierung

Das Influenzamodell verwendet unterschiedlichste Parameter. Parameterstudien wurden durchgeführt um *„die Einflüsse von Parametern auf die Ergebnisse der Simulation zu untersuchen“*.[2] Die detaillierten Ergebnisse zu dieser Parameterstudien sind in [FM12] und [Pic13] angeführt. Parameter wie die Altersverteilung der Bevölkerung oder die durchschnittlichen Kontakte in Haushalt, Arbeitsstätte und Schule, können aus Volkszählungen, Statistiken und Studien entnommen bzw. errechnet werden. Parameter wie die Infektionswahrscheinlichkeit hingegen sind formal sehr schwer zu bestimmen und wurden deshalb versucht, mit Hilfe von bekannten Daten der Influenza-Saison 2006/07 durch Kalibrierung zu bestimmen. Hierfür wurde die Infektionswahrscheinlichkeit variiert und die Ergebnisse mit den Daten aus der Saison 2006/07 verglichen (siehe Abbildung 4.4 und Abbildung 4.5)[3].

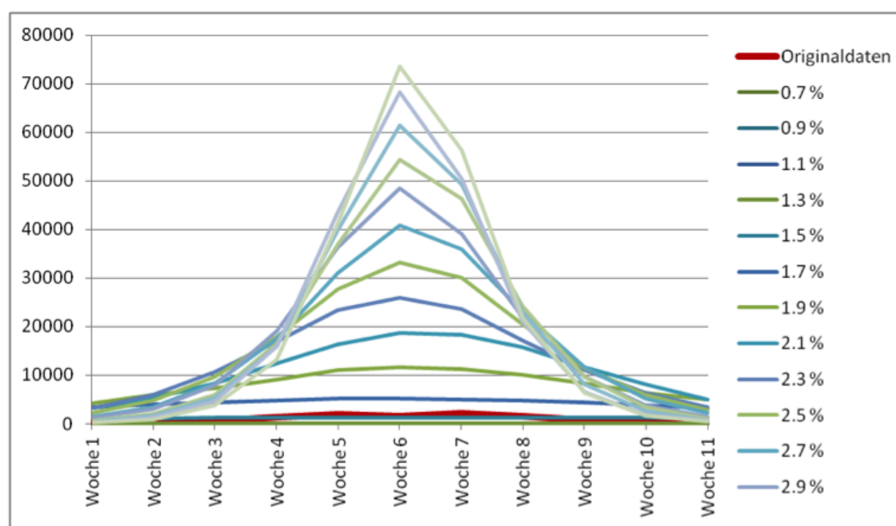

Abbildung 4.4: Kalibrierung der Infektionswahrscheinlichkeit aus [Pic13]

Pichler interpretiert die Ergebnisse folgendermaßen[4]:

„Das Problem bei einer Infektionswahrscheinlichkeit in diesem Bereich ist, dass

- *eine zu hohe Infizierten-Zahl erreicht wird. In der Abbildung 4.5 verdeutlicht dies [die] blaue Kurve mit 1.7%iger Infektionswahrscheinlichkeit.*
- *Neuinfizierten-Zahlen erzeugt werden, die keinem epidemieähnlichen Verhalten folgen. Ein epidemieähnliches Verhalten wären mehr infizierte Personen in der Mitte der Epidemie; am Anfang und am Ende weniger infizierte Personen.*

[2] [Pic13] p.60.
[3] Beide Abbildungen zeigen die Anzahl der neu Infizierten pro Woche.
[4] [Pic13] p.70 - 71.

- *die Anzahl an neu infizierten Personen so niedrig ist, dass es nie zu einem Ausbruch der Epidemie kommt.*

Das heißt, die Daten sind alleine durch die Infektionswahrscheinlichkeit nicht reproduzierbar.“

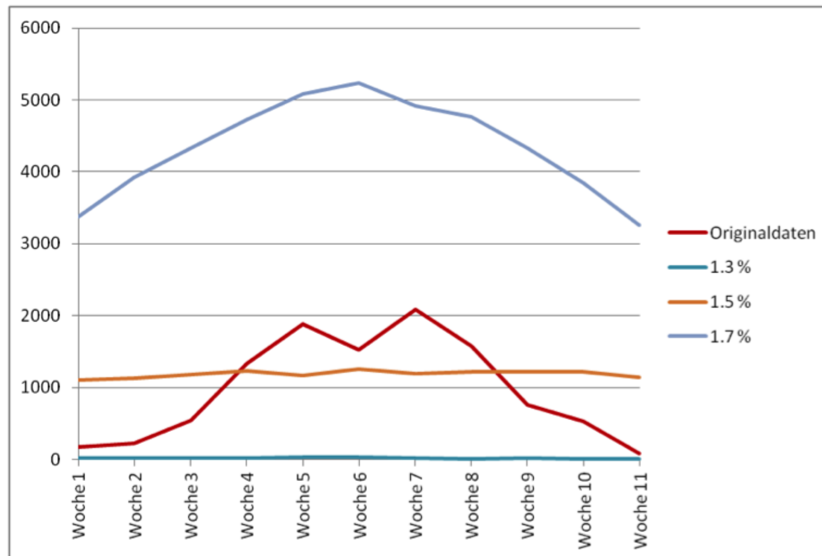

Abbildung 4.5: Kalibrierung der Infektionswahrscheinlichkeit (selektierte Auswahl) aus [Pic13]

In weiteren Experimenten wurde versucht, die Ergebnisse „nur“ qualitativ wiederzugeben, bzw. zusätzlich zur Infektionswahrscheinlichkeit auch die Anzahl der natürlich immunen Person variabel zu wählen, um die Originaldaten möglichst genau zu reproduzieren (die Genauigkeit der reproduzierten Daten wurde anhand einer Bewertungsfunktion bestimmt). Keiner der Versuche erbrachte allerdings zufrieden stellende Resultate, da entweder der Anteil der natürlich immunen Personen im Modell unrealistisch hoch gewählt werden musste (teilweise liegt dieser Wert bei 79%), oder die Anzahl der Infizierten im Zuge der Grippewelle zu hoch ist. Pichler kommt zum Schluss, dass *„eine Reproduktion der Originaldaten nicht möglich ist“*[5] und womöglich eine genauere Modellierung notwendig ist. Weiters schreibt er an derselben Stelle: *„Da das Modell eine Größe erreicht hat, bei der es immer schwieriger wird den Überblick zu behalten, steht die Entwicklung von Methoden im Vordergrund, damit eine Validierung leichter möglich wird. Am sinnvollsten wäre es, ein System zu entwerfen, dass auf den VOMAS-Ansatz aufbaut. Optimal wäre dabei, wenn der Experte / die Expertin bei Anwendung dieses Validierungssystems genau einstellen kann, was er / sie wissen möchte. Im Idealfall wäre es wünschenswert, wenn das Tool soweit entwickelt wird, dass der Experte wirklich das Programm selbst steuern kann und genau sagen kann, was er / sie wissen möchte.“*

[5] [Pic13] p.87.

4.2 Umsetzung des VOMAS-Designs für das Influenzamodell

Die vorgeschlagene Methode der Entwicklung eines VOMAS, wie sie in Abschnitt 3.5 präsentiert wurde, wird in Abschnitt 4.2.1 am Beispiel des Influenzamodells vorgeführt. Das entwickelte VOMAS arbeitet mit einer Datenbank – genauer gesagt, speichert der Logger-Agent alle Informationen in einer Datenbank ab. Das Design dieser Datenbank wird in Abschnitt 4.2.2 angeführt. Da das VOMAS erst im Nachhinein entworfen wurde, mussten notwendige Änderungen am Modell vorgenommen werden. Diese werden kurz in Abschnitt 4.2.3 beschrieben. Diverse Probleme und Erkenntnisse, die vor und während der Implementierung auftraten, werden in Abschnitt 4.2.4 diskutiert.

4.2.1 Struktur des VOMAS

Die Validierung mit VOMAS setzt die Zusammenarbeit mit einem Subject Matter Expert (SME) voraus. Diese Rolle wurde von Kollegen übernommen, mit denen gemeinsam validierungsrelevante Informationen definiert wurden. Wie in Abschnitt 3.5 dargelegt, werden die einzelnen Phasen der Modellierung durchlaufen und das VOMAS wird sukzessive ausgebaut.

Problem Entity Phase

- **Formulated Problem:** In einem Epidemiologiemodell – wie es das Influenzamodell ist – steht die Übertragung der Krankheit im Zentrum. Für die Validierung von Interesse ist beispielsweise, welche Bevölkerungsgruppen einander anstecken und an welchen Orten die meisten Ansteckungen stattfinden. Diese Informationen können dann von Experten auf Plausabilität bewertet werden. Deshalb sollen Informationen zu allen Ansteckungen gesammelt werden. Ein watch-value mit der Bezeichnung **Infection** wird definiert (siehe Tabelle 4.1).

  Weiters von Interesse könnte auch der Krankheitsverlauf infizierter Personen sein. Allerdings sind die Paramter, die den Krankheitsverlauf bestimmen, für jede Altersgruppe gleich gewählt. Um das Prinzip der watch-values zu demonstrieren, sollen aber trotzdem Informationen dazu gesammelt werden (siehe Tabelle 4.2). Der NonContactTriggeredStateChange-value hätte auch auf alle einzelnen Statuswechsel aufgeteilt werden können – also ein watch-value, der nur auf Entwicklung von Symptomen überprüft und ein watch-value, der nur auf das Ende der Genesung überprüft. Es wurde aber entschieden, alle diese Aufgaben von einem Agenten erledigen zu lassen.

  Die Logging-Informationen werden nun der Reihe nach auf Veränderbarkeit untersucht. Jene Attribute, die sich im Verlauf der Simulation nicht mehr ändern, werden in das *start-up* Formular eingetragen (siehe Tabelle 4.3). Außerdem werden dort Informationen notiert, die ebenso noch von Interesse sein könnten.

Alle Infomationen müssen für eine spätere Analyse dauerhaft gespeichert werden. Es wird also ein Logger-Agent benötigt.

	Infection		
A	Beschreibung des watch-value		
	Die Infektion einer Person.		
B	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
C	Wann ist der watch-value von Interesse?		
	Sobald die Person infiziert wurde.		
D	Beschreibung der logging Attribute		
	Attribut	Datentyp	Zugriff
	Tag der Infektion		
	Ort der Infektion		
	Infizierte Person		
	Infizierende Person		
	Latente Infektion		
E	Minimaler Zeitschritt		
	Eine Person kann an jedem Tag infiziert werden.		
F	Löschung des Agenten		
	Ein Person kann nicht an mehreren Tagen infiziert werden. Nach der Infektion kann der Agent aus der Target-Liste gelöscht werden.		

Tabelle 4.1: Infection Formular

	NonContactTriggeredStateChange		
A	Beschreibung des watch-value		
	Der Krankheitsverlauf einer Person.		
B	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
C	Wann ist der watch-value von Interesse?		
	Sobald die Person infiziert wurde.		
D	Beschreibung der logging Attribute		
	Attribut	Datentyp	Zugriff
	Person, der die Statusänderung durchläuft		
	Tag der Statusänderung		
	Typ der Statusänderung		
	Ob die Person zu Hause bleibt (nur bei der Entwicklung von Symptomen)		
	Wirkung einer möglichen Impfung		
E	Minimaler Zeitschritt		
	Eine Person kann an jedem Tag infiziert werden.		
F	Löschung des Agenten		
	Ein Person durchläuft den Verlauf der Krankheit immer nur einmal.		

Tabelle 4.2: NonContactTriggeredStateChange Formular

	start-up Informationen		
A	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
	Name jeder Person (ID-Nummer)		
	Alter jeder Person		
	Geschlecht jeder Person		
	Tätigkeit jeder Person		
	Name jedes Ortes (ID-Nummer)		
	Typ jedes Ortes		
	Liste der Orte, der eine Person zugeteilt ist		

Tabelle 4.3: Start-up Informationen

- **System Knowledge and Objectives:** Es wird davon ausgegangen, dass Personen nach einer Infektion und der darauffolgenden Genesung natürlich immun sind, also nicht noch einmal infiziert werden können. Um sicherzugehen, dass bei der Implementierung diesbezüglich keine Fehler passieren, wird ein constraint dafür definiert (siehe Tabelle 4.4).

  Weitere mögliche constraints könnten die Zuteilung der Orte an die Agenten überprüfen. So muss eine Person vom Typ „Worker“ sowohl einem Haushalt, als auch einem Arbeitsplatz zugeteilt werden.

Conceptual Model Phase

- **Conceptual Model:** Das Influenzamodell ist link-based (siehe Abschnitt 3.3.1 – **Link Based**) und die VO-Agenten müssen die Simulationsagenten nur beobachten, also nicht direkt an der Simulation teilnehmen. Die Taktung des VOMAS ergibt sich aus dem minimalen Wert der Kategorie **E** und der Taktung der Simulation an sich. Das VOMAS muss also täglich gestartet werden. Aus den Informationen der Kategorie **D** der watch-values und constraints, sowie den Informationen des start-up Formulars, werden die Log Entries entworfen. Für jeden watch-value und constraint wird eine Unterklasse der LogEntry Klasse entworfen. Selektive Attribute der Personen werden definiert. Die Verwendung einer Datenbank wird beschlossen und deren Design entworfen (siehe Abschnitt 4.2.2). Aus dem konzeptuellen Modell kann in dieser Phase auch abgelesen werden, über welche Schnittstellen die notwendigen Informationen zur Verfügung stehen. Die Kategorie **B** kann ausgefüllt und damit Kategorie **C** konkreter formuliert werden (siehe Tabelle 4.6, 4.5, 4.7 und 4.8).

	IllegalStatusChange		
A	Beschreibung des constraint		
	Eine genesene Person darf nicht erneut angesteckt werden.		
B	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
C	Anzeichen für den constraint		
	Die Person wird nach einer Genesung erneut infiziert.		
D	Beschreibung der logging Attribute		
	Attribut	Datentyp	Zugriff
	Person, die erneut infiziert wurde		
	Tag der erneuten Infektion		
	Ort der erneuten Infektion		
	Person, die die erneute Infektion verursacht hat		
E	Minimaler Zeitschritt		
	Eine Person kann jeden Tag infiziert werden.		
F	Nachricht an den Benutzer		
	Eine Person wurde illegalerweise nach einer Genesung erneut infiziert.		
F	Abbruch der Simulation. Wenn ja, warum?		
	Ja. Falsches Modellverhalten		

Tabelle 4.4: IllegalStatusChange Formular

	NonContactTriggeredStateChange		
A	Beschreibung des watch-value		
	Der Krankheitsverlauf einer Person.		
B	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
	shedding	**boolean**	**Sim-Agent**
	symptoms	**boolean**	**Sim-Agent**
	severe symptoms	**boolean**	**Sim-Agent**
	stays at home	**boolean**	**Sim-Agent**
	naturally immune	**boolean**	**Sim-Agent**
	vaccinated	**boolean**	**Sim-Agent**
C	Wann ist der watch-value von Interesse?		
	Sobald die Person infiziert wurde, und nach jeder Statusänderung.		
D	Beschreibung der logging Attribute		
	Attribut	Datentyp	Zugriff
	Person, die die Statusänderung durchläuft	**int**	**Sim-Agent**
	Tag der Statusänderung	**int**	**Simulation**
	Typ der Statusänderung	**String**	**Sim-Agent**
	Ob die Person zu Hause bleibt (nur bei der Entwicklung von Symptomen) = **stays at home**	**boolean**	**Sim-Agent**
	Wirkung einer möglichen Impfung	**boolean**	**Sim-Agent**
E	Minimaler Zeitschritt		
	Eine Person kann an jedem Tag infiziert werden.		
F	Löschung des Agenten		
	Ein Person durchläuft den Verlauf der Krankheit immer nur einmal.		

Tabelle 4.5: NonContactTriggeredStateChange Formular

	Infection		
A	Beschreibung des watch-value		
	Die Infektion einer Person.		
B	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
	shedding	**boolean**	**Sim-Agent**
C	Wann ist der watch-value von Interesse?		
	Sobald die Person infiziert wurde. **shedding = true**		
D	Beschreibung der logging Attribute		
	Attribut	Datentyp	Zugriff
	Tag der Infektion	**int**	**Simulation**
	Ort der Infektion	**int**	**Sim-Agent**
	Infizierte Person	**int**	**Sim-Agent**
	Infizierende Person	**int**	**Sim-Agent**
	Latente Infektion	**boolean**	**Sim-Agent**
E	Minimaler Zeitschritt		
	Eine Person kann an jedem Tag infiziert werden.		
F	Löschung des Agenten		
	Ein Person kann nicht an mehreren Tagen infiziert werden. Nach der Infektion kann der Agent aus der Target-Liste gelöscht werden.		

Tabelle 4.6: Infection Formular

	start-up Informationen		
A	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
	Name jeder Person (ID-Nummer)	**int**	**Sim-Agent**
	Alter jeder Person	**int**	**Sim-Agent**
	Geschlecht jeder Person	**int**	**Sim-Agent**
	Tätigkeit jeder Person	**String**	**Sim-Agent**
	Name jedes Ortes (ID-Nummer)	**int**	**Simulation**
	Typ jedes Ortes	**String**	**Simulation**
	Liste der Orte, der eine Person zugeteilt ist	**List<int>**	**Simulation**

Tabelle 4.7: Start-up Informationen

	IllegalStatusChange		
A	Beschreibung des constraint		
	Eine genesene Person darf nicht erneut angesteckt werden.		
B	Beschreibung der notwendigen Attribute		
	Attribut	Datentyp	Zugriff
	naturally immune	**boolean**	**Sim-Agent**
	shedding	**boolean**	**Sim-Agent**
C	Anzeichen für den constraint		
	Die Person wird nach einer Genesung erneut infiziert. **naturally immune = true and shedding = true**		
D	Beschreibung der logging Attribute		
	Attribut	Datentyp	Zugriff
	Person, die erneut infiziert wurde	**int**	**Sim-Agent**
	Tag der erneuten Infektion	**int**	**Simulation**
	Ort der erneuten Infektion	**int**	**Sim-Agent**
	Person, die die erneute Infektion verursacht hat	**int**	**Sim-Agent**
E	Minimaler Zeitschritt		
	Eine Person kann jeden Tag infiziert werden.		
F	Nachricht an den Benutzer		
	Eine Person wurde illegalerweise nach einer Genesung erneut infiziert.		
F	Abbruch der Simulation. Wenn ja, warum?		
	Ja. Falsches Modellverhalten		

Tabelle 4.8: IllegalStatusChange Formular

- **Communicative Model:** Alle Formulare und das Datenbankdesign werden nochmals überprüft.

Computerized Model Phase

- **Programmed Model:** In dieser Phase werden die abstrakten Methoden der Klassen implementiert und das Parameter File (siehe Abbildung 4.6) erstellt. Dieses enthält außer den selektiven Attributen auch Informationen zur Datenbank, die der Logger-Agent benötigt.

- **Experimental Model:** Durch Änderungen im Parameter File können unterschiedliche Simulationsagenten beobachtet und auf verschiedene watch-values überprüft werden.
- **Simulation Results:** Einige Ergebnisse, die aus den Daten, die das VOMAS gesammelt hat, entstanden sind, werden am Ende dieses Kapitels präsentiert.

Element	Value
VOAMSparams	
!--	Database settings
databaseSetting	
connection	jdbc:mysql://localhost:8889/influenza?rewriteBatchedSt...
user	root
password	root
!--	Watches and constraint settings
watchesAndContraints	
watches	
type	
continuousLogging	false
	VOMAS.VOMAS_Influenza_Watch_Infection
type	
constraints	
!--	Target settings
targets	
sex	
male	
observed	true
female	
observed	true
occupation	
infant	
observed	true
pupil	
worker	
observed	true
workless	
retired	
ageSpan	
span	
fromAge	0
toAge	99
span	
span	

Abbildung 4.6: Parameter File des VOMAS für das Influenzamodell

4.2.2 Datenbankdesign

Mit dem Protokollmodul des Influenzamodells wurden bisher Informationen zum Modellverhalten in einem Excel File gespeichert (siehe Tabelle 4.9 und Tabelle 4.10). Es wird zu jedem Tag (150 Tage) und jeder Altersgruppe (in 1 Jahresabständen) notiert, wieviele Personen in welchem Zustand waren (S = susceptible, SH w/o SY = shedding without symptoms, SH mild SY = shedding with mild symptoms, SH severe SY = shedding with sever symptoms, NATI = naturally immune, V = vaccinated) und wieviele Personen welche Statusänderungen vollzogen haben (Start SH = start shedding, Start mild SY = start mild symptoms, Start severe SY = start severe symptoms, Stop SH = stop shedding, Stop mild SY = stop mild symptoms, Stop severe SY = stop severe symptoms, Get V = get vaccinated).

	Age 0-99						
Time	Total	S	SH w/o SY	SH mild SY	SH severe SY	NATI	V
Initial							
1							
2							
...							

Tabelle 4.9: Informationen über die Simulation über das Protokollmodul

	Age 0-99						
Time	Start SH	Start mild SY	Start severe SY	Stop SH	Stop mild SY	Stop severe SY	Get V
1							
2							
3							
...							

Tabelle 4.10: Informationen über die Simulation über das Protokollmodul

Es wurden also vor allem Informationen über den Gesamtzustand der Bevölkerung in Erfahrung gebracht (man spricht von Informationen auf dem Makro-Level). Diese Informationen wurden aber noch nicht mit einzelnen Personen in Verbindung gebracht. Es muss mehr Wissen auf dem Mikro-Level der Simulation gesammelt werden, um die stillstehende Validierung voran zu bringen. In Abschnitt 4.2.1 wurden bereits Validierungsfragen formuliert:

- *Wer* wird von *wem*, *wann* und *wo* infiziert?
- Hätte sich die Person auch woanders anstecken können?
- Wie sieht der Krankheitsverlauf der Person aus?
- Bleibt die Person zu Hause?

Die Speicherung dieser Informationen bei über 8 Millionen Simulationsagenten übersteigt die Möglichkeiten eines Excel-Sheets. Die schiere Menge an Daten und der Zusammenhang zwischen den Informationen legt die Benutzung einer Datenbank nahe. Diese ermöglicht effizientere Speicherung der Daten, schnelleren Zugriff auf Informationen und einfachere Formulierung komplexer, tabellenübergreifender Abfragen. Außerdem ist es eine bessere Schnittstelle zwischen dem VOMAS und möglichen Visualisierungstools, welche die gesammelten Daten grafisch darstellen können.

Die entworfene Datenbank besteht aus 5 Tabellen: Person, Location, Person_Location, Infection und NonContactTriggeredStateChange. Die Zusammenhänge der Tabellen sind in einem Entity-Relationship Diagramm abgebildet (siehe Abbildung 4.7). Es wird beispielsweise folgendermaßen gelesen: An einem Ort können mehrere Infektionen stattfinden – eine Infektion findet immer genau an einem Ort statt.

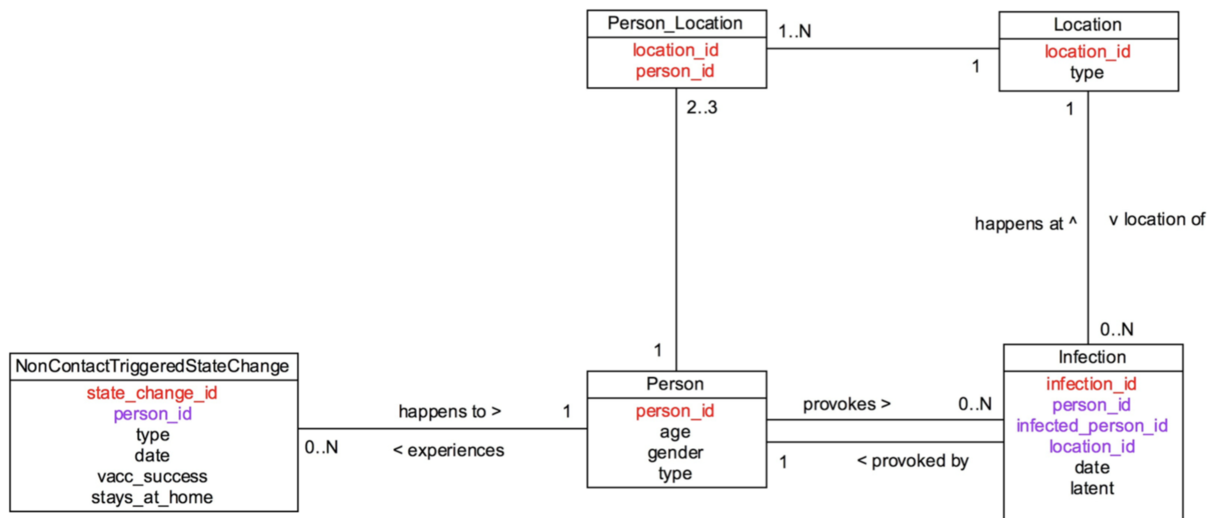

Abbildung 4.7: Entity-Relationship Diagramm des Datenbankdesign

Person

In der Tabelle **Person** werden Grundinformationen zu den Personen im Modell gespeichert (siehe Tabelle 4.11). Diese Tabelle wird bereits in der Intitialisierungsphase des Modells befüllt und im Laufe der Simulation nicht mehr bearbeitet. Der primary key ist *person_id*.

- **Person_id:** Eine eindeutige Identifikationsnummer der Person. Wird bei der Initialisierung zugeordnet.
- **Age:** Das Alter der Person. Wird bei der Initialisierung zugeordnet.
- **Gender:** Das Geschlecht der Person. Wird bei der Initialisierung zugeordnet.

- **Type:** Typ der Person. Wird bei der Initialisierung zugeordnet.

Person	
person_id	int
age	int
gender	0/1 (male/female)
type	string (pupil, infant, worker, workless, retired)

Tabelle 4.11: Attribute und Datentypen von Person

Infection

In der Tabelle **Infection** werden Informationen zu jeder Ansteckung gespeichert (siehe Tabelle 4.12). Diese Tabelle wird während der Laufzeit stetig erweitert. Der primary key ist *infection_id*.

- **Infection_id:** Wird mit jeder neuen Infektion erhöht.
- **Person_id:** Person, welche die Ansteckung verursacht hat. Hat den Wert -1, falls Person bereits im Zuge der Initialisierung infiziert wurde[6]
- **Infected_person_id:** Person, welche angesteckt wurde.
- **Location_id:** Ort, an dem die Infektion passiert ist. Hat den Wert -1 für eine Ansteckung im Zuge der Initialisierung[7]
- **Date:** Tag, an dem die Infektion stattgefunden hat.
- **Latent:** Hat den Wert 1, falls die Person am selben Tag bereits durch eine andere Person angesteckt worden ist. Dies ist möglich, da angesteckte Personen erst am Folgetag ihren Infektionsstatus ändern können. Dadurch werden Kettenreaktionen von Ansteckungen innerhalb desselben Tages vermieden.

Infection	
infection_id	int
person_id	int
infected_person_id	int
location_id	int
date	int (day)
latent	int (0,1)

Tabelle 4.12: Attribute und Datentypen von Infection

[6] Die Person mit person_id = -1 hat den Typ *God*.
[7] Der Ort mit location_id = -1 hat den Typ *Heaven*.

Location

In der Tabelle **Location** werden Informationen zu jedem Ort gespeichert (siehe Tabelle 4.13). Diese Tabelle wird gänzlich während der Initialisierung befüllt. Der primary key ist *location_id*.

- **Location_id:** Eine eindeutige Identifikationsnummer des Ortes. Wird bei der Initialisierung zugeordnet.
- **Type:** Typ des Ortes. Wird bei der Initialisierung zugeordnet.

Location	
location_id	int
type	String (leisure, school, workplace, household)

Tabelle 4.13: Attribute und Datentypen von Location

Person_Location

Die Tabelle **Person_Location** ist eine Querverweistabelle um die *many-to-many*-Beziehung zwischen einer Person und einem Ort darzustellen[8] (siehe Tabelle 4.14).

Person_Location	
location_id	int
person_id	int

Tabelle 4.14: Querverweistabelle von Person und Location

NonContactTriggeredStateChange

In der Tabelle **NonContactTriggeredStateChange** werden Informationen zu allen Statusänderungen gespeichert (siehe Tabelle 4.15), welche nicht von anderen Personen initiiert werden (Genesung, Impfung, natürliche Immunität, Entwicklung von Symptomen). Diese Tabelle wird während der Laufzeit gefüllt. Der primary key ist *state_change_id*.

- **State_change_id:** Wird mit jeder neuen Infektion erhöht.
- **Person_id:** Person bei der eine Statusänderung passiert.
- **Type:** Art der Statusänderung (vaccinated, naturally immune, symptoms, severe symptoms, shedding, end shedding)
- **Date:** Tag an dem die Statusänderung stattgefunden hat.

[8] Diese Tabelle musste eingefügt werden um die Anforderungen der zweiten Normalform zu erfüllen.

- **Vacc_success:** Mit NULL belegt, da im aktuellen Modell nur erfolgreiche Impfungen erfasst werden.
- **Stays_at_home:** 1 falls die Person zu Hause bleibt, sonst 0.[9]

NonContactTriggeredStateChange	
state_change_id	int
person_id	int
type	String
date	int (day)
vacc_success	NULL
stays_at_home	int

Tabelle 4.15: Attribute und Datentypen von NonContactTriggeredStateChange

4.2.3 Notwendige Änderungen am Modell

Aufgrund der Tatsache, dass das VOMAS erst im Nachhinein entworfen wurde, mussten einige Änderungen am Modell gemacht werden. Bei einem großen Teil der Änderungen handelte es sich lediglich um einfache get-Funktionen, die Zugriff zu diversen Attributen, wie Personenalter, Personengeschlecht und Personentyp ermöglichen. Für einige Änderungen musste allerdings sehr tief in die Programmarchitektur vorgedrungen werden, da für das VOMAS notwendige Informationen in den Methodenketten teils bereits sehr früh nicht weitergeleitet wurden. Dieser Abschnitt hat wenig modelltechnische Relevanz, soll aber veranschaulichen, welche Mehrarbeit eine erst nachträgliche Entwicklung eines VOMAS mit sich bringt.

Person: Für das Sammeln der Attribute einer Person mussten lediglich passende get-Funktionen implementiert werden. Um die modellinternen Methoden von jenen Methoden zu unterscheiden, zu welchen die VOMAS-Agenten Zugriff haben, wurde vor die Methodenbezeichnung die Zeichenkette VOMAS_ gesetzt. Das gleiche Prinzip wird auch bei Attributen verwendet, welche zusätzlich wegen dem VOMAS eingeführt werden mussten.

Location: Die Orte im Modell hatten bisher noch kein Attribut, das sie eindeutig unterscheidbar gemacht hat. Deshalb wurde ein Klassenattribut (VOMAS_locationCounter) eingeführt, mit dem jedem Ort eine eindeutige ID zugewiesen wird. Zugang zu einer Liste der Orte im Modell erhält der VO-Manager über die Schnittstelle zur Simulation.

Person_Location: In der Klasse der Personen wurde ein Methode implementiert, die eine Liste der zugehörigen Orte des Agenten liefert. Auf diese Liste hat der VO-Agent Zugriff.

[9] Der Datentyp *boolean* wäre hierfür zweckdienlicher. Allerdings wurde die Datenbank mit MySQL erstellt, welches den Datentyp *boolean* zum jetzigen Zeitpunkt nicht unterstützt.

NonContactTriggeredStateChange: Für das Einholen der Information der Statusänderungen ohne Fremdeinwirken mussten nur an passenden Stellen get-Funktionen implementiert werden. Die Abfrage der Information, ob eine Impfung erfolgreich war oder nicht, wäre in dem Modell mit zu vielen Änderungen verbunden gewesen. Es werden deshalb nur erfolgreiche Impfungen registriert. Diese Tatsache hätte bei einer parallelen Entwicklung von VOMAS und Modell leicht verhindert werden können. Zugriff auf das Datum erhält der Watch-Agent über die Schnittstelle des VO-Managers mit der Simulation.

Infection: Um die Informationen für diese Tabelle zu sammeln, mussten die tiefgehendsten Änderungen am Modell vorgenommen werden. Da für jede Infektion beide Parteien und außerdem der Ort des infizierenden Kontaktes in Erfahrung gebracht werden sollte, musste jeder Ort zunächst unterscheidbar gemacht werden (wie bereits oben weiter erwähnt wurde). Da im bisherigen Kontaktmodul weder Informationen zum Ort noch zur Person, mit der man Kontakt hatte weitergeleitet wurden, mussten hierfür einige Methoden erweitert werden.

4.2.4 Erkenntnisse und Herausforderungen bei der Implementierung

- Die größte Unsicherheit zu Beginn der praktischen Umsetzung war der Aspekt der Performance. Wie viel mehr Zeit würde ein VOMAS in einem Simulationslauf in Anspruch nehmen, da mindestens 8 Millionen weitere Entitäten in der Simulation tätig sind? Wie sich herausstellte, dauert die Initialisierung und das Sammeln der start-up Informationen im Schnitt 773% länger als die Initialisierung der Simulation davor. Die Gesamtdauer des Setup fällt im Vergleich zur Simulationsdauer allerdings wenig ins Gewicht (13%). Von der Dauer des eigentlichen Simulationslaufes verfallen im Schnitt lediglich 4,4% auf die Tätigkeiten des VOMAS. Das liegt nicht unbedeutend daran, dass die Evaluierungsmethoden der Watch- und Constraint-Agenten sehr simpel sind – in den meisten Fällen müssen nur einige Attribute auf deren Wert überprüft werden. In anderen Fällen kann es sich hierbei aber auch um komplexe Berechnungen handeln, welche die Performance deutlich beeinflussen könnten. In diesem Fall sollten, wenn möglich die Methoden *deleteTarget* und *deleteWatchAgent* eingesetzt werden. Außerdem kann mit Hilfe der selektiven Attribute die Zielagentenmenge eingeschränkt werden. Das kann besonders bei Parameterstudien mit mehreren Simulationsläufen eine Zeitersparnis ermöglichen.

- Der Arbeitsspeicherverbrauch (RAM-Verbrauch) durch das VOMAS ist in Abbildung 4.8 dargestellt. Der Speicherbedarf der Datenbank beträgt bei einer Bevölkerungszahl von 200.000 ca. 235MB[10].

- Die Arbeit des Logger-Agenten musste deutlich optimiert werden. Die Schnittstelle von JAVA mit SQL stellt hierfür eine Vielzahl von Zugängen zur Verfügung (prepared statements, batch execution, ...).

[10] Gerechnet auf einem MacBook Pro (early 2013), 2,6 GHz Intel Core i5, 8 GB 1600 MHz DDR3.

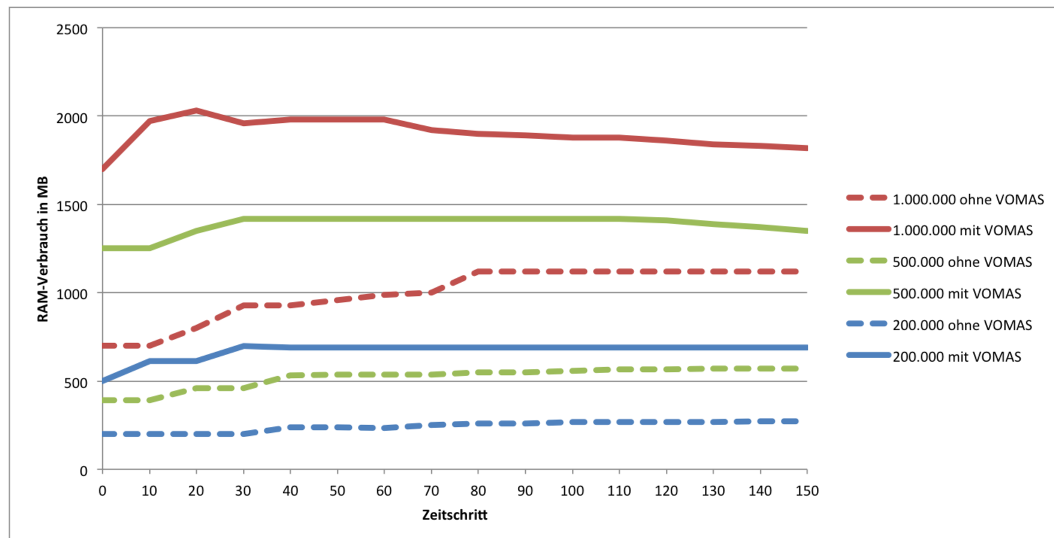

Abbildung 4.8: RAM-Verbrauch bei variierender Bevölkerungszahl

- Wie von Beginn an verdeutlicht, sollte ein VOMAS immer parallel mit dem Modell entwickelt werden. Relevante Informationen sollten von Beginn an festgelegt und leicht erreichbar sein. Der Vorteil daran ist, dass die endgültige Informationsspeicherung zunächst irrelevant ist. Der Modellierungsexperte kann also volles Augenmerk auf die Programmierung der Verhaltensregeln der Agenten legen und muss sich nicht sofort mit der Implementierung eines Protokollmoduls beschäftigen. Sind die Schnittstellen sauber definiert und implementiert, kann mit wenig Aufwand ein Logger-Agent erstellt werden, der notwendige Informationen dauerhaft speichern kann.

- Die leichtverständliche und gut kommunizierbare Struktur eines VOMAS erleichtert vor allem modellierungsfremden Personen (wie dem SME) den Zugang zum Thema der Validierung eines Modells.

- Die Verwendung eines VOMAS hat, wie zu erwarten, gewisse Vor- und Nachteile. Bei einem Versuch diese Vor- und Nachteile herauszufiltern, wurde offensichtlich, dass dies immer nur im Vergleich zu einer anderen Art der Protokollierung möglich ist[11]. Anstatt also Vor- und Nachteile eines VOMAS gegenüber der Verwendung verschiedener anderer Ansätze zu untersuchen, ist es womöglich zweckdienlicher aufzulisten, was die Verwendung eines VOMAS alles ermöglicht und mit welchen Folgen zu rechnen ist (siehe Tabelle 4.16).

[11] Das VOMAS erfüllt bei der Influenzasimulation vor allem die Rolle eines Protokollmoduls.

Möglichkeiten	**Negative Folgen**
Dynamische Datenaggregierung	Rechenzeit
Erweiterungen benötigen nur minimale Änderungen	Nachträgliche Implementierung um vieles aufwendiger als parallele Implementierung
Art der Speicherung beliebig	
Verifizierung und Validierung während der Modellentwicklung	
Abstrahiert auch anderweitig anwendbar	
Plattformübergreifender Zugriff möglich	

Tabelle 4.16: Erkenntnisse aus der Arbeit mit einem VOMAS

4.3 Ergebnisse

Die zusätzlichen Daten, die mit dem VOMAS gesammelt werden konnten, erlauben einen detaillierten Einblick in den Fluss der Infektionen, also welche Person von welcher angesteckt wurde. Außerdem können neue Erkenntnisse über die Infektionsorte gesammelt werden und welche Rolle der „Beruf" – also der Personentyp – einer Person bei der Infektionsübertragung spielt.

Die Parameter, mit denen die Simulationsläufe gestartet wurden, sind aus [FM12][12] entnommen. Wie in Abschnitt 4.1.2 bereits erwähnt, war das Modell auch nach einer feinen Kalibrierung nicht in der Lage, die Originaldaten – unter vernünftigen Annahmen der Anzahl von infektiösen Personen und dem Prozentanteil natürlich immuner Personen zu Simulationsbeginn im System – wiederzugeben. Die folgenden Ergebnisse ermöglichen daher vor allem einen Blick auf das qualitative Verhalten des Modells ermöglichen unter Umständen grundsätzliche Fehler in der Modellstruktur zu finden.

Die Simulationsexperimente wurden mit einer Bevölkerungszahl von 200.000 durchgeführt. Es sei angemerkt, dass die Ergebnisse qualitativ unabhängig von der gewählten Bevölkerungszahl sind. Die nicht festgelegten Parameter des Modells wurden folgendermaßen gewählt:

- Infektionswahrscheinlichkeit: 12%
- Anteil anfangs infektiöser Personen in der Bevölkerung: 0,02%

[12] p.58 ff.

- Anteil natürlich immuner Personen in der Bevölkerung: 15,55%

Zum ersten Mal können Infektionen mit den Orten im Modell in Verbindung gebracht werden, wie in Abbildung 4.9 dargestellt ist.

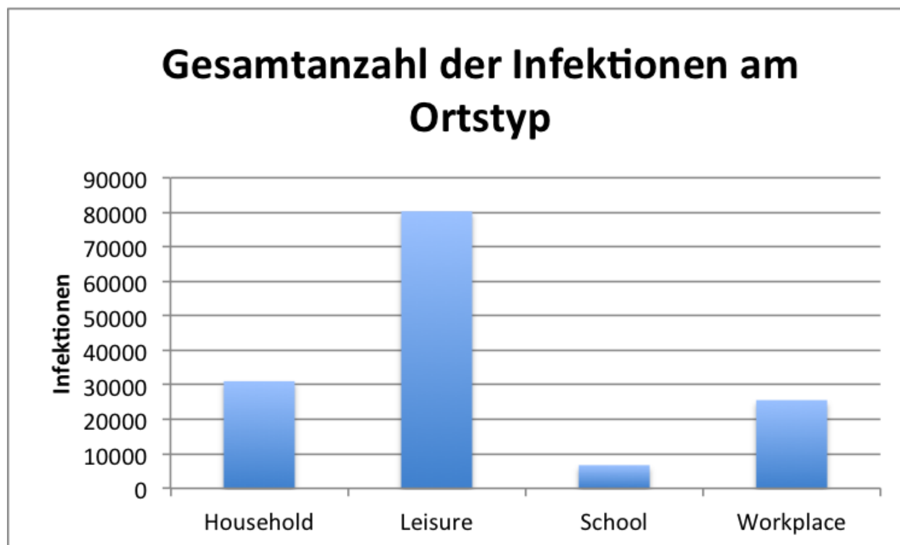

Abbildung 4.9: Anzahl der Infektionen pro Ortstyp

Da nicht unbedingt jeder Ort einen Infektionskontakt hervorgerufen haben muss, können die „infektionsfreien“ Orte heraus selektiert werden (siehe Abbildung 4.10):

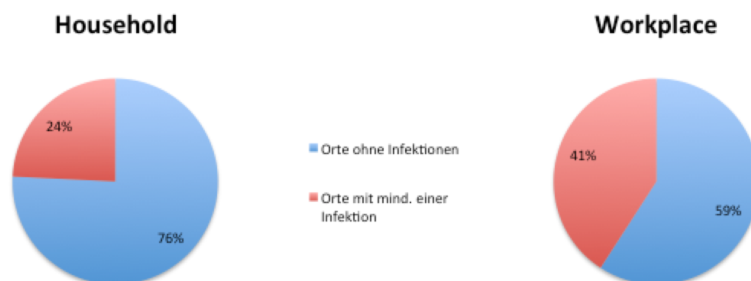

Abbildung 4.10: Anteil der infektionsfreien Orte nach Ortstyp

Die Grafiken für die Freizeit (Leisure) und die Schulen (School) wurden ausgelassen, da an jedem dieser Ortstypen Infektionen stattgefunden haben.

Ebenso wie die Orte, können auch die Personen stärker in Verbindung mit den Infektionen gebracht werden. Bisher konnten höchstens Informationen zu der Anzahl der infizierten Personen je Altersgruppe gesammelt werden. Mit Hilfe des VOMAS kann nun in Erweiterung

auch verfolgt werden, wieviele Personen eines Personentyps weitere Infektionen verursacht haben (siehe Abbildung 4.11).

Abbildung 4.11: Anteil der infizierenden Personen nach Personentyp

Die verursachten Infektionen können, wie in Abbildung 4.12, 4.13, 4.14 und 4.15 präsentiert, noch detaillierter analysiert werden:

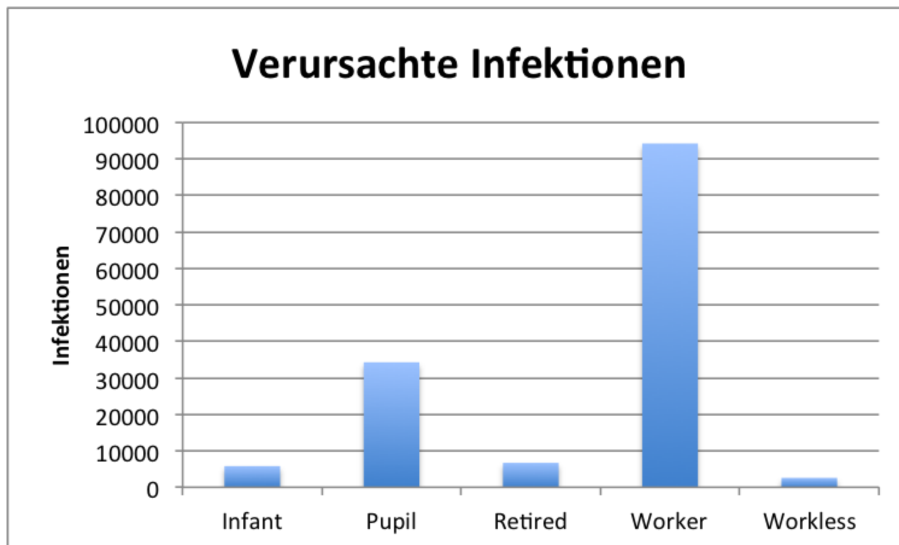

Abbildung 4.12: Verursachte Infektionen (Typ)

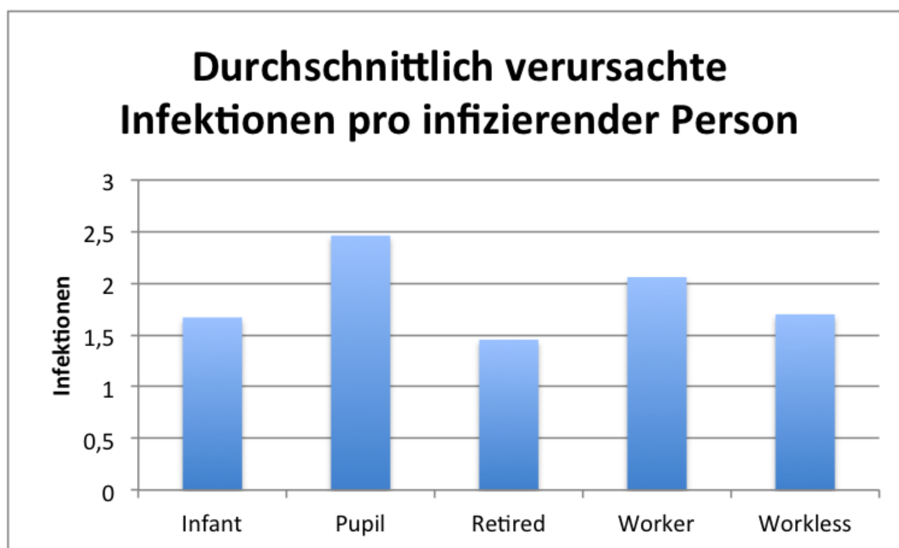

Abbildung 4.13: Verursachte Infektionen pro infizierender Person (Typ)

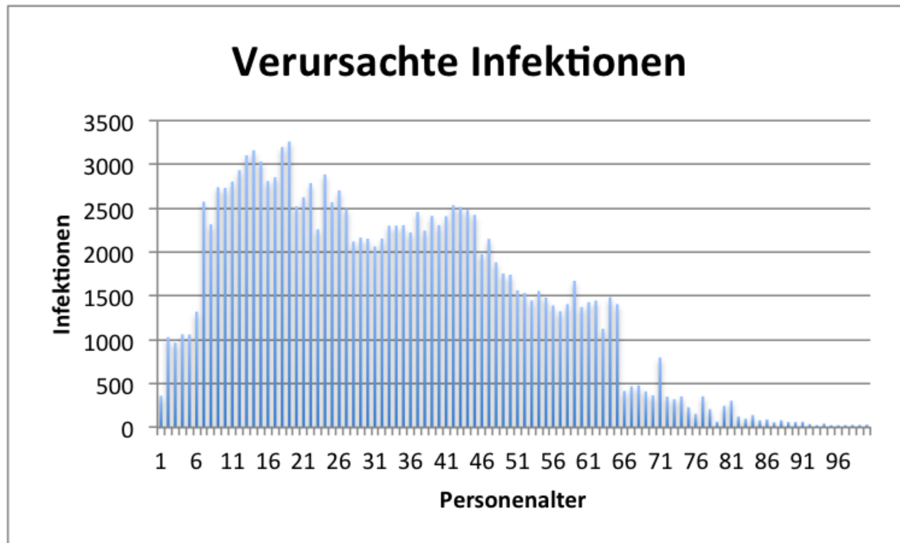

Abbildung 4.14: Verursachte Infektionen (Alter)

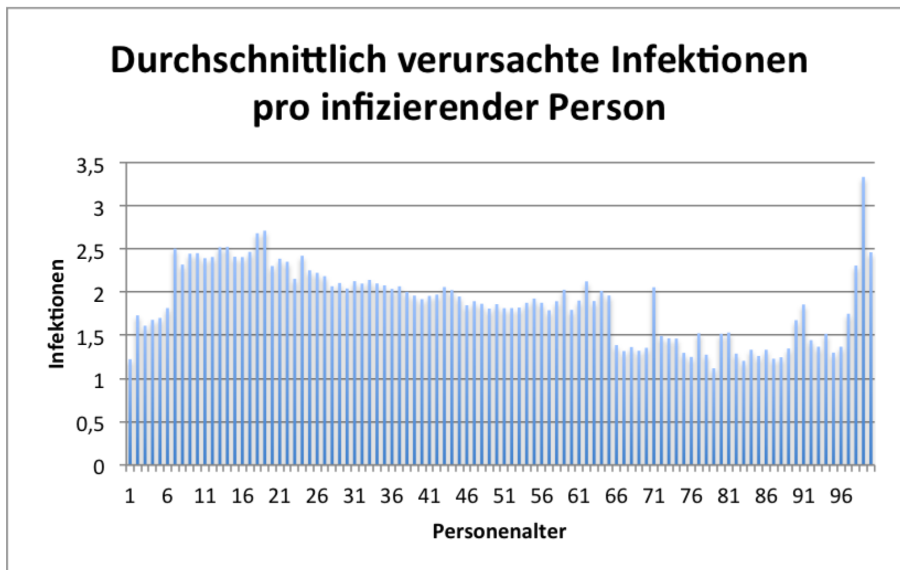

Abbildung 4.15: Verursachte Infektionen pro infizierender Person (Alter)

All diese und weitere mögliche grafische Darstellungen (siehe Kapitel 6 – Diskussion) können von Experten auf Plausibilität untersucht oder mit möglichen vorhandenen Daten verglichen werden.

KAPITEL 5

Zusammenfassung

Da die Wirklichkeit nie vollkommen in einem Modell abgebildet werden kann, muss das Ziel der Modellbildung und Simulation jenes sein, ein Modell zu finden oder zu konstruieren, das die zugrundeliegende Problematik hinreichend gut löst. Der Prozess der Validierung ist hierbei ein wichtiges Hilfsmittel. Die Modellentwicklung entspricht einer zyklischen Entwicklung – dem Modellierungskreislauf. Die Validierung eines Modells kann in diesen Kreislauf aufgenommen werden. Auf den speziellen Modellierungsansatz der agentenbasierten Modellierung können von den allgemeinen Validierungsmethoden nur wenige angewandt werden – vorwiegend aus dem Bereich der informalen und dynamischen Techniken. Neue Methoden wurden deshalb speziell für die Validierung agentenbasierter Modelle entwickelt. Hierunter sind vor allem Arbeiten von Klügl zu erwähnen (siehe [Klü08] und [LK12]).

Ein weiterer Ansatz – beschrieben in [NHK09] – bedient sich eines Virtual Overlay Multiagent System (kurz: VOMAS). Ein VOMAS besteht aus Agenten, die verschiedensten Aufgaben nachgehen können. Je nach Aufgaben, erhalten diese Agenten verschiedene Bezeichnungen. Die VO-Agenten können mit Hilfe von Watch-Agenten Attribute von Simulationsagenten beobachten und Änderungen in deren Verhalten oder Zustände falls gewünscht für weitere Analysen zur Speicherung an einen Logger-Agenten weiterleiten. Constraint-Agenten sind in der Lage Verletzungen von Modellannahmen zu registrieren. Ein Manager – der VO-Manager – delegiert die Abläufe innerhalb des VOMAS und ist in der Lage, globale Attribute der Simulation zu beobachten. Da wenig Informationen zur genauen Struktur eines VOMAS oder Erfahrung zur Arbeit mit einem solchen existieren, wurde eine genau Analyse durchgeführt. Die Aufgaben der Agenten eines VOMAS wurden isoliert und mittels Klassen- und Sequenzdiagrammen detailliert beschrieben. Im Zuge dessen wurde ein Vorschlag für ein plattformunabhängiges VOMAS-Design entwickelt, zusammen mit einer Anleitung für die endgültige Implementierung anwendungsabhängiger Komponenten. Diese Schritte konnten in den allgemeinen Modellierungskreislauf aufgenommen werden.

Die Funktionalität des entwickelten VOMAS-Design wurde am Beispiel einer Influenzasi-

mulation getestet. Das VOMAS stellte sich als sehr flexibles Datenaggregierungswerkzeug heraus, mit dessen Hilfe wesentlich mehr und detailliertere Informationen zum Verhalten der Agenten gesammelt werden konnte als mit dem bis dahin verwendeten Protokollmodul. Mit Hilfe des VOMAS können unter anderem erstmals beide Seiten einer Infektion (aus Sicht der Person, die eine andere infiziert, und aus Sicht jener, die infiziert wird) protokolliert werden und diese Infektionen mit den Orten an denen sie stattgefunden haben in Verbindung gebracht werden. Diese Informationen können in Zukunft dabei helfen, die Validierung des Influenzamodells voranzutreiben.

KAPITEL 6
Diskussion

Die agentenbasierte Modellierung steht trotz vielversprechenden Anwendungsmöglichkeiten und Potential vor dem Problem des fehlenden Vertrauens in die vergleichsweise neue Modellierungstechnik. Besonderes Augenmerk muss deshalb auf Weiterentwicklungen im Bereich der Validierung von agentenbasierten Modellen gelegt werden. „Klassische“ Validierungsmethoden lassen sich aus verschiedensten Gründen – welche im Zuge dieser Arbeit präsentiert wurden – nicht zufriedenstellend auf diese Modelle anwenden. Das VOMAS-Konzept bietet ein Framework, mit dem verschiedenste – für agentenbasierte Modelle in Frage kommende – Validierungsmethoden effizient ermöglicht werden können.

Der VOMAS-Ansatz verlangt, dass der Subject Matter Expert von Anfang an an der Modellbildung beteiligt ist. In diesem Sinne sollte ihm ein Werkzeug zur Steuerung des VOMAS zur Vefügung gestellt werden – in Form einer grafischen Oberfläche mit Steuerelementen. Das entwickelte VOMAS-Design sollte bei anderen Modellen angewandt und weiterentwickelt werden. Vor allem die Frage, in wie weit VO-Agenten direkt in der Simulationsumgebung platziert werden können um Informationen zu sammeln, sollte in Form weiterer Forschungsarbeiten untersucht werden.

Für die Influenzasimulation war das VOMAS in der Lage, viele neue Daten zum Verhalten der Agenten zu sammeln. Die Information, die in diesen Daten steckt, ist aber nicht mehr wie bisher in einfachen Plots und Diagrammen darstellbar. Hier müssen Techniken und Möglichkeiten der Visualierung genutzt werden. Infektionsnetzwerke könnten damit dargestellt werden, sowie Infektionsflussdiagramme, die beschreiben, zwischen welchen Bevölkerungsgruppen Ansteckungen passieren (ähnliche Grafiken kommen beispielsweise bei Wählerstromanalysen zum Einsatz).

Der Einfluss einer bestimmten Parametermenge wurde bislang vernachlässigt: die *location intensities*, also die Rolle, die der Ort bei einem möglicherweise ansteckenden Kontakt spielt. Mit Hilfe des VOMAS kann deren Einfluss auf das Modellverhalten nun festgestellt

werden. In Abbildung 6.1 ist dies exemplarisch dargestellt. Die location intensities der Orte vom Typ Household, Leisure und School wurden mit dem Wert 1 gewählt. Der Wert für die Orte des Typ Workplace wurde variiert (von links nach rechts: 1, 0.1, 10). Die Änderungen der Infektionsanzahl an den Ortstypen Workplace ist zu erwarten, interessant ist jedoch, in wie fern es die Anzahl der Infektionen an den anderen Ortstypen beeinflusst. In diesem Bereich können weitere Parameterstudien angestellt werden.

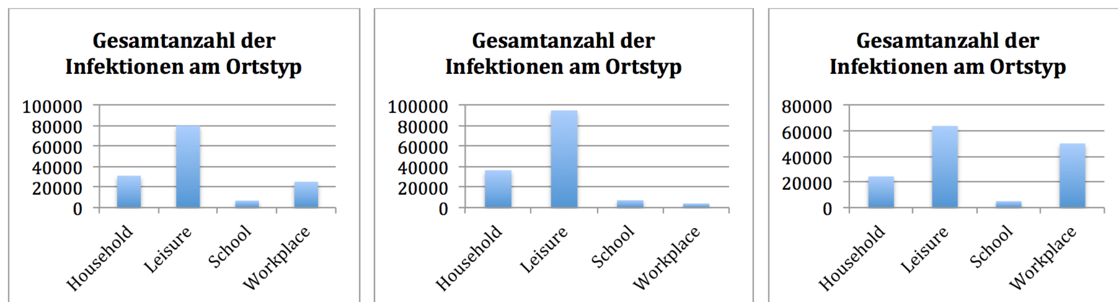

Abbildung 6.1: Einfluss der location intensity auf die Infektionen am Beispiel Workplace

Abbildungsverzeichnis

1.1 Problemlösung mit Hilfe von Modellen 2
1.2 Einfache Regeln für ein Schwarmverhalten aus [Rey87] 3
1.3 Agenten und ihre Umwelt 5
1.4 Topologien für die Interaktion zwischen Agenten aus [MN10] 5

2.1 Darstellung des Modellierungskreislauf nach [PP15] 10
2.2 Validierungsmethode aus [Klü08] 17
2.3 Entscheidungsdiagramm aus [LK12] 20

3.1 Darstellung der VOMAS- und der Simulationsebene nach [NHK09] 24
3.2 Klassendiagramm des VOMAS-Design 26
3.3 Zugriff auf die Sim-Agenten durch die VO-,Watch- und Constraint-Agenten 30
3.4 Zugriff auf das VOMAS durch die Simulation 31
3.5 Zugriff auf die Simulation durch den VO-Manager 32
3.6 Zugriff auf den VO-Manager 34
3.7 Attribute und Methoden der VOMAS_anyManager Klasse 36
3.8 Interner Ablauf des Konstruktors der VOMAS_anyManager Klasse 39
3.9 Interner Ablauf der Methoden *startVomas* und *stopSimulation* 39
3.10 Zugriff auf den Logger-Agent 40
3.11 Attribute und Methoden der VOMAS_anyLoggerAgent Klasse 41
3.12 Zugriff auf den Console-Agent 41
3.13 Zugriff auf den VO-Agent 42
3.14 Attribute und Methoden der VOMAS_anyAgent Klasse 44
3.15 Interner Ablauf des Konstruktors der VOMAS_anyAgent Klasse 46
3.16 Interner Ablauf der Methoden *gatherStartUpData* und *gatherData* 47
3.17 Zugriff auf den Watch-Agent 47
3.18 Attribute und Methoden der VOMAS_anyWatchAgent Klasse 48
3.19 Interner Ablauf der Methoden *gatherStartUpData* und *gatherData* 50
3.20 Zugriff auf den Constraint-Agent 51
3.21 Attribute und Methoden der VOMAS_anyConstraintAgent Klasse 52
3.22 Interner Ablauf der Methoden *checkStartUpData* und *checkData* 53
3.23 Sequenzdiagramm der Initialisierung eines VOMAS 55
3.24 Sequenzdiagramm der Informationssammlung eines VOMAS 56

4.1 Bevölkerungsmodul aus [FM12] 66
4.2 Kontaktmodul aus [FM12] 67
4.3 Krankheitsmodul aus [FM12] 68
4.4 Kalibrierung der Infektionswahrscheinlichkeit aus [Pic13] 69
4.5 Kalibrierung der Infektionswahrscheinlichkeit (selektierte Auswahl) aus [Pic13] . . 70
4.6 Parameter File des VOMAS für das Influenzamodell 79
4.7 Entity-Relationship Diagramm des Datenbankdesign 81
4.8 RAM-Verbrauch bei variierender Bevölkerungszahl 86
4.9 Anzahl der Infektionen pro Ortstyp 88
4.10 Anteil der infektionsfreien Orte nach Ortstyp 88
4.11 Anteil der infizierenden Personen nach Personentyp 89
4.12 Verursachte Infektionen (Typ) 90
4.13 Verursachte Infektionen pro infizierender Person (Typ) 90
4.14 Verursachte Infektionen (Alter) 91
4.15 Verursachte Infektionen pro infizierender Person (Alter) 91

6.1 Einfluss der location intensity auf die Infektionen am Beispiel Workplace 95

Tabellenverzeichnis

3.1 Watch-value Formular 62
3.2 Constraint Formular 64

4.1 Infection Formular 72
4.2 NonContactTriggeredStateChange Formular 73
4.3 Start-up Informationen 74
4.4 IllegalStatusChange Formular 75
4.5 NonContactTriggeredStateChange Formular 76
4.6 Infection Formular 77
4.7 Start-up Informationen 77
4.8 IllegalStatusChange Formular 78
4.9 Informationen über die Simulation über das Protokollmodul 80
4.10 Informationen über die Simulation über das Protokollmodul 80
4.11 Attribute und Datentypen von Person 82
4.12 Attribute und Datentypen von Infection 82
4.13 Attribute und Datentypen von Location 83
4.14 Querverweistabelle von Person und Location 83

4.15 Attribute und Datentypen von NonContactTriggeredStateChange 84
4.16 Erkenntnisse aus der Arbeit mit einem VOMAS 87

Literaturverzeichnis

[B$^+$03] Olivier Barreteau et al. Our companion modelling approach. *Journal of Artificial Societies and Social Simulation*, 6(2), 2003.

[Bal94] Osman Balci. Validation, verification, and testing techniques throughout the life cycle of a simulation study. *Annals of Operations Research*, 53:121 – 173, December 1994.

[Bal97] Osman Balci. Verification validation and accreditation of simulation models. In *Proceedings of the 29th Conference on Winter Simulation*, WSC '97, pages 135–141, Washington, DC, USA, 1997. IEEE Computer Society.

[DP95] Jim Doran and Mike Palmer. The eos project integrating two models of palaeolithic social change. In N. Gilbert and R. Conte, editors, *Artificial Societies - The Computer Simulation of Social Life*, chapter 6, pages 103 – 125. UCL, 1995.

[FM12] Philipp Pichler Christoph Urach Niki Popper Florian Miksch, Günther Zauner. Endbericht des influenza projekts, February 2012.

[GH06] Paul Guyot and Shinichi Honiden. Agent-based participatory simulations: Merging multi-agent systems and role-playing games. *Journal of Artificial Societies and Social Simulation*, 9(4), 2006.

[KL00] W David Kelton and Averill M Law. *Simulation modeling and analysis*. McGraw Hill Boston, 2000.

[Klü08] Franziska Klügl. A validation methodology for agent-based simulations. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 39–43. ACM, 2008.

[LK12] Athanasia Louloudi and Franziska Klügl. Immersive face validation: A new validation technique for agent-based simulation. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, pages 1255–1260. IEEE, 2012.

[MN06] Charles M. Macal and Michael J. North. Tutorial on agent-based modeling and simulation part 2: How to model with agents. In *Proceedings of the 38th Conference on Winter Simulation*, WSC '06, pages 73–83. Winter Simulation Conference, 2006.

[MN10] Charles M Macal and Michael J North. Tutorial on agent-based modelling and simulation. *Journal of simulation*, 4(3):151–162, 2010.

[NF67] Thomas H Naylor and Joseph Michael Finger. Verification of computer simulation models. *Management Science*, 14(2):B–92, 1967.

[NHK09] Muaz A K Niazi, Amir Hussain, and Mario Kolberg. Verification and validation of agent based simulations using the vomas (virtual overlay multi-agent system) approach. *Computing Science and Mathematics Conference Papers and Proceedings*, 2009.

[Nia11] Muaz AK Niazi. *Towards a novel unified framework for developing formal, network and validated agent-based simulation models of complex adaptive systems*. PhD thesis, University of Stirling, 2011.

[NSHK10] Muaz A. Niazi, Qasim Siddique, Amir Hussain, and Mario Kolberg. Verification & validation of an agent-based forest fire simulation model. In *Proceedings of the 2010 Spring Simulation Multiconference*, SpringSim '10, pages 1:1–1:8, San Diego, CA, USA, 2010. Society for Computer Simulation International.

[Pic13] Philipp Pichler. Simulation und validierung von agentenbasierten modellen für die ausbreitung von epidemien. Master's thesis, Technische Universität Wien, Mai 2013.

[PP15] Niki Popper and Philipp Pichler. *Agent-based Modeling and Simulation in Archaeology*, chapter Reproducibility, pages 77–98. Springer International Publishing, Cham, 2015.

[Rey87] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, August 1987.

[Sar10] R.G. Sargent. Verification and validation of simulation models. In *Proceedings of the 2010 Winter Simulation Conference (WSC)*, pages 166–183, Dec 2010.

[SF80] Peter M Senge and Jay W Forrester. Tests for building confidence in system dynamics models. *System dynamics, TIMS studies in management sciences*, 14:209–228, 1980.

[Sha98] Robert E. Shannon. Introduction to the art and science of simulation. In *Proceedings of the 30th Conference on Winter Simulation*, WSC '98, pages 7–14, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[STG12] Ramzi Suleiman, Klaus G Troitzsch, and Nigel Gilbert. *Tools and techniques for social science simulation*. Springer Science & Business Media, 2012.

[Tro04] Klaus G Troitzsch. Validating simulation models. In *18th European Simulation Multiconference. Networked Simulations and Simulation Networks*, pages 265–270, 2004.

[ZKP00] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2000.