HAW HAMBURG

MASTERTHESIS
Jan-Niklas Voß

Konzeption und Entwicklung einer didaktischen Modellierungs- und Simulationsplattform für MARS

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Jan-Niklas Voß

Konzeption und Entwicklung einer didaktischen Modellierungs- und Simulationsplattform für MARS

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Eingereicht am: 10. Januar 2022

Jan-Niklas Voß

Thema der Arbeit

Konzeption und Entwicklung einer didaktischen Modellierungs- und Simulationsplattform für MARS

Stichworte

Multiagentensysteme, Simulation, Modellierung

Kurzzusammenfassung

Das aufstrebende Forschungsumfeld der Multi-Agenten-Simulation wird in verschiedenen Kursen an der Hochschule für angewandte Wissenschaften Hamburg anhand des von der MARS-Forschungsgruppe entwickelten Framework gelehrt. Hierbei sehen sich die Studierenden sowohl mit der Komplexität des Themas selbst als auch mit der Komplexität der Werkzeuge konfrontiert. Um den Lernprozess zu vereinfachen, wurde im Rahmen dieser Arbeit eine Plattform konzipiert und implementiert, die nachweislich in der Lage ist, die Studierenden ganzheitlich in der Simulation von MARS-Modellen zu unterstützen.

Jan-Niklas Voß

Title of Thesis

Design and Development of a Didactic Modeling and Simulation Platform for MARS

Keywords

Multiagent systems, simulation, modeling

Abstract

The emerging research area of multi-agent simulation is taught in various courses at the Hamburg University of Applied Sciences using the framework developed by the MARS research group. Through this process, students are confronted with both the complexity of the topic itself and the complexity of the tools. In order to simplify the learning process, a platform was designed and implemented in the context of this thesis, which is proven to be able to support students comprehensively in the simulation of MARS models.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungen

ABM Agenten-basierte Modellierung

AI Artificial Intelligence

API Application Programming Interface

AT Agent-Toolkit

CSS Cascading Style Sheets

CSUQ Computer System Usability Questionnaire

CSV Comma-separated values

DSL Domain-specific Language

FA funktionale Anforderung

GIS Geographic Information System

GUI Graphical user interface

HAW Hochschule für Angewandte Wissenschaften

HCI Human Computer Interaction

HTML Hypertext Markup Language

IDE Integrated Development Environment

IPC Interprocess communication

JS JavaScript

JSON JavaScript Object Notation

LSP Language Server Protocol

MARS Multi-Agent Research and Simulation

MAS Multiagentensystem

MOBIDYC MOdeling Based on Individuals for the DYnamics of Communities

MVP Model-View-Presenter Muster

NFA nicht-funktionale Anforderung

PID Process identifier

SDK Software Development Kit

SoC Separation of Concerns

UI User interface

UX User experience

XSS Cross-site-scripting

Lesehinweis

Zur besseren Lesbarkeit wird in der vorliegenden Arbeit auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Es wird entweder das Neutrum oder das generische Maskulinum verwendet, womit jedoch alle Geschlechter gleichermaßen gemeint sind.

1 Einleitung

Das Forschungsfeld um die Agenten-basierte Modellierung (ABM) und Multiagentensysteme (MAS) gewinnt in diversen Disziplinen immer mehr an Bedeutung, um komplexe Sachverhalte zu modellieren und vielschichtige Fragestellungen zu beantworten. Dabei ist dieses Fachgebiet mittlerweile nicht mehr nur für Wissenschaftler der Informatik interessant, sondern bspw. ebenso für Soziologen oder Ökologen, um komplexe Systeme zu modellieren (Bodine u. a., 2020, Janssen u. a., 2008). Zur Beantwortung der jeweiligen Forschungsfrage, werden innerhalb eines entsprechenden Systems autonome Einheiten (sog. Agenten) definiert, die ihre Handlungen von ihren individuellen Zielen, den Umwelteinflüssen und von anderen Agenten ableiten (Dorri u. a., 2018). MAS kommen zum Einsatz, wenn das zu betrachtende System überaus komplex, unvorhersehbar oder sehr rechenintensiv ist (Sycara, 1998).

Eine beispielhafte Aufgabe, die für ein MAS geeignet wäre, könnte wie folgt lauten: *Wie verhält sich der Straßenverkehr innerhalb eines definierten Bereiches, in dem die Straßenführung minimal verändert wird?* Um eine solche Frage zu beantworten, würde ein digitaler Klon eines entsprechenden Verkehrsnetzes mithilfe der zuvor genannten Agenten und weiteren Softwareentitäten modelliert werden. Die Agenten wären hier bspw. die Verkehrsteilnehmer – wie Autofahrer, Fußgänger und Radfahrer – die sich unter Beachtung definierter Regeln und anderer Verkehrsteilnehmer auf einem Straßenverkehrsnetz bewegen. Dieses Modell würde dann verwendet werden, um eine Simulation über einen festgelegten Zeitraum durchzuführen, wodurch Bewegungspunkte zu jeder Instanz eines Agenten erzeugt würden. Die Ergebnisse der Simulation könnten dann wiederum analysiert und für die Beantwortung der aufgestellten Forschungsfrage verwendet werden.

Für die Ausführung einer solchen Simulation muss das zu simulierende Szenario von seiner Domäne in ein ausführbares Agenten-basiertes Modell übersetzt werden. Die Modelle unterscheiden sich dabei je nach gewählter Technologie und den Werkzeugen in ihrer Definition stark. Während einige auf eine hohe Mächtigkeit mithilfe von bekannten Programmiersprachen wie Java setzen (Argonne National Laboratory, 2021), oder andere

eine eigene Domain-specific Language (DSL) entwickelt haben, um ihre Modelle mithilfe natürlicher Sprache beschreiben zu können (Tisue und Wilensky, 2004), setzen wiederum andere auf Malwerkzeuge, die eine graphische Beschreibung eines Modells erlauben (Bollen und Van Joolingen, 2013).

Im Rahmen der Forschungsgruppe Multi-Agent Research and Simulation (MARS) der Hochschule für Angewandte Wissenschaften (HAW) Hamburg wurde das gleichnamige .NET-Framework entwickelt. Mithilfe dieses Framework können Modelle in der objektorientierten Programmiersprache C# implementiert und simuliert werden. Des Weiteren beinhaltet das Framework eine Trennung zwischen dem Modell und dem zu simulierenden Szenario, welches mithilfe einer Datei im Format JSON beschrieben werden kann. Die Form der daraus resultierenden Ergebnisse ist abhängig von der zuvor erwähnten Konfiguration und rangiert vom einfachen Comma-separated values (CSV) Format bis zur Speicherung der Ergebnisse in Online-Datenbanken (MARS Group, 2021a).

Um diese aufstrebende Forschungsdisziplin weiter voranzubringen, lehren Mitglieder der Forschungsgruppe MARS dieses Themengebiet in mehreren Modulen (z.B. *Artificial Intelligence & Software Agents*) an der HAW und in internationalen Kooperationen an weiteren Hochschulen.

1.1 Problemstellung

Um den Studierenden im Rahmen der oben genannten Module praktische Eindrücke in das Themenfeld von MAS zu geben, wird das genannte MARS-Framework verwendet. Dabei wurde den Mitgliedern der Forschungsgruppe deutlich, dass der Einstieg für die Studierenden durch die gleichzeitige Vermittlung der Lerninhalte und der Vermittlung des Umgangs mit den vielzähligen Werkzeugen erschwert wird. Schwierigkeiten in der Vermittlung von MAS bzw. ABM konnten ebenso in den Untersuchungen von Bodine u. a. (2020) und Janssen u. a. (2008) festgestellt werden.

Da es sich bei den Modellen innerhalb des Framework um standardmäßige .NET-Projekte handelt, wird zum einen das .NET-Software Development Kit (SDK) und zum anderen optimalerweise eine Integrated Development Environment (IDE) benötigt, welche bei der Entwicklung in Form von Syntax-Validierung und weiteren Funktionen unterstützt. Um die zuvor erwähnte Konfiguration zu bearbeiten, kann ebenso ein Quelltext-Editor

vorzugsweise mit Unterstützung des JSON-Formats verwendet werden. Dies sorgt allerdings nur für eine syntaktische Validierung in Form von Zeichensetzung, überprüft aber nicht die semantische Korrektheit einer solchen Konfiguration. Die Felder der Konfiguration haben ein vorgeschriebenes Schema, welches zum einen die Datentypen, aber auch Bedingungen definiert. Innerhalb einer JSON-Datei kann in ein Feld bspw. eine -1 oder ein String `"yes"` eingetragen werden. Diese Werte können syntaktisch korrekt sein, können jedoch aufgrund der Semantik zur Laufzeit der Simulation Fehler produzieren, falls bspw. eine natürliche Zahl erwartet wird. Dies führt ggf. zu Irritationen auf Seiten des Modellierenden, da die verwendete IDE keine Fehler anzeigt, die Simulation aber dennoch fehlgeschlagen ist. Des Weiteren bestehen mittlerweile eine große Anzahl an Konfigurationsmöglichkeiten, die innerhalb einer einfachen IDE weder erwähnt, noch erklärt werden. Um außerdem die Ergebnisse der Simulation im Anschluss zu analysieren, bietet sich eine Visualisierung an, die jedoch ebenfalls nicht von der IDE, sondern von externen Werkzeugen übernommen werden muss. Die Studierenden müssen also eine Vielzahl an Werkzeugen installieren und verstehen.

Darüber hinaus existiert die Komplexität der Fachlichkeit des Themas MAS und des MARS-Framework selber. Dazu gehört unter anderem die Orientierung im Modellierungszyklus, welcher aus vielfachen Iterationen der folgenden Aktionen besteht: Code schreiben, Szenario konfigurieren, Simulation ausführen und Ergebnisse analysieren. Da diese Schritte innerhalb einer einfachen IDE meist in einem einzigen Fenster geschehen, erschwert dies den Studierenden die Unterscheidung und Orientierung zwischen den verschiedenen Phasen. Außerdem liefert das MARS-Framework verschiedene Schnittstellen zur Definition von Objekten, die innerhalb der Simulation verwendet werden. Neben den Agenten gehören dazu auch sog. Entities, die einfache Objekte innerhalb der Simulation darstellen, sowie sog. Layer, welche die Umwelt definieren, in der sich die Agenten und die Entities bewegen. Dabei ist den Studierenden zu Anfang häufig unklar, welche Typen für die jeweilige Aufgabe verwendet und wie diese implementiert werden müssen.

1.2 Zielsetzung

Im vorherige Abschnitt wurden einige Probleme aufgezeigt, die während der Vermittlung von Lerninhalten bzgl. des Themas MAS in Bezug auf das MARS-Framework aufgetreten sind. Diese Probleme gilt es innerhalb dieser Arbeit zu lösen, um den Lernprozess der

Studierenden zu verbessern. Vor diesem Hintergrund wurde die folgende Forschungsfrage formuliert:

> Wie können fachfremde Benutzer des MARS-Framework ganzheitlich in der Simulation von Modellen unterstützt werden?

Um diese forschungsleitende Frage zu beantworten, werden in Kapitel 4.2 vier Hypothesen aufgestellt, die zum einen den Ergebnissen der Literaturrecherche aus Kapitel 3 sowie den identifizierten Lernbedarfen aus Kapitel 4.1.1 entspringen. Zugrunde dieser Hypothesen liegt die der Literaturrecherche entnommene Annahme, dass eine didaktische Lernplattform in Form einer Anwendung mit einer graphischen Benutzeroberfläche zum Lernerfolg der Studierenden beitragen kann. Dementsprechend wurden folgende Hypothesen formuliert:

Hypothese 1. Die Verwendung einer Lernplattform fördert das Verständnis von den verschiedenen Objekten (Agent, Layer, Entität).

Hypothese 2. Die Verwendung einer Lernplattform fördert das Verständnis von den verschiedenen Phasen der Multi-Agenten-Simulation (Modellierung, Konfiguration, Simulation, Analyse).

Hypothese 3. Die Verwendung einer Lernplattform führt dazu, dass Fehler innerhalb der Modellierung oder Parametrisierung vom Benutzer schneller identifiziert und behoben werden.

Hypothese 4. Die Verwendung einer Lernplattform führt dazu, dass seltener Experten zur Unterstützung benötigt werden.

Im Rahmen dieser Arbeit wurde eine entsprechende Lernplattform konzipiert sowie implementiert und im Anschluss die Hypothesen anhand von Umfrageergebnissen eines Nutzertests untersucht. Neben der effektiven Vermittlung von Lehrinhalten wurde bei der Konzeption auf die Langlebigkeit der Lernplattform geachtet, da diese in Zukunft in Lehrveranstaltungen an der HAW und in Kooperation an weiteren Hochschulen eingesetzt werden soll.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: Zuerst werden in Kapitel 2 die theoretischen Grundlagen zum Verständnis dieser Arbeit geschaffen. Dazu zählt zum einen die Einführung in das Thema MAS und zum anderen in das Thema Usability innerhalb der Anwendungsentwicklung, welche im Folgenden eine bedeutende Rolle einnehmen wird. Im Rahmen des Kapitels *Verwandte Arbeiten* wird das MARS-Framework detailliert erläutert, existierende Modellierungswerkzeuge in der Bildung aufgezeigt und für die Konzeption relevante Anforderungen herausgearbeitet, welche die Literatur an Software im Bildungsbereich stellt. Anschließend beschäftigt sich Kapitel 4 mit der Analyse der Lernziele und der daraus resultierenden Hypothesen. Des Weiteren werden sowohl funktionale als auch nicht-funktionale Anforderungen an die Lernplattform gestellt, die sich aus der Analyse von typischen Anwendungsfällen ergeben. Im Rahmen der Analyse werden diverse Entwürfe zum Design der Benutzeroberfläche (sog. Mockups) vorgestellt und begründet. Kapitel 5 stellt die architektonischen Konzepte der Implementierung der Lernplattform u.a. in Form des von Starke (2015) definierten Sichtenmodells vor. In Kapitel 6 wird auf Details der Implementierung, wie bspw. die gewählten Technologien oder das interne Datenmanagement eingegangen. Innerhalb von Kapitel 7 werden die in der Analyse gestellten Anforderungen auf ihre Umsetzung überprüft. Die resultierende Lernplattform wird innerhalb des Kapitels 8 evaluiert. Dies geschieht in Form einer Nutzerumfrage, dessen Konzept und Aufbau im Detail erläutert werden. Auf Basis der Umfrageergebnisse werden die Hypothesen im Rahmen der *Diskussion* innerhalb des Kapitels *Evaluation des MARS-Explorer* verifiziert bzw. falsifiziert und die Forschungsfrage beantwortet. Das letzte Kapitel fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf die Zukunft der Lernplattform.

2 Grundlagen

Das folgende Kapitel erklärt die Grundlagen der MAS, welche den Grundbaustein für den weiteren Verlauf dieser Arbeit darstellen. Außerdem wird der Begriff der Usability im Detail erläutert, da es sich dabei um einen bedeutsamen Faktor für die Konzeption, Entwicklung und Evaluation handeln wird.

2.1 Multiagentensystem

Die übergeordnete Forschungsdisziplin von MAS ist die Distributed Artificial Intelligence. Diese beschäftigt sich damit, wie komplexe Fragestellungen mithilfe von Artificial Intelligence (AI) gelöst werden können. Ein Ansatz, um solche komplexen Probleme zu lösen, sind MAS (Dorri u. a., 2018). Im Gegensatz zum Distributed Problem Solving, bei dem die komplexe Aufgabe in kleine Teile heruntergebrochen, auf mehrere Knoten verteilt und die Ergebnisse in einem Hauptknoten gesammelt und ausgewertet werden (Wooldridge, 2009, S. 10), setzen MAS auf verteilte autonome Entitäten, welche als Agenten bezeichnet werden (Sycara, 1998).

Eine Definition der Charakteristiken eines MAS wurde von Sycara (1998) aufgestellt:

Definition 2.1.1 (Charakteristiken eines Multiagentsystem nach Sycara (1998))**.** [...] (1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (2) there is no system global control; (3) data are decentralized; and (4) computation is asynchronous.

2.1.1 Abgrenzung

Die ABM ist ein Forschungsfeld, welches eng mit MAS verwandt ist. Aufgrund einer fehlenden klaren Abgrenzung der beiden Begriffe, werden diese in der Literatur häufig gleichgesetzt und als Synonyme verwendet (Niazi und Hussain, 2011, Macal, 2016). Laut

Macal (2016) beschäftigt sich MAS vor allem mit dem Design und der Entwicklung, während es bei ABM um das Erklären eines auftretenden Verhaltens eines komplexen Systems und das Ableiten eines Regelwerks für die beteiligten Agenten geht (Bodine u. a., 2020).

2.1.2 Agenten

Eine allgemein akzeptierte Definition eines Agenten existiert nicht, da diese immer stark vom Kontext abhängig ist (Macal, 2016). Mit Definition 2.1.2 haben Dorri u. a. (2018) in ihrem Survey aber eine möglichst allgemeingültige Definition aufgestellt.

Definition 2.1.2 (Agent nach Dorri u. a. (2018))**.** An entity which is placed in an environment and senses different parameters that are used to make a decision based on the goal of the entity. The entity performs the necessary action on the environment based on this decision.

Nach Wooldridge und Jennings (1995) erfüllen Agenten außerdem die folgenden Eigenschaften:

autonomy Verwalten ihren eigenen Zustand und funktionieren ohne jegliche Fremdeinwirkung.

social ability Kommunizieren mit anderen Agenten, ihrer Umwelt und ggf. dem Benutzer.

reactivity Reagieren auf ihre Umwelt und zeitnah auf Änderungen, die in dieser auftreten.

pro-activeness Handeln proaktiv und ergreifen Initiative, um ihr individuelles Ziel zu erreichen.

2.1.3 Anwendung

Die Anwendung von MAS bzw. ABM findet sich in diversen Forschungsdisziplinen wie bspw. der Archäologie, Biologie oder auch der Informatik wieder (Macal und North, 2013), weshalb das Thema immer mehr Aufmerksamkeit in diversen naturwissenschaftlichen Hochschulkursen erlangt (Bodine u. a., 2020).

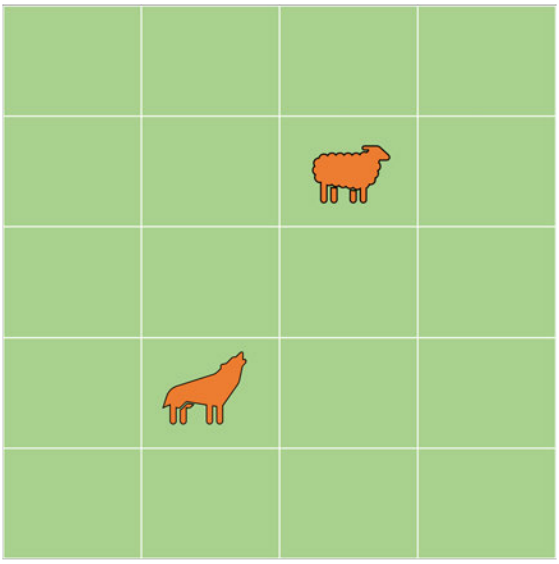


Abbildung 2.1: Beispiel-Darstellung eines Multiagentensystems anhand eines einfachen Wolf-Schaf-Models

Als Beispiel für ein solches MAS kann die Abbildung 2.1 betrachtet werden. In diesem einfachen Modell existiert je eine Instanz der Agenten-Typen Schaf und Wolf in einer $4x4$ Matrix, die mit Gras bedeckt ist. Beide Agenten bewegen sich selbstständig innerhalb ihrer Umgebung und halten einen Zustand, der ihr Energie-Level beschreibt (autonomy). Das Energie-Level verringert sich mit jeder Bewegung und kann je nach Agenten-Typ auf verschiedene Art und Weise erhöht werden. Im Fall des Schafs erhöht sich dessen Wert durch das Fressen von Gras, welches sich in der Umgebung befindet; im Fall des Wolfs wiederum kann das Energie-Level nur durch das Fressen von Schafen aufgefüllt werden, was dazu führt, dass der Schaf-Agent aus dem System entfernt wird (social ability). Befindet sich ein Schaf im Umfeld von zwei Feldern des Wolfs, bewegt sich dieser auf das Schaf zu (proactivity), während das Schaf sich daraufhin wegbewegt (reactivity).

2.2 Usability

Anwendungen, die Lehrinhalte vermitteln wollen, stehen immer vor dem zentralen Problem, dass dem Benutzer während der Verwendung zwei Themen gleichzeitig vermittelt werden, welche um die kognitive Aufmerksamkeit des Benutzers konkurrieren: Zum einen die Lehrinhalte und zum anderen die Handhabung der Benutzeroberfläche (Baddeley, 1981). Aus diesem Grund kommt es vor, dass eine solche Anwendung relevante

Informationen für den Benutzer beinhaltet, der Lernerfolg aber dennoch aufgrund von Schwierigkeiten mit der Benutzeroberfläche gebremst wird (Parlangeli u. a., 1999).

Die Usability ist ein Faktor aus der Forschungsdisziplin der Human Computer Interaction (HCI), welcher dazu verwendet werden kann, die Qualität einer Benutzeroberfläche zu bemessen (Moore und Redmond-Pyle, 1995). Die ISO 9241:11 (2018) definiert Usability folgendermaßen:

Definition 2.2.1 (Usability nach ISO 9241:11 (2018))**.** The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context.

Effectiveness Accuracy and completeness with which users achieve specified goals.

Efficiency Resources used in relation to the results achieved.

Satisfaction Extent to which the user's physical, cognitive and emotional responses that result from the use of a system, product or service meet the user's needs and expectations.

Studien mehrerer Autoren aus diversen Domänen im Kontext von MAS oder ABM heben den Faktor der Usability ebenfalls als ausschlaggebend für den Lernerfolg hervor (Serenko und Detlor, 2002, Sandars und Lafferty, 2010, Bressan u. a., 2017). Nach Grudin (1992) ist neben der Usability, die Utility für die Akzeptanz des Systems verantwortlich. Beide Aspekte lassen sich folgendermaßen beschreiben: Utility, ob die geforderten Funktionen des Systems prinzipiell ausgeführt werden können; und Usability, wie gut die Funktionen genutzt werden können (Grudin, 1992). Nielsen (1994b, S. 25) unterstützt diese These durch seine Beobachtungen. Er stellt fest, dass ein System, welches zwar alle Funktionen bietet, dessen Benutzeroberfläche aber zu schwierig zu bedienen ist, genau so wenig Mehrwert bietet, wie ein System, welches eine hervorragende Benutzeroberfläche besitzt, aber dessen Funktionen nicht ausreichend sind. Zusammen bilden die Usability und Utility die Usefulness eines Systems (Grudin, 1992).

In einer aktuelleren Veröffentlichung unterteilt Nielsen (2012) die Faktoren der Usability etwas detaillierter. Das Modell, welches in Abbildung 2.2 dargestellt ist, zeigt das Modell der Usefulness nach Nielsen. Er beschreibt diese Aspekte folgendermaßen:

Learnability How easy is it for users to accomplish basic tasks the first time they encounter the design?

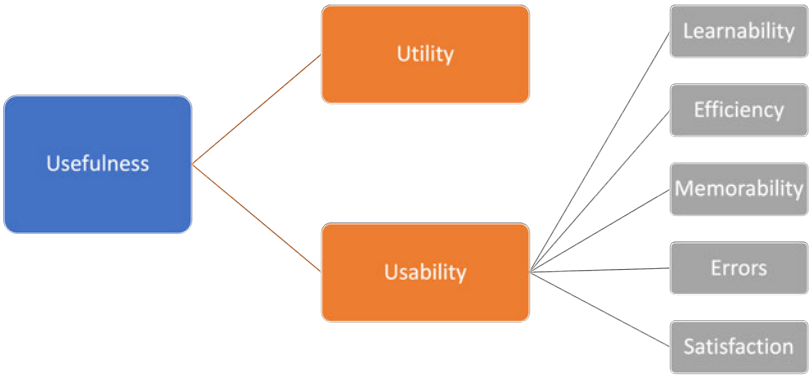


Abbildung 2.2: Modell der Usefulness eines Systems in Anlehnung an Nielsen (2012)

Efficiency Once users have learned the design, how quickly can they perform tasks?

Memorability When users return to the design after a period of not using it, how easily can they reestablish proficiency?

Errors How many errors do users make, how severe are these errors, and how easily can they recover from the errors?

Satisfaction How pleasant is it to use the design?

Um ein System bzw. dessen Benutzeroberfläche hinsichtlich der Usability zu überprüfen, wurde von Nielsen und Molich das Konzept der *Heuristischen Evaluation* eingeführt. Zu diesem Zwecke wurden ursprünglich neun Heuristiken definiert, die sich mit verschiedenen Aspekten einer Benutzeroberfläche beschäftigen (Nielsen und Molich, 1990). Diese Heuristiken können im Rahmen der *Heuristischen Evaluation* als Checklist durchlaufen werden, um die Usability eines Systems zu bewerten. Eine detaillierte Erläuterung des Konzeptes findet in Kapitel 8 statt.

3 Verwandte Arbeiten

Innerhalb dieses Kapitels werden für diese Ausarbeitung relevante Arbeiten vorgestellt. Dazu gehört zum einen das zuvor mehrfach erwähnte MARS-Framework, welches den Grundbaustein für die gesamte Ausarbeitung legt. Zum anderen werden existierende Lernplattformen zum Thema MAS vorgestellt und aufgezeigt, wie sich diese von der in dieser Arbeit zu entwickelnden Plattform unterscheiden. In einem weiteren Abschnitt werden verschiedene Untersuchungen und Leitfäden zu Bildungssoftware vorgestellt, die den Rahmen für die Konzeption und die Entwicklung der Lernplattform liefern.

3.1 MARS-Framework

Das MARS Framework wird von der gleichnamigen Forschungsgruppe an der HAW Hamburg entwickelt und findet in diversen Forschungsprojekten Anwendung (MARS Group, 2021b,c). Es handelt sich dabei um ein Agenten-basiertes Framework zur Implementierung eines MAS mit dem hoch-skalierbare und verteilte Simulationen ausgeführt werden können (Hüning u. a., 2016). Das Framework ist in C# implementiert und kann mithilfe der .NET Runtime ausgeführt werden.

3.1.1 Motivation

In ihrer ersten Veröffentlichung zum MARS-Framework identifizierten Hüning u. a. (2014) verschiedene Probleme, die mit den bestehenden Tools zur ABM einhergehen, wie u.a. die Skalierbarkeit der Systeme oder Einfachheit der Benutzung. Laut Hüning u. a. müsse ein geeignetes Tool dem Benutzer bspw. eine einfache Benutzung des Systems bieten, indem ein Domänen-spezifisches Modell auf einfache Art und Weise in das jeweilige Simulationssystem integriert werden kann. Um die identifizierten Probleme zu lösen, wurde in der ersten Iteration das MARS-Framework implementiert, welches in zwei Komponenten aufgeteilt war: Zum einen das Simulationssystem mit dem Namen LIFE, welches

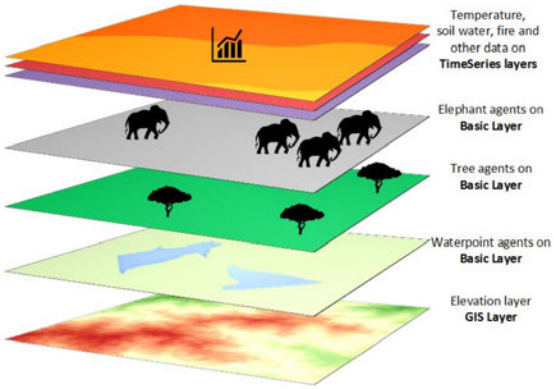


Abbildung 3.1: Beispielhafter Aufbau eines afrikanischen Nationalparks abgebildet in der ersten Iteration des MARS-Framework aus Hüning u. a. (2016, S. 4)

für die Ausführung der Simulation verantwortlich war und zum anderen der MARS-Websuite, welcher eine benutzerfreundliche Web-GUI darstellt und von den Benutzern des Framework verwendet werden kann, um ihre Daten zu verwalten, die Simulation zu konfigurieren, zu starten und die Ergebnisse letztendlich zu visualisieren (Hüning u. a., 2016). Das eigentliche Modell muss zuvor vom Domänen-Experten konzipiert und von einem Informatiker mithilfe einer IDE in ein MARS-Modell übersetzt werden, woraufhin es im Websuite hochgeladen und verwendet werden kann (Hüning u. a., 2016).

3.1.2 Modellierung

Das Konzept des MARS-Framework stützt sich neben den in Kapitel 2.1.2 erläuterten Agenten, auf so genannte Layer, auf denen die Agenten platziert und Umweltdaten gehalten werden. Agenten können mit den Schnittstellen der Layer interagieren. Layer basieren auf dem selben Prinzip wie die Datenanordnung innerhalb eines Geographic Information System (GIS), in dem unterschiedliche Layer Teilinformationen des Gesamtsystems halten. Zur Zeit der ersten Iteration von MARS existierten drei Typen von Layern, die von Hüning u. a. (2016) definiert worden sind:

Basic-Layer Bietet keine eigenen Funkionen und muss vom Benutzer selbst implementiert werden.

TimeSeries-Layer Ermöglicht die Speicherung von Zeitreihen und Abfragen nach Zeit und Position.

GIS-Layer Ermöglicht die Verwendung von GIS-Daten und geometrische sowie spatiale Abfragen.

In Abbildung 3.1 ist der Aufbau eines beispielhaften MARS-Modells dargestellt. In diesem Beispiel existieren die drei Agententypen *Waterpoint*, *Tree* und *Elephant*, welche jeweils auf einem Basic Layer platziert worden sind. Neben den Basic Layern existieren außerdem noch ein TimeSeries Layer, um diverse Umweltbedingungen zu speichern, sowie ein GIS Layer für weitere spatiale Daten.

In der zweiten Iteration des Framework wurde sich u.a. dem Modellierungs-Prozess angenommen, welcher im vorherigen Ansatz zum einen den Domänen-Experten und zum anderen einen Informatiker benötigte. Der Domänen-Experte entwarf das konzeptuelle Modell, welches dann aufgrund der technischen Komplexität vom Informatiker zum MARS-Modell transformiert worden ist. Um diese Transformation den Domänen-Experten zugänglicher zu machen, wurde in der zweiten Iteration des MARS-Framework eine eigene DSL entwickelt. Für komplexere Logiken innerhalb eines Modells, musste dennoch auf eine Implementierung mithilfe von C# zurückgegriffen werden. Aus diesem Grund und den zeitlich hohen Wartungskosten wurde sich letztendlich gegen die Weiterentwicklung der DSL entschieden. Auch Teil der zweiten Iteration des MARS-Framework ist die Einführung eines weiteren Typs neben Agent und Layer: Entity. Dieser Typ verhält sich wie ein Agent, nur dass dieser nicht bei jedem Simulationsschritt eine Aktion ausführt (MARS Group, 2021a). Dadurch eignen sich Entities zur Implementierung von statischen Objekten, wie bspw. Autos, die lediglich von Agenten verwendet werden und von sich aus keine Aktion ausführen können.

3.1.3 Konfiguration

Neben der Möglichkeit zur Definition von Modellen, bietet das Framework außerdem ein Laufzeitsystem mit dem das Modell simuliert werden kann, sowie eine Möglichkeit zur Konfiguration dieser Simulationsläufe. Diese Konfiguration kann mithilfe von Code oder einer JSON Datei beschrieben werden, wobei letzteres aufgrund der Übersicht und der Separation of Concerns (SoC) die präferierte Methode darstellt. Das Schema der Konfiguration ist ausführlich in der Dokumentation des MARS-Framework erläutert (MARS Group, 2021a).

Listing 3.1: Beispielhafte Konfiguration einer Simulation als JSON

```
1 {
```

```
2   "globals": {
3     "steps": 500,
4     "deltaT": 1,
5     "output": "csv",
6   },
7   "agents": [
8     {
9       "name": "Sheep",
10      "count": 50,
11      "mapping": [
12        {
13          "parameter": "SheepGainFromFood",
14          "value": 5
15        }
16      ]
17    }
18  ],
19  "layers": [
20    {
21      "name": "GrasslandLayer",
22      "file": "Resources/grid.csv"
23    }
24  ]
25 }
```

In Listing 3.1 ist eine solche Konfiguration beispielhaft dargestellt. In dieser werden von Zeile 2 bis 6 zunächst globale Einstellungen getroffen, wie die Anzahl der Simulationsschritte und die Ausgabemethode der Ergebnisse. Von Zeile 7 bis 18 können Einstellungen zu den Agenten innerhalb des Modells getroffen werden. Die hier gezeigten Einstellungen beinhalten die Konfiguration des Agenten *Sheep*. In Zeile 10 wird bspw. festgelegt, dass zu Beginn der Simulation 50 Instanzen dieses Agenten existieren werden. Innerhalb des Modells können Attribute als Parameter gekennzeichnet werden, wodurch in Zeile 13 dem Attribut *SheepGainFromFood* der Wert *5* zugewiesen werden kann. Ähnlich funktioniert auch die Konfiguration der im Modell vorkommenden Layer. In diesem Fall wird zur Initialisierung des *GrasslandLayer* allerdings die Datei *grid.csv* verwendet, um eine umfangreiche Zuweisung der Werte zu vereinfachen.

3.1.4 Ausführung

Nachdem die Modellierung und die Konfiguration abgeschlossen sind, kann das im Framework enthaltene Laufzeitsystem dazu verwendet werden, eine Simulation des Modells auszuführen. Für die Ausführung wird eine Installation der .NET Runtime auf dem Zielsystem benötigt. Die Form der Ergebnisse unterscheidet sich je nach konfigurierter Ausgabe. Während einer Simulation ist es möglich per Websocket auf die Ergebnisse zur Laufzeit zuzugreifen. Sofern nicht anderweitig konfiguriert, werden zu den Objekten innerhalb einer Simulation verschiedene Werte gespeichert, die von den Parametern und implementierten Schnittstellen abhängen.

3.1.5 Anwendung in Forschungsprojekten

Wie einleitend erwähnt, findet das MARS-Framework in unterschiedlichen Domänen Einsatz. Eine dieser Domänen ist der Verkehr, welcher innerhalb von Hamburg mithilfe des Framework in Form unterschiedlichster Agenten (Autos, Fußgänger, Radfahrer, ...) modelliert wird. Ergebnisse der Simulation dieses Modells werden verwendet, um in einer vorangegangenen Analyse Optimierungen am Verkehr vorzunehmen und somit das Verkehrsnetz der Stadt Hamburg nachhaltig zu entlasten (MARS Group, 2021b). Ein weiteres Forschungsprojekt ist die Modellierung des Ökosystems der Savanne in Südafrika. Mit dem Modell sollen die Einflüsse und die Beziehungen diverser Faktoren, die innerhalb dieses Ökosystems wirken, untersucht werden (MARS Group, 2021b, Hüning u. a., 2016).

3.2 Agenten-basierte Systeme in der Bildung

Das letztendliche Ziel dieser Arbeit ist es, die resultierende Plattform im Rahmen von verschiedenen Lehrveranstaltungen einzusetzen, um somit das Thema von MAS effektiver vermitteln zu können. Ähnliche Ziele haben sich auch andere Plattformen gesetzt, die sich mit MAS bzw. ABM beschäftigen. In der durchgeführten Studie von Abar u. a. (2017) wurden alle bis zum Jahr 2017 bestehenden MAS- bzw. ABM-Plattformen untersucht und ihnen diverse Charakteristiken zugeordnet. Innerhalb dieser Einordnung wurden von Abar u. a. 18 Werkzeuge identifiziert, die sich auf den Bereich Education bzw. Teaching konzentriert haben, wovon drei vorgestellt werden sollen. Diese drei Werkzeuge wurden

bewusst ausgewählt, da sie unterschiedliche Herangehensweisen an das Thema von MAS in Bezug auf die Bildung aufzeigen.

3.2.1 NetLogo

Eines der prominentesten und am weitesten verbreiteten Werkzeuge aus dieser Domäne ist NetLogo, welches aus zwei Komponenten besteht: Einer DSL, die auf auf dem Dialekt Logo der Programmiersprache LISP basiert und die zur Beschreibung der Modelle verwendet werden kann. Sowie einer graphischen Modellierungsumgebung, mit der Simulationen konfiguriert, ausgeführt und ausgewertet werden können (Tisue und Wilensky, 2004). Laut Tisue und Wilensky liegt der Fokus von NetLogo auf Einfachheit, sodass selbst Studierende oder Forschende ohne professionelle Programmiererfahrung ihre eigenen Modelle erstellen können. Als grundlegendes Konzept verwendet NetLogo so genannte *turtles* und *patches*, welche das Pendant zu den in Kapitel 3.1 erläuterten Agenten und Layern des MARS-Framework darstellen.

Die Abbildung 3.2 zeigt die graphische Benutzeroberfläche von NetLogo nach dem Öffnen des Beispielmodells *Rabbits Grass Weeds*. In der obersten Zeile bietet die Oberfläche eine Möglichkeit zwischen diversen Beispielmodellen aus verschiedenen Domänen zu wählen oder ein eigenes Modell hochzuladen. Unterhalb dessen kann der Faktor der Simulationsgeschwindigkeit und die Anzahl der Agenten innerhalb der Simulation angepasst werden. Mithilfe von weiteren Reglern können unterhalb der Buttons *Setup* und *Go* die verschiedenen Eigenschaften der *turtles* und *patches* konfiguriert werden. In der zweidimensionalen Grafik rechts von den Einstellungsmöglichkeiten findet die Live-Visualisierung der Simulationsergebnisse statt. In diesem Beispiel sind die Hasen-Turtles als Hasenköpfe dargestellt, während die Grass-Patches als grüne Pixel markiert sind. Eine weitere Visualisierung in Form eines Live-Graphen kann unter den Einstellungsmöglichkeiten betrachtet werden. Im untersten Drittel des Fenster befindet sich ein Akkordeon-Element, welches dem Benutzer die Möglichkeit gibt, Befehle während der Simulation auszuführen (Command Center), das Modell zu überarbeiten (NetLogo Code) oder sich Informationen zu dem geöffneten Modell anzeigen zu lassen (Model Info).

3.2.2 SimSketch

Einen anderen Ansatz haben dagegen Bollen und Van Joolingen (2013). Das Problem der Lehre von ABM sei eine Kombination der folgenden Komplexitäten: der Tools, die

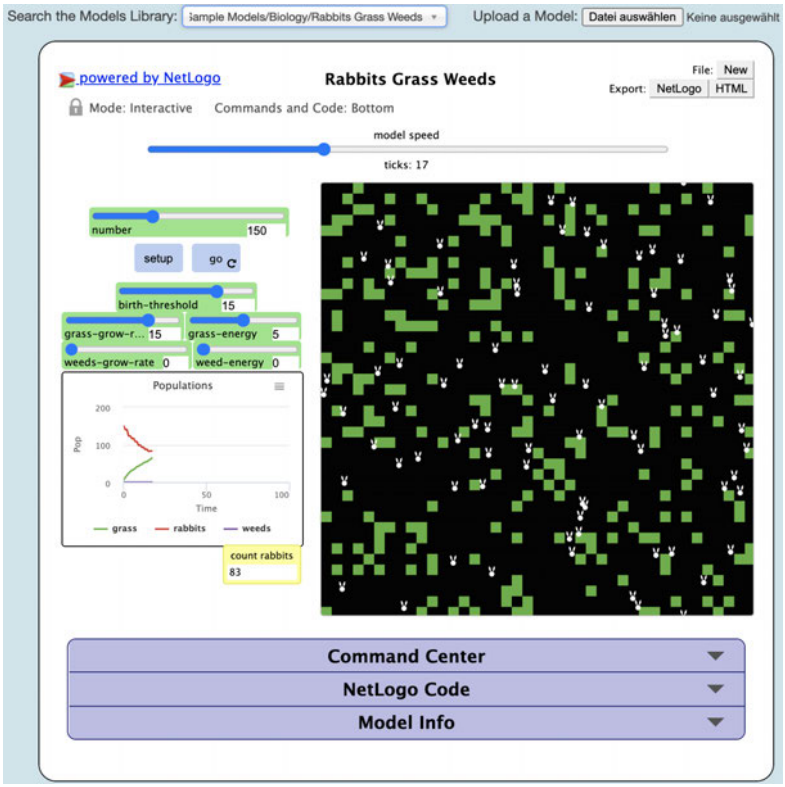


Abbildung 3.2: Graphische Benutzeroberfläche von NetLogo am Beispielmodell *Rabbit Grass Weeds* aus NetLogo (2021)

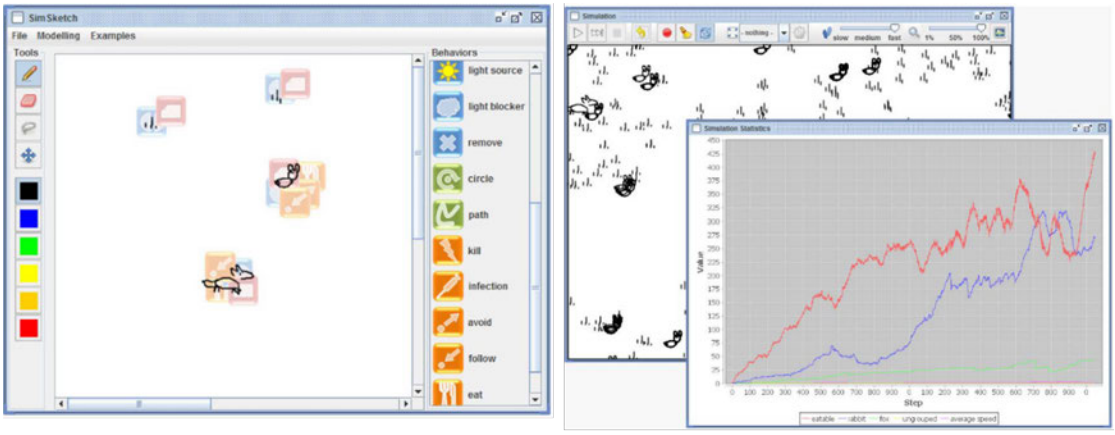


Abbildung 3.3: GUI von SimSketch am Beispielmodell *Predetor-Pray*. Links: Fenster zur Modellierung, Rechts: Darstellung der Simulationsergebnisse. Quelle: Bollen und Van Joolingen (2013, S. 214)

das Einarbeiten in die jeweiligen Syntax in Form von Programmcode, Gleichungen oder Grafiken erfordert; sowie der Konzepte des Modellierungsprozesses, welche die Iteration aus Modellierung, Ausführung und Bewertung der Resultate beinhaltet (Bollen und Van Joolingen, 2013, Louca u. a., 2011, Louca und Zacharia, 2012).

Mit SimSketch haben die Autoren ein Werkzeug entwickelt, welches vor allem Schülern in der Primärstufe dabei helfen soll, Modelle mithilfe von Zeichnungen (Sketches) zu definieren. Ainsworth u. a. (2011) konnten verschiedene Vorteile identifizieren, die das Lernen mithilfe des Zeichnens mit sich bringt. Dazu zählen u.a. eine erhöhte Motivation, Unterstützung bei der Organisation des eigenen Wissens und ein vereinfachter Ideenaustausch mit Anderen.

Die gezeichneten Modelle sind allerdings nur der Ausgangspunkt, der die im Modell bestehenden Objekte und dessen Beziehungen zueinander beschreibt. Die Definition des Verhaltens der verschiedenen Objekte geschieht mithilfe von vorgefertigten Bausteinen und dessen Parametrisierung in Form von Benutzereingaben innerhalb eines Dialogs. Die tatsächliche Simulation geschieht mithilfe eines Framework.

Die Oberfläche von SimSketch ist in Abbildung 3.3 zu sehen. Dort wird ein Modell vorgestellt, welches dem aus Kapitel 2.1.1 vorgestellten Wolf-Schaf-Model sehr ähnlich sieht. Hier werden jedoch Hasen anstelle von Schafen modelliert. Im linken Fenster ist die Zeichenfläche von SimSketch zu sehen, in der mithilfe der Werkzeuge an der linken Seite die Objekte gezeichnet werden können. In diesem Fall wurde ein Hasen-Agent, ein Wolf-

Agent und zwei Gras-Agenten gezeichnet. Die Vierecke, die leicht transparent hinter den Zeichnungen zu sehen sind, zeigen die Verhaltens-Bausteine, die dem jeweiligen Agenten zugewiesen worden sind. Die Bausteine wurden aus der rechten Leiste entnommen.

Das rechte Fenster in Abbildung 3.3 zeigt die Ergebnisse der Simulation in zwei Darstellungen: zum einen sind im Hintergrund die Ergebnisse anhand der eigenen Zeichnung angeordnet, zum anderen befindet sich im Vordergrund ein Liniendiagramm, welches die Population der Agenten-Typen in Form von verschiedenfarbigen Kurven darstellt.

SimSketch konzentriert sich auf Einfachheit, Intuition und Spaß, um Inhalte der ABM Schülern in der Primärstufe zu vermitteln. Es verzichtet dabei aber bewusst auf die wissenschaftlichen Genauigkeit und im Vergleich mit NetLogo auch auf Mächtigkeit und Flexibilität (Bollen und Van Joolingen, 2013).

3.2.3 MOBIDYC

MOdeling Based on Individuals for the DYnamics of Communities (MOBIDYC) ist eine auf der Programmiersprache *Smalltalk* basierende MAS-Plattform für Wissenschaftler außerhalb der Informatik zur Modellierung von Populationsdynamiken (Ginot und Le Page, 1998). Nach Ginot und Le Page sei die Rolle des Informatikers damals die Übersetzung des Domänen-Modells in ein ausführbares Computer-Modell und dessen Weiterentwicklung gewesen. Die Aufgabe der Informatiker hätte aber laut Ginot und Le Page vielmehr die Bereitstellung generischer Modelle sein müssen, die von den Wissenschaftlern anderer Domänen verwendet und selbst hätten verwaltet werden können.

Das Grundprinzip basiert auf biologischen Agenten, die i.d.R. eine Spezies zu einem gewissen Zeitpunkt seiner Entwicklung abbilden. Neben dem Alter, verfügen die Agenten standardmäßig über Attribute, wie einen Namen, die Position oder die Anzahl der Individuen, aus denen der jeweilige Agent zusammengesetzt ist (Ginot und Le Page, 1998). Die Attribute können auch dynamische Werte in Form von mathematischen Funktionen beinhalten, die bspw. das Gewicht eines Agenten in Abhängigkeit zum Simulationszeitraum darstellen. Zusätzlich zu den Attributen, können den Agenten sog. *Tasks* zugewiesen werden, welche das Verhalten der Agenten innerhalb der Simulation widerspiegeln. Für einfache Modelle existieren vorgefertigte Verhaltensweisen, wie bspw. die Metamorphose eines Agenten. Es können jedoch auch eigene *Tasks* in Form von mathematischen Funktionen definiert werden. Wie auch NetLogo verfügt MOBIDYC über eine Entität zur

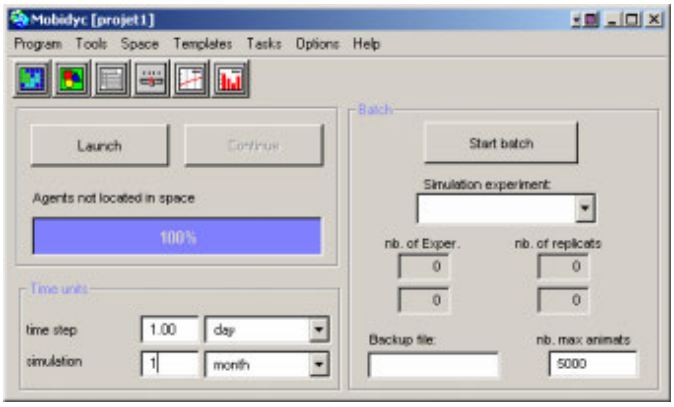


Abbildung 3.4: GUI von MOBIDYC: Start der Anwendung. Quelle: Ginot und Souissi (2003, S. 5)

Modellierung von Umgebungen (hier: *Environments*) mit der die Agenten interagieren können.

In Abbildung 3.4 ist das Startfenster der Plattform dargestellt. Innerhalb dieses Fenster können diverse Einstellungen bzgl. der Simulation getroffen werden. Mithilfe der verschiedenen Symbole unter dem Anwendungsmenü (*Programm*, *Tools*, etc.) können verschiedene Funktionen in Form weiterer Fenster ausgeführt werden. Dazu gehört bspw. das Fenster zu Erstellung einer Population, welches in Abbildung 3.5 gezeigt wird. Der Agent *Sugar* besitzt demnach die Attribute *age*, *number* und *location*, sowie den Task *Grow older*. Sind die vorgefertigten Tasks und Attribute für den jeweiligen Anwendungsfall nicht ausreichend, können diese mithilfe von Code in Smalltalk implementiert werden.

3.2.4 Einordnung

Genau so wie die vorgestellten Werkzeuge, hat die vorliegende Arbeit das Ziel, die resultierende Plattform im Rahmen von verschiedenen Lehrveranstaltungen einzusetzen, um somit das Thema von MAS effektiver vermitteln zu können. Der Unterschied zu den zuvor erläuterten System liegt in der unterliegenden Technologie und wie diese genutzt wird, um die Modelle zu beschreiben. Während NetLogo bspw. auf eine eigene DSL setzt, werden für SimSketch einfache Zeichnungen verwendet. MOBIDYC setzt wiederum auf eine umfangreiche GUI mit der die Modelle mithilfe von vielen Klicks zusammengestellt werden können. Das MARS-Framework hingegen ist ein reines Software-Framework, welches C# als Modellierungssprache und die .NET-Runtime als Ausführungsumgebung

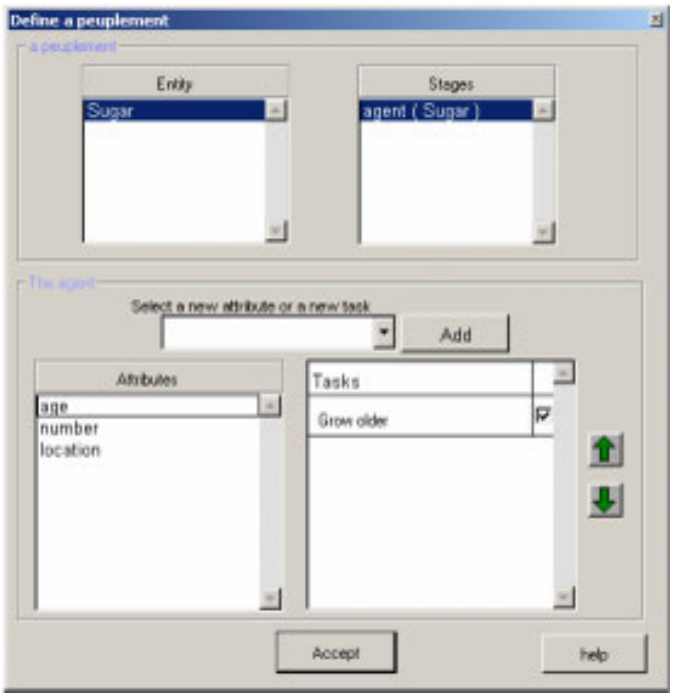


Abbildung 3.5: GUI von MOBIDYC: Definition einer Population. Quelle: Ginot und Souissi (2003, S. 13)

verwendet. Die tiefergehenden Gründe für die Entwicklung des MARS-Framework liegen außerhalb des Umfangs dieser Arbeit. Sie wurden jedoch in Kapitel 3.1 angerissen und können in Hüning u. a. (2014, 2016) sowie in Glake u. a. (2017) nachvollzogen werden.

3.3 Didaktische Faktoren einer Lernsoftware

Um einen positiven Effekt auf den Lernerfolg des Anwenders zu haben, müssen während der Konzeption und Entwicklung von Software im Bildungsbereich diverse Faktoren berücksichtigt werden. Viele Autoren beschäftigten sich in ihren Untersuchungen mit der Definition entsprechender Faktoren (Phillips, 1997, Beale und Sharples, 2002, Serenko und Detlor, 2002, Van Nuland u. a., 2017), die im Folgenden erläutert und als Grundlage für mehrere Anforderungen in der Analyse dienen werden. Die von den Autoren gestellten Anforderung unterteilen sich zum einen in Anforderung an das Design der Software und zum anderen an den Prozess, der von der Konzeption bis zur Auslieferung der Anwendung durchlaufen wird. In diesem Zusammenhang ist mit *Design* sowohl der visuelle als auch der architektonische Aufbau der Anwendung gemeint.

3.3.1 Design

Einer der wichtigsten Faktoren, der von Serenko und Detlor, Beale und Sharples und Van Nuland u. a. genannt wird, ist die Usability einer Anwendung. Eine detaillierte Erläuterung des Begriffs ist bereits in Kapitel 2.2 gegeben worden. Nach Van Nuland u. a. (2017) ist die Usability der Faktor, der aussagt, wie effektiv die Anwendung vom Benutzer verwendet werden kann und der somit ausschlaggebend den Lernerfolg mitbestimmt.

Die Relevanz des Faktors der *User Interaction*, welcher im weiteren Verlauf mit der Usability gleichgesetzt worden ist, wurde besonders von Serenko und Detlor (2002) in ihren Untersuchungen herausgearbeitet. Serenko und Detlor (2002) haben im Jahr 2002 existierende Agent-Toolkits (ATs) untersucht und wichtige Erkenntnisse darüber gesammelt, ob und wie Toolkits dieser Art am effektivsten in der Lehre von ABM unterstützen können. Solche ATs werden von Serenko und Detlor folgendermaßen definiert:

Definition 3.3.1 (Agent-Toolkit nach Serenko und Detlor (2002))**.** [...] any software package, application or development environment that provides agent builders with a sufficient level of abstraction to allow them to implement intelligent agents with desired attributes, features and rules.

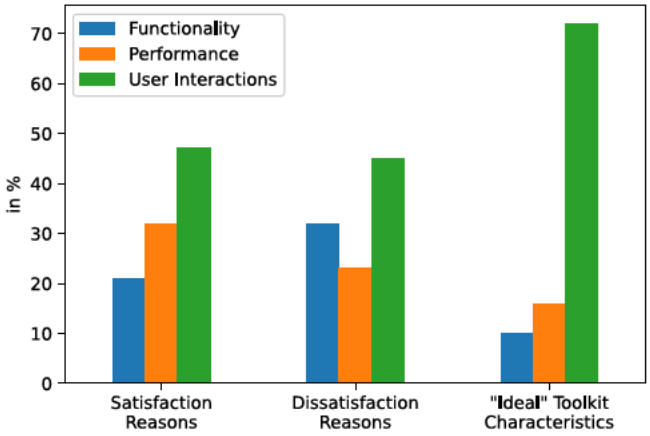


Abbildung 3.6: Zusammenfassung der Antworten der Lehrenden bezüglich verschiedener Usability Characteristics von ATs in Anlehnung an Serenko und Detlor (2002, S. 34)

Der Definition 3.3.1 zufolge kann die in dieser Arbeit zu entwickelnde Plattform mit einem AT gleichgesetzt werden.

Abbildung 3.6 zeigt die Ergebnisse einer Umfrage, die von Serenko und Detlor unter den Lehrenden von ATs durchgeführt worden ist. Daraus geht die User Interaction als wertvollster Faktor hervor, der sowohl den größten Einfluss auf die Zufriedenheit als auch auf die Unzufriedenheit hat. Außerdem wurde dieser Faktor mit 72% als am wünschenswertesten in einem AT erachtet.

Darüber hinaus liefern weiterführende Ergebnisse von Serenko und Detlor, die in Abbildung 3.7 dargestellt sind, weitere Erkenntnisse darüber, welche Features der User Interaction ausschlaggebend sind: Am wichtigsten erscheinen den Lehrenden mit 32% funktionsfähige Beispiele innerhalb des Toolkits. Mit einer minimal niedrigeren Gewichtung von 28% werden eine ausführliche Dokumentation, eine einfache Bedienung, sowie eine benutzerfreundliche Oberfläche angemerkt. Des Weiteren merkten die Lehrenden an, dass innerhalb eines solchen Toolkits viele Hilfestellungen bspw. in Form von Code-Beispielen oder Vorlagen zur Generierung von neuen Agenten gegeben werden sollten. Außerdem müsse ein solches Toolkit zugänglich gemacht werden, indem die Installation auf verschiedenen Systemen ermöglicht wird. Laut Serenko und Detlor (2002) ist es für die Lernenden ebenfalls von großem Vorteil, wenn die Toolkits ebenfalls Funktionen anbieten, die beim Ausführen, Überwachen, Analysieren und Testen von agenten-basierten Systemen helfen.

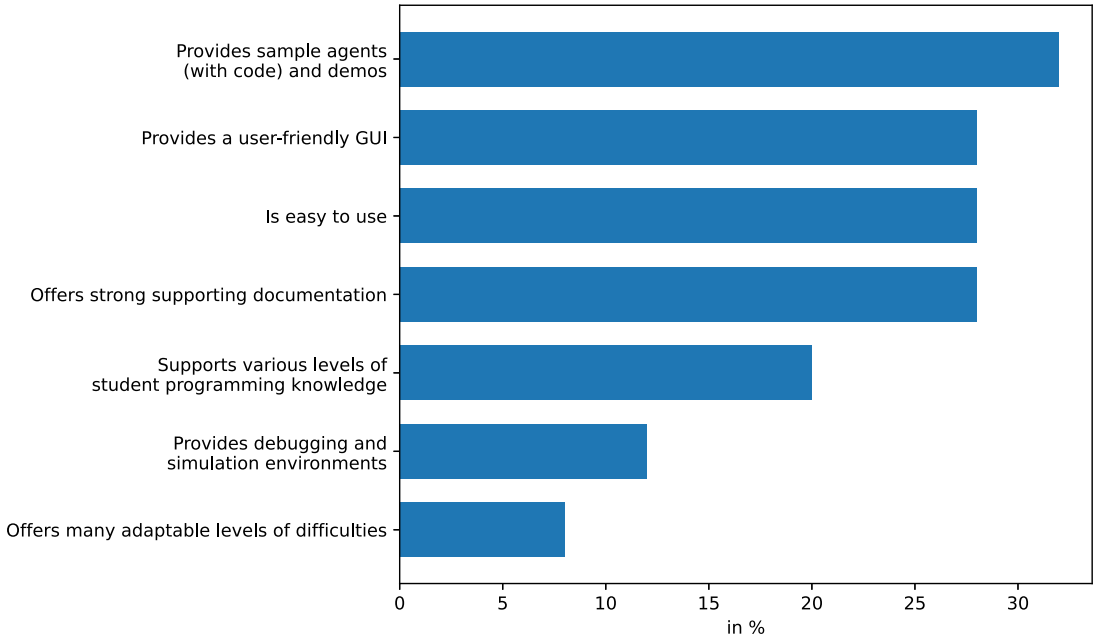


Abbildung 3.7: Wünschenswerte Features des Faktors User Interaction in Anlehnung an Serenko und Detlor (2002, S. 33)

In ihrem Leitfaden geben Beale und Sharples (2002) mehrere Hinweise, um die Usability einer Lernanwendung zu verbessern. Demnach sollen Informationen klar strukturiert und gruppiert sein, sodass diese von den Benutzern einfacher in Zusammenhang gesetzt werden können. Auch soll das Design der Benutzeroberfläche wohl überlegt sein, da die Motivation zur Benutzung der Lehranwendung stark von der ästhetischen Wahrnehmung des Benutzers abhängig ist. Die Navigation soll neben der klaren Strukturierung der Informationen so aufgebaut sein, dass die Rückkehr zum Startbildschirm jederzeit möglich ist. Sofern sinnvoll, sollten Undo- und Redo-Aktionen verwendet werden können. Ein weiterer Aspekt, welcher für den Lernerfolg sehr vorteilhaft sei, ist die Verwendung von Grafiken in Form von bspw. Bildern, Tabellen oder Graphen. Diese reduzieren die mentale Anstrengung, ermöglichen es dem Benutzer Beziehungen, Cluster oder Trends zu erkennen und vereinfachen so die Interpretation der Daten.

Neben der Usability spielen auch andere Faktoren für Van Nuland u. a. eine große Rolle. Die Anwendung müsse verlässlich sein und auf diversen Betriebssystemen funktionieren. Auch sollte darauf geachtet werden, dass die Anwendung erweiterbar gestaltet wird, sodass zukünftige Änderungen mit akzeptablem Zeitaufwand durchgeführt werden können.

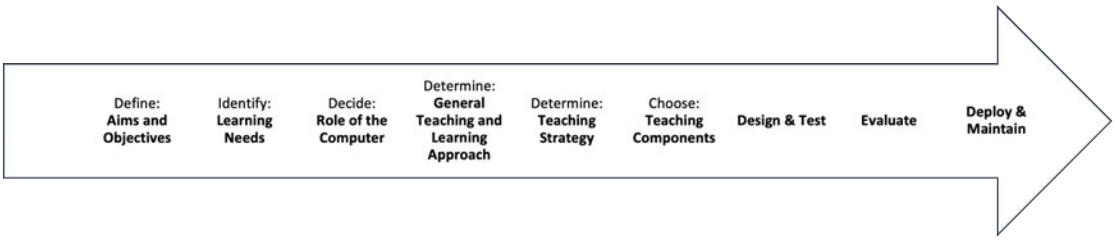


Abbildung 3.8: Schritte zu einer erfolgreichen Lernsoftware in Anlehnung an Beale und Sharples (2002)

3.3.2 Prozess

Neben den Anforderungen an die Software, stellen Beale und Sharples, Phillips sowie Van Nuland u. a. ebenso Anforderungen an den Prozess, der von der Konzeption bis zur Auslieferung der Lernsoftware durchlaufen wird.

In ihrem Leitfaden verfassen Beale und Sharples (2002) eine Reihe von Schritten, die bis hin zur Auslieferung der Anwendung durchlaufen werden sollten, um eine erfolgreiche Lernsoftware zu entwerfen. Diese Schritte sind in Abbildung 3.8 dargestellt und werden im Folgende erläutert:

Aims and Objectives Lernziele festlegen, welche die Anwendung verfolgt. Um diese Ziele zu definieren, sollte geklärt werden, welche Themen behandelt werden und auf welchem Niveau diese Themen vermittelt werden sollen. Soll die Anwendung eine Einführung in die Themen geben oder eher ein Werkzeug für Fortgeschrittene sein?

Learning Needs Verständnisschwierigkeiten des Themas identifizieren. Außerdem sollte zwischen Wissensvermittlung und dem Erlernen einer Fähigkeit unterschieden werden.

Role of the Computer Klären, welche Schwierigkeiten bei der Vermittlung des Lernstoffes ohne einen Computer bestehen und wie der Computer bei der Vermittlung unterstützen kann.

General Teaching and Learning Approach Abbildung 3.9 zeigt die von Beale und Sharples definierten Vorgehensweisen, die von einer Lernsoftware verfolgt werden können.

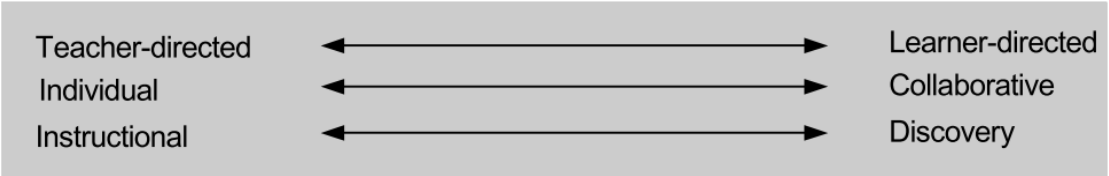


Abbildung 3.9: Dimensionen des Lernens und der Lehre von Beale und Sharples (2002, S. 6)

Dabei bewegt sich diese immer zwischen den abgebildeten Aspekten. Bei *Teacher-directed* ist die Software für den Einsatz unter der Kontrolle eines Lehrers vorgesehen, während bei *Learner-directed* kein Lehrer direkt vorgesehen ist. *Individual* und *Collaborative* unterscheidet zwischen Einzel- und Gruppenarbeit. Während bei einer *Instructional* Strategie dem Lernenden die Lehrinhalte direkt beigebracht werden, ist es bei *Discovery* die Aufgabe des Lernenden durch Erkundung der Anwendung oder durch praktische Aufgaben, die Lernziele selber zu erarbeiten.

Teaching Strategy Die Teaching Strategy beschreibt die Festlegung auf die Art und Weise, wie die Lehrinhalte vermittelt und Lernziele erreicht werden sollen. Laut Beale und Sharples (2002) sei bspw. die Teach-And-Test Strategy typisch für eine Lernsoftware. In dieser wird dem Lernenden das zu lehrende Thema vorgestellt, durch Lerninhalte beigebracht, das Lernziel getestet und am Ende das Ergebnis des Tests vorgestellt. Bei fehlerhaften Ergebnissen werden dem Benutzer die Lösung, sowie Hinweise angezeigt, warum die Lösung falsch war.

Teaching Components Aus einer Menge aus typischen Lernkomponenten eine Auswahl treffen, die innerhalb der Lernanwendung verwendet wird. Solche Komponenten sind bspw.: Learning Resources, Dokumente oder Webseiten mit Informationen zu dem jeweiligen Themengebiet; Computer-marked assignment, Tests, die das eigene Verständnis anhand von vordefinierten Fragen prüfen und die korrekten Lösungen liefern. Auch Simulation und Modellierung sind laut Beale und Sharples mögliche Komponenten.

Design & Test Geregelten Entwurfs- und Testablauf einführen, indem kontinuierlich Anforderungen abgestimmt, die Anwendung entwickelt und regelmäßig auf die Umsetzung der Anforderungen sowie auf die Usability getestet wird.

Evaluate Überprüfung der Lernsoftware auf ihre Effektivität bzgl. der Vermittlung von Lehrinhalten und weiterer Faktoren. Mögliche Methoden sind eine formative und eine summative Evaluierung.

Deploy & Maintain Auslieferung der Software durchführen. Dabei gilt es sicherzustellen, dass die Anwendung auf den Zielsystemen funktioniert und die Anwender instruiert werden, wie diese Anwendung zu verwenden ist. Außerdem sollte die Verknüpfung zum Lehrmaterial stattfinden. Wie bei jeder langfristigen Software sollte die Anwendung an neue Technologien angepasst werden und alle verwendeten Ressourcen aktualisiert werden, wenn notwendig.

Die Richtlinien von Phillips decken sich mit den zuvor aufgeführten Schritten. Für ihn spielt aber ebenso der Ablauf der Entwicklung eine große Rolle. Nach Phillips (1997, S. 38-39) sollte dieser Prozess von der Idee bis zur Implementierung wohl definiert sein und eine Trennung des Design- und des Entwicklungsprozesses vorgenommen werden. Bevor das Design des User Interfaces implementiert wird, müsse es erst vollkommen ausgearbeitet sein, um teure Änderungen an der Implementierung zu vermeiden, die entstehen, falls das Design im späteren Verlauf angepasst wird. Vorweg sollte eine *Feasibility Study* durchgeführt werden, die im Kern die Rahmenbedingungen und die Machbarkeit einer solchen Anwendung prüfen soll. Dies entspricht ebenso den ersten sechs Schritten des Leitfadens von Beale und Sharples (2002).

Nach Van Nuland u. a. (2017) sollte der Service der Anwendung nach der Veröffentlichung sichergestellt werden. Die Autoren sehen die entwickelnde Entität in der Pflicht. Dabei geht es Van Nuland u. a. um schnelle Reaktion der Entität auf Rückfragen, um die Aktualität der Soft- und Hardware die für die Entwicklung verwendet worden ist, sowie um die allgemeine Verlässlichkeit der Entität in Bezug auf die Software. Außerdem sehen auch sie es für zwingend notwendig, dass die Anwendung in Hinblick auf den Lernerfolg im Anschluss an die Entwicklung evaluiert wird. Dabei sollen sowohl die positiven als auch die negativen Auswirkung der Lernanwendung bemessen werden. Wichtige Mittel für die Messung sind dabei die Zeitersparnis, die bei der Verwendung einer erfolgreichen Lernanwendung auftritt; die Festigung des Fachwissen; sowie das Maß an Unterstützung, welche die Anwendung dem Lernenden bietet.

4 Analyse

In diesem Kapitel werden verschiedene Faktoren bzgl. der zu entwickelnden Anwendung analysiert. Auf Basis des in Kapitel 3.3 erläuterten Prozesses zur Entwicklung einer Lernsoftware werden den didaktischen Faktoren verschiedene Attribute dieser Arbeit zugewiesen. Darauf basierend werden die zur Beantwortung der Forschungsfrage verwendeten Hypothesen aufgestellt. Im nächsten Schritt werden verschiedene Anwendungsfälle der Modellierenden vorgestellt, die während der Implementierung eines MARS-Modells durchlaufen werden. Aus den Anwendungsfällen werden konkrete Anforderungen abgeleitet, die im späteren Verlauf dieser Arbeit auf ihre Umsetzung überprüft werden. Aufbauend auf den gestellten Anforderungen werden sog. Mockups – also Entwürfe der Benutzeroberfläche – vorgestellt, die das Ergebnis des Konzeptionsprozesses sind. Innerhalb der Implementierung wurden diese als Vorlage verwendet. Am Ende des Kapitels wird verdeutlicht, worauf der Fokus der zu entwickelnden Lernplattform liegt und welche Aspekte im Rahmen dieser Ausarbeitung bewusst vernachlässigt werden.

4.1 Bestimmung und Identifikation der didaktischen Faktoren

In diesem Abschnitt werden die in Kapitel *Didaktische Faktoren einer Lernsoftware* erläuterten Schritte des Prozesses zu einer effektiven Lernsoftware durchlaufen, die von Beale und Sharples (2002) verfasst worden sind. Diese sind ausschlaggebend für den didaktischen Rahmen der Anwendung und haben direkten Einfluss auf die Anforderungen, die im weiteren Verlauf der Arbeit aufgestellt werden. Im ersten Schritt werden die Lernbedarfe und -Ziele identifiziert bzw. definiert. Daraufhin wird die Rolle des Computers bzw. der Software erläutert. Abschließend werden geeignete Lernstrategien und -komponenten zur Vermittlung der Lehrinhalte und Erfüllung der aufgestellten Lernziele ausgewählt.

4.1.1 Lernbedarf

Gemäß der forschungsleitenden Frage verfolgt die vorliegende Arbeit das Ziel, Studierenden den Einstieg in das Thema der MAS und in das Framework MARS zu erleichtern. Dies beinhaltet in erster Linie das Erlernen von Modellierungsfähigkeiten, also die Überführung einer Fachdomäne in Software bzw. Quellcode. Diesbezüglich konnten die Mitglieder der Forschungsgruppe MARS folgende Schwierigkeiten bzw. Bedarfe der Studierenden während der relevanten Lehreinsätze beobachten:

Orientierung im Modellierungszyklus

Um komplexe Fragestellungen mithilfe eines MAS zu beantworten, durchläuft der Modellierende drei grundlegende Arbeitsschritte (Bodine u. a., 2020):

1. Modell entwerfen
2. Modell simulieren
3. Simulationsergebnisse analysieren

Da es sich dabei um die Entwicklung sehr komplexer Systeme handelt, wird das Modell in der ersten Version i.d.R. noch nicht vollständig sein oder eventuell Fehler beinhalten. Aus diesem Grund werden die zuvor genannten Arbeitsschritte zyklisch durchlaufen, bis die gestellte Forschungsfrage beantwortet ist (Bodine u. a., 2020). Innerhalb dieses Zyklus kommt es unter unerfahrenen Studierenden häufig zu Orientierungsschwierigkeiten. Die Modellierenden sind sich nicht bewusst in welcher dieser drei Arbeitsschritte bzw. Phasen sie sich aktuell befinden (Bodine u. a., 2020). Außerdem existiert im Zyklus des MARS-Framework noch ein weiterer Schritt, der vor der Ausführung der Simulation stattfindet: die Konfiguration des zu simulierenden Szenarios (MARS Group, 2021a). Dieser Schritt muss ebenfalls von den Studierenden verstanden und umgesetzt werden.

Weite Tool-Landschaft

Innerhalb des Kontextes von MARS existiert eine Vielzahl an Werkzeugen, von denen jedes seinen eigenen speziellen Anwendungsfall hat. Während bspw. Python-Skripte zur Visualisierung von Ergebnissen verwendet werden, kommen IDEs zum Einsatz, um das Modell zu definieren, die Konfiguration zu bearbeiten und die Simulation auszuführen.

Das Ausführen der Simulation erfordert außerdem das .NET-SDK, welches vom Modellierenden zuvor installiert werden muss. Da das Framework für die Studierenden anfangs unbekannt ist und die meisten Anwendungsfälle i.d.R. noch nicht verstanden worden sind, ist die Reduzierung der verschiedenen Werkzeuge ein erster Schritt, um die kognitiven Anstrengungen des Studierenden auf das Thema MAS und das MARS Framework zu reduzieren.

Schnittstellen des MARS-Framework

Das Framework liefert neben dem Laufzeitsystem viele Schnittstellen, die von den Modellierenden verwendet werden müssen (s. Kapitel 3.1). Dazu gehören besonders die C# Interfaces für Agenten, Layer und auch Entities. Bei den Studierenden treten häufig folgende Unsicherheiten auf:

- Welchen Typen sollen sie für ihre jeweilige Aufgabe verwenden?
- Wie genau müssen die verschiedenen Typen implementiert werden?
- Welche Funktionalitäten bringen die Typen mit sich?

Fehlkonfiguration der Simulation

Ebenso Teil des Framework ist die Möglichkeit einen Simulationsdurchlauf zu konfigurieren. Da die Konfiguration allerdings über eine JSON Datei abgebildet wird, werden die Einträge bis auf die korrekte Syntax nicht weiter validiert. Dies ermöglicht Eingaben, die zwar syntaktisch korrekt sind, semantisch allerdings keinen Sinn ergeben. Ein Beispiel ist die Konfiguration der Anzahl der Simulationsschritte, die eine Simulation durchlaufen soll. Innerhalb einer JSON Datei kann dort bspw. eine -1 oder ein String `"yes"` eingetragen werden. Diese Werte sind syntaktisch korrekt, werden jedoch aufgrund der Semantik zur Laufzeit der Simulation Fehler produzieren. Dies führt ggf. zur Verwirrung des Modellierenden, da die verwendete IDE keine Fehler anzeigt, die Simulation aber dennoch fehlgeschlagen ist.

Anzahl der Konfigurationsmöglichkeiten

Ein weiteres Problem für angehende Modellierende ist laut den Beobachtungen der Forschungsgruppe MARS die Vielzahl an Konfigurationsmöglichkeiten, die zwar Online wohl dokumentiert sind, allerdings aufgrund ihrer großen Anzahl überwältigend wirken können. Unter den Konfigurationsmöglichkeiten befinden sich Einstellungen, die eher für Experten relevant sind, welche aber von den Modellierenden anfangs noch nicht eingeordnet werden können. Als Beispiel für eine solche Konfigurationsmöglichkeit kann die Möglichkeit der Ausgabe betrachtet werden. Mithilfe des Feldes `output` ist es möglich, die resultierenden Daten einer Simulation statt als CSV Datei in einer Online-Datenbank zu speichern. Ein solches Feature wird höchstwahrscheinlich nicht von einem angehenden Benutzer des Framework verwendet werden, belastet aber dennoch die kognitive Aufmerksamkeit des Modellierenden.

4.1.2 Lernziele

Aus den zuvor erläuterten Lernbedarfen lassen sich die folgenden Lernziele ableiten:

- Förderung der Orientierung im Modellierungszyklus
- Förderung des allgemeinen Verständnisses über die Schnittstellen des MARS-Framework
- Förderung des Verständnisses der Typen, die in einem MAS innerhalb des MARS-Framework verwendet werden
- Förderung des Verständnisses zur Konfiguration einer Simulation
- Förderung der Identifikation von Fehlern innerhalb der Konfiguration einer Simulation

Besonders hervorzuheben ist, dass es sich bei der Zielgruppe dieser Lernziele, um fachfremde Studierende handelt, die in das Thema eingeführt werden sollen.

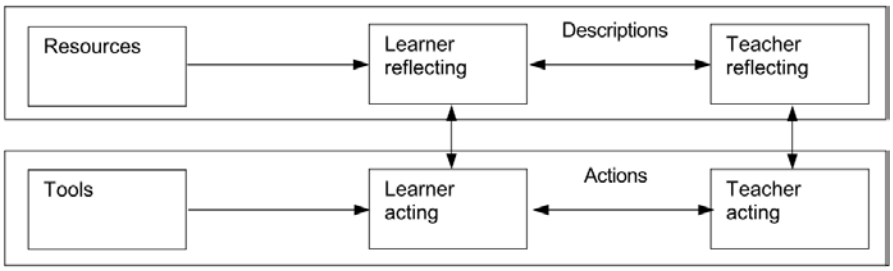


Abbildung 4.1: Modell zur idealen Lehre aus Beale und Sharples (2002, S. 9)

4.1.3 Rolle des Computers

Ein weiterer Aspekt, der laut Beale und Sharples (2002) geklärt werden sollte, ist die Rolle, die vom Computer eingenommen wird, um die Lernziele zu erreichen. Es ist offensichtlich, dass die Rolle des Computers hier essentiell ist, da agenten-basierte Modelle sich theoretisch zwar konzipieren lassen, aber die Simulation entsprechender Modelle realistisch nur mithilfe eines Computers möglich ist. Außerdem handelt es sich bei den Lehrinhalten um Themen, die ein Software-Framework betreffen, dessen Verwendung einen Computer impliziert.

4.1.4 Lehrstrategien und -komponenten

Die verschiedenen Dimensionen der Lehre wurden bereits in Kapitel 3.3 erläutert. Mithilfe der zu entwickelnden Anwendung soll der individuelle Lernprozess des Modellierenden unterstützt werden, sodass das Fachwissen ohne direkte Anwesenheit des Lehrenden vertieft werden kann. Außerdem ist es Ziel der Anwendung, dass sich der Benutzer das Wissen selber erarbeitet, ohne dass eine statische Menge von Anweisungen durchlaufen werden muss. Natürlich wird es innerhalb der Anwendung Hinweise oder Erklärungen geben, sodass der Modellierende trotzdem in gewisser Weise gelenkt wird und somit einfache Fehler vermieden werden. Das Ziel der Anwendung ist es aber, nicht den Lehrenden zu ersetzen, sondern vielmehr zu ergänzen. Die Strategie entspricht dem in Abbildung 4.1 dargestellten Modell. Der Lehrende liefert die Grundlagen zum allgemeinen Thema von MAS und MARS, woraufhin die Studierenden das Gelernte mithilfe von externen Ressourcen und der Lernanwendung praktisch umsetzen. Als Ressourcen existieren bspw. Aufzeichnungen von vergangenen Vorlesungen oder die Dokumentation des MARS-Framework.

Die Lernstrategie die mit der Anwendung verfolgt werden soll, entspricht der Trial-and-Error Strategie. Diese verfolgt den Ansatz, dass der Benutzer eine Aufgabe solange bearbeitet bis keine Fehler mehr auftreten. In diesem Zusammenhang ist unter *Fehler* kein Fehler der Software bzw. des Modells gemeint, sondern ein Ergebnis, welches die gestellte Forschungsfrage nicht beantwortet. Sind die Ergebnisse in Bezug auf die Forschungsfrage nicht zufriedenstellend, wird das Modell verfeinert und die Ergebnisse erneut analysiert. Dieser Zyklus wurde unter anderem von Bodine u. a. (2020) und Hüning u. a. (2016) identifiziert.

Die relevanten Lehrkomponenten, die innerhalb der Anwendung verwendet werden, sind die Modellierung und die Simulation, welche bereits Teil des zu vermittelnden Themengebiets sind. Des Weiteren besteht für die Benutzer die Möglichkeit, die Online-Dokumentation des MARS-Framework zu verwenden. In der Dokumentation können Beschreibungen, Begründungen und Beispiele zu diversen Schnittstellen des Framework eingesehen werden. Da es sich bei der zu entwickelnden Anwendung lediglich um eine praktische Einführung in das Thema handelt, eignet sich die Dokumentation ergänzend als Nachschlagewerk.

4.2 Aufstellung der Forschungsfrage und der Hypothesen

Die in Kapitel 4.1.1 identifizierten Lernbedarfe zeigen mehrere Probleme fachfremder Studierender mit dem MARS-Framework auf. Um den Lernprozess in Hinblick auf das allgemeine Thema MAS und besonders das MARS-Framework zu verbessern, gilt es innerhalb dieser Arbeit eine Lösung zu finden. Aus diesem Grund wurde die folgende Forschungsfrage formuliert, die im Rahmen der vorliegenden Arbeit beantwortet werden soll: *Wie können fachfremde Benutzer des MARS-Framework ganzheitlich in der Simulation von Modellen unterstützt werden?* Die vorgestellten Untersuchungen aus dem Kapitel 3 lassen vermuten, dass eine Anwendung mit graphischer Benutzeroberfläche verwendet werden kann, um Modellierende innerhalb des Lernprozesses effektiv zu unterstützen. Um diese Vermutung zu bestätigen und die Forschungsfrage somit zu beantworten, wurden vier Hypothesen aufgestellt, die zum einen auf den vorgestellten Untersuchungen aus Kapitel 3, aber vor allem auch auf den identifizierten Problemen des MARS-Framework basieren, welche von Hüning u. a. (2016), der MARS-Forschungsgruppe und innerhalb dieser Arbeit im vorherigen Kapitel als Lernbedarfe identifiziert worden sind. Folgende Hypothesen wurden aufgestellt, die es im Folgenden zu untersuchen gilt:

Hypothese 1. Die Verwendung einer Lernplattform fördert das Verständnis von den verschiedenen Objekten (Agent, Layer, Entität).

Hypothese 2. Die Verwendung einer Lernplattform fördert das Verständnis von den verschiedenen Phasen der Multi-Agenten-Simulation (Modellierung, Konfiguration, Simulation, Analyse).

Hypothese 3. Die Verwendung einer Lernplattform führt dazu, dass Fehler innerhalb der Modellierung oder Parametrisierung vom Benutzer schneller identifiziert und behoben werden.

Hypothese 4. Die Verwendung einer Lernplattform führt dazu, dass seltener Experten zur Unterstützung benötigt werden.

Eine entsprechende Lernplattform mit dem Namen *MARS-Explorer* wird im Rahmen dieser Arbeit unter Berücksichtigung der Lehrfunktion konzipiert, implementiert und anschließend evaluiert, um die Hypothesen zu verifizieren oder ggf. zu falsifizieren und somit die Forschungsfrage zu beantworten.

4.3 Aufstellung der funktionalen Anforderungen

Im Rahmen dieses Kapitels werden funktionale Anforderungen (FA) formuliert, die für die Entwicklung der Anwendung relevant sind. Dazu wird auf die aktuellen Anwendungsfälle eingegangen, die von den fachfremden Modellierenden während der Implementierung eines MARS-Modells unter der Verwendung der bestehenden Werkzeuge durchlaufen werden. Die für den MARS-Explorer relevanten Funktionen werden als FA festgehalten. Des Weiteren werden FA ergänzt, welche die Usability und den Lernerfolg der Anwendung erhöhen sollen. Am Ende jedes Abschnitts werden die aufgestellten FA innerhalb eines Kastens festgehalten. Die Anforderungen werden in dem von Cohn (2004, S. 81) vorgestellten Format zur Formulierung von User Stories definiert: *Als [Rolle] möchte ich [Funktion], damit/um [Nutzen] (ursprünglich: I as a (role) want (function) so that (business value)).*

Auf Basis des von Hüning u. a. (2016) analysierten Modellierungsablaufs und in Zusammenarbeit mit der MARS-Forschungsgruppe konnten vier Phasen identifiziert werden, die von einem Modellierenden während der Verwendung des MARS-Framework durchlaufen werden:

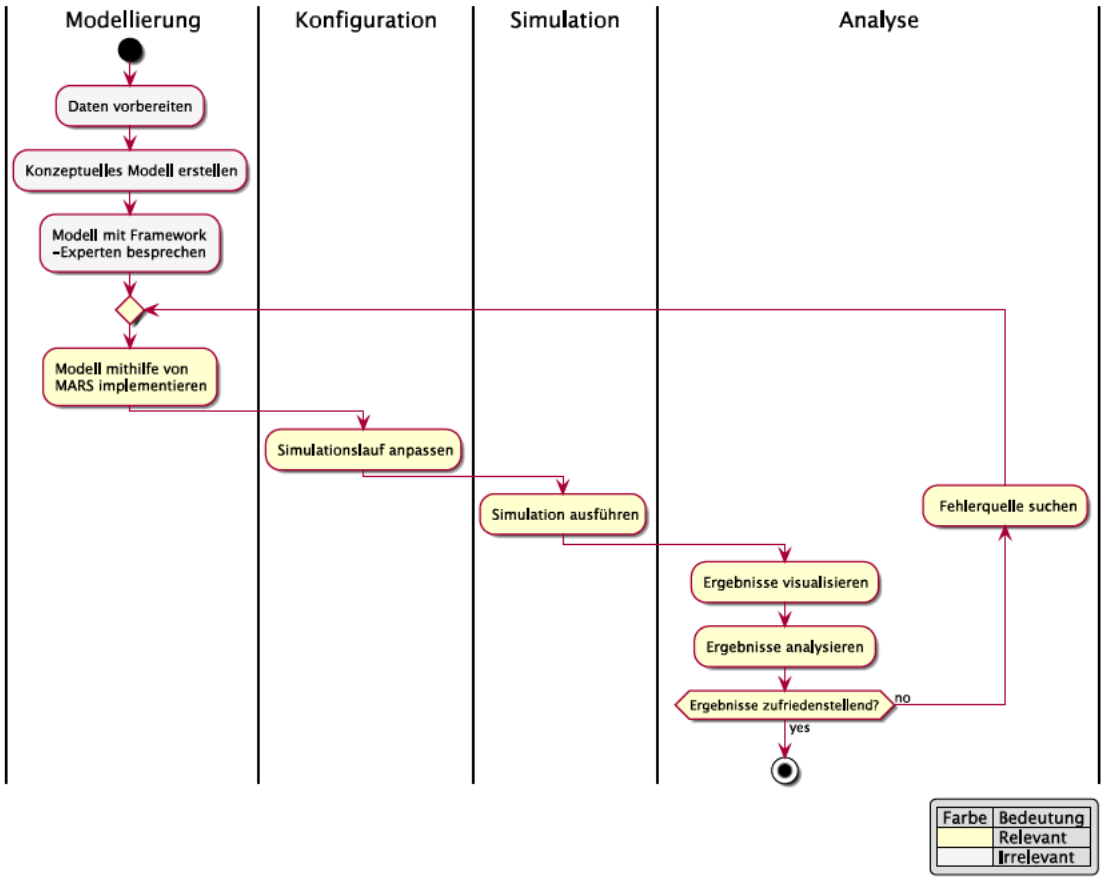


Abbildung 4.2: Aktivitätsdiagramm zum Ablauf der Modellierung mithilfe des MARS-Framework in Anlehnung an Hüning u. a. (2016, S. 33)

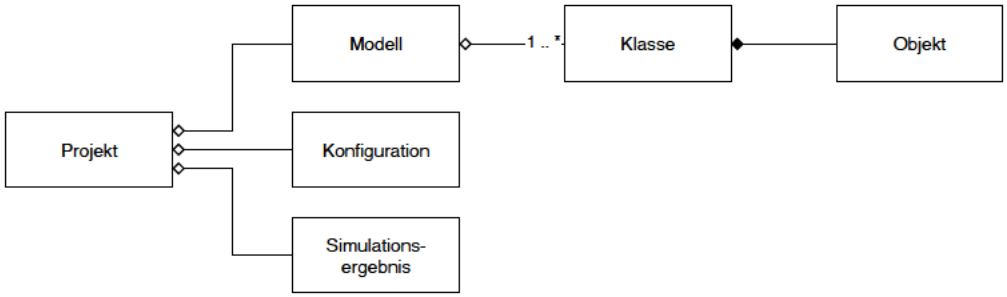


Abbildung 4.3: Taxonomie der in dieser Arbeit verwendeten Begriffe im Kontext des MARS-Framework

1. Modellierung
2. Konfiguration
3. Simulation
4. Analyse

Dieser Arbeitsablauf ist in Abbildung 4.2 dargestellt und zeigt neben den vier Phasen eine Unterscheidung zwischen relevanten und irrelevanten Arbeitsschritten für den MARS-Explorer. Die Schritte *Daten vorbereiten*, *Konzeptuelles Modell erstellen* und *Modell mit Framework-Experten besprechen* werden als irrelevant angesehen, da die Zielgruppe des MARS-Explorer zunächst fachfremde Informatiker sind, die mithilfe des MARS-Explorer den technischen Umgang mit MAS bzw. mit dem MARS-Framework erlernen sollen. Deswegen ist davon auszugehen, dass ein konzeptuelles Modell vorgegeben und dazugehörigen Daten bereits vorhanden sind, sodass das Modell nur noch innerhalb des Framework umgesetzt werden muss. Die identifizierten Phasen decken sich ebenso mit den Beobachtungen von Bodine u. a. (2020), welche die Schritte *Modellierung* und *Konfiguration* allgemeiner als das *Schreiben von Code* zusammenfassen. Zusätzlich wurde ein weiterer implizierter Arbeitsablauf identifiziert, der sich mit der Verwaltung des gesamten Projekts beschäftigt.

4.3.1 Taxonomie

Zur eindeutigen Einordnung der Fachwörter, die innerhalb dieses Kapitels verwendet werden, wurde eine Taxonomie aufgestellt. Diese Taxonomie ist in Abbildung 4.3 zu sehen. Innerhalb des MARS-Framework werden Agenten, Layer und Entities nach den

Prinzipien der Objekt-orientierten Programmierung mithilfe von *Klassen* als *Objekte* implementiert. In ihrer Menge bilden die *Klassen* ein *Modell*. Auf gleicher Ebene wie das *Modell* befinden sich die *Konfiguration* und das *Simulationsergebnis*, welche zusammen mit dem *Modell* ein *Projekt* bilden.

Im nachfolgenden Abschnitt ist mit *Benutzer* gleichermaßen der *Modellierende* gemeint.

4.3.2 Modellierung

Die Modellierung ist die Kernaufgabe des Benutzers und sollte im Rahmen dieser Ausarbeitung dementsprechend fokussiert werden. Neben dem Schreiben von Code spielt auch die Verwaltung des Modells eine große Rolle. Im Rahmen der Modellierung werden neue Objekte in Form einer *cs*-Datei erstellt. Die von den Modellierenden verwendeten Werkzeuge – wie bspw. IDEs – unterstützen den Modellierenden in der syntaktischen Validierung und in Form von weiteren Funktionen, wie *Undo*, *Redo* oder Autovervollständigungen.

Klassen erstellen

Die Objekte *Agent*, *Layer* und *Entity* sind für ein MARS-Modell essentiell und haben jeweils einen für sie typischen Aufbau. Dazu gehört vor allem die Implementierung der entsprechenden Interfaces, die das MARS-Framework zur Verfügung stellt.

Listing 4.1: Beispielhafte Implementierung eines Agenten-Objektes mithilfe des MARS-Framework

```
1  // ...
2
3  namespace MyProject
4  {
5      public class MyAgentName : IAgent<MyLayerName>,
           IPositionable
6      {
7
8          public void Init(MyLayerName layer)
9          {
10             // initialization logic
```

```
11          }
12
13          // position of the agent
14          public Position Position { get; set; }
15
16
17          public void Tick()
18          {
19              // what does the agent do on each tick?
20          }
21
22          // identifies the agent
23          public Guid ID { get; set; }
24      }
25  }
```

Ein Beispiel für die Implementierung eines Agenten ist in Listing 4.1 zu sehen. Das Objekt `MyAgentName` muss sowohl das Interface `IAgent` als auch `IPositionable` implementieren, wodurch der Agent (`MyAgentName`) auf seinem designierten Layer (`MyLayerName`) platziert wird. Das Verhalten kann daraufhin durch die Implementierung der `Init`- und `Tick`-Methode definiert werden.

Listing 4.2: Beispielhafter Einstiegspunkt eines Modells, in dem je ein Layer (`MyLayerName`) und ein Agent (`MyAgentName`) registriert wird

```
1  // ...
2
3  namespace MyProject
4  {
5      internal static class Program
6      {
7          private static void Main()
8          {
9              var description = new ModelDescription();
10             description.AddLayer<MyLayerName>();
11             description.AddAgent<MyAgentName, MyLayerName>();
12             // ...
13         }
```

```
14      }
15  }
```

Neben der korrekten Implementierung der Interfaces, muss das neue Objekt noch dem Modell bekannt gemacht werden. Dies geschieht innerhalb des Einstiegspunktes des Projekts, welcher meistens eine für .NET typische `Program.cs` Datei ist. Dies geschieht wie in Listing 4.2 beispielhaft dargestellt. In Zeile 9 wird eine `ModelDescription` instanziiert, welche den Ausgangspunkt für die Simulation bildet. Auf dieser Instanz werden alle weiteren Agenten, Layer und Entities registriert, indem – wie in Zeile 10 und 11 dargestellt – die `AddLayer`, `AddAgent` oder für Entities die `AddEntity` Methode mit dem zu registrierenden Objekt als Argument ausgeführt wird.

Wie bereits in den Lernbedarfen erwähnt, kommt es häufig zu Unsicherheiten, über welche Funktionalitäten die verschiedenen Objekte verfügen. Aus diesem Grund sollte dem Benutzer bei der Erstellung eines neuen Objektes Informationen über dessen Funktionen und Einsatzgebiete angezeigt werden. In ihrer Untersuchung bzgl. eines allgemeinen Community-Framework zur ABM fanden Janssen u. a. (2008) heraus, dass angehende Modellierende Schwierigkeiten haben, Modelle von Grund auf neu zu schreiben. Dem soll innerhalb des MARS-Explorer durch Verwendung von Vorlagen bei der Erstellung eines neuen Objektes entgegengewirkt werden. Des Weiteren sollten Objekte, die neu erstellt worden sind, automatisch dem Einstiegspunkt des Modells hinzugefügt werden.

FA 1 **(Klasse erstellen)**: Als Modellierender möchte ich eine neue Klasse erstellen können, um mein Modell zu erweitern.

FA 2 **(Hilfestellung zur Auswahl der Objekte erhalten)**: Als Modellierender möchte ich Hilfestellung zu den verschiedenen Objekten erhalten, um zu verstehen, welches das geeignete Objekt für meinen jeweiligen Anwendungsfall ist.

FA 3 **(Objekt-Vorlagen verwenden)**: Als Modellierender möchte ich bei der Erstellung eines neuen Objektes nicht von vorne beginnen und auf das Nötigste, was für die Implementierung notwendig ist, hingewiesen werden, um das jeweilige Objekt mit weniger Aufwand korrekt implementieren zu können.

FA 4 **(Neues Objekt registrieren)**: Als Modellierender möchte ich, dass das erstellte Objekt dem Einstiegspunkt des Modells automatisch hinzugefügt wird, um diesen Schritt nicht selbst durchführen zu müssen.

Klassen verwalten

Innerhalb der verwendeten Werkzeuge existiert i.d.R. eine Übersicht, die alle Dateien eines Projekts aufzeigt. Die Dateien können ausgewählt, der Inhalt dargestellt und daraufhin bearbeitet werden. Unter *Bearbeiten* ist in diesem Kontext das Schreiben des C# Codes zu verstehen. Außerdem können die Dateien umbenannt oder gelöscht werden, falls sie bspw. nicht mehr benötigt werden oder mittlerweile einen anderen Zweck erfüllen, als ursprünglich bei der Erstellung gedacht.

Um sich einen Überblick über ein bestehendes Modell verschaffen zu können, sollte innerhalb des MARS-Explorer ersichtlich sein, welche Klassen dem Modell angehören. Typische Aktionen – wie das Bearbeiten, Umbenennen und Löschen von Klassen – sind Standard-Funktionen, die von Werkzeugen wie IDEs unterstützt werden müssen. Neben Syntaxkorrekturen bietet eine IDE diverse Funktionen, welche die Usability stark erhöhen, wie bspw. *Eingabe rückgängig machen/wiederholen*, *Multi-Cursor* oder *Autocompletion*. Funktionen dieser Art gehören mittlerweile zum Standardrepertoire eines jeden Entwicklungswerkzeugs, weshalb diese zumindest in Teilen im MARS-Explorer vorhanden sein sollten.

FA 5 **(Klassen ansehen)**: Als Modellierender möchte ich alle Klassen eines Projekts sehen, um einen Überblick über die Gesamtstruktur des Projekts zur erhalten.

FA 6 **(Klasse ansehen)**: Als Modellierender möchte ich den Quellcode einer Klasse betrachten können, um die Funktionsweise des Modells zu verstehen.

FA 7 **(Klasse bearbeiten)**: Als Modellierender möchte ich den Quellcode einer Klasse bearbeiten können, um die Funktionsweise des Modells anzupassen.

FA 8 **(Klasse umbenennen)**: Als Modellierender möchte ich eine Klasse umbenennen können, um ihr einen aussagekräftigeren Titel zu geben.

FA 9 **(Klasse löschen)**: Als Modellierender möchte ich eine Klasse löschen können, um sie aus meinem Projekt zu entfernen, wenn sie nicht mehr benötigt wird.

FA 10 **(Eingabe rückgängig machen)**: Als Modellierender möchte ich eine getätigte Eingabe rückgängig machen können, um bei Fehlern o.ä. wieder in den Ausgangszustand der Klasse zurückzukehren.

FA 11 **(Eingabe wiederholen)**: Als Modellierender möchte ich eine getätigte Eingabe wiederholen können, um versehentlich getätigte Undo-Aktionen umzukehren.

FA 12 **(Eingabe automatisch vervollständigen)**: Als Modellierender möchte ich, dass ich auswählbare Vorschläge zu meiner getätigten Eingabe bekomme, um Zeit zu sparen.

Validierung

Während der Verwendung der oben genannten Werkzeuge hat die Bearbeitung einer Klasse eine Validierung zur Folge, um den Benutzer darüber zu informieren, ob der geschriebene Code Fehler beinhaltet. Dieses typische Feature sollte besonders in diesem Kontext nicht fehlen. Die Abwesenheit einer solchen Kontrolle hätte zur Folge, dass sich der Modellierende bis zur Ausführung der Simulation nie sicher sein kann, ob das geschriebene Modell funktionsfähig ist. Dies kann zu Frustration mit der Anwendung und schlussendlich zur Ablehnung führen. Eine Forderung, die von Nielsen (2012) im Rahmen der von ihm aufgestellten Heuristiken formuliert worden ist, besagt, dass Fehler dem Benutzer deutlich kommuniziert und vor allem Lösungswege bereitgestellt werden müssen, um den Fehler selbstständig beheben zu können.

FA 13 **(Stetige Validierung des Modells)**: Als Modellierender möchte ich, dass meine Klassen stetig validiert werden, damit ich fehlerhafte Änderungen korrigieren kann und mein Modell dadurch stets lauffähig bleibt.

FA 14 **(Status anzeigen)**: Als Modellierender möchte ich den Status der Validierung stets angezeigt bekommen, um zu sehen, ob mein aktuelles Modell läuffähig ist oder Fehler enthält.

FA 15 **(Fehler ansehen)**: Als Modellierender möchte ich Fehler, die Ergebnis der Validierung sind, an den jeweiligen Stellen mitsamt einer Begründung im Quellcode angezeigt bekommen, damit ich verstehe, warum diese Fehler auftreten und wie ich sie beheben kann.

Einführung von Beispielmodellen

Da der Umfang der Funktionen, die das MARS-Framework liefert, anfänglich schwer zu überblicken ist, sollen Beispielmodelle helfen, verschiedene Funktionen zu demonstrieren. Tutoren sollen diese per einfacher Schnittstelle im MARS-Explorer zur Verfügung stellen können. Die Beispielmodelle sollen gleichzeitig als ständiger Orientierungspunkt für Syntax und anderweitige Vorraussetzungen für die korrekte Funktionsweise des Modells dienen. Die Untersuchungen von Serenko und Detlor (2002) zeigen, dass die Nennung von Beispielen das am wünschenswerteste Feature eines Toolkits sei.

Damit der Benutzer auch das Verhalten eines Beispielmodells versteht und daraus Schlüsse für das eigene Modell ziehen kann, sollen Informationen zu jedem Beispielmodell angezeigt werden können. Die Informationen bieten eine Übersicht über die enthaltenen Agenten, Layer und Entitäten sowie die Interaktionen zwischen den Klassen.

FA 16 **(Beispielmodelle einstellen)**: Als Lehrender möchte ich den Modellierenden innerhalb des MARS-Explorer verschiedene Beispielprojekte zur Verfügung stellen können, um ihnen einen Überblick über diverse Schnittstellen des Framework zu geben.

FA 17 **(Beispielmodelle ansehen)**: Als Modellierender möchte ich den Quelltext von Beispielmodellen einsehen können, damit ich mich an diesem orientieren kann und einen Überblick über die Funktionen des Framework erlange.

FA 18 **(Informationen zu Beispielmodellen einsehen)**: Als Modellierender möchte ich Informationen zu den verschiedenen Beispielmodellen einsehen können, um zu verstehen, wie die Modelle funktionieren und welche Rolle die verschiedenen Objekte spielen.

4.3.3 Konfiguration

Mit der Konfiguration eines Szenarios werden verschiedene Einstellungen getroffen, die für den jeweiligen Simulationslauf relevant sind. Die von den angehenden Modellierenden verwendeten Werkzeuge stellen zur Bearbeitung eines Szenarios einen Texteditor mit Syntax-Unterstützung für JSON-Dateien zur Verfügung.

Konfiguration einsehen

Da sich die Unterstützung der verwendeten Werkzeuge lediglich auf die Syntax beschränkt, bedeutet das, dass der Benutzer sich den diversen Konfigurationsmöglichkeiten bewusst sein muss. Unter den Konfigurationsmöglichkeiten befinden sich jedoch einige, die lediglich für Experten relevant sind. Dazu gehört bspw. die Konfiguration der Ausgabedaten, mit der das Zielformat bestimmt werden kann. Da der Modellierende zunächst voraussichtlich kleine Simulationen durchführen wird, um sich mit der Technologie und dem Thema vertraut zu machen, wird bspw. keine Unterstützung von Datenbanken benötigt, die der Benutzer zunächst aufsetzen und konfigurieren müsste.

Innerhalb des MARS-Explorer soll es also möglich sein, die Konfiguration des Projekts einzusehen. Aufgrund der sehr umfangreichen Anzahl an Optionen sollte eine entsprechende Übersicht geschaffen werden, welche die Optionen logisch gruppiert. Um die kognitiven Anstrengungen des Modellierenden zu entlasten, sollten außerdem nicht alle Konfigurationsmöglichkeiten dargestellt werden, die für Anfänger weniger relevant sind.

FA 19 **(Konfigurationsmöglichkeiten logisch anordnen)**: Als Modellierender möchte ich die Konfiguration meines Modells in einer logischen Übersicht dargestellt bekommen, um einen Überblick über die Zusammengehörigkeit verschiedener Einstellungen zu erlangen.

Konfiguration bearbeiten

Die Konfiguration der Simulation eines Modells wird mithilfe einer einfachen JSON-Datei vorgenommen, die einem definierten Schema folgt. Bei einer Bearbeitung mit den genannten Werkzeugen muss der Benutzer selbst darauf achten, dass die Namen der getroffenen Einstellungen mit denen im Schema übereinstimmen. Um diese Fehlerquellen auszuschließen, sollen alle Einstellungsmöglichkeiten mithilfe von geeigneten Eingabeelementen dargestellt werden können, sodass der Benutzer lediglich die Werte eintragen muss. Ebenso sollten die Eingabeelemente den Benutzer in seinen Möglichkeiten einschränken, um fehlerhafte Werte zu verhindern. Beispielsweise ist es sinnvoll, dass für eine Option, die eine natürliche Zahl als Wert erwartet, ein Eingabeelement gewählt wird, welches lediglich Zahlen größer Null entgegennimmt.

Zu den möglichen Konfigurationen gehören die globalen Einstellungen sowie die Einstellung bzgl. der drei Objekt-Typen (Agent, Layer, Entity). Die Konfiguration der Objekt-Typen wird auch *Mapping* genannt. Das *Mapping* beschreibt die Abbildung der in der Konfiguration hinterlegten Werte auf die Attribute der Objekte. Die Attribute müssen im Modell dementsprechend gekennzeichnet werden. In der Konfiguration ist es bspw. möglich, die Anzahl der Agenten oder Initialisierungswerte für entsprechend gekennzeichnete Attribute zu setzen. Ebenso können Dateien verschiedener Formate als Mapping von Objekten verwendet werden.

FA 20 **(Konfiguration darstellen)**: Als Modellierender möchte ich die Konfiguration meiner Simulation in Form von Eingabefeldern dargestellt bekommen, um den internen Aufbau der JSON-Datei nicht beachten zu müssen, von einer erleichterten Eingabe zu profitieren und Fehler durch falsche Feldernamen innerhalb der Datei zu verhindern.

FA 21 **(Globale Einstellungen bearbeiten)**: Als Modellierender möchte ich die globalen Einstellungen bearbeiten können, um allgemeine Einstellungen der Simulation vorzunehmen.

FA 22 **(Agenten-Mappings bearbeiten)**: Als Modellierender möchte ich die Mappings der Agenten bearbeiten können, um Einstellungen bzgl. der Agenten in meiner Simulation vorzunehmen.

FA 23 **(Layer-Mappings bearbeiten)**: Als Modellierender möchte ich die Mappings der Layer bearbeiten können, um Einstellungen bzgl. der Layer in meiner Simulation vorzunehmen.

FA 24 **(Entity-Mappings bearbeiten)**: Als Modellierender möchte ich die Mappings der Entities bearbeiten können, um Einstellungen bzgl. der Entities in meiner Simulation vorzunehmen.

FA 25 **(Eingaben in der Konfiguration rückgängig machen)**: Als Modellierender möchte ich in der Konfiguration getätigte Eingaben rückgängig machen können, um bei Fehlern in den ursprünglichen Zustand zurückzukehren.

FA 26 **(Eingaben in der Konfiguration wiederholen)**: Als Modellierender möchte ich in der Konfiguration getätigte Eingaben wiederholen können, um irrtümlich rückgängig gemachte Eingaben wiederherzustellen.

Validierung

Da es innerhalb einer einfachen JSON-Datei nicht möglich ist, eine Typisierung oder ähnliche Validierungsmechanismen einzuführen, soll der MARS-Explorer dabei unterstützen, die korrekten Werte für die jeweiligen Felder einzutragen. Dabei muss zwischen semantischen und syntaktischen Fehlern unterschieden werden.

Beispielsweise existiert für die Konfiguration innerhalb des Objektes *globals* ein Feld *step*, welches die Anzahl der in der Simulation zu durchlaufenden Schritte bestimmt und demnach eine natürliche Zahl als Wert erwartet. Während ein Wert von `-1` für das Feld *step* syntaktisch korrekt wäre, würde dieser bei Ausführung der Simulation zu einem Fehler führen, da es keine negative Anzahl an Simulationsschritten geben kann. Die Simulation würde daraufhin abbrechen und einen entsprechenden Fehler werfen. Eine solche Validierung wird bspw. bei IDEs erst zur Laufzeit durchgeführt.

Fast jedes Feld hat Bedingungen und muss dementsprechend vor der Ausführung validiert werden. Um zu verhindern, dass der Modellierende semantisch oder syntaktisch falsche Eingaben innerhalb der Konfiguration treffen kann, soll der MARS-Explorer eine

Validierung dieser Werte nach jeder Bearbeitung eines Feldes durchführen. Die Anwendung sollte daraufhin auf Fehler hinweisen, sodass diese vom Modellierenden frühzeitig erkannt und in Eigenarbeit behoben werden können.

Eine wichtige Information, die ebenso dargestellt werden soll, ist der allgemeine Zustand der Konfiguration, der dem Modellierenden zeigen soll, ob sie in ihrem aktuellen Zustand valide oder invalide ist. Dieser Zustand soll außerhalb der Konfiguration angezeigt werden, sodass dem Modellierenden jederzeit klar ist, ob eine Simulation ausgeführt werden kann.

FA 27 **(Fehler in der Konfiguration anzeigen)**: Als Modellierender möchte ich, dass meine Eingabe nach der Bearbeitung eines Feldes validiert und das Feld bei falscher Eingabe markiert wird, um zu wissen, in welchem Feld ich eine falsche Eingabe getätigt habe.

FA 28 **(Grund für Fehler anzeigen)**: Als Modellierender möchte ich, dass bei falscher Eingabe Informationen über den Fehler angezeigt werden, um zu verstehen, warum dieser Fehler aufgetreten ist und diesen anschließend zu korrigieren.

FA 29 **(Status der Konfiguration darstellen)**: Als Modellierender möchte ich angezeigt bekommen, ob meine Konfiguration valide ist, um sicherzustellen, dass meine Simulation ausgeführt werden kann.

4.3.4 Simulation

Ein Simulationsdurchlauf eines Modells wird von Anwendern des MARS-Framework mithilfe des .NET-SDKs ausgeführt. Dies wird i.d.R. von den jeweiligen IDEs unterstützt, da es sich bei dem Projekt lediglich um ein .NET-Projekt handelt, welches standardmäßig mit dem Befehl `dotnet run` gestartet werden kann.

Simulation starten/stoppen

Eine Simulation soll vom Modellierenden auf einfache Art und Weise gestartet werden können, sodass dieser nicht über die entsprechenden .NET-Befehle informiert sein muss.

Um Fehler aufgrund eines fehlerbehafteten Modells oder einer Konfiguration zu vermeiden, sollte das Starten einer Simulation nur möglich sein, wenn das Modell und die Konfiguration validiert worden sind und keine Fehler beinhalten. Auch die Möglichkeit des Stoppens einer Simulation ist sinnvoll, bspw. wenn während des Simulationslaufes vom Modellierenden ein logischer Fehler innerhalb des Modells identifiziert wird.

FA 30 **(Simulation starten)**: Als Modellierender möchte ich eine Simulation jederzeit mit einem einfachen Knopfdruck starten können, um mich nicht weiter mit der .NET-Schnittstelle beschäftigen zu müssen.

FA 31 **(Simulationsstart überprüfen)**: Als Modellierender möchte ich, dass eine Simulation nur gestartet werden kann, wenn mein Modell und meine Konfiguration valide sind, um vorhersehbare Fehler direkt zu verhindern.

FA 32 **(Simulation stoppen)**: Als Modellierender möchte ich eine gestartete Simulation anhalten können, um ggf. Fehler zu korrigieren, die nach dem Start der Simulation aufgefallen sind.

Zustand der Simulation

Die Heuristiken von Nielsen (1994a) beinhalten die kontinuierliche Benachrichtigung des Modellierenden über den aktuellen Zustand des Systems. Aus diesem Grund soll der Modellierende nach dem Starten eines Simulationslaufs stets darüber informiert sein, in welchem Zustand sich die Simulation befindet.

Einer dieser Zustände ist der Bau bzw. die Kompilation des Projekts. Die Kompilation findet implizit vor der Ausführung der Simulation statt. Die Dauer dieses Prozesses hängt von der Größe des Projekts und vorhandenen Ressourcen der verwendeten Maschine ab und kann mehrere Sekunden oder auch Minuten dauern. Der darauf folgende Zustand ist die laufende Simulation. Dessen Fortschritt sollte dem Modellierenden kommuniziert werden, sodass dieser grob abschätzen kann, wie lange die Simulation noch benötigt. Tritt während der Simulation ein Fehler auf, so sollte der Simulationsabbruch dem Modellierenden entsprechend dargestellt werden.

Innerhalb des Terminals, in dem der Start-Befehl ausgeführt worden ist, wird die Ausgabe der Kompilation und der Ausführung der Simulation angezeigt. Dazu gehören zum einen

Ausgaben, die über die Konsole innerhalb des Modells (bspw. `Console.WriteLine(...)`) ausgegeben werden und zum anderen Ausgaben von Fehlern, die nach dem Start der Simulation aufgetreten sind. Dabei kann es sich um Kompilierungsfehler, die bereits beim Bauen der Anwendung aufgetreten oder Laufzeitfehler, die während der Simulation aufgetreten sind, handeln. Zu einem Kompilierungsfehler kann es bspw. bei falscher .NET Version kommen, während für einen Laufzeitfehler bereits ein fehlerhafter Indexzugriff ausreichend ist.

Nachdem der Simulationslauf abgeschlossen ist, sollen die Ausgaben der Simulation dem Modellierenden angezeigt werden. Treten Fehler während der Simulation auf, so müssen sie dem Modellierenden angezeigt werden, damit er sie entsprechend beheben kann.

FA 33 **(Simulationsstatus anzeigen)**: Als Modellierender möchte ich jederzeit über den Status der Simulation informiert werden, um zu wissen, in welcher Phase sich die Simulation befindet.

FA 34 **(Fortschritt anzeigen)**: Als Modellierender möchte ich jederzeit über den prozentualen Fortschritt der Simulation informiert werden, um abzuschätzen, wie viel Zeit die Simulation noch benötigt.

FA 35 **(Ausgabe einsehen)**: Als Modellierender möchte ich die Ausgabe meiner Simulation im Anschluss einsehen können, um mir relevante Werte zur Laufzeit auszugeben.

FA 36 **(Fehler einsehen)**: Als Modellierender möchte ich nach dem Abbruch einer Simulation Fehlermeldungen angezeigt bekommen, um zu verstehen, warum es zu diesem Fehler kam.

4.3.5 Analyse

Der letzte Schritt des Workflows eines Modellierenden ist die Analyse. Die Analyse der Simulationsergebnisse ist stark von der jeweiligen Domäne abhängig: Ist bei Simulationen eines Nationalparks bspw. die Anzahl der Elefanten relevant, so kann bei einem Verkehrsnetz die Positionierung der Autos von Interesse sein. Aus diesem Grund existiert im Kontext vom MARS-Framework kein festgelegtes Werkzeug, welches ausschließlich

für die Analyse verwendet wird. Für einfache Darstellungen werden Python-Skripte geschrieben, die auf die Ergebnisdaten der jeweiligen Simulation zugeschnitten sind.

Da sich die Darstellungen je nach Domäne unterscheiden, soll der MARS-Explorer möglichst einfache und allgemeine Darstellungsformen unterstützen, die sich auf die Agenten konzentrieren.

Eine der gewählten Darstellungsformen ist ein zweidimensionaler Graph, der auf der X-Achse den Simulationsfortschritt in Prozent und auf der Y-Achse die Anzahl der ausgewählten Agenten darstellt. Jeder Agent, der innerhalb der Simulation vorkommt, kann ausgewählt werden und wird als eigene Linie innerhalb des Graphen dargestellt. So lässt sich nachvollziehen, wie sich die Population eines jeden Agenten über den Simulationszeitraum entwickelt.

Eine zweite allgemeingültige Darstellungsform, die innerhalb des MARS-Explorer Anwendung finden soll, ist ein dreidimensionales Blasendiagramm. Die X- und Y-Achsen stellen jeweils örtliche Koordinaten innerhalb der Simulation zu einem bestimmten Simulationsfortschritt dar. Befindet sich bspw. ein Agent zum Simulationsfortschritt 1% an den Koordinaten $(1, 1)$, so wird eine Blase im Graphen an dieser Stelle dargestellt. Die Größe dieser Blase ist dabei abhängig von der Anzahl der Agenten, die sich zu dem zuvor ausgewählten Zeitpunkt auf dieser Koordinate befinden. Je mehr Agenten sich dort befinden, desto größer ist diese im Verhältnis zu den Blasen mit weniger Agenten. Diese Darstellungsform ermöglicht eine räumliche Einordnung der Agenten innerhalb der Simulation.

FA 37 **(Agentenanzahl einsehen)**: Als Modellierender möchte ich die Anzahl der Agenten zu einem bestimmten Fortschritt innerhalb der Simulation anhand eines Graphen ablesen können, um die Entwicklung der Agenten über einen bestimmten Zeitraum zu beobachten.

FA 38 **(Agentenpositionen einsehen)**: Als Modellierender möchte ich die Position der Agenten zu einem bestimmten Fortschritt innerhalb der Simulation anhand eines Graphen ablesen können, um die räumliche Entwicklung der Agenten über einen bestimmten Zeitraum zu beobachten.

FA 39 **(Ergebnisse filtern)**: Als Modellierender möchte ich die Ergebnisse nach den in der Simulation vorkommenden Agenten filtern können, um für meine aktuelle Forschungsfrage irrelvante Daten auszublenden.

4.3.6 Projektverwaltung

Die Verwaltung verschiedener Projekte ist ein Workflow, der indirekt von den Modellierenden durchlaufen wird, indem ein neues Projekt in Form einer vorgegebenen Ordnerstruktur erstellt wird.

Da ein MARS-Modell gewisse Vorraussetzungen – wie bspw. die installierte Abhängigkeit des Framework oder die Existenz eines Einstiegspunktes – mit sich bringt, wird den angehenden Modellierenden i.d.R. ein vorgefertigtes Projekt zur Verfügung gestellt, welches im weiteren Verlauf der Modellierung erweitert wird. Dem Benutzer sollte es mithilfe des MARS-Explorer jederzeit möglich sein, eigene Projekte zu erstellen, ohne über ein ausgeprägtes Verständnis des internen Aufbaus zu verfügen. Dazu wird zur Erstellung eines neuen Projekts ein minimales Modell aufgesetzt und eine Konfiguration mit Standardwerten gefüllt.

Untersuchungen von Bodine u. a. (2020) zeigen, dass das Erstellen eines neues Projekts ohne jegliche Vorlagen oder Hilfestellungen für angehende Modellierende ein Problem darstellen kann. Deshalb sollte es ebenfalls ermöglicht werden, ein funktionierendes Beispielprojekt als Vorlage zu verwenden. Genau wie bei der Modellierung sollten standardmäßige Aktionen, wie das Umbenennen und Löschen eines Projekts möglich sein.

Die Anforderungen *Beispielmodelle ansehen* (FA 17) und *Informationen zu Beispielmodellen einsehen* (FA 18) wurden bereits in der Modellierung aufgestellt und sind ebenso für diesen Anwendungsfall gültig.

FA 40 **(Projekte einsehen)**: Als Modellierender möchte ich alle meine Projekte dargestellt bekommen, um eine Übersicht über meine bisherigen Projekte zu erlangen.

FA 41 **(Projekt erstellen)**: Als Modellierender möchte ich ein neues und lauffähiges Projekt erstellen können, um neue Forschungsfragen bearbeiten zu

können, ohne meine bisherigen Projekte löschen zu müssen und ohne über Fachwissen bzgl. des internen Aufbaus zu verfügen.

FA 42 **(Beispielmodelle kopieren)**: Als Modellierender möchte ich alle Beispielprojekte kopieren können, um mich an diesen orientieren und sie als Ausgangspunkt verwenden zu können.

FA 43 **(Projekte umbenennen)**: Als Modellierender möchte ich meine bisherigen Projekte umbenennen können, um einen aussagekräftigeren Titel für mein Projekt zu wählen.

FA 44 **(Projekte löschen)**: Als Modellierender möchte ich meine bisherigen Projekte löschen können, um nicht mehr benötigte Projekte zu entfernen.

4.4 Aufstellung der nicht-funktionalen Anforderungen

Neben den zuvor aufgestellten funktionalen Anforderungen werden innerhalb dieses Kapitels nicht-funktionale Anforderungen (NFA) aufgestellt, die die Rahmenbedingungen des MARS-Explorer bestimmen. Das Aufstellen entsprechender Rahmenbedingungen wird von Phillips (1997, S. 39-40) in dem von ihm beschriebenen Prozess zu einer erfolgreichen Lernsoftware gefordert.

Diese Rahmenbedingungen basieren auf der ISO/IEC 25010 (2011), welche verschiedene Charakteristiken von Software kategorisiert und beschreibt. Innerhalb der ISO existieren Charakteristiken, die für die Beantwortung der Forschungsfrage nicht essentiell sind. Dazu gehört bspw. eine optimierte Ressourcennutzung, die zwar in Zukunft beachtet werden sollte, aber für die Beantwortung der Forschungsfrage keine Rolle spielt. Die relevanten Charakteristiken wurden anhand der folgenden Faktoren bestimmt:

- Analyse der Anwendungsfälle (Kapitel 4.3)
- Anforderungen an didaktische Software (Kapitel 3.3)
- Beobachtungen der Forschungsgruppe

Folgende Rahmenbedingungen bzw. NFA wurden aufgestellt:

Tabelle 4.1: Abgeleitete nicht-funktionale Anforderungen aus der Analyse der Workflows und den zuvor beschriebenen didaktischen Anforderungen basierend auf ISO/IEC 25010 (2011)

Anforderung	Funktionale Tauglichkeit
Beschreibung	Der MARS-Explorer kann den Modellierenden nur einen Mehrwert bieten, wenn die analysierten funktionalen Anforderungen umgesetzt wurden. Außerdem muss die korrekte Funktionsweise sichergestellt werden.
Test-Kriterien	• Es werden alle funktionalen Anforderungen aus Kapitel 4.3 umgesetzt • Es existieren Tests, die Teile der Funktionalitäten testen • Es werden Nutzertests durchgeführt, die ebenfalls Aufschluss über die korrekte Funktionsweise geben
Anforderung	Kompatibilität
Beschreibung	Neben dem .NET-SDK existieren voraussichtlich weitere externe Systeme mit denen der MARS-Explorer kommunizieren muss. Im Rahmen der Kompatibilität soll sichergestellt werden, dass das Zusammenspiel zwischen dem MARS-Explorer und den externen Systemen gewährleistet ist.
Test-Kriterien	• Externe Systeme, die im Zusammenhang mit dem MARS-Explorer stehen, werden von der Anwendung auf ihre Existenz geprüft • Sollten Abhängigkeiten nicht installiert oder fehlerhaft sein, sodass sie nicht von der Anwendung verwendet werden können, wird der Benutzer darauf hingewiesen
Anforderung	Performanz

Beschreibung	Die im Rahmen der Anwendung ausgeführten Funktionen sollen im angemessenen Zeitrahmen durchgeführt werden. Da die Ressourcen-intensiven Operationen, wie bspw. die Ausführung einer Simulation, allerdings vom Framework bzw. .NET-SDK ausgeführt werden, ist die Performanz abhängig von diesen externen Systemen und nicht Teil der Anforderung an diese Anwendung. Da es sich bei der vorliegenden Arbeit lediglich um eine erste Iteration der Anwendung handelt, liegt ein ausführliches Profiling der Performanz außerhalb dieser Arbeit. Stattdessen wird der Fokus auf den Funktionsumfang, die Vermittlung von Lehrinhalten sowie auf die Usability gelegt. Dennoch sollten Maßnahmen ergriffen werden, die einen flüssigen Arbeitsablauf ermöglichen.
Test-Kriterien	• Es werden Nutzertests auf verschiedenen Systemen durchgeführt, um eine ausreichende Performanz zu verifizieren • Es werden bei den Nutzertests keine Auffälligkeiten im Bezug auf die Performanz festgestellt
Anforderung	Usability
Beschreibung	Die Usability und dessen wichtige Rolle innerhalb der Lehre wurde bereits weitreichend in Kapitel 2.2 und 3.3 erläutert. Im Bezug auf den MARS-Explorer wird die Usability mithilfe von ausführlichen Diskussionen über Designentwürfe, die basierend auf dem erhaltenen Feedback iterativ verbessert werden, sichergestellt. Weitere Instrumente, welche die Usability des Systems sicherstellen sollen, sind die Vorstellung von Zwischenständen der Implementierung, sowie einzelne Benutzertests mit Mitgliedern der MARS Forschungsgruppe. Außerdem soll eine Umfrage durchgeführt werden, die sicherstellt, dass eine gute Usability gewährleistet ist.

Test-Kriterien	• Es werden Design-Entwürfe erstellt, die die Usability berücksichtigen • Es werden während der Entwicklung Nutzertests mithilfe der Forschungsgruppe durchgeführt, um die Usability kontinuierlich sicherzustellen und ggf. Änderungen rechtzeitig durchführen zu können • Es wird eine Umfrage an den Nutzern durchgeführt, wovon ein Teilaspekt der Usability dediziert wird
Anforderung	Sicherheit
Beschreibung	Die der Security zugeordneten Faktoren beziehen sich größtenteils auf Online-Systeme. Der Sicherheits-Faktor *Integrität* ist aber dennoch für den MARS-Explorer relevant. Er beschreibt, wie gut eine Anwendung im Falle der Übernahme durch einen unbefugten Dritten die Daten des Anwenders schützt. Dieser Fall muss berücksichtigt und entsprechende Maßnahmen getroffen werden.
Test-Kriterien	• Sicherheitsaspekte werden während des Architekturdesigns berücksichtigt • Es werden nachweisliche Maßnahmen getroffen, um den Zugriff auf die Daten des Anwenders zu verhindern
Anforderung	Wartbarkeit
Beschreibung	Durch eine wohl durchdachte Architektur soll sichergestellt werden, dass verschiedene Bausteine innerhalb der Anwendung austauschbar sind und somit Modifikationen in Zukunft unkompliziert durchgeführt werden können. Außerdem sollen Werkzeuge zur statischen Code-Analyse eingesetzt werden, um typische Problemfälle der eingesetzten Programmiersprache zu vermeiden.

Test-Kriterien	• Es werden Werkzeuge zur statischen Code-Analyse eingesetzt • Die Architektur der Anwendung folgt einem detailliert beschriebenen Konzept und/oder bekannten Entwurfsmustern
Anforderung	Portabilität
Beschreibung	Bei der Zielgruppe des MARS-Explorer handelt es sich um Studierende der Informatik. Da die Studierenden i.d.R. über eigene Endgeräte verfügen, schließt dies eine Fokussierung auf eine einzige Plattform aus. Um die Verfügbarkeit zu gewährleisten, soll deshalb eine entsprechende Technologie in der Entwicklung verwendet werden, die es ermöglicht, die Anwendung für die gängigen Plattformen MacOS, Windows und Linux bereitzustellen. Da die Anwendung in Zukunft in der Lehre verwendet werden soll, spielt außerdem die Anpassbarkeit eine große Rolle. Ist eine Anwendung anpassbar, so kann diese für neue Soft- oder Hardware in zumutbarer Zeit angepasst werden. Dieser Faktor ist ebenso von der verwendeten Technologie abhängig. Deswegen soll bei der Wahl der Technologie auf eine lange Historie, sowie auf regelmäßige Updates geachtet werden.
Test-Kriterien	• Es werden verschiedene Technologien abgewägt und der Faktor der Langlebigkeit mit hohem Gewicht bewertet • Es existiert eine installierbare Datei für die aktuellste Version von Windows • Es existiert eine installierbare Datei für mind. eine aktuelle Linux-Distribution • Es existiert eine installierbare Datei für die aktuellste Version von MacOS

4.5 Abgrenzung

Die zu entwickelnde Anwendung soll keine vollumfängliche IDE oder andere Experten-Tools ersetzen, sondern lediglich als Einstiegspunkt in das Thema von MAS sowie in das MARS-Framework dienen. Der MARS-Explorer abstrahiert den internen Aufbau solcher Projekte und legt den Fokus auf die bessere Verständlichkeit und Übersichtlichkeit statt auf Flexibilität und Mächtigkeit. Sobald sich die Modellierenden in die beiden Themen ausführlich eingearbeitet haben, wird es Anwendungsfälle geben, die der MARS-Explorer nicht abdeckt und für die der volle Zugriff auf alle Dateien des Projekts benötigt wird. Aus diesem Grund soll der Modellierende jederzeit vollen Zugriff auf die Dateien haben, um die Projekte auch nach der Lernphase verwenden zu können. Dies wird dadurch gewährleistet, dass im Hintergrund keine eigenen Dateiformate erzeugt werden, sondern lediglich mit der Reinform eines .NET Projekts gearbeitet wird.

Aus Zeitgründen werden Aspekte wie Performanz oder ausführliche Testumgebungen vernachlässigt, um die funktionale Vollständigkeit und eine gute Usability zu gewährleisten. Das Ergebnis dieser Ausarbeitung soll damit ein erster Schritt zu einer langlebigen Lehrplattform für die zuvor genannten Themen sein.

4.6 Fachliches Datenmodell

Das in Abbildung 4.4 dargestellte fachliche Datenmodell gibt einen groben Überblick über die Daten, die innerhalb des MARS-Explorer verwendet werden. Das Modell wurde auf Basis der FA aufgestellt.

Das Objekt *Project* ist die übergeordnete Entität und beinhaltet alle weiteren Daten, die zur Modellierung und zur Ausführung der weiteren Workflows benötigt werden. Zu den Daten gehört eine beliebig große Menge von *Class*-Objekten. Für ein funktionsfähiges Modell muss mindestens der Einstiegspunkt der Simulation vorhanden sein. Eine *Class* besitzt neben dem eigenen Namen ein Attribut, welches den Inhalt der eigenen Datei darstellt (*Content*). Da alle Klassen nach der Bearbeitung validiert werden, wodurch der Inhalt auf syntaktische Fehler überprüft wird, beinhaltet *Class* ein *Errors*-Attribut. In diesem werden die Fehlermeldungen zu der jeweiligen Klasse gespeichert, sodass sie dem Benutzer dargestellt werden können.

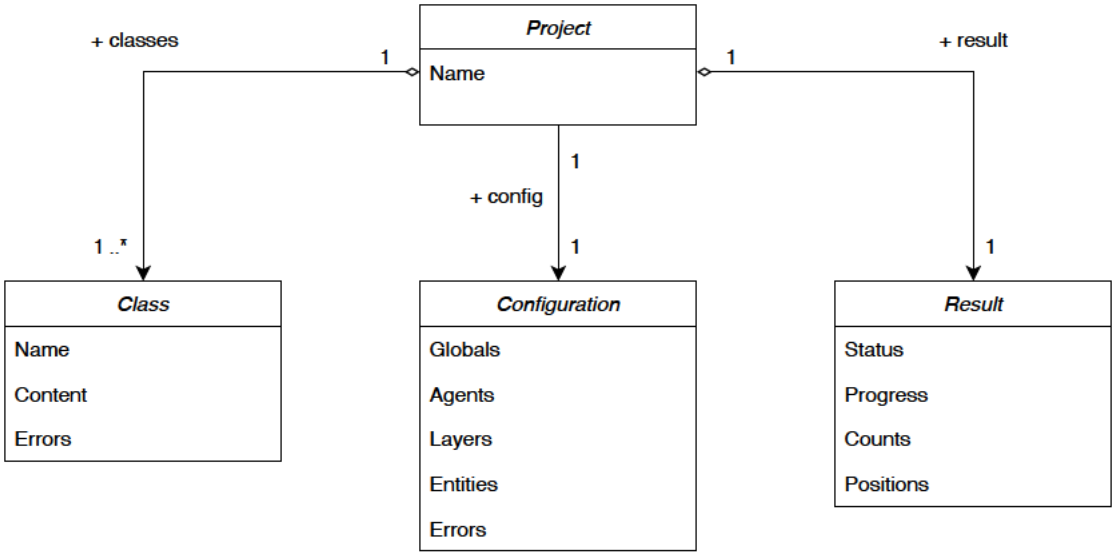


Abbildung 4.4: Fachliches Datenmodell zu den im MARS-Explorer verwendeten Daten

Neben dem Modell muss innerhalb eines Projekts die Konfiguration gespeichert werden, die alle Einstellungsmöglichkeiten beinhaltet. Für eine verbesserte Übersichtlichkeit werden innerhalb der Abbildung 4.4 nur die übergeordneten Properties aufgeführt (*Globals*, *Agents*, etc.). Das vollständige Schema kann in der Dokumentation des Framework betrachtet werden (MARS Group, 2021a). Ein *Project* besitzt nur eine einzige Konfiguration, die vom Benutzer im Laufe der Modellierung bearbeitet wird.

Die Ergebnisse einer Simulation werden im Datenmodell mit dem *Result*-Objekt abgebildet. Um wie in den FA gefordert, den Zustand einer Simulation anzeigen zu können, existiert ein Attribut *Status*, sowie ein Attribut *Progress*. *Status* spiegelt den allgemeinen Zustand der Simulation wider, während *Progress* dazu verwendet wird, um den genauen Fortschritt einer Simulation zu speichern. Die Ergebnisse werden während der Simulation kontinuierlich in *Counts* und *Positions* aggregiert, sodass die Daten zur Laufzeit in Form der erläuterten Graphen dargestellt werden können. *Counts* beinhaltet die Anzahl der Agenten und *Positions* die Positionsdaten der Agenten pro Simulationsschritt.

4.7 Mockups

Im Rahmen der Analyse wurde der in Abbildung 4.5 dargestellte Prozess etabliert. Zusammen mit Mitgliedern der MARS-Forschungsgruppe wurden in einem ersten Schritt die

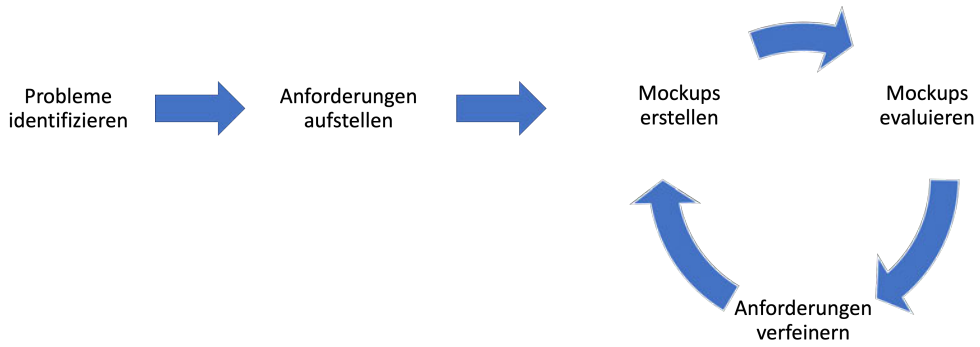


Abbildung 4.5: Kontinuierlicher Design-Prozess, der während der Konzeption des MARS-Explorer zusammen mit Mitgliedern der MARS Group angewandt worden ist

Probleme im Zusammenhang mit der Lehre von MAS identifiziert und daraufhin Anforderungen aufgestellt. Die Anforderungen wurden verwendet, um Mockups anzufertigen, welche die durchdachten Features in einem graphischen Entwurf zeigen. Somit wurde eine Basis für weitere Diskussionen geschaffen. Im Rahmen von Diskussionen wurden die Mockups evaluiert, und ggf. zusammen mit den Anforderungen angepasst. Dieser Prozess wurde wiederholt durchlaufen, sodass die Anforderungen und die Mockups immer weiter verfeinert wurden. Ein derartiger Ablauf wird auch von Beale und Sharples (2002) sowie von Phillips (1997, S. 38-39) dringend empfohlen.

Beim Entwurf der Mockups wurde besondere Rücksicht auf die Usability genommen, da sie – wie bereits mehrfach hervorgehoben – einen besonderen Einfluss auf den Lernerfolg hat. Außerdem wurde basierend auf den Erfahrungswerten der Mitglieder der MARS-Forschungsgruppe darauf geachtet, dass nicht zu viele Navigationsebenen eingeführt werden. So soll verhindert werden, dass der Benutzer erst durch viele Schritte klicken muss, um schlussendlich eine Simulation zu starten.

Zur Erläuterung der Mockups werden nummerierte rote Punkte verwendet, um sich innerhalb des Textes eindeutig auf die jeweiligen Bereiche einer Abbildung beziehen zu können. Die innerhalb dieses Kapitels vorgestellten Mockups gelten als Ausgangspunkt für die Implementierung des MARS-Explorer. Im Laufe der Entwicklung sind jedoch noch weitere Verbesserungsmöglichkeiten aufgefallen, die ohne eine erneute Iteration über das Design eingebaut worden sind, sodass die implementierte Oberfläche in Teilen leicht von den hier gezeigten Mockups abweicht. Insbesondere das Farbkonzept, sowie kleine Än-

Abbildung 4.6: Mockup zum Grundaufbau des MARS-Explorer

derungen an Abständen und Formen sind davon betroffen. Die Navigation sowie die Funktionalitäten bleiben prinzipiell aber unverändert.

4.7.1 Grundlegender Aufbau

Die Abbildung 4.6 zeigt den grundlegenden Aufbau des MARS-Explorer, welcher auf allen Mockups zu sehen sein wird.

Punkt 1 befindet sich innerhalb der globalen Navigationsleiste. Eine solche muss laut Phillips (1997, S. 64) und Beale und Sharples (2002) gut durchdacht und einfach strukturiert sein. Phillips (1997, S. 65-66) unterscheidet dabei zwischen verschiedenen Navigationskonzepten, wovon das Hierarchische innerhalb des MARS-Explorer Anwendung findet. Bei dieser Art der Navigation sind die Informationen in Form von Menüs in Unterpunkte gruppiert, um die Komplexität zu reduzieren. Dies hat den Vorteil, dass eine Rückkehr zu zuvor verarbeiteten Informationen sehr einfach ist. Außerdem wird durch die Anordnung indirekt eine Reihenfolge angegeben, da der Benutzer dazu tendiert, von oben nach unten durch die dargestellten Menüpunkte zu navigieren.

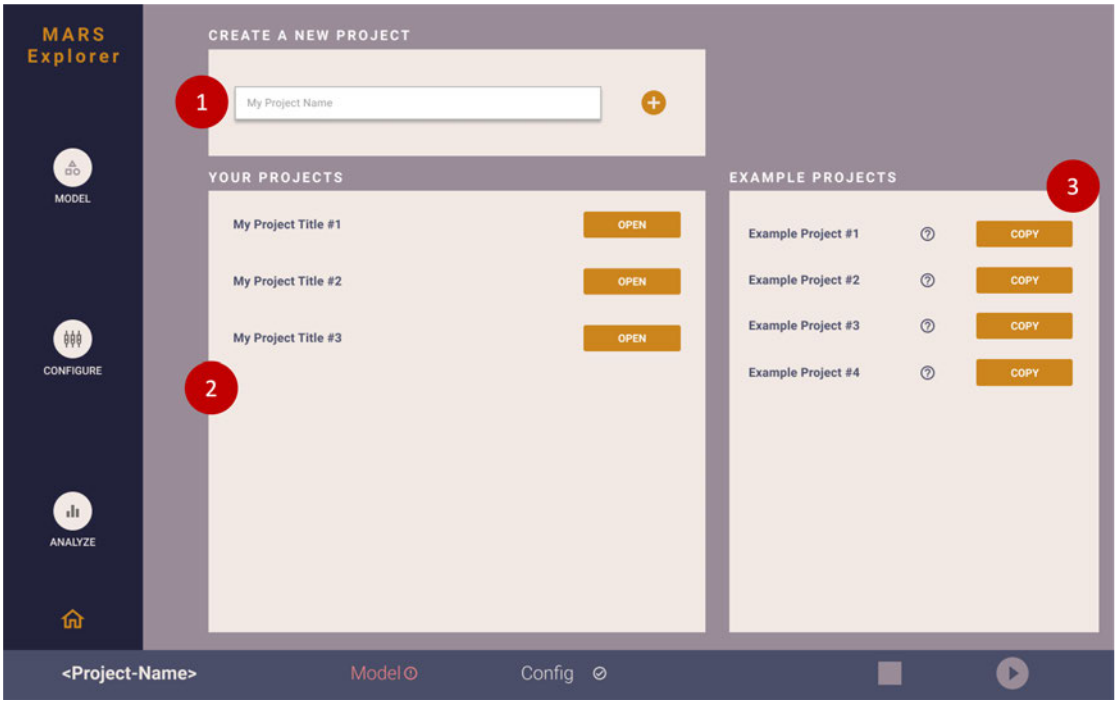


Abbildung 4.7: Mockup zur Projektverwaltung

Wie in der Navigationsleiste zu sehen, sind die einzelnen Menüpunkte mit Ausnahme von *Simulation* in die zuvor erläuterten Arbeitsabläufe innerhalb des MARS-Framework unterteilt. Da es bei den Arbeitsabläufen ebenfalls eine Reihenfolge gibt, wie diese durchlaufen werden, ist das hierarchische Navigationskonzept besonders geeignet. Der unterste Menüpunkt mit dem Haus-Symbol stellt die Navigation zum Startbildschirm dar. Innerhalb dieses Abschnitts können alle Projekte verwaltet werden. Nach Beale und Sharples (2002) sollte es jederzeit möglich sein, zum Startbildschirm der Anwendung zurückzukehren, was durch das Haus-Symbol erfüllt wird. Dieses Symbol ist im Gegensatz zu den anderen in Orange eingefärbt, da sich der Benutzer aktuell innerhalb dieses Navigationspunktes befindet.

Der Bereich in dem Punkt 2 dargestellt ist, wird immer dazu verwendet, um den Hauptinhalt des jeweils ausgewählten Navigationspunktes darzustellen. Punkt 3 zeigt die so genannte Quick-Start-Leiste, die im Kapitel 4.7.5 näher erläutert wird.

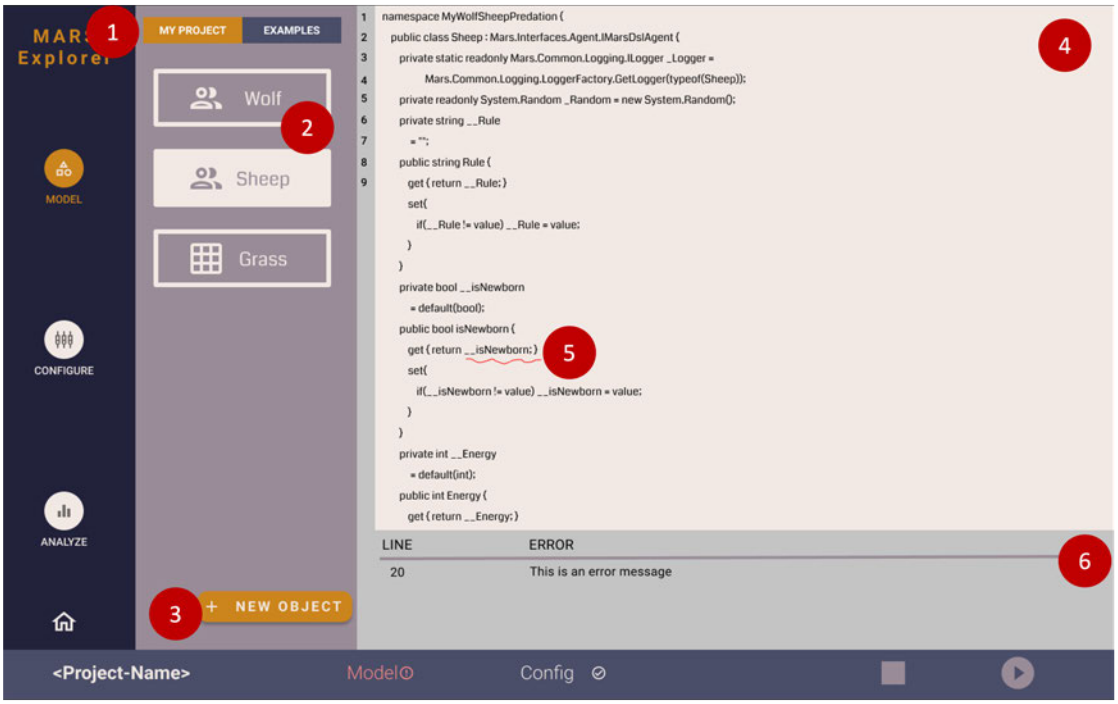


Abbildung 4.8: Mockup zur Modellierung: Eigenes Projekt

4.7.2 Projektverwaltung

Abbildung 4.7 zeigt die Projektverwaltung, welche in drei Bereiche aufgeteilt ist. Punkt 1 zeigt den ersten Bereich, in dem durch die Eingabe eines einzigartigen Namens ein Projekt erstellt werden kann. Im Bereich von Punkt 2 werden alle eigenen Projekte angezeigt, die mit einem Klick auf den Button *Open* geöffnet werden können. Mithilfe eines klassischen Kontextmenüs können mit einem Rechtsklick auf das jeweilige Projekt weitere Aktionen – wie das Löschen – ausgeführt werden. Punkt 3 zeigt den Bereich für die Beispielprojekte. Diese werden ebenso in einer Liste dargestellt und können mit einem Klick auf den Button *Copy* kopiert und daraufhin geöffnet werden. Durch das Hovern über das Fragezeichen-Icon werden genauere Details zum jeweiligen Beispielmodell erläutert.

4.7.3 Modellierung

Das Mockup aus Abbildung 4.8 zeigt den Abschnitt zur Modellierung innerhalb des MARS-Explorer. In der obersten Leiste bei Punkt 1 besteht die Möglichkeit, zwischen der Ansicht des eigenen Projekts und der Ansicht der Beispielprojekte zu wechseln. In der hier

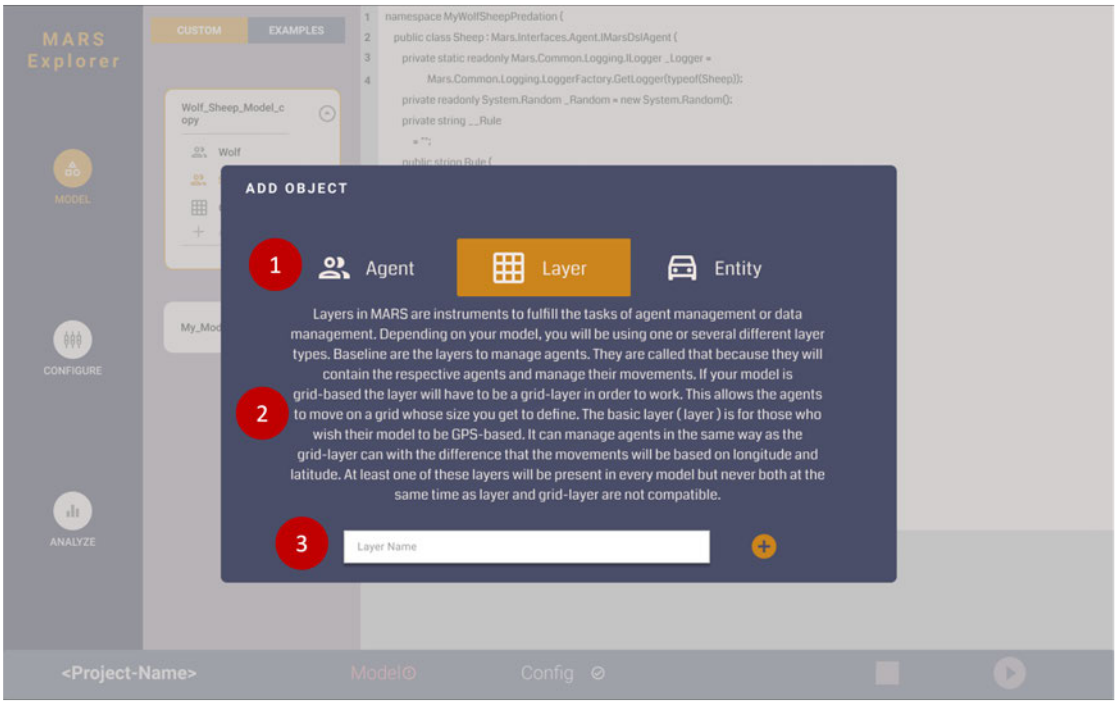


Abbildung 4.9: Mockup zur Modellierung: Objekt hinzufügen

dargestellten Auswahl werden die eigenen Klassen in der Liste bei Punkt 2 dargestellt. Hier geben die Symbole einen Hinweis darauf, um was für einen Objekt-Typen es sich handelt. Während *Wolf* und *Sheep* Agenten-Objekte sind, ist *Grass* ein Layer. Auch hier kann der Rechtsklick verwendet werden, um die Klassen zu löschen oder umzubenennen. Punkt 4 zeigt das Fenster zur Bearbeitung des Quellcodes. In diesem Beispiel ist der Sheep-Agent geöffnet. Fehler innerhalb des Quellcodes werden durch rote Linien – wie sie bei Punkt 5 zu sehen sind – dargestellt. Der Grund für den Fehler kann entweder durch Hovern über diese Linie oder durch das untere Fenster bei Punkt 6 nachvollzogen werden. Im unteren Fenster werden alle Fehlermeldungen der aktuellen Klasse gesammelt dargestellt.

Mithilfe des Buttons bei Punkt 3 kann ein Dialog geöffnet werden, in dem neue Objekte erstellt werden können. Dieser Dialog ist in Abbildung 4.9 dargestellt. Innerhalb des Dialogs hat der Benutzer bei Punkt 1 die Möglichkeit zwischen den drei Objekt-Typen zu wählen und sich Erklärungen bei Punkt 2 durchzulesen. Innerhalb des Eingabefeldes bei Punkt 3 muss lediglich noch ein Name für das ausgewählte Objekt eingegeben und dieser mit dem nebenstehenden Plus-Button bestätigt werden.

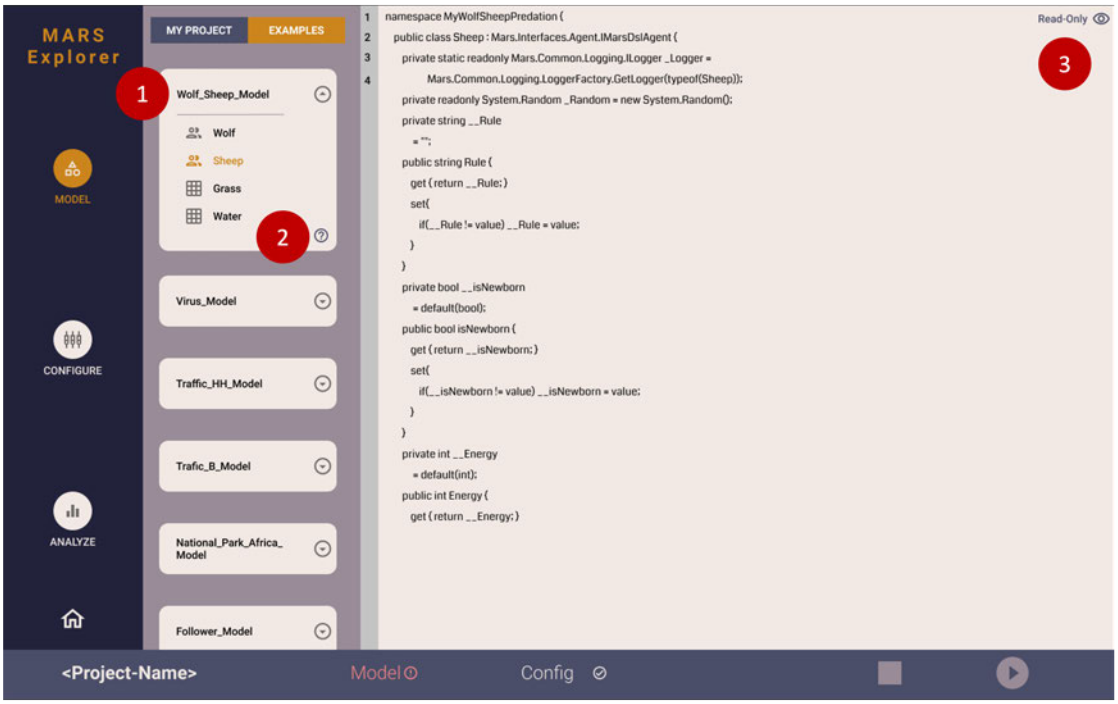


Abbildung 4.10: Mockup zur Modellierung: Beispielprojekte

Abbildung 4.10 zeigt die Ansicht der Beispielmodelle innerhalb der Modellierung. Diese werden in einer Liste von aufklappbaren Elementen dargestellt. Neben Punkt 1 ist das Beispielmodell *Wolf_Sheep_Model* aufgeklappt, wodurch die enthaltenen Klassen offenbart werden. In diesem Fall befinden sich darin zwei Agenten- und zwei Layer-Objekte. Mit einem Klick auf das Fragezeichen-Symbol bei Punkt 2 kann eine Erklärung zu diesem Projekt geöffnet werden. Der Bereich um Punkt 3 zeigt den geöffneten Agenten *Sheep* aus dem Beispielprojekt, welcher im Read-Only Modus geöffnet wurde. Dadurch können die Inhalte nicht bearbeitet, aber angesehen und kopiert werden.

4.7.4 Konfiguration

In Abbildung 4.11 ist der Tab zur Konfiguration des Szenarios dargestellt. Dieser ist in drei größere Bereiche unterteilt:

General (Bereich 1) Allgemeine Einstellungen zur Simulation, wie die Dauer, der Simulationszeitraum, die Einheiten, etc.

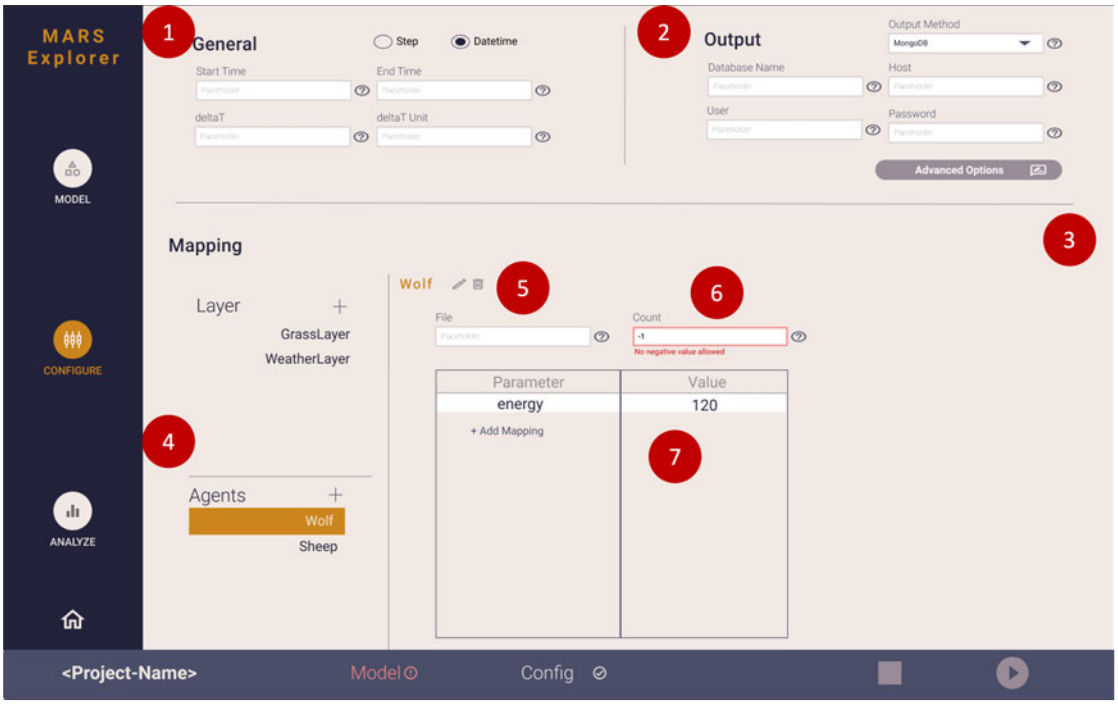


Abbildung 4.11: Mockup zur Konfiguration

Output (Bereich 2) Einstellungen zu verschiedenen Ausgabemöglichkeiten, die zum Zeitpunkt der Mockups noch mit eingeplant, letztendlich aus Gründen der entstehenden Komplexität für den Modellierenden aber gestrichen worden sind.

Mapping (Bereich 3) Einstellungsmöglichkeiten zu den verschiedenen Objekten innerhalb des Modells.

Im Bereich *General* können verschiedene allgemeine Einstellungen bzgl. der Konfiguration getroffen werden. Durch die Auswahl des Radio-Buttons *Datetime* werden die dazugehörigen Felder für den Benutzer eingeblendet (*Start Time* und *End Time*). Jedes Feld verfügt über ein Eingabeelement und ein Fragezeichen-Symbol. Nach einem Klick auf das Fragezeichen-Symbol werden dem Benutzer Informationen zum jeweiligen Feld angezeigt.

Innerhalb des Mappings (Bereich 3) befinden sich vier weitere Punkte, denen es der Erläuterung bedarf. Bei Punkt 4 ist eine Liste der innerhalb der Konfiguration verwendeten Objekt-Typen zu sehen. In diesem Fall befinden sich keine Entitäten in den Listen, da sie innerhalb des Modells nicht verwendet worden sind. Mit einem Klick auf das Plus-Symbol

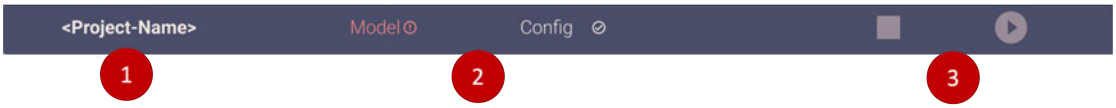


Abbildung 4.12: Mockup zur Simulation

können weitere Objekte innerhalb der Konfiguration parametrisiert und eingestellt werden. Links neben Punkt 5 sieht man bspw. den aktuell ausgewählten Agenten *Wolf*, welcher mit dem Stift-Symbol umbenannt oder mithilfe des Mülleimer-Symbols gelöscht werden kann. Eine solche Änderung hat keine Auswirkung auf das Modell, sondern lediglich auf die Einträge innerhalb der Konfiguration. Unter Punkt 5 befindet sich ein Datei-Eingabefeld mit dessen Hilfe vorgefertigte Daten zur Initialisierung bereitgestellt werden können. Unter Punkt 6 ist ein beispielhaftes fehlerhaftes Feld zu sehen, das validiert wurde und den Grund für den invaliden Zustand angibt. Innerhalb der Tabelle bei Punkt 7 können Attribute, die im Modell entsprechend markiert worden sind, parametrisiert werden. Dazu muss zum einen der identische Name des Parameters, sowie der Wert, der dem entsprechenden Attribut zugeordnet werden soll, angegeben werden.

4.7.5 Simulation

Der Teil der Oberfläche, der die Simulation verwaltet, wurde bereits innerhalb der zuvor dargestellten Mockups gezeigt. Dieser Abschnitt ist isoliert in Abbildung 4.12 als Schnellstart-Leiste dargestellt. Am Anfang dieser Leiste wird bei Punkt 1 der Name des aktuellen Projekts angezeigt, um dem Benutzer einen globalen Orientierungspunkt zu geben, in welchem Projekt er sich aktuell befindet. Punkt 2 zeigt die Zustände des Modells und der Konfiguration. Wie in den Anforderungen zuvor erwähnt, sollte jederzeit zu sehen sein, ob eine der beiden Komponenten einen oder mehrere Fehler beinhaltet. Bei erfolgreicher Validierung wird neben der jeweiligen Komponente ein grüner Haken und bei fehlgeschlagener Validierung ein rotes Kreuz dargestellt. Die beiden Zustände haben auch Einfluss auf die bei Punkt 3 gezeigten Elemente, die für das Starten und Stoppen der Simulation verantwortlich sind. Der Stop-Knopf kann nur verwendet werden, wenn eine Simulation gestartet worden ist, während der Start-Knopf nur betätigt werden kann, wenn noch keine Simulation läuft und wenn die Komponenten *Modell* und *Konfiguration* beide valide sind. Dies soll Frust verhindern, indem vorhersehbare Fehler ausgeschlossen werden. Wurde eine Simulation gestartet, wird rechts neben dem Start-Knopf eine Fort-

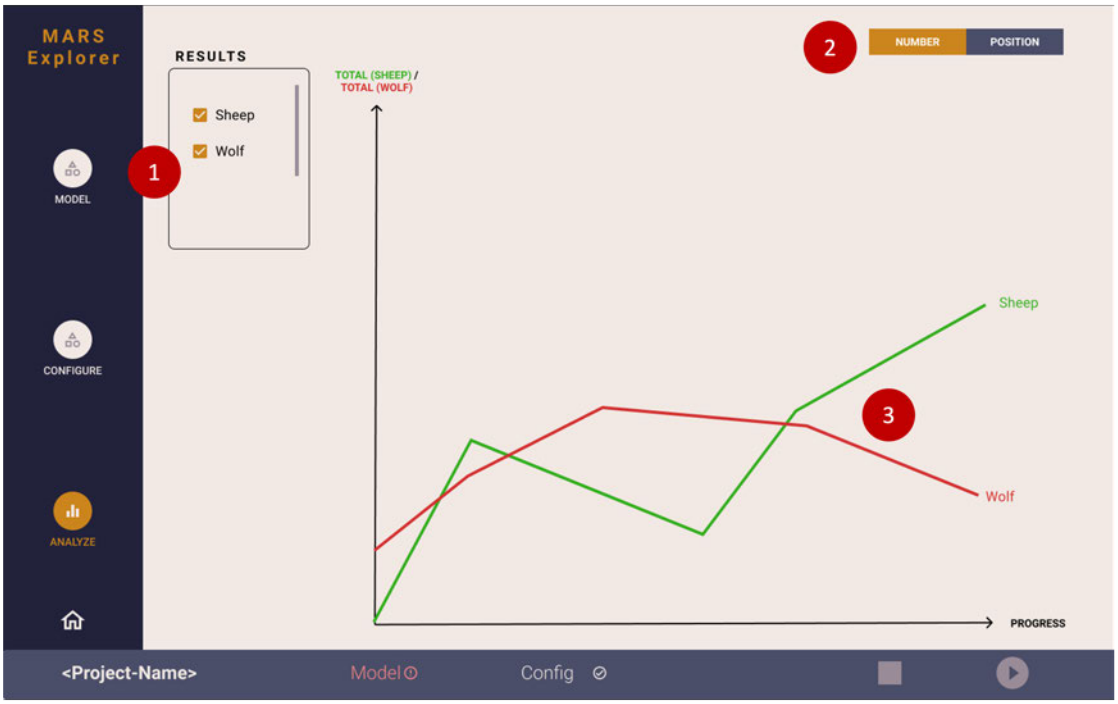


Abbildung 4.13: Mockup zur Analyse

schrittsanzeige dargestellt. Das Ziel dieser Komponente war es, dass der Benutzer die Möglichkeit bekommt, sein Modell aus jeder Phase testen zu können.

4.7.6 Analyse

Das letzte Mockup ist in Abbildung 4.13 dargestellt und zeigt den Analyse-Tab. In diesem werden die Ergebnisse der letzten oder der laufenden Simulation visualisiert. Die Liste von Checkboxes bei Punkt 1 wird dazu verwendet, um die Ergebnismenge zu filtern. In diesem Fall werden die Ergebnisse von den Agenten *Sheep* und *Wolf* im Graphen in Punkt 3 visualisiert. Punkt 2 zeigt die Möglichkeit zwischen den Darstellungsformen zu wechseln. Der aktuelle Graph zeigt die Anzahl der Agenten pro Simulationsfortschritt. Wird *Position* ausgewählt, wird mithilfe eines Blasengraphs die Position der ausgewählten Agenten pro Simulationsfortschritt dargestellt.

5 Design

In diesem Kapitel wird die Architektur des MARS-Explorer im Detail vorgestellt. Begonnen wird mit der Erklärung des verwendeten Entwurfsmusters. Daraufhin folgt eine schrittweise Vertiefung der Architektur der Anwendung in Form des von Starke (2015, S. 154-159) definierten Sichtenmodells. Abschließend werden architektonische Maßnahmen erläutert, die in Bezug auf die NFA *Sicherheit* getroffen worden sind.

5.1 Entwurfsmuster

Der visuelle Teil der Anwendung, die der Modellierende sehen wird, macht den mit Abstand größten Anteil an Codeumfang innerhalb des Projekts aus. Um die Komplexität für diesen Teil der Anwendung dennoch gering zu halten, sowie die NFA *Wartbarkeit* zu erfüllen, wurde sich dazu entschieden, das Model-View-Presenter Muster (MVP) in Form der sog. *Passive View* zu verwenden, welche von Fowler (2006) definiert worden ist. Im Kern sorgt dieses Muster für eine strikte Trennung zwischen der Darstellung der Inhalte und der Funktionalität. Die Funktionsweise des Musters ist in Abbildung 5.1 zu sehen. Nach Fowler existieren innerhalb dieses Musters folgende drei zentrale Rollen:

Model Enthält die Domain-Logik und kann auch Teile der Geschäftslogik enthalten. Das *Model* ist komplett unabhängig und unwissend vom *Presenter* und der *View*.

View Stellt lediglich die Interaktionselemente und Inhalte dar, die vom *Presenter* weitergegeben wurden. Beinhaltet keinerlei Logik und leitet jegliche Interaktion des Benutzers an den *Presenter* weiter.

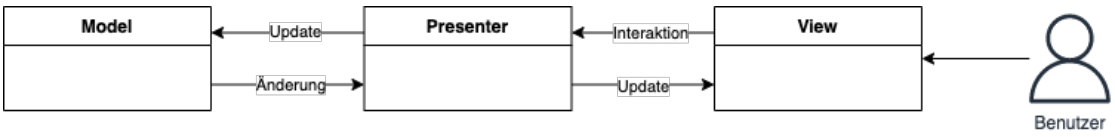


Abbildung 5.1: Funktionsweise des Model-View-Presenter Entwurfsmusters

Presenter Verbindet die *View* und das *Model*, indem es die logischen Abläufe steuert. Reagiert auf die Interaktionen des Benutzers und führt auf Basis dessen eine oder mehrere Aktionen aus, die Auswirkungen auf das *Model* haben. Das *Model* benachrichtigt den *Presenter* im Falle von Änderungen mithilfe eines *Observer*-Mechanismus. Die Änderungen werden wiederum im *Presenter* verarbeitet und ggf. an den *Presenter* weitergereicht.

Im Gegensatz zum Ansatz der *Passive View*, existiert die *Variante Supervising Controller*, in welcher ein großer Teil der Befugnisse des *Presenters* auf die *View* übertragen wird (Fowler, 2006).

Zusätzlich zu dem grundlegenden MVP werden die Entwurfsmuster *Observer* und *Mediator* angewandt. Das *Observer*-Muster beschreibt eine Möglichkeit, wie sog. *Observer* Interesse am Zustand eines sog. *Subjects* bekunden können (Gamma, 2004, 360-368). Dazu stellt das *Subject* eine Möglichkeit bereit, wie sich Interessenten zur Benachrichtigung über Änderungen am besagten *Subject* registrieren können (Gamma, 2004, 360-368). Wie bereits zuvor erwähnt, werden Updates des Models mithilfe des *Observer*-Mechanismus bekannt gegeben. Dadurch muss das *Model* nichts von seinen *Presenter*-Komponenten wissen, da sie sich für die Aktualisierung eigenhändig registrieren.

Das *Mediator*-Muster wird verwendet, um das Zusammenspiel einer Menge von Objekten zu koordinieren, indem die Ausführung der Funktionen der einzelnen Teilnehmer dem sog. *Mediator* überlassen wird (Gamma, 2004, S. 338-343). Innerhalb der Implementierung des MARS-Explorer ist aufgefallen, dass einzelne Funktionen durchaus komplex werden können. Diese Funktionen können mithilfe des *Mediators* aufgeteilt werden, sodass Teilfunktionen getrennt voneinander implementiert und innerhalb des *Mediators* zusammengesetzt werden können. Dies sorgt für eine bessere Übersichtlichkeit, Austauschbarkeit und Testbarkeit.

5.2 Sichten

Für eine verständliche Darstellung der entwickelten Lösung wird das Sichten-Konzept herangezogen, welches von Starke (2015, S. 154-159) konzipiert worden ist. Mithilfe dieses Konzeptes wird das entwickelte System aus verschiedenen Blickwinkeln betrachtet, um die Darstellungskomplexität des Gesamtsystems zu senken.

Starke vergleicht diese Art der Dokumentation eines Softwaresystems mit der einer Immobilie: Verschiedene Parteien benötigen vor der Umsetzung ihrer Tätigkeit in einer neuen Immobilie verschiedene Pläne, die für ihre jeweilige Aufgabe relevant sind und die andere Aspekte bewusst ausblenden. So benötigt bspw. ein Elektriker einen nach einer DIN formatierten Elektroplan, während ein Sanitärinstallateur bspw. Pläne der Wasserleitungen benötigt (Starke, 2015, S. 155). Ebenso verhalte es sich laut Starke mit der Systemdokumentation.

In dem Sichten-Konzept werden konkret vier Sichten definiert, die zum Gesamtverständnis des System essentiell sind: die Kontextsicht (bzw. Kontextabgrenzung), Bausteinsicht, Laufzeitsicht sowie die Verteilungssicht. Diese vier Sichten werden in den folgenden Unterkapiteln mithilfe von Diagrammen erläutert.

5.2.1 Kontextabgrenzung

In Abbildung 5.2 ist das entworfene System in der Kontextabgrenzung zu sehen. In dieser werden die für das System relevanten externen Systeme in Form einer Blackbox dargestellt, um vorerst einen Überblick über alle beteiligten Parteien zu erlangen (Starke, 2015, S. 162-163). Es werden außerdem die Schnittstellen des MARS-Explorer zu den Blackboxes in Form des Datenflusses und des verwendeten Kommunikationsmechanismus definiert.

Links in der Mitte der Abbildung ist der MARS-Explorer abgebildet, der dem Modellierenden Möglichkeiten zur Interaktion bietet. Wie der Anzahl der Pfeile vom und zum .NET SDK zu entnehmen ist, spielt dieses eine wichtige Rolle innerhalb des Gesamtsystems. Dem MARS-Explorer bietet das SDK, welches u.a. die .NET Kommandozeile beinhaltet, über den Kanal `stdin` Möglichkeiten zur Ausführung diverser .NET-spezifischer Befehle, wie der Installation von Abhängigkeiten (`dotnet add ...`) oder dem Bauen (`dotnet build`) und Starten der geschriebenen Modelle (`dotnet run`). Die Schnittstelle liefert dem MARS-Explorer daraufhin entweder die Ausgabe über `stdout` oder die Fehlermeldung über `stderr`, die der jeweilige Befehl erzeugt.

Mithilfe des SDK bzw. der von .NET bereitgestellten Kommandozeile wird die Simulation über das MARS-Framework gestartet. Das Framework öffnet vor dem Start der Simulation unter `ws://127.0.0.1:4567` einen Websocket, der es dem MARS-Explorer ermöglicht, eine Verbindung zu der Simulation herzustellen und dessen Ergebnisse während der Ausführung zu verarbeiten. Ausgaben, die während der Ausführung auftreten,

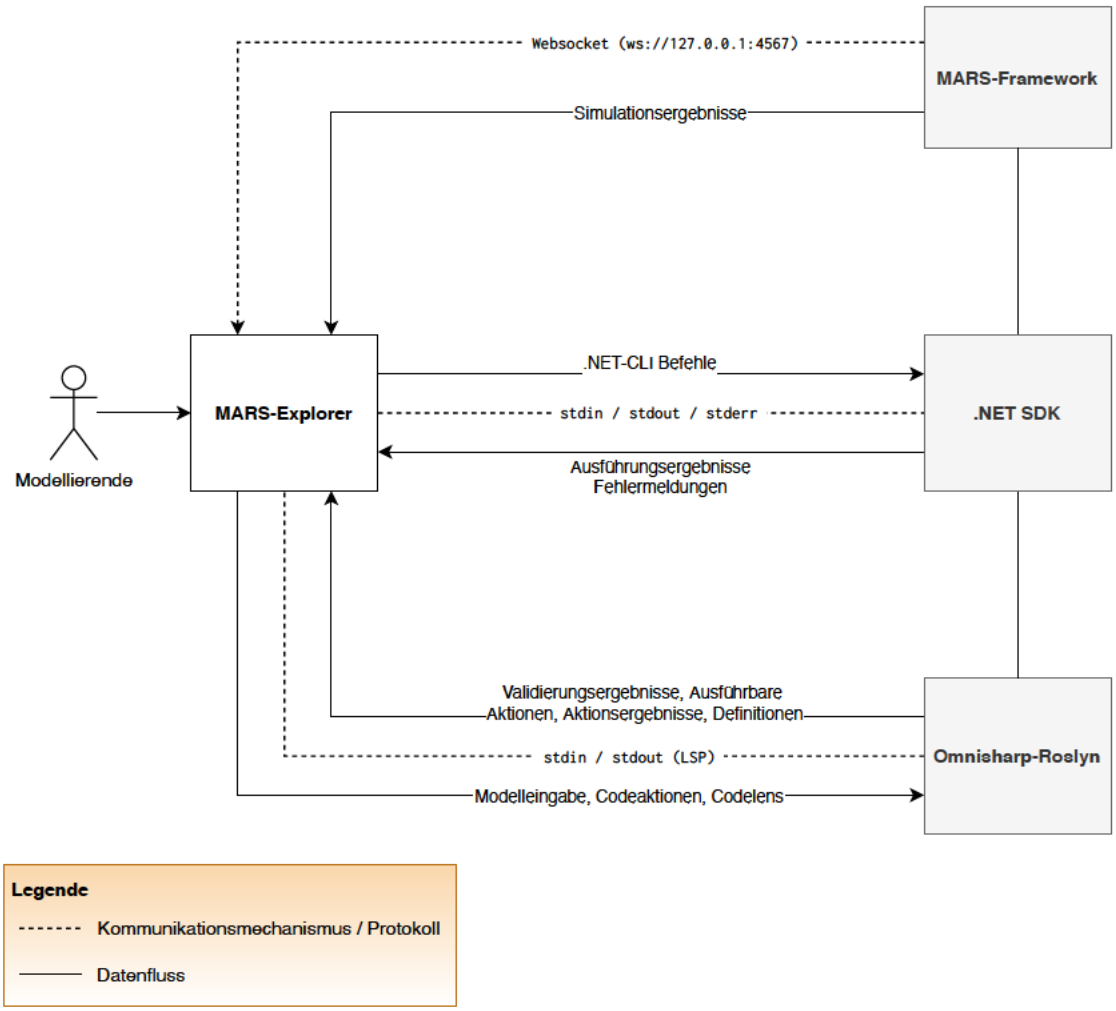


Abbildung 5.2: Diagramm zur Kontextabgrenzung

werden nach der Beendigung der Ausführung über die Standard-Kanäle der Kommandozeile (`stdout` oder `stderr`) an den MARS-Explorer geleitet, sodass diese Ausgaben bspw. dem Modellierenden angezeigt werden können.

Omnisharp-Roslyn verwendet die Implementierung von Roslyn, um eine Entwicklungsplattform für .NET über verschiedene Kommunikationsmechanismen, wie `stdin` und `stdout` oder `HTTP`, anzubieten (.NET Foundation, 2021). Roslyn ist eine Open-Source Implementierung des .NET Kompilators mit diversen Funktionalitäten wie u.a. Codeanalysen und Refactorings (Microsoft, 2021b). Die Kommunikation zwischen dem MARS-Explorer und Omnisharp-Roslyn findet über die Kanäle `stdin`, `stdout` und das von Microsoft definierte Language Server Protocol (LSP) statt.

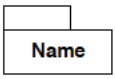
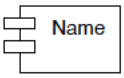
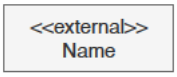
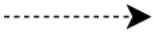
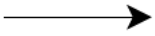
Das LSP wurde entworfen, um die Spezifika einer Programmiersprache von den verwendeten Entwicklungswerkzeugen zu trennen (Microsoft, 2021d). Durch das Protokoll ist es möglich, technologieunabhängig mittels einer Interprocess communication (IPC) eigene Entwicklungswerkzeuge zu implementieren, ohne dass zusätzlich aufwendige Schnittstellen zu Compilern oder Interpretern der jeweiligen Programmiersprache entwickelt werden müssen (Microsoft, 2021d).

Der MARS-Explorer verwendet Omnisharp-Roslyn, indem vom LSP definierte Nachrichten an das System gesendet oder vom System empfangen werden. Beispielsweise sendet der MARS-Explorer das bearbeitete Modell an Omnisharp-Roslyn, woraufhin mit Validierungsergebnissen geantwortet wird, die dem Modellierenden dann in Form von Text und farblichen Markierungen angezeigt werden. Das System wird bspw. auch dazu verwendet, um dem Modellierende Definitionen von Klassen oder Variablen anzuzeigen oder gewisse Aktionen für die Zeile anzubieten, über die sich der Modellierende aktuell mit dem Mauszeiger befindet. Weitere Details zur Kommunikation mittels des LSP werden in Kapitel 5.2.3 erläutert.

5.2.2 Bausteinsicht

Die Bausteinsicht wird dazu verwendet, um die Struktur des Quellcodes auf Komponenten abzubilden (Starke, 2015, S. 163-164). Diese Sicht besteht i.d.R. aus drei Ebenen, in denen das Abstraktionsniveau steig verringert wird, indem die Komponenten immer weiter konkretisiert werden.

Tabelle 5.1: Legende zu den Elementen, die in den folgenden Diagrammen zur Bausteinsicht vorkommen

Element	Beschreibung
	Blackbox
	Whitebox
	Modul
	Schnittstelle außerhalb der Whitebox
	Externes System als Blackbox
	Benutzer des Systems
	Entität innerhalb des Systems
	Abhängigkeit
	Assoziation

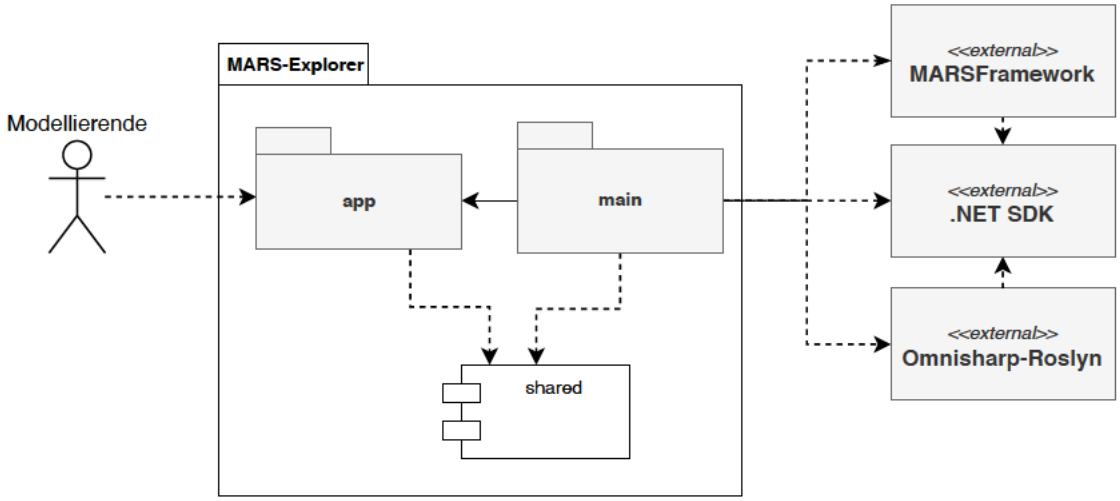


Abbildung 5.3: Whitebox-Darstellung des MARS-Explorer, Level-1

Die Tabelle 5.1 zeigt eine Legende zu den Elementen, die in den folgenden Darstellungen verwendet werden. Von Blackbox-Elementen sind auf der aktuellen Ebene lediglich die Schnittstellen bekannt und der Inhalt wird ggf. erst in der nächsten Ebene offenbart. Whitebox-Elemente hingegen offenbare ihren internen Aufbau mithilfe von den gezeigten Elementen. Komponenten, dessen Inhalt in dem jeweiligen Abschnitt vollständig erläutert wird, werden als Module dargestellt.

Neben den dargestellten Komponenten existieren noch weitere Module und Bibliotheken, die bei der Implementierung zur Unterstützung herangezogen worden sind. Für die Darstellung dieser Sicht wurde sich jedoch lediglich auf die ausschlaggebenden Abhängigkeiten konzentriert, um die Anzahl der Elemente innerhalb der Diagramme gering und die Darstellung somit übersichtlich zu halten.

Level 1

Das erste Level der Bausteinsicht ist in Abbildung 5.3 dargestellt und entspricht, bis auf ein paar Erweiterungen, den Komponenten der Kontextsicht des vorangegangenen Unterkapitels.

Der MARS-Explorer wird im Gegensatz zur Kontextabgrenzung konkretisiert und zeigt die Aufteilung des Systems in zwei Subsysteme: *main* und *app*. Diese Aufteilung entspricht dem *Multi-Prozess Model* von Chromium, welches einen Hauptprozess (hier:

main) besitzt, der wiederum einen oder mehrere Unterprozesse verwaltet (hier: *app*) (OpenJS Foundation, 2021d). Die Komponente *app* steht für den Prozess, der dem Modellierenden visuell angezeigt wird, während *main* als Hauptprozess, das Fenster der *app*, sowie den Zugriff auf die Betriebssystem-spezifischen Schnittstellen verwaltet.

Teil des MARS-Explorer ist außerdem die Komponente *shared*. Diese Komponente wird lediglich von den anderen beiden Komponenten des MARS-Explorer verwendet, um Modelle und Typen zu teilen. Dabei ist *shared* nicht als eigenständiger Prozess, sondern viel mehr als eine Sammlung an geteilten Ressourcen zu sehen.

Als externes Systeme besteht das MARS-Framework, welches als Abhängigkeit zu den Modellen hinzugefügt wird. Das Framework bietet dem Modellierenden verschiedene Schnittstellen innerhalb des Quellcodes. Außerdem stellt es die Ausführungsumgebung, mit der die geschriebenen Modelle simuliert werden können, zur Verfügung. Wie bereits im vorherigen Abschnitt erwähnt, sorgt Omnisharp-Roslyn für die Validierung der Syntax des Quellcodes und für weitere optionale Features, die für die Modellierung von Nutzen sind. Das .NET SDK liefert den wichtigsten externen Baustein, der die vorangegangenen Features erst ermöglicht.

Im Gegensatz zu den zuvor genannten externen Systemen, muss das .NET SDK zwingend auf dem System des Modellierenden installiert sein. Die anderen Systeme werden vom MARS-Explorer mit ausgeliefert.

Level 2 - Main

In Abbildung 5.4 ist die Whitebox des Bausteins *main* abgebildet. Innerhalb dieses Bausteins ist das Modul *index* der Ausgangspunkt, der zum einen *main* und zum anderen *IPCMainHandler* initialisiert.

Die Aufgabe des *Main* Moduls ist die Verwaltung des Fensters (hier: *app*), welches letztendlich dem Modellierenden angezeigt und mit welchem interagiert wird. Außerdem werden neben der Fenster-Verwaltung ebenso weitere Initialisierungsschritte ausgeführt, die den korrekten Ablauf der Anwendung sicherstellen, wie bspw. das Erstellen benötigter Verzeichnisse auf dem System des Modellierenden.

Innerhalb der verwendeten Frameworks existiert zur Kommunikation zwischen den Prozessen ein Kanal für den Hauptprozess (`ipcMain`), sowie ein Kanal für alle Unterprozesse (`ipcRenderer`), welche vom jeweiligen Prozess sowohl zum Senden als auch zum

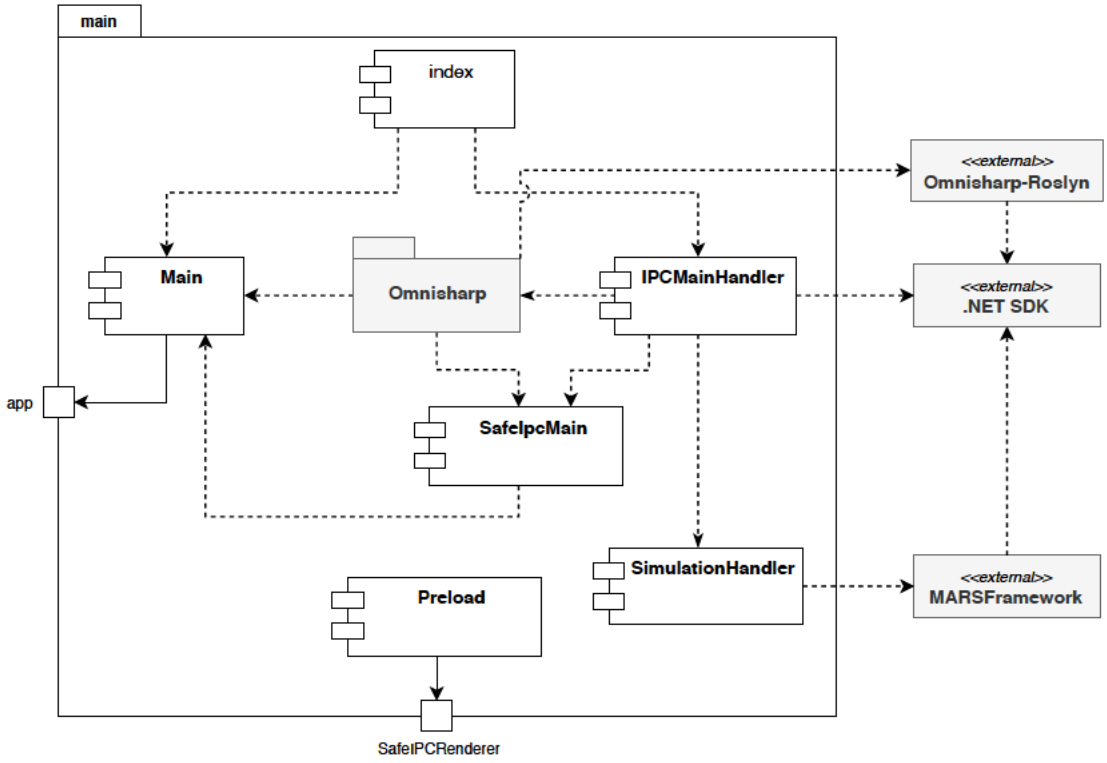


Abbildung 5.4: Whitebox-Darstellung von main, Level-2

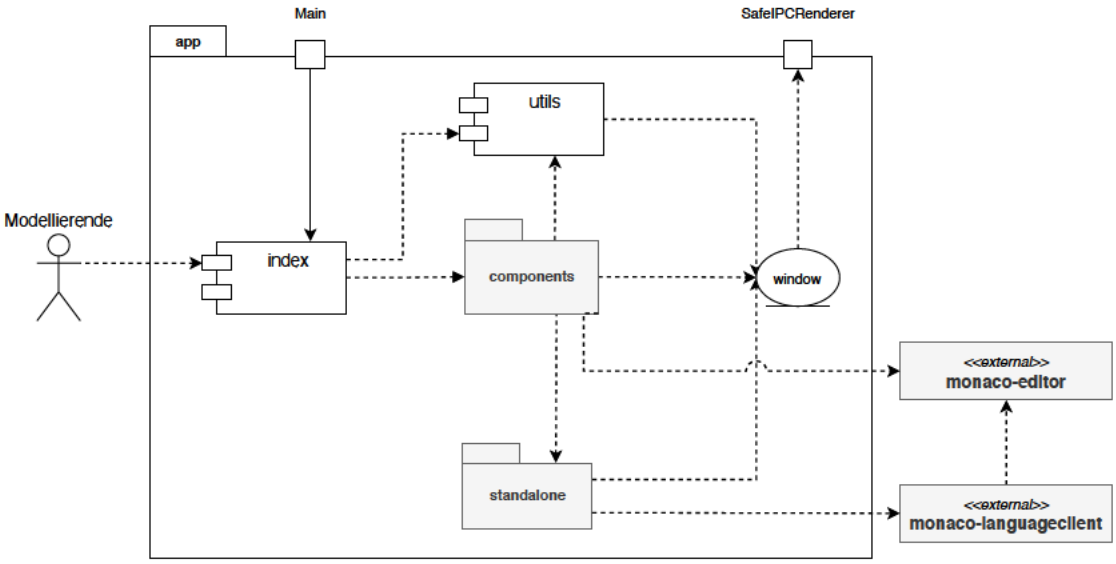


Abbildung 5.5: Whitebox-Darstellung von app, Level-2

Empfangen von Nachrichten verwendet werden kann. Das Modul *SafeIpcMain* stellt die selben Funktionen, wie der `ipcMain` zur Verfügung und ergänzt fehlende Typeninformationen, die im Framework selber nicht enthalten sind. Ähnlich funktioniert auch das *Preload*-Modul, welches im *SafeIpcRenderer* den `ipcRenderer` um Typeninformationen erweitert. Die Aufgabe des *Preload*-Moduls ist es zusätzlich dem darstellenden Prozess eine Schnittstelle zum *SafeIpcRenderer* zur Verfügung zu stellen. Diese umständlich erscheinende Maßnahme am `ipcRenderer` bieten neben den Typeninformationen weitere sicherheitsrelevante Vorteile, die im Kapitel 5.3 erläutert werden.

Level 2 - App

Die Whitebox-Darstellung der Komponente *App* ist in Abbildung 5.5 dargestellt. Wie auch zuvor, ist *index* hier der Einstiegspunkt von *app*. Dieses Modul wird vom Main-Prozess initialisiert und daraufhin dem Modellierenden angezeigt, woraufhin dieser mit der Anwendung interagieren kann. Auf der darunterliegenden Ebene befindet sich u.a. das Modul *utils*, welches neben einer Sammlung von Funktionen, die von allen Modulen verwendet werden kann, auch die Initialisierung eines globalen Zustandsspeichers beinhaltet. Dieser Speicher wird verwendet, um Daten über Module hinweg zu teilen.

Auf der gleichen Ebene befindet sich die Blackbox *components*, welche die darzustellenden Elemente beinhaltet. Dieses umfangreiche Modul verwendet sowohl das *utils* Modul als auch das *standalone* Modul, welches wiederum eine Sammlung von Modulen beinhaltet, die aufgrund ihrer wenigen Abhängigkeiten in Zukunft ggf. dazu geeignet sind, sie später als eigene Pakete zu veröffentlichen und als Abhängigkeiten innerhalb der Anwendung zu beziehen. Eine weitere Abhängigkeit des *components* Moduls besteht zum externen *monaco-editor*.

Der *monaco-editor* ist ein Open-Source Quellcode-Editor, welcher von Microsoft entwickelt worden ist und als Basis für den bekannten Visual Studio Code Editor verwendet wird (Microsoft, 2021a). Dieser kommt zum Einsatz, um die MARS-Modelle zu bearbeiten. Wie bereits erläutert, definiert das LSP eine Kommunikation zwischen einem Languageserver (hier: *Omnisharp-Roslyn*) und einem Klienten (hier: *monaco-editor*), um dem Benutzer des Klienten diverse Funktionalitäten, wie die Anzeige von Syntax-Fehlern, oder Autokorrekturen, bieten zu können. Um die Kommunikation auf Seite des Klienten zu implementieren und zu verwalten, wird die externe Bibliothek *monaco-languageclient* verwendet (TypeFox GmbH, 2021).

Alle Komponenten, die auf eine Kommunikation zum Prozess *main* angewiesen sind, verwenden das Objekt *window*, welches eine global zugängliche Schnittstelle zum *SafeIpcRenderer* bietet.

Level 3 - Omnisharp

Abbildung 5.6 zeigt die Whitebox-Darstellung der Komponente *Omnisharp*. Der Einstiegspunkt ist der *server-launcher*, welcher angesprochen wird, sobald die entsprechende Nachricht im *IPCMainHandler* empfangen wurde. Die Startkonfiguration für den Start des Languageservers von *Omnisharp-Roslyn* befindet sich in der *server-config*.

Mithilfe der Module *LspWriter* und *LspReader* wird die Kommunikation von und zum Languageserver geregelt. Während der *LSPReader* für das Schreiben von Nachrichten vom Languageserver zum Klienten über den IPC Kanal verantwortlich ist, ist es die Aufgabe des *LspWriter*, Nachrichten vom Klienten zu empfangen und an den Languageserver weiterzuleiten.

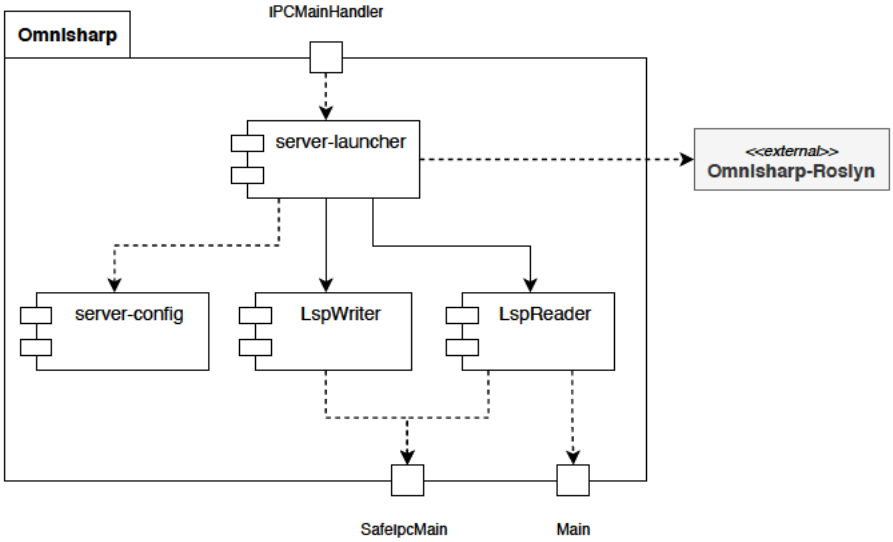


Abbildung 5.6: Whitebox-Darstellung von Omnisharp, Level-3

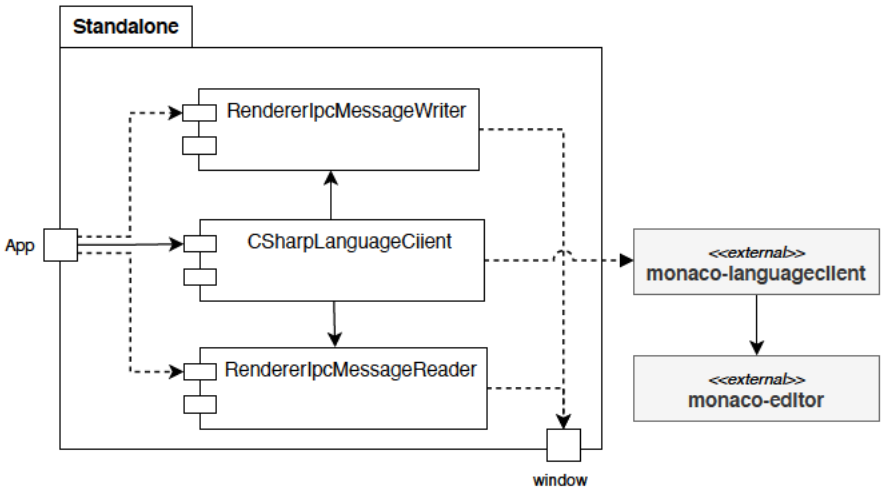


Abbildung 5.7: Whitebox-Darstellung von Standalone, Level-3

Level 3 - Standalone

Das Paket *Standalone*, dessen Whitebox in Abbildung 5.7 dargestellt ist, stellt das Gegenstück zum *Omnisharp* Paket dar und regelt die LSP Kommunikation auf Klientenseite. Die *app* sorgt für die Initialisierung von allen drei Komponenten innerhalb des Pakets.

Während der *RendererIpcMessageReader* die Nachrichten des Klienten entgegennimmt und über das *window* Objekt an den Languageserver sendet, empfängt der *RendererIpcMessageWriter* die Nachrichten des Servers und leitet sie an den *CSharpLanguageClient* weiter. Der *CSharpLanguageClient*, welcher eine Implementierung des *monaco-languageclient* ist, verarbeitet diese Nachrichten und sorgt für entsprechende Aktionen innerhalb des *monaco-editors*.

Level 3 - Components

Die Darstellung der Whitebox zum Modul *components* ist in Abbildung 5.8 zu sehen. Wie bereits erwähnt, befinden sich innerhalb von *components* alle Komponenten, die dem Benutzer angezeigt werden. Einstiegspunkt ist hier die Komponente *App*, welche von *index* initialisiert wird. Die Aufgabe von *App* ist vergleichbar mit dem *Main*-Modul aus Abbildung 5.4. Dem Modul obliegt die Verwaltung der in der Mitte der Abbildung dargestellten Module (*Home*, *Model*, *Configure*, etc.), indem es die Navigation der gesamten Anwendung implementiert und somit bestimmt, welches Modul zu welcher Zeit aktiv ist. Außerdem sorgt sie im Hintergrund für die korrekte Initialisierung von diversen Funktionalitäten, wie bspw. des Languageservers oder der Überprüfung der Installation des .NET-SDK.

In der Mitte der Abbildung befindet sich eine Whitebox zum internen Aufbau der verschiedenen Unterkomponenten, die je nach ihren abgebildeten Features benannt sind. Der Aufbau ist dabei für alle Unterkomponenten derselbe. Einstiegspunkt ist ein weiteres Mal der *index*, der die eigentliche Komponente initialisiert. Die Komponente hat wiederum Zugriff auf untergeordnete Komponenten, die nur innerhalb des eigenen Moduls eingesetzt werden, sowie auf Funktionalitäten aus dem *utils* und *hooks* Modul. Im Kontext der verwendeten Technologie, sind Hooks ein Konzept, wodurch das Schreiben von funktionalen Komponenten vereinfacht werden soll (Facebook Inc., 2021a). Während

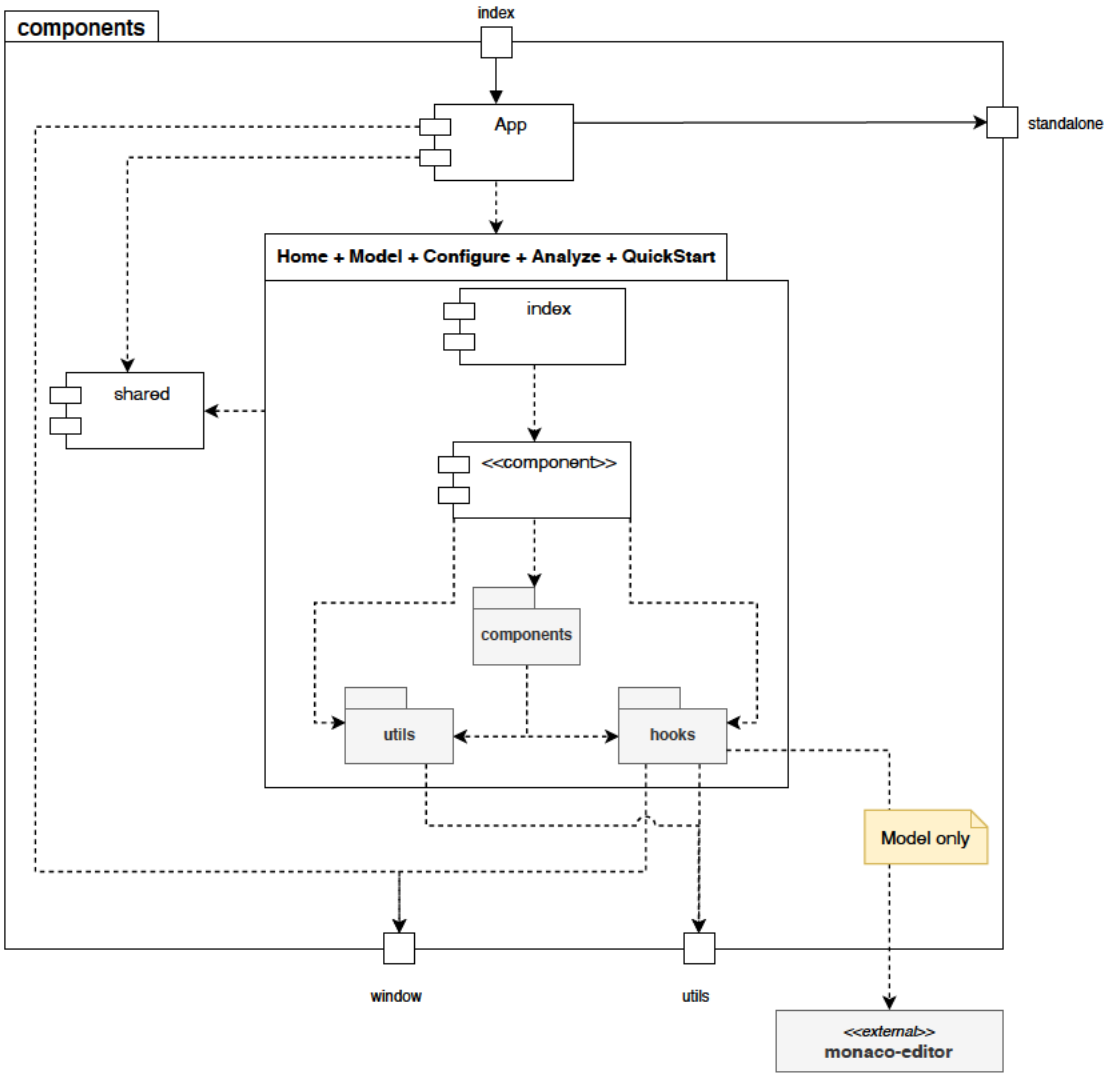


Abbildung 5.8: Whitebox-Darstellung von Components, Level-3

die Komponenten im MVP Muster die Rolle der View einnehmen, werden Hooks als Presenter eingesetzt, welche die Kontrolle über die dargestellte Komponente und dessen Interaktionen hat.

Ähnlich zu dem Modul *shared* aus Level 1, handelt es sich bei *shared* auf dieser Ebene um eine Sammlung von Komponenten, die von allen Modulen verwendet wird.

5.2.3 Laufzeitsicht

Die Laufzeitsicht zeigt das Zusammenspiel der zuvor dargestellten Komponenten während der Ausführung der relevantesten Funktionen (Starke, 2015, S. 170). Als Darstellungsform wurde das Sequenzdiagramm des UML-Standards ausgewählt. Jeder Schritt innerhalb jedes Diagramms ist mit einer Nummer gekennzeichnet, welche innerhalb des Textes verwendet wird, um Bezug auf die jeweilige Aktion zu nehmen.

Da der Fokus dieser Sicht auf dem Zusammenspiel der Komponenten liegt, wurden Details wie die Fehlerbehandlung oder das *Logging* bewusst nicht dargestellt. Auch wurden zur Vereinfachung triviale Module wie *shared* oder *utils* ausgelassen, da diese lediglich eine Sammlung von geteilten Funktionalitäten sind und keine gesonderte Rolle im Verhalten der Anwendung übernehmen. Außerdem werden alle darstellenden Komponenten des *app*-Prozesses innerhalb der Laufzeitsicht zu einer Komponente *App* zusammengefasst, da die Komponenten zum Teil kleinteilig implementiert sind und aus diesem Grund für Darstellungsprobleme sorgen würden.

Laufzeitsicht - Anwendung starten

Die in Abbildung 5.9 dargestellte Laufzeitsicht erläutert den Start der Anwendung.

(1) Der Benutzer startet die Anwendung, indem die Betriebssystem-abhängige ausführende Datei ausgeführt wird.

(2–4) Das verwendete Framework startet den *main*-Prozess, führt parallel die Preload-Funktion aus und erzeugt die *SafeIpcMain* Instanz.

(5–12) Die *Main*-Klasse wird instanziiert und verschiedene Funktionen ausgeführt, die zur Initialisierung der Anwendung notwendig sind. Im Anschluss wird das Fenster der Anwendung geöffnet. Gleichzeitig werden innerhalb des *IpcMainHandler* für diverse

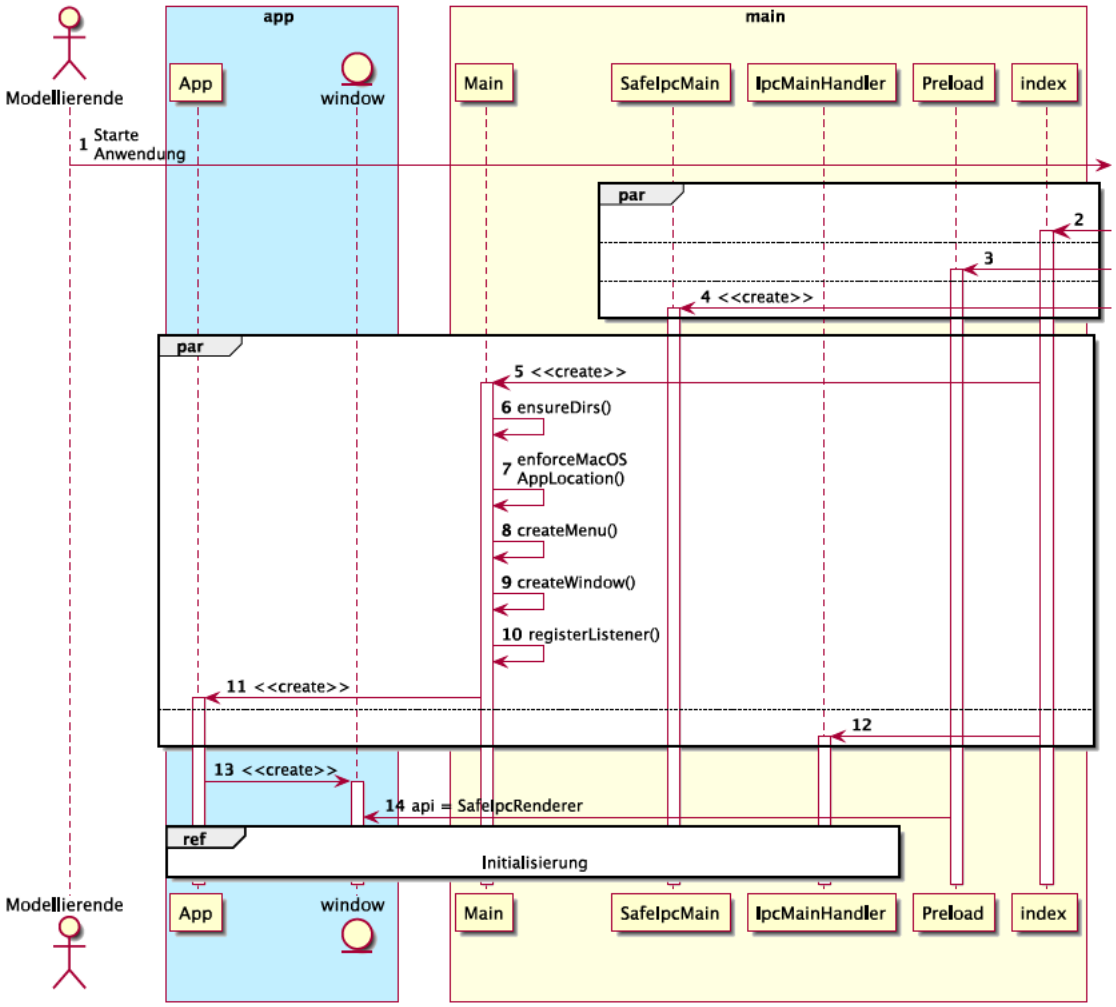


Abbildung 5.9: Laufzeitsicht des Vorgangs *Anwendung starten*

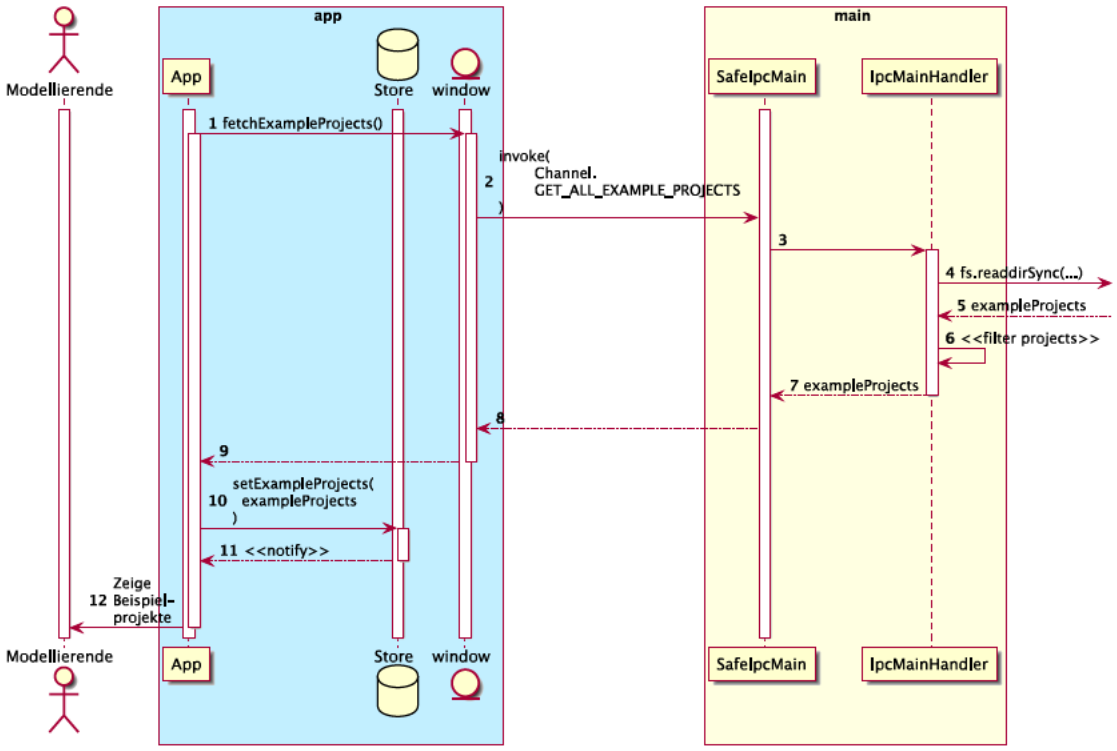


Abbildung 5.10: Laufzeitsicht des Vorgangs *Kommunikation zwischen den Prozessen*

Kanäle Funktionen registriert, die ausgeführt werden, sobald eine Nachricht über diesen empfangen wird.

(13+14) Das Objekt *window* wurde erstellt und die Preload-Funktion stellt die Schnittstelle des *SafeIpcRenderer* über dieses Objekt zur Verfügung.

Im Anschluss folgen weitere Initialisierungsschritte innerhalb des *app*-Prozesses, die aus Gründen der Übersichtlichkeit in den folgenden Abschnitten ausführlicher erläutert werden.

Laufzeitsicht - Kommunikation zwischen den Prozessen

Eine der Aufgaben, die die Anwendung nach dem Start ausführt, ist u.a. das Darstellen von Beispielprojekten, welche mit der Anwendung ausgeliefert worden sind. Diese beispielhafte Aufgabe soll das Zusammenspiel zwischen den beiden Prozessen *main* und *app*

verdeutlichen. Da die IPC Kommunikation innerhalb dieses Ablaufs ausführlich vorgestellt wird, wird sie in den folgenden Diagrammen aufgrund der Übersichtlichkeit impliziert.

(1–3) Um die Beispielprojekte darzustellen, ruft die Anwendung die *fetchExampleProjects()* Funktion auf, welche mithilfe des *window*-Objektes eine Anfrage über den Kanal *GET_ALL_EXAMPLE_PROJECTS* an den *main*-Prozess sendet.

(4–9) Diese Anfrage wird von einer auf den Channel registrierten Funktion innerhalb des *IpcMainHandler* empfangen. Daraufhin wird mithilfe des nativen Moduls *fs* von NodeJS das Verzeichnis mit den Beispielprojekten innerhalb des Ressourcen-Ordners des MARS-Explorer durchsucht und gefiltert. Die gefilterten *exampleProjects* werden daraufhin über den selben Kanal an den *App*-Prozess zurückgesendet.

(10–12) Die Beispielprojekte werden im globalen Speicher der Applikation hinterlegt (*setExampleProjects*), wodurch die für die Darstellung der Projekte verantwortlichen Komponenten über die neuen Daten benachrichtigt und dem Modellierenden schlussendlich angezeigt werden.

Laufzeitsicht - Initialisierung des Languageservers

Die Funktionen des Languageservers gehören zu den Kernfunktionen des MARS-Explorer, weshalb die korrekte Initialisierung eine große Rolle spielt und mithilfe des in Abbildung 5.11 dargestellten Ablaufs erläutert werden soll. Um die Übersichtlichkeit zu gewährleisten, wurden die in der Bausteinsicht erwähnten Komponenten Writer und Reader sowohl auf Seiten des Languageserver als auch auf Seiten des Klienten entfernt.

(1–2) Nachdem der Modellierende zur Modellierung navigiert ist, bekommt dieser Indikatoren dafür angezeigt, dass der Languageserver gestartet und initialisiert wird.

(3–8) Die App sendet über den jeweiligen Kanal eine Anfrage an den *ServerLauncher*, welcher daraufhin mithilfe des nativen NodeJS Moduls *ChildProcess* die ausführbare Datei von Omnisharp mit Referenz zu dem übergebenen Pfad (*projectUri*) startet. Der jeweilige Befehl unterscheidet sich zwischen Unix- und Windows-Systemen. Der daraufhin gestartete Prozess wird dazu verwendet, um auf Basis der Standard-Kanäle (*stdin* und *stdout*) die Reader und Writer zu erstellen. Außerdem wird ein dynamischer Kanal erzeugt, der auf dem Process identifier (PID) basiert und über den der Klient und der Server miteinander kommunizieren werden. Der Name des

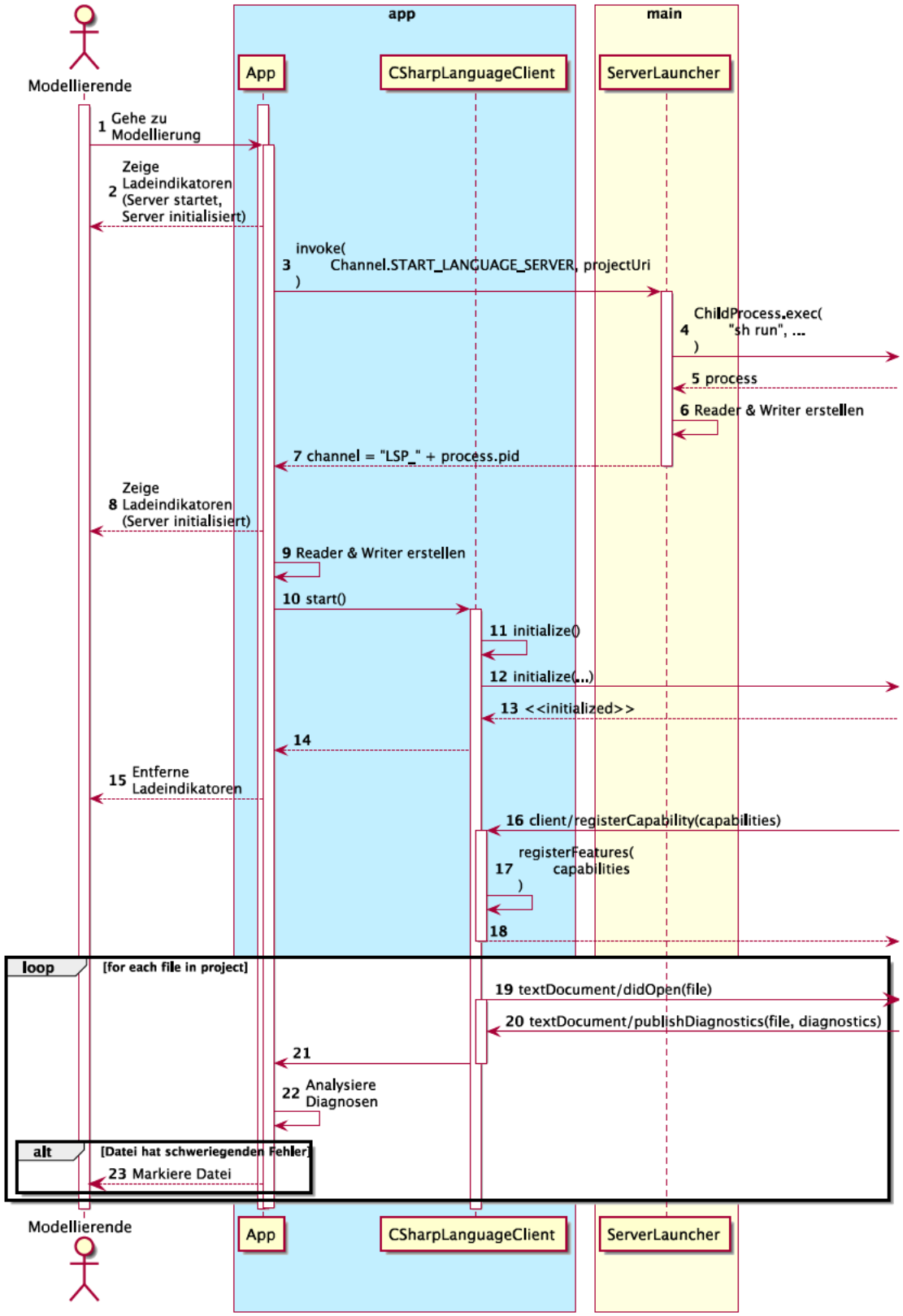


Abbildung 5.11: Laufzeitsicht des Vorgangs *Initialisierung des Languageservers*

erstellten Kanals wird an die App gesendet, um ihn dem Klienten bekannt zu machen. Dadurch ist das Starten des Servers abgeschlossen und es wird lediglich noch der Indikator der Initialisierung angezeigt.

(9–15) Reader und Writer für den besagten Kanal werden ebenso erstellt und dem Klienten übergeben. Daraufhin wird die Methode *start* des Klienten aufgerufen, welche zur eigenen Initialisierung führt und ebenfalls eine Aufforderung zur Initialisierung an den Languageserver sendet. Eine solche Initialisierunganfrage beinhaltet Informationen über den Klienten, wie bspw. Angaben über die Funktionen die von ihm unterstützt werden oder über die Sprache des Systems. Nach ein paar Sekunden meldet der Languageserver die erfolgreiche Initialisierung an den Klienten und die Ladeindikatoren werden komplett entfernt.

(16–18) Des Weiteren meldet der Languageserver, welche Funktionalitäten von ihm unterstützt werden. Diese werden wiederum vom Klienten registriert, sodass sowohl Klient als auch Server die Funktionen des jeweils anderen kennen.

(19–20) Um nun den Status des aktuellen Modells zu validieren, wird für jede Klasse innerhalb des Projekts eine *textDocument/didOpen* Benachrichtigung an den Languageserver gesendet, was wiederum dazu führt, dass der Languageserver die Klasse validiert und das Ergebnis dieser Validierung über eine *textDocument/publishDiagnostics* Benachrichtigung an den Klienten sendet. Diese Benachrichtigung beinhaltet zum einen die Klasse und zum anderen eine Menge von Diagnosen, welche jeweils eine betroffene Stelle im Code, eine dazugehörige Nachricht, sowie einen Schweregrad der jeweiligen Diagnose angibt. Der Grad der Schwere wird in Fehler, Warnung, Information und Hinweis unterschieden. Diese Diagnosen werden dann vom Klienten verarbeitet und im Quelltext-Editor markiert.

(21-23) Die Diagnosen werden allerdings auch von der App selbst nochmal überprüft, da die funktionalen Anforderungen fordern, dass zu jeder Zeit eingesehen werden können soll, ob das aktuelle Modell valide ist oder nicht (FA 14: Status anzeigen). Dazu wird also überprüft, ob sich innerhalb der erhaltenen Diagnosen eine vom Schweregrad Fehler befindet. Falls dies der Fall ist, wird dem Modellierenden signalisiert, dass eine Klasse bzw. eine Datei einen Fehler beinhaltet.

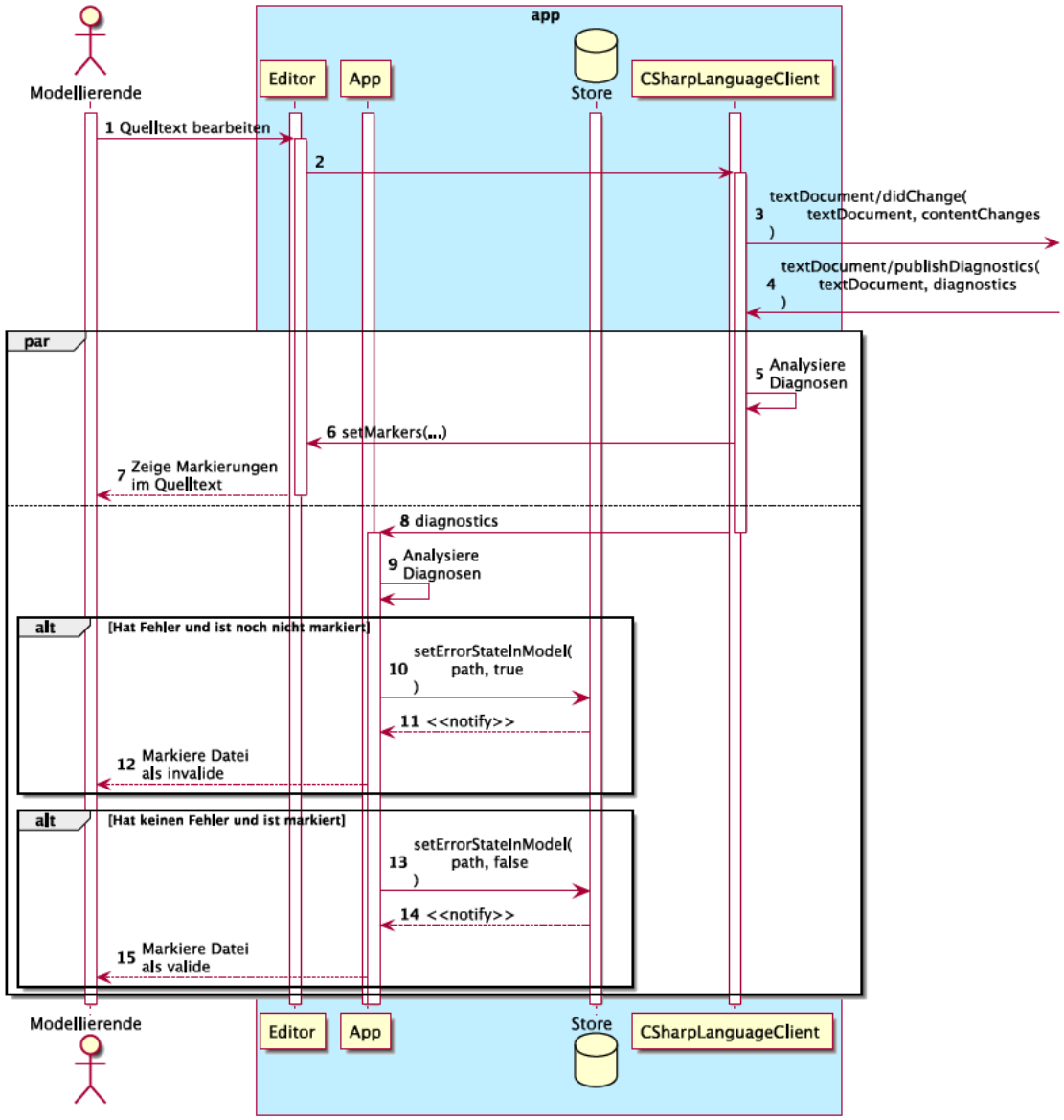


Abbildung 5.12: Laufzeitsicht des Vorgangs *Quelltext bearbeiten*

Laufzeitsicht - Quelltext bearbeiten

Innerhalb dieser Sicht soll anhand von Abbildung 5.12 erläutert werden, wie das Zusammenspiel zwischen dem Editor und dem Languageclient bzw. Languageserver während Bearbeitung des Quelltextes funktioniert.

(1–4) Der Ablauf startet, indem der Modellierende Änderungen am Modell durch die Editor-Komponente vornimmt. Diese Änderungen werden vom *CSharpLanguageClient* verarbeitet und eine *textDocument/didChange* Benachrichtigung an den Languageserver gesendet. Diese Benachrichtigung hält die Informationen darüber, welche Datei bearbeitet worden ist und welche Änderungen diese Bearbeitung enthält. Der Server sendet nach der internen Analyse die zuvor beschriebene *textDocument/publishDiagnostics* Benachrichtigung an den Klienten.

(5–7) Daraufhin analysiert der Klient die empfangenen Diagnosen und markiert die entsprechenden Zeilen innerhalb des Editors. Diese werden dem Modellierenden folgend entsprechend anzeigt.

(8–12) Parallel dazu werden die Diagnosen von der App-Komponente analysiert, um die zuvor beschriebene Validierung durchzuführen. Enthalten die Diagnosen schwerwiegende Fehler, wird die Datei als fehlerhaft im globalen Speicher hinterlegt, was dazu führt, dass die entsprechenden Komponenten auf diesen Zustand reagieren und dem Benutzer dementsprechend den Fehler signalisieren.

(13–15) Andersherum kann eine zuvor fehlerhafte Datei keine Fehler in der Diagnose beinhalten, was dazu führt, dass der valide Zustand im globalen Speicher festgehalten und von anderen Komponenten dargestellt wird.

Laufzeitsicht - Simulationsablauf

Als letzte Laufzeitsicht wird der Ablauf einer Simulation mit Abbildung 5.13 erläutert. Dabei gilt zu beachten, dass aus Platzgründen das Speichern der Simulationsergebnisse nicht abgebildet ist. Am Ende eines Simulationslaufes werden diese im globalen Speicher und im sog. *LocalStorage* hinterlegt, um eine Analyse der Daten auch nach dem Neustart der Anwendung zu ermöglichen. Außerdem wird zur Bewahrung der Übersichtlichkeit lediglich die Handhabung der Count-Daten berücksichtigt, welche die Anzahl der Agenten pro Simulationsfortschritt angibt. Die Verwaltung der Positionsdaten funktioniert auf

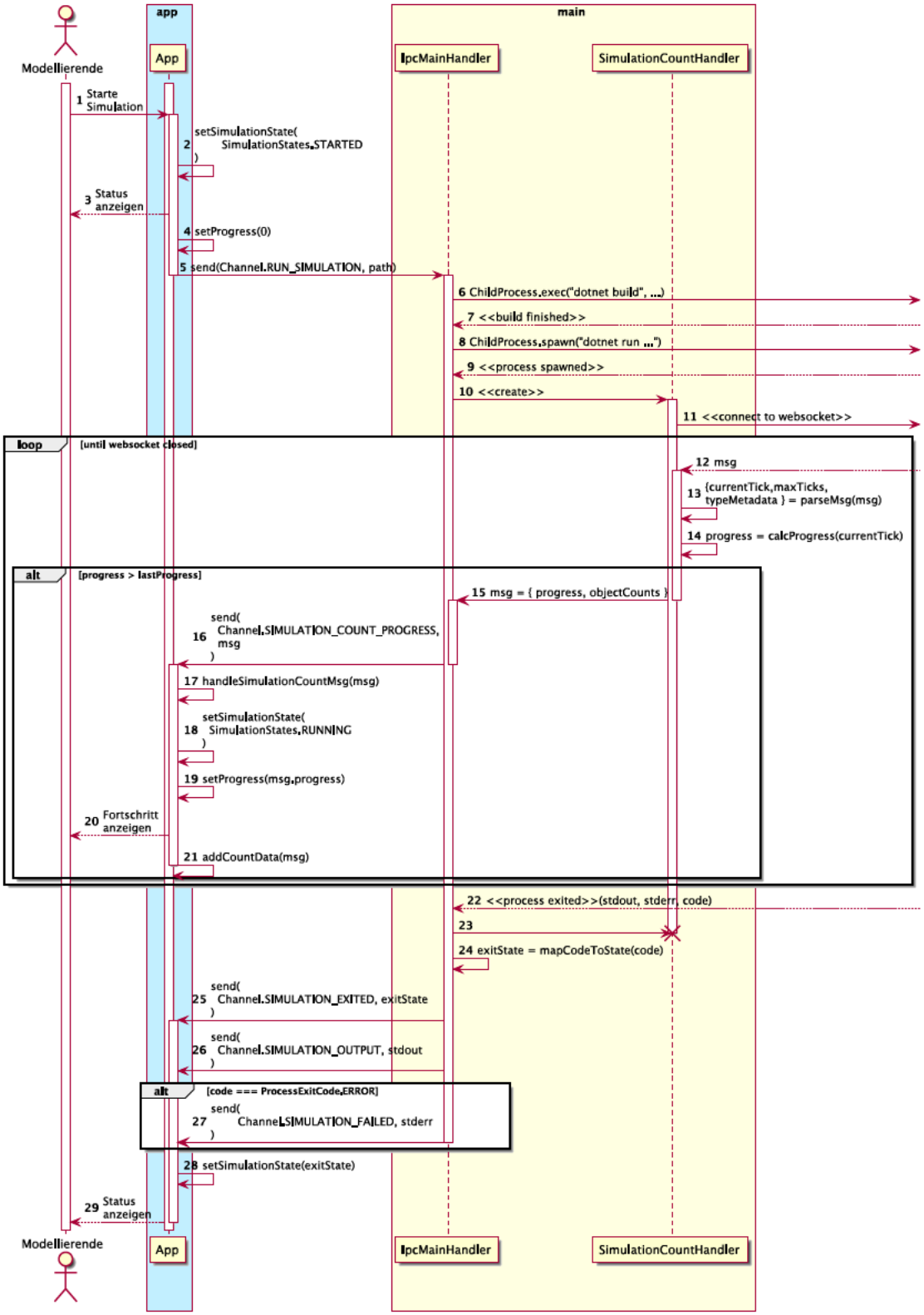


Abbildung 5.13: Laufzeitsicht des Vorgangs *Simulationsablauf*

gleiche Art und Weise, abgesehen davon, dass zusätzlich zum *SimulationCountHandler* eine weitere Klasse *SimulationVisHandler* verwendet wird, um die Daten zu aggregieren.

(1–4) Mit einem Klick auf den Start-Button wird die Simulation angestoßen. Dadurch wird im ersten Schritt der Zustand zu *Started* geändert und dem Modellierenden signalisiert, dass die Simulation gestartet wird. Außerdem wird der vorherige *Progress* zurückgesetzt.

(5–11) Die App sendet unter Angabe des Pfades zum aktuellen Projekt über den *RUN_SIMULATION* Kanal eine Nachricht an den *main*-Prozess. Die registrierte Funktion im *IpcMainHandler* reagiert auf diese Nachricht und stößt im ersten Schritt das Bauen der Anwendung an. Dies geschieht mithilfe des installierten .NET SDK und dem Befehl *dotnet build*. Ist das Bauen abgeschlossen, wird der Befehl *dotnet run –no-build* verwendet, um den Simulations-Prozess zu starten. Nachdem der Prozess gestartet wurde, wird eine Instanz der Klasse *SimulationCountHandler* erstellt, welche dafür verantwortlich ist, die Daten, welche die Anzahl der Agenten pro Simulationsfortschritt betreffen, zu verarbeiten. Nach der Instanziierung wird kontinuierlich versucht, sich mit dem Websocket zu verbinden, der vom Simulationsprozess geöffnet wird.

(12–14) Nachdem die Verbindung zur Simulation zustande gekommen ist, werden bis zum Ende der Simulation Nachrichten vom Simulationsprozess empfangen, welche Daten über die Anzahl der Agenten beinhalten. Die empfangenen Nachrichten (*msg*) werden interpretiert und die relevanten Daten extrahiert. Neben der Anzahl der Agenten wird der aktuelle und der maximale Tick der Simulation übermittelt. Aus diesen Daten kann der aktuelle Simulationsfortschritt berechnet werden. Um die Anzahl der anzuzeigenden Daten möglichst gering zu halten und so die Performanz zu erhöhen, werden die Daten nur an die *App* gesendet, falls es sich um einen neuen Simulationsschritt handelt. Um den Simulationsfortschritt zu berechnen, wird die folgende Formel angewandt: $Math.floor(currentTick/maxTicks) * 100$. Diese Formel liefert immer eine natürliche Zahl zwischen 0 und 100. Das Ergebnis der Formel wird dann wiederum mit dem letzten Simulationsfortschritt verglichen. Ist der neue Fortschritt größer als der alte Fortschritt, werden die Daten zur weiteren Verarbeitung freigegeben. Dadurch werden höchstens 100 Datenobjekte an die *App* gesendet, was zu einer geringen Rechenlast und einer flüssigen Performanz führt. Im Falle der hier ausgelassenen Positionsdaten können diese Datenobjekte – je nach

Anzahl der Agenten – natürlich auch beliebig groß werden, wodurch nicht immer eine flüssige Performanz garantiert werden kann.

(15–21) Bei einem neuen Simulationsfortschritt werden die Daten an den *IpcMainHandler* weitergegeben, um diese an die *App* zu senden. Dort werden sie verarbeitet, indem der Status der Simulation zu *RUNNING* geändert wird, sofern noch nicht geschehen. Außerdem wird der *Progress* auf den erhaltenen Wert gesetzt, welcher dann wiederum dem Modellierenden angezeigt wird. Die eigentlichen Daten zur Anzahl der Agenten werden im hier nicht abgebildeten globalen Speicher hinterlegt, was wiederum von der ggf. geöffneten Graphen-Komponente erfasst und dargestellt wird. Dieser Vorgang wiederholt sich, bis die Simulation abgeschlossen ist und alle Datenpunkte übertragen worden sind.

(22-28) Nachdem der Prozess beendet worden ist, wird die zuvor erstellte Instanz des *SimulationCountHandler* zerstört. Der erhaltene Exit-Code, welcher dem jeweiligen Betriebssystem entspringt, wird auf einen internen Zustand abgebildet, welcher an die *App* übertragen wird. Ebenso wird die Ausgabe des Simulationsprozesses an die *App* übertragen, welche bspw. Konsolen-Ausgaben enthalten kann, die vom Modellierenden in das Modell eingebaut worden sind. Handelt es sich bei dem abgebildeten Exit-Code um einen Fehlercode, so wird die Fehlerausgabe des Prozesses ebenfalls an die *App* gesendet, um diese dem Benutzer darzustellen. Der empfangene Status wird schlussendlich in der *App* gesetzt und dem Modellierenden präsentiert.

5.2.4 Verteilungssicht

Abbildung 5.14 zeigt die Verteilungssicht des MARS-Explorer, welche die technische Infrastruktur der Anwendung darstellen soll (Starke, 2015, S. 173-174).

Innerhalb dieses Diagramms ist auf der linken Seite der MARS-Explorer als Software-Paket abgebildet. Dieses beinhaltet alle Pakete, die für die Auslieferung der Anwendung benötigt werden. Zum einen die Quelldateien unter *src/*, welche in die zuvor erläuterten Prozesse *main* und *app* aufgeteilt sind. Zum anderen alle Ressourcen, die innerhalb der Anwendung benötigt werden unter *Ressourcen/*. Dazu gehören die ausführbaren Dateien von Omnisharp-Roslyn, welche sich je nach Ziel-System unterscheiden. Dazu kommen noch folgende weitere Ressourcen:

assets Bilder und Icons, die von der Anwendung verwendet werden.

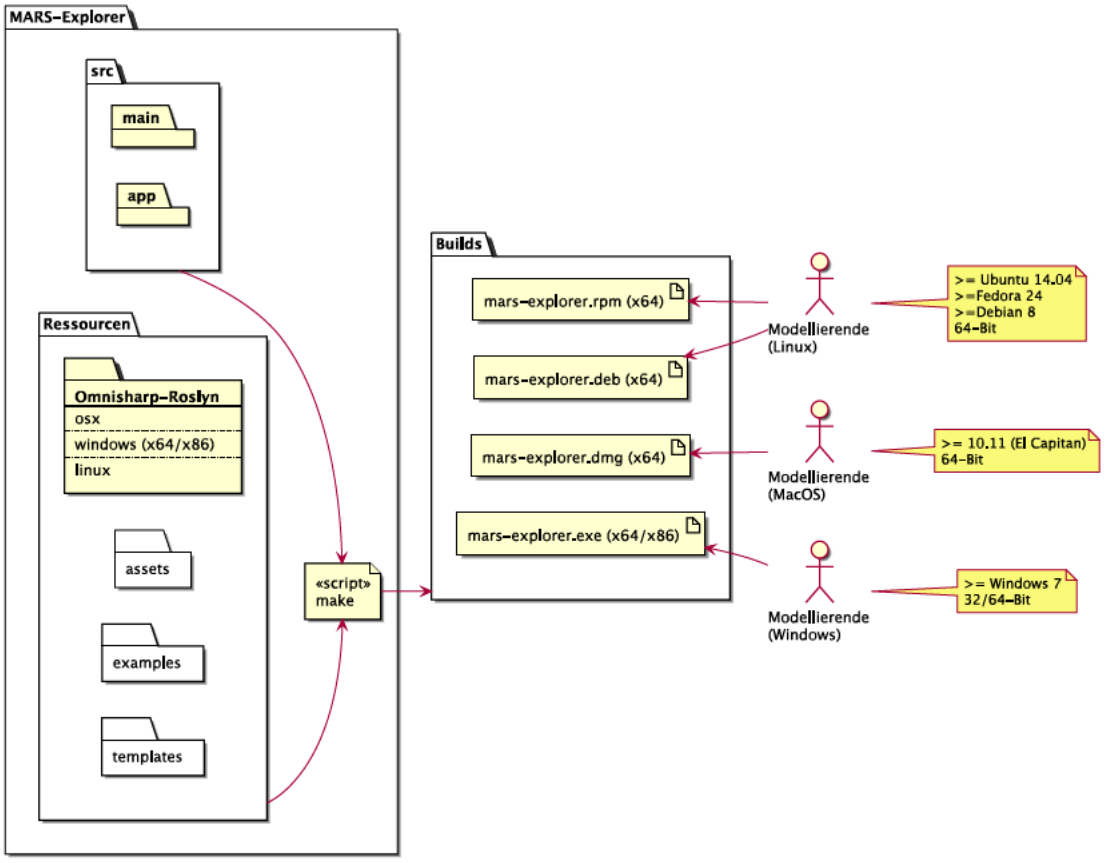


Abbildung 5.14: Verteilungssicht des MARS-Explorer

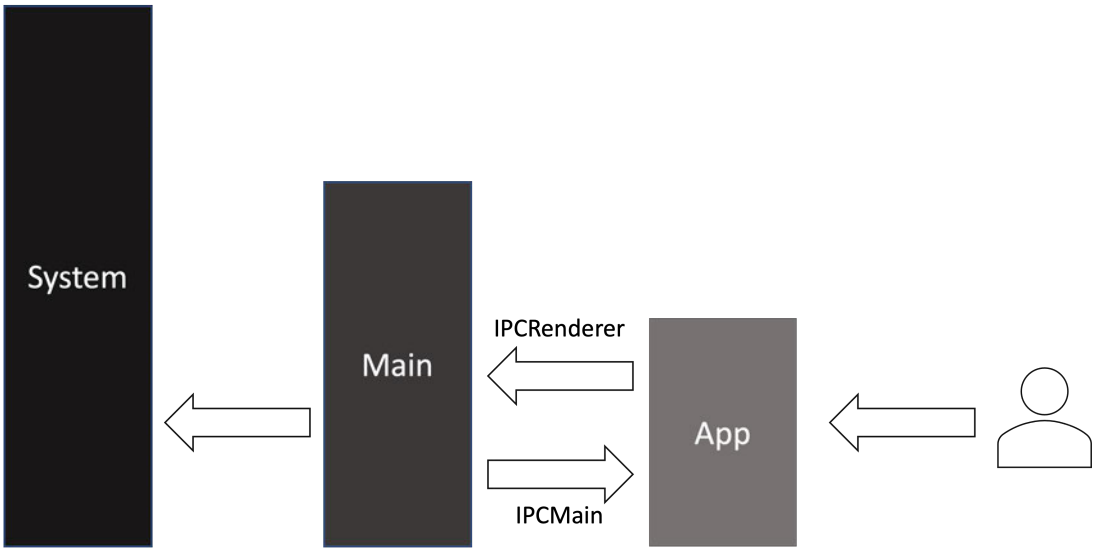


Abbildung 5.15: Standardmäßige Schnittstellen der Prozesse der verwendeten Technologie

examples Beispielprojekte, die von den Lehrenden eingesetzt werden, um sie den Modellierenden als Orientierungspunkte zur Verfügung zu stellen.

templates Vorgefertigte Dateien, die kopiert und punktuell ersetzt werden, um neue Projekte oder Objekte zu erstellen.

Mithilfe eines Skriptes werden aus den Paketen verschiedene Installationspakete (*Builds*) erstellt, welche von den Modellierenden für das jeweilige Betriebssystem verwendet werden können. In Abhängigkeit zur verwendeten Technologie sind mögliche Systeme: Windows ab Version 7, MacOS ab Version 10.11 El Capitan sowie Linux-basierte Systeme, wobei eine Installation unter Ubuntu ab Version 14.04, Fedora ab Version 24 und Debian ab Version 8 möglich ist (OpenJS Foundation, 2021b).

5.3 Gesicherte Interprozesskommunikation

Wie bereits in der Bausteinsicht erklärt, existieren innerhalb des verwendeten Frameworks zwei Prozesse, welche standardmäßig über unterschiedliche Berechtigungen verfügen. Diese Prozesse werden in Abbildung 5.15 als Schaubild gezeigt. Der Prozess *App* wird dem Benutzer als graphische Benutzeroberfläche dargestellt und bietet Möglichkeiten zur Interaktion an. Muss der Prozess mit Schnittstellen des Betriebssystems interagieren, so

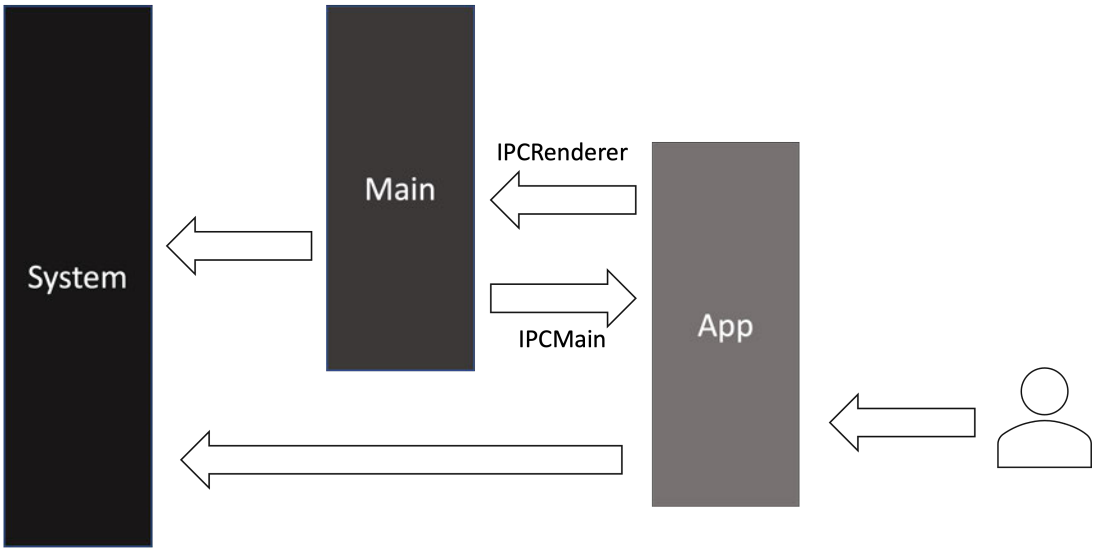


Abbildung 5.16: Unsichere Schnittstellen der Prozesse der verwendeten Technologie

muss über den *IPCRenderer*-Kanal mit dem *Main*-Prozess kommuniziert werden, welcher wiederum Zugriff auf die Funktionen des Betriebssystems hat.

Es existieren auch Anwendungsfälle bei dem der *Main*-Prozess die Interaktion über den *IPCMain* initiiert. Dies kann der Fall sein, wenn bspw. die Anwendung geschlossen wird. In dem Fall wird aufgrund der Verwaltungstätigkeit des *Main*-Prozesses das Event zuerst dort verarbeitet. Soll dem Benutzer dann ein Bildschirm angezeigt werden, der das Schließen der Anwendung signalisiert, wird über den *IPCMain* Kanal eine Meldung an den *App*-Prozess gesendet.

Mithilfe verschiedener Konfigurationsmöglichkeiten kann der *App*-Prozess allerdings auch direkt auf das System zugreifen, wie in Abbildung 5.16 dargestellt. Im ersten Moment erscheint dies sehr praktisch, da der Umweg über den *Main*-Prozess entfällt. Eine solche Konfiguration macht die Anwendung aber sehr anfällig für diverse Attacken, wie bspw. eine Cross-site-scripting (XSS) Attacke (OpenJS Foundation, 2021e). Hat ein Angreifer Zugriff auf den *App*-Prozess, so kann dieser mithilfe einer XSS Attacke einfach genutzt werden, um auf die Schnittstellen des Betriebssystems zuzugreifen und beliebige Kommandos auszuführen.

Der standardmäßige Aufbau mit dem *IPCRender* als Schnittstelle, wie er in Abbildung 5.15 zu sehen ist, kann allerdings ebenso anfällig für sog. Prototype Solution Attacks sein, welche wiederum andere Attacken nach sich ziehen können (OpenJS Foundation, 2021e).

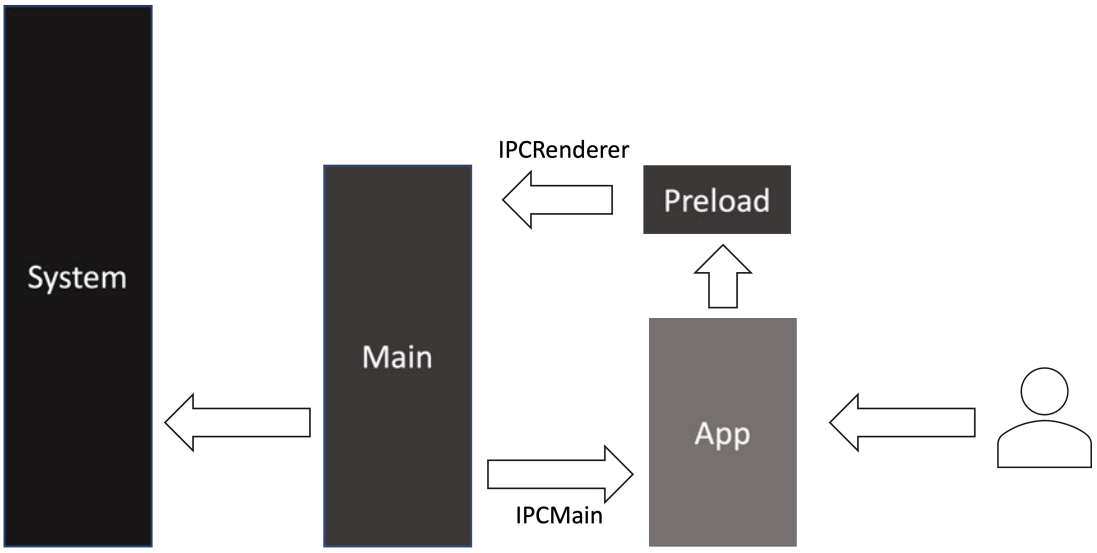


Abbildung 5.17: Gesicherte Schnittstellen der Prozesse der verwendeten Technologie mit vorgeschaltetem Proxy

Aus diesem Grund wird die *App* komplett isoliert und ein sog. *Preload* Skript eingeführt, welches die Möglichkeit besitzt, Schnittstellen explizit für den *App*-Prozess freizugeben. Um sicherzustellen, dass nach einer solchen Freigabe nur Nachrichten durchkommen, die vom Entwickler definiert worden sind, wird eine Art Proxy vor den freigegebenen *IPCRenderer*-Kanal geschaltet. Dieser Proxy kontrolliert, ob die eingehende Nachricht, die an den *Main*-Prozess gesendet werden soll, eine zugelassene Nachricht ist und leitet diese nur in diesem Fall weiter. Ein Schaubild des gesicherten Kommunikationsflusses ist in Abbildung 5.17 dargestellt.

Diese Sicherheitsvorkehrungen sind vor allem notwendig, wenn Online-Inhalte in der Anwendung bezogen werden. Dies ist beim MARS-Explorer aktuell nicht der Fall. Für die Zukunft ist es allerdings denkbar, dass innerhalb der Anwendung mit Online-Services kommuniziert wird. Mithilfe der genannten Maßnahmen wären somit die Anforderungen NFA *Wartbarkeit* und NFA *Sicherheit* gewährleistet.

6 Implementierung

Innerhalb dieses Kapitels wird auf Details der Implementierung eingegangen. Dazu gehören zum einen die verwendeten Technologien und die Gründe für diese Auswahl, zum anderen wird auf das innerhalb der Anwendung betriebene Datenmanagement thematisiert, welches essentiell für das detaillierte Verständnis der Implementierung ist. Außerdem wird im letzten Abschnitt auf Learnings eingegangen, welche bei der Implementierung gemacht worden sind.

6.1 Verwendete Technologien

In Folgenden werden die Technologien, die für die Implementierung des MARS-Explorer verwendet worden sind, kurz vorgestellt.

6.1.1 Framework zur Entwicklung von plattformübergreifenden Desktop-Anwendungen

Die Anforderung, welche die Wahl der Technologie am meisten beeinflusst, ist die NFA *Portabilität*, da kein einzelnes Betriebssystem definiert werden kann, dass von allen Anwendern benutzt wird. Außerdem handelt es sich bei dem MARS-Explorer, um eine erste Iteration hin zu einer langlebigen didaktischen Simulationsplattform, weshalb vor allem auch die Langlebigkeit eine übergeordnete Rolle spielt. Gleichzeitig sollte die Anwendung aber auch ansprechend für den Benutzer sein, was eine gewisse Modernität der Anwendung voraussetzt.

In der Theorie wäre es möglich, jeweils eine Anwendung für die drei spezifizierten Betriebssysteme zu entwickeln. Dies würde jedoch hohe Wartungskosten mit sich ziehen. Durch die Wahl einer sog. plattformunabhängigen Technologie kann dieser Aufwand im

Gegensatz zu drei individuellen Anwendungen stark reduziert werden. Für die Entwicklung wurden die folgenden Lösungen untereinander abgewägt:

Electron Ein Framework, was auf NodeJS basiert und durch den integrierten Webbrowser Chromium eine Web- als Desktop-Anwendung erscheinen lassen kann. Mithilfe der bekannten Web-Technologien JavaScript (JS), Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS), sowie von NodeJS als Schnittstelle zu den nativen Schnittstellen des jeweiligen Betriebssystems, können somit plattformunabhängige Anwendungen entwickelt werden (OpenJS Foundation, 2021a).

Electron.NET Eine Technologie, die sowohl auf dem zuvor erwähnten Electron-Framework als auch auf .NET bzw. einer ASP.NET Core Anwendung basiert (Biswanger und Muehsig, 2021). Eine solche Anwendung wird mithilfe von C#, JS und HTML geschrieben.

Avalonia Ein .NET Framework, welches die Elemente der Oberfläche mit XAML, einer von Microsoft eingeführten Sprache, definiert (AvaloniaUI OÜ, 2021).

NodeGUI Mithilfe des plattformunabhängigen QT Framework und einer modifizierten NodeJS Runtime können Desktop-Anwendungen für die gängigen Betriebssysteme entwickelt werden.

Aufgrund der langjährigen Erfahrungen des Autors mit Web-Technologien, sowie der allgemeinen Popularität und Einfachheit dieser (Stack Exchange Inc., 2021), wurde sich letzten Endes für das Electron-Framework entschieden, welches im folgenden auch kurz Electron genannt wird. Neben den Faktoren der Popularität und Einfachheit weist Electron ebenso eine lange Historie und kontinuierliche Updates mit dem ersten Release im Jahr 2013 auf, welches ursprünglich als Grundlage für den Atom-Editor geplant war (OpenJS Foundation, 2021a). Des Weiteren wird das Framework ebenfalls in der Entwicklung diverser verbreiteter Anwendungen, wie Slack oder Visual Studio Code eingesetzt (OpenJS Foundation, 2021a).

6.1.2 UI-Framework

Electron bietet die Möglichkeit, die User Interfaces (UI) mithilfe der bekannten Web-Technologien zu implementieren. Innerhalb dieses Kontextes können deshalb auch weitere Frameworks eingebunden werden, welche die Entwicklung der UI einfacher gestalten. Auch hier sind die Faktoren Popularität, Langlebigkeit, sowie die Erfahrungen des Autors

ausschlaggebend. Innerhalb dieser drei Faktoren hat sich die von Facebook entwickelte UI Bibliothek *React* herausgestellt. React ist im Jahr 2021 die beliebteste Bibliothek zur Erstellung von Benutzeroberflächen im Kontext der Web-Entwicklung (Stack Exchange Inc., 2021, Liu, 2021). Der erste Release hat ebenfalls im Jahr 2013 stattgefunden und seitdem werden kontinuierlich Updates ausgeliefert (Facebook Inc., 2021b). Die Bibliothek löst einige Probleme, die mit den nativen Technologien der Web-Entwicklung einhergehen, wie die Reaktivität, Performanz, oder die Kapselung (Facebook Inc., 2021b).

6.1.3 Programmiersprache

Wie zuvor erwähnt, sind Electron und React beides JavaScript Frameworks bzw. Bibliotheken. Da JS allerdings keine statische Typisierung mit sich bringt, ist es schwerer für Entwickler, den geschriebenen Code eines anderen nachzuvollziehen. Aus diesem Grund wurde von Microsoft die Programmiersprache *Typescript* eingeführt, welche auf JS aufbaut und somit über die gleichen Features verfügt (Microsoft, 2021c). Wie der Name bereits verrät, wird die ursprüngliche Sprache um ein statisches Typisierungssystem erweitert, was zu besser lesbarem Code und zu einer vereinfachten Fehleridentifikation innerhalb der eigenen Entwicklungsumgebung führt (Microsoft, 2021c). Somit wird die NFA *Wartbarkeit* deutlich verbessert.

6.1.4 Linter

Um die NFA *Wartbarkeit* auch Tool-gestützt zu verbessern, wird auf der gesamten Projektebene ein sog. *Linter* eingesetzt, welcher mithilfe eines Regelsatzes eine statische Code-Analyse durchführt, um typische Fehler während der Entwicklung zu finden (OpenJS Foundation, 2021c). Somit können bspw. Zugriffe auf undefinierte Variablen oder ungenutzte Konstanten identifiziert werden. Neben der Identifikation von Fehlern, können im Zusammenspiel mit der jeweiligen IDE des Entwicklers ebenso automatische Korrekturen durchgeführt werden. In Kombination mit TypeScript sorgt das für klare Regelungen innerhalb des Teams und besser lesbaren Code.

6.1.5 Komponentenbibliothek

Nach Nielsen (1994b, S. 132-134) wird die Usability eines Produktes gefördert, wenn für den jeweiligen Kontext bekannte Elemente konsistent verwendet werden. Um die-

ser Vorschrift zu folgen und ebenso die Zeit der Entwicklung herabzusetzen, wurde die *Material-UI* Bibliothek für React verwendet. Neben diversen typischen Komponenten für eine Desktop- bzw. Web-Anwendung (Eingabefelder, Buttons, Slider, ...) bietet die Bibliothek ein ausgeklügeltes Design-System, welches eine stilistische Konsistenz über das gesamte System sicherstellt (Material-UI SAS, 2021). Auch hier wurde sorgfältig auf die Frequenz der Updates, sowie auf die Langlebigkeit der Bibliothek geachtet. Seit der ersten Veröffentlichung im Jahr 2014 werden bis heute kontinuierlich Updates ausgeliefert, sodass auch hier die NFA *Wartbarkeit* sichergestellt ist.

6.1.6 Logging

Da innerhalb des MARS-Explorer mehrere Prozesse gleichzeitig aktiv sind (*Main*, *Renderer*, *Languageserver*, etc.) und diese miteinander interagieren, kann sich das Debugging einzelner Prozeduren in einigen Fällen als schwierig erweisen. Um im Sinne der NFA *Wartbarkeit* die Funktionen des Systems auch bei Prozess-übergreifenden Prozeduren nachvollziehen zu können, wurde die Logging-Bibliothek *electron-log* eingeführt (Prokhorov, 2021). Bei schwerwiegenden Fehlern können die Logs herangezogen werden, um die zuvor durchlaufenen Prozeduren zu verstehen. Mithilfe der Bibliothek werden folgende Logs generiert:

main Ausgaben des *main*-Prozesses von Electron

renderer Ausgaben des *renderer*-Prozesses von Electron

lsp-server Ausgaben des Languageservers

lsp-client Ausgaben des Languageclients

ipc Ausgaben bzgl. der Kommunikation vom *renderer*- zum *main*-Prozess

Auf den Geräten des Anwenders bzw. Entwicklers werden Logs sowohl für den Entwicklungs- als auch den Production-Modus erstellt. Die Pfade zu den jeweiligen Logs unterscheiden sich je nach Betriebssystem und können der Dokumention der Bibliothek entnommen werden (Prokhorov, 2021).

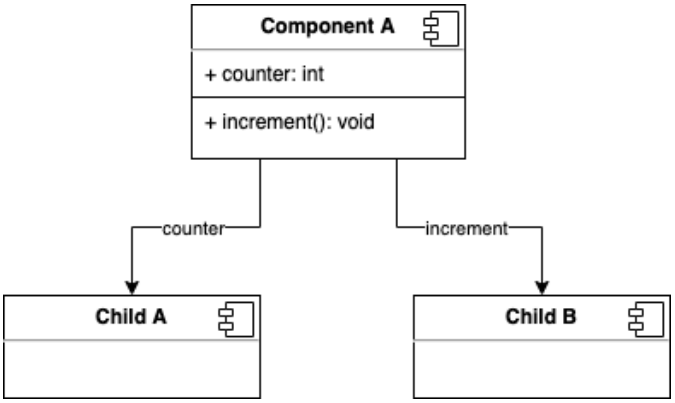


Abbildung 6.1: Einfaches Beispiel für das Datenmanagement innerhalb einer React-Anwendung

6.2 Datenmanagement

Das Datenmanagement innerhalb der React Anwendung wird mithilfe von der JS Bibliothek *Redux* betrieben. Um die Funktionsweise von Redux zu erklären, wird in einem ersten Schritt die typische Funktionsweise des Datenmanagements innerhalb von React anhand des in Abbildung 6.1 dargestellten Schaubildes erläutert. In diesem Schaubild soll innerhalb der Komponente *Child A* die Zahl *counter* dargestellt werden, während innerhalb von Komponente *Child B* dieser Wert mithilfe der Funktion *increment* erhöht werden soll. Um diesen Zustand zu synchronisieren, wird der Zustand typischerweise in einer Eltern-Komponente *Component A* gehalten und der Wert, sowie die Funktion zur Berechnung des Inkrements von dort aus an die Kinder-Komponenten weitergegeben.

Innerhalb des zuvor beschriebenen Szenarios kann es allerdings vorkommen, dass mehrere Komponenten auf den Wert *counter* zugreifen müssen und nicht direkte Kinderelemente von *Component A* sind. Aus diesem Grund wurde die Bibliothek *Redux* hinzugefügt, die es ermöglicht, solche Zustände, die für viele Komponenten relevant sind, innerhalb eines Speichers zu halten. Ein Schaubild für den Aufbau einer entsprechenden Datenverwaltung ist in Abbildung 6.2 dargestellt. Dort ist neben den zuvor gezeigten Komponenten auch ein *Store*-Objekt, sowie ein sog. *Slice* zu sehen. Ein solches *Slice* stellt immer nur einen Teil des globale Speichers dar. In diesem Beispiel enthält das Slice den gekapselten Zustand *counter* und stellt Funktionen zum Lesen als auch zum Bearbeiten des Zustandes bereit. Um einen Zustand zu Bearbeiten, muss die Methode *dispatch* verwendet werden, welche als Argument eine *Action* entgegennimmt, die wiederum selbst eine Funktion ist. Die Actions werden von den Slices selbst definiert, sodass die Kontrolle über die Bear-

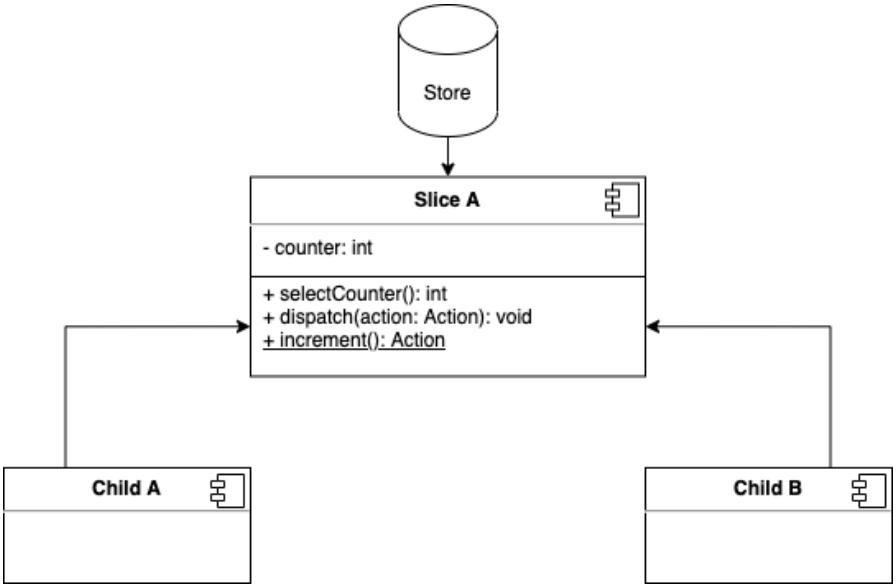


Abbildung 6.2: Beispiel für das Datenmanagement innerhalb einer React-Anwendung mithilfe der Bibliothek *Redux*

beitung stets direkt bei den Slices liegt. Bei der Methode *selectCounter* handelt es sich nicht um einen gewöhnlichen Getter, sondern um einen Beobachtungsmechanismus im Stile des *Observer*-Patterns. Durch die Verwendung der Methode *selectCounter* registrieren sich die Komponenten automatisch für Updates, sodass bei einem Aufruf der Action *increment* alle Komponenten, welche die *selectCounter* Methode verwenden, benachrichtigt werden. Da der Zustand im Slice verwaltet wird, ist die zuvor gezeigte Komponente *Component A* für die Datenhaltung obsolet.

Die Rolle von der Bibliothek Redux ist die Bereitstellung von Funktionen und Methoden zur einfachen Definition von Stores und Slices, sowie der Actions und der *Dispatch*-Funktion (Abramov, 2021). Außerdem sorgt Redux für ein konsistentes Verhalten des globalen Speichers, ein deutlich verbessertes Testing der Business-Logik und hilfreiche Debugging-Funktionen durch diverse Logs und Browser-Erweiterungen (Abramov, 2021).

Übertragen auf das zuvor genannte Entwurfsmuster MVP stellt das *Slice A* somit das Modell dar, während die Kinderelemente die Rolle der View einnehmen. In diesem Beispiel fehlt aber noch die Presenter-Komponente, welche die gesamte Kommunikation zwischen Model und View übernimmt. Dafür wurden die Hooks eingeführt, welche bereits im vorherigen Kapitel erwähnt worden sind. Mit der Einführung der Hooks sieht

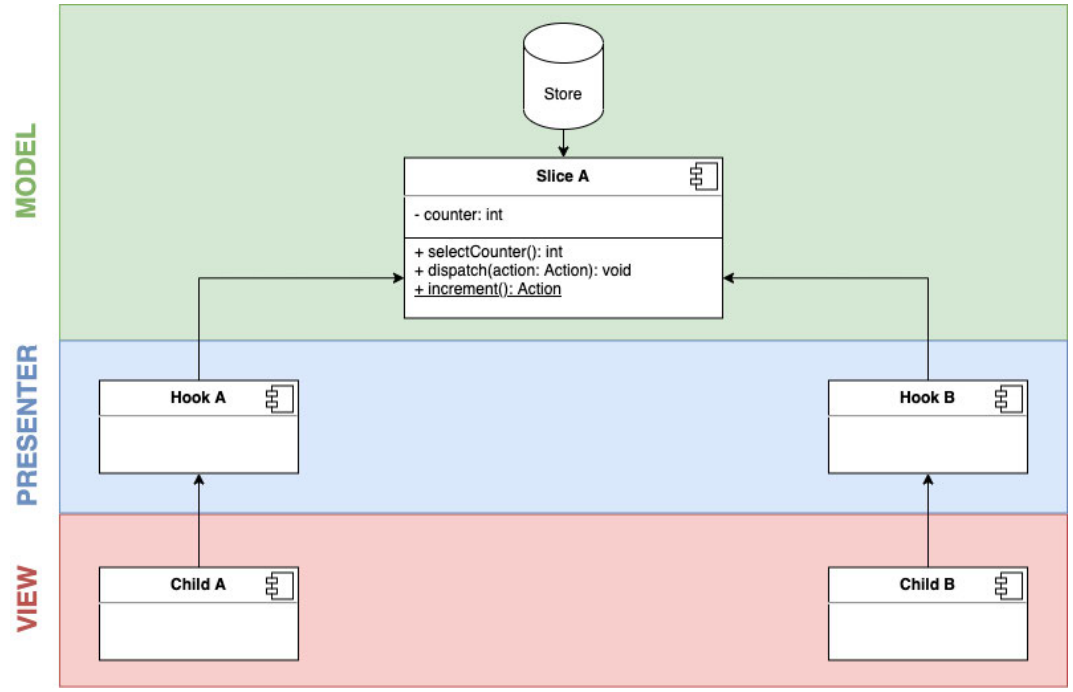


Abbildung 6.3: Beispiel für das Datenmanagement innerhalb einer React-Anwendung mithilfe der Bibliothek Redux und dem Konzept von Hooks angewandt

der Aufbau eines Features ungefähr so wie in Schaubild 6.3 dargestellt aus. Zwischen den Kinder-Elementen und dem Slice existiert jeweils eine Hook-Komponente, welche auf die Daten und Methoden aus *Slice A* zugreift und die für die Darstellung relevanten Daten an *Child A* bzw. *Child B* weiterleitet. Interagiert der Benutzer mit einer der beiden Child-Komponenten, wird das Event an die jeweilige Hook weitergegeben, welche die dazugehörige Logik ausführt. Somit kann die View jederzeit leicht ausgetauscht und der Presenter unabhängig von der View getestet werden. Falls die Komplexität einer Hook zu groß wird, kann eine solche Hook nach dem Mediator-Muster, welches in Kapitel 5.1 vorgestellt worden ist, in eine Mediator- und viele kleinere Hooks aufgeteilt werden.

Das MVP Muster ist stringent für die umfänglichen Features (Modeling, Simulation, ...) umgesetzt worden. Beispielsweise existieren aber auch Komponenten, die von mehreren Komponenten ohne jegliche Logik zur Darstellung verwendet werden und aus diesem Grund keine eigene Hook besitzen. Auch lokale Zustände, die keine weiteren Komponenten betreffen, werden nicht im globalen *Store* bzw. im jeweiligen Slice gespeichert, sondern in der eigenen Hook.

6.3 Learnings

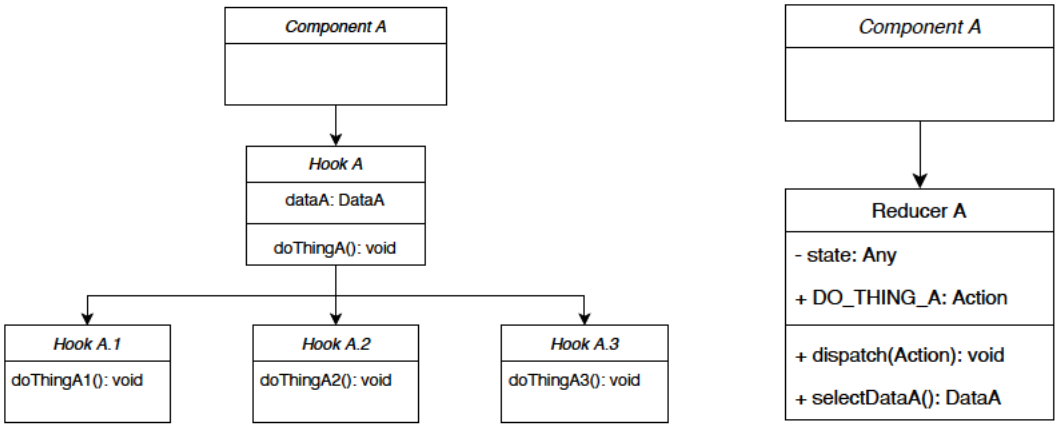
Innerhalb dieses Unterkapitels wird darauf eingegangen, welche Lehren aus der Umsetzung des MARS-Explorer gezogen werden können bzw. welche Aspekte in Zukunft noch verbessert werden könnten.

6.3.1 Fehlende standardmäßige Projektstruktur

Insgesamt hat sich der verwendete Technologie-Stack als sehr zufriedenstellend erwiesen. Die grundlegenden Technologien, wie Electron, React und Redux verfügen über eine ausführliche Dokumentation, sowie eine sehr aktive Entwicklergemeinschaft, wodurch i.d.R. Probleme schnell recherchiert und gelöst werden können. Ein Kritikpunkt, der über Electron und React geäußert werden könnte, aber auch gleichzeitig ein Vorteil für die meisten Entwickler darstellt, ist die Flexibilität der beiden Frameworks. Indem keine Annahmen über den Technologie-Stack gemacht und auch keine eindeutigen Strukturen vorgeschrieben werden, stecken beide Frameworks nur einen groben Rahmen ab, wie eine Anwendung implementiert werden kann. Dies ist für das schnelle Prototyping von Anwendung sehr vorteilhaft; in langlebigen Projekten, welches der MARS-Explorer werden soll, erfordert es vor allem im Team aber strikte, festgelegte Muster und Strukturen, die vom Team akzeptiert, eingehalten und kontrolliert werden müssen.

6.3.2 Abhängigkeiten zu Betreibern der externen Systeme

Eine erste Version des Microsoft spezifizierten Language Server Protocols wurde im Jahr 2017 veröffentlicht (Microsoft, 2021d). Mittlerweile existieren zwar insgesamt eine Vielzahl an Implementierung für verschiedenste Programmiersprachen, beim genauen Inspizieren fällt allerdings auf, dass für eine Programmiersprache meistens aber nur eine einzige vollständige Implementierung eines Servers existiert (Microsoft, 2021d). Da eine eigene Implementierung eines solchen Servers zu aufwendig wäre, ist man auf diese eine Implementierung angewiesen. Genau wie bei jeder Software dieser Größenordnung existieren aber auch hier einige Probleme, die erst über einen gewissen Zeitraum auffallen, gemeldet und dann behoben werden (.NET Foundation, 2021). Im Falle des hier verwendeten Omnisharp-Roslyn-Servers existiert neben vereinzelten Einträgen im Wiki des Repositories auch keine Dokumentation.



(a) Initialer Ansatz

(b) Verbesserter Ansatz

Abbildung 6.4: Generisches Beispiel zur Trennung der Logik einer Komponente

Ähnliches gilt auch bei der Wahl des Klienten zur Kommunikation mit dem Languageserver. Für den Monaco-Editor existiert lediglich eine externe Bibliothek, die als Adapter zwischen dem Server und dem Editor fungiert, sodass auch hier wieder eine extreme Abhängigkeit zur verwendeten Bibliothek und vor allem zu den Entwicklern der Bibliothek besteht (Microsoft, 2021d, TypeFox GmbH, 2021). Als stellvertretende Dokumentation existieren innerhalb des Repositories zumindest einige Beispiele, die als Orientierungspunkt verwendet werden können (TypeFox GmbH, 2021). Theoretisch ist es auch hier möglich, die Funktionalitäten eigenhändig zu implementieren, aus Zeitgründen liegt das allerdings außerhalb des Umfangs dieser Arbeit.

Um in Zukunft einen solchen Übergang zu einer eigenen Implementierung zu vereinfachen, wurde innerhalb des MARS-Explorer die Ordnerstruktur *standalone* angelegt. Innerhalb dieser Ordnerstruktur befinden sich alle Funktionalitäten, die den Languageclient betreffen, sodass theoretisch ein daraus resultierendes Paket zu einem späteren Zeitpunkt veröffentlicht werden könnte.

6.3.3 Unübersichtlichkeit durch kleinteilige Logik

Im Kapitel *Entwurfsmuster* wurde das *Mediator*-Entwurfsmuster beschrieben, welches verwendet wurde, um komplexe Zustände und Abläufe innerhalb der Komponenten aufzuteilen. Ein generisch Beispiel ist dazu in Abbildung 6.4a dargestellt. In diesem Beispiel existiert eine *Component A*. Wie bereits ausführlich in Kapitel 5 beschrieben, existiert

zu jeder hinreichend großen Komponente eine eigene *Hook*, welche die Business-Logik von der Darstellungs-Logik der Komponente trennt. Aus diesem Grund befinden sich die Funktionen, die von *Component A* aufgerufen werden, in *Hook A*. In diesem Fall stellt die Hook die Daten *dataA* sowie Funktion *doThingA* zur Verfügung. Diese Funktion besteht wiederum aus kleinteiligen Funktionen, die in den Hooks *A.1*, *A.2* und *A.3* definiert sind. Im Laufe der Entwicklung hat sich gezeigt, dass eine solche Aufteilung zwar für eine vorteilhafte Trennung der Zuständigkeiten sorgt, die Nachvollziehbarkeit aber stark vermindert wird.

Aus diesem Grund werden die im Kapitel *Datenmanagement* erläuterten *Reducer* nicht nur auf globaler Ebene, sondern auch auf Lokaler eingesetzt. Eine aktualisierte Darstellung des generischen Beispiels ist in Abbildung 6.4b zu sehen. Die *Component A* verfügt lediglich über einen *Reducer A*. Die Daten, die von der Komponente benötigt werden, können über im Reducer definierte Selektoren – in diesem Fall *selectDataA* – bezogen werden. Änderungen an diesem Zustand werden mithilfe der definierten Aktionen (*DO_ THING_ A*) und der *dispatch*-Funktion des *Reducers* veranlasst. Die Anwendung dieses Prinzips sorgt für eine besser Lesbarkeit, Nachvollziehbarkeit und Testbarkeit.

7 Überprüfung der Anforderungen

Innerhalb dieses Kapitels werden die im Kapitel *Analyse* aufgestellten Anforderungen auf ihre Umsetzung überprüft.

7.1 Funktionale Anforderungen

Um die in Kapitel 4.3 aufgestellten funktionalen Anforderungen auf ihre Umsetzung zu überprüfen und gleichzeitig die Effektivität der Anwendung aufzuzeigen, wird innerhalb dieses Abschnitts die Funktionsweise des MARS-Explorer vorgestellt. Die Beschreibung der umgesetzten Anforderungen erfolgt in Form von roten Markierungen auf Bildschirmfotos der Anwendung. Des Weiteren wird in Form von (FA: $< Titelder Anforderung >$) auf die jeweilige FA verwiesen, die durch das beschriebene Feature umgesetzt wurde. Am Ende dieses Abschnitts folgt eine Zusammenfassung darüber, welche Anforderungen umgesetzt wurden und welche nicht.

7.1.1 Projektverwaltung

Das in Abbildung 7.1 dargestellte Bildschirmfoto zeigt den MARS-Explorer nach dem ersten Start der Anwendung. Auf der linken Seite beim Punkt 1 ist eine Leiste dargestellt, die zur Navigation zwischen den verschiedenen Phasen verwendet werden kann. Alle Navigationspunkte bis auf *Home* sind deaktiviert, da zu diesem Zeitpunkt kein Projekt ausgewählt ist. Ohne ein ausgewähltes Projekt kann kein Modell erstellt und keine Konfiguration bearbeitet werden. Diese Maßnahmen folgen den von Nielsen (2012) aufgestellten Heuristiken, die u.a. besagen, dass ein Design, welches Fehler gar nicht erst zulässt, wichtiger ist als eine gut formulierte Fehlermeldung. Aus diesem Grund sollen potentielle Fehlerquellen identifiziert und so früh wie möglich unterbunden werden. In diesem Fall wäre es zum Beispiel unvorteilhaft, dem Benutzer die Navigation zu einer Phase zu erlauben, zu der noch keine Daten vorliegen. Lediglich der Navigationspunkt

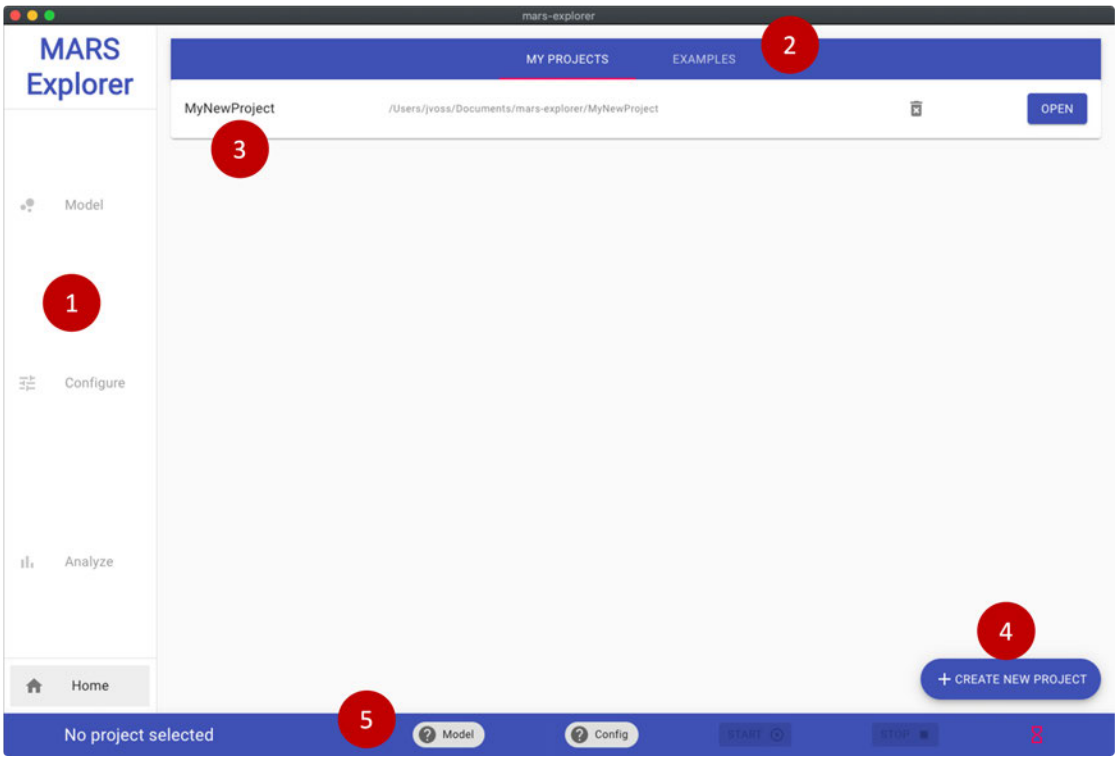


Abbildung 7.1: Bildschirmfoto MARS-Explorer: Darstellung der eigenen Projekte innerhalb der Projektverwaltung

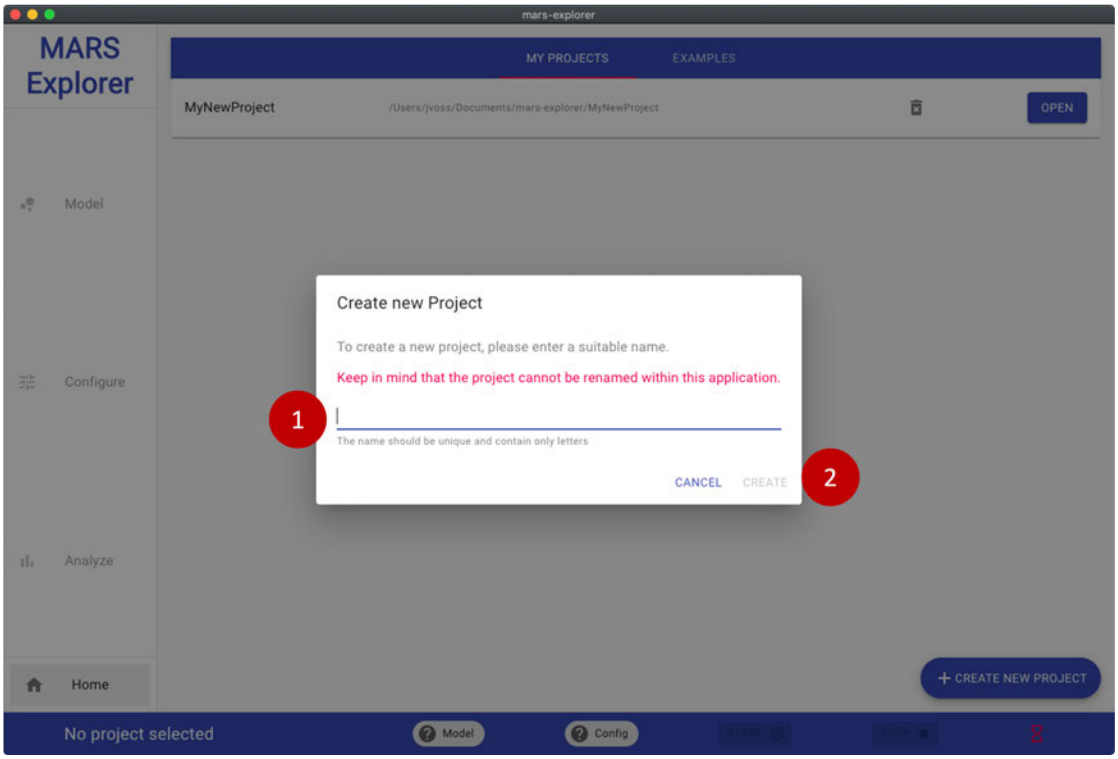


Abbildung 7.2: Bildschirmfoto MARS-Explorer: Projektverwaltung mit geöffnetem Dialog zur Erstellung eines neuen Projekts

Home kann jederzeit ausgewählt werden, um auf den in Abbildung 7.1 dargestellten Startbildschirm zurückzukehren. Bei Punkt 2 ist eine horizontale Leiste zum Umschalten zwischen den eigenen und den Beispielprojekten zu sehen. In der aktuellen Abbildung ist die Ansicht für die eigenen Projekte ausgewählt, in der bei Punkt 3 das Projekt *MyNewProject* angezeigt wird (FA 40: Projekte einsehen). Dieses kann durch einen Klick auf den Button *OPEN* geöffnet oder mit einem Klick auf das Mülleimer-Symbol gelöscht werden (FA 44: Projekte löschen).

Mit einem Klick auf den Button *CREATE NEW PROJECT* wird das in Abbildung 7.2 dargestellte Dialogfenster geöffnet. In diesem muss der Benutzer einen Namen in das bei Punkt 1 dargestellte Textfeld eingeben und mit einem Klick auf *CREATE* bestätigen. Im Hintergrund des Systems wird daraufhin mithilfe der .NET CLI ein neues Projekt erstellt und in der Liste angezeigt (FA 41: Projekt erstellen). Da ein solches Projekt mit wachsender Größe innerhalb des Quellcodes eine unvorhersehbare Menge an Verweisen auf den eigenen Projektnamen beinhalten kann (bspw. in Form von *Namespaces*), ist die Umbenennung eines Projekts eine komplexe Aufgabe, die umfassend getestet werden

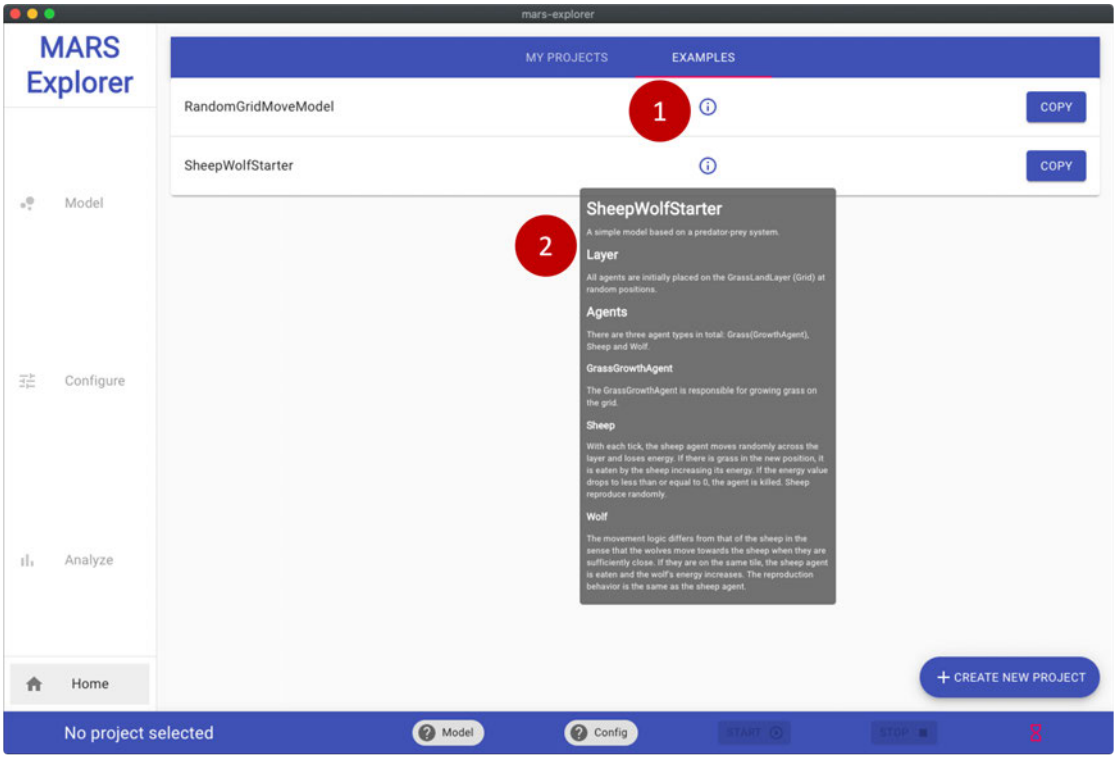


Abbildung 7.3: Bildschirmfoto MARS-Explorer: Darstellung der Beispielprojekte innerhalb der Projektverwaltung

muss. Falls der Projektname innerhalb der zahlreichen Dateien fälschlicherweise geändert oder übersehen wird, kann dies zu einem nicht funktionsfähigen Projekt führen. Da bei diesem Feature eine Kosten-Nutzen-Abwägung negativ ausfiel, wurde *Projekte umbenennen* (FA 43) für die erste Version des MARS-Explorer nicht implementiert.

Abbildung 7.3 zeigt die Beispielprojekte, die zu diesem Zeitpunkt innerhalb des MARS-Explorer existieren (FA 17: Beispielmodelle ansehen). Informationen über die Beispielprojekte, wie bei Punkt 2 dargestellt, können durch das Hovern über das Informations-Symbol (Punkt 1) angezeigt werden (FA 18: Informationen zu Beispielmodellen einsehen). Außerdem können die Beispielprojekte mit einem Klick auf den Button *COPY* kopiert werden, sodass sie als eigene Projekte geöffnet und anschließend erweitert werden können (FA 42: Beispielmodelle kopieren). Um als Lehrender neue Beispielprojekte zu hinterlegen, kann ein kompilierbares Projekt in eine dafür vorgesehene Ordnerstruktur innerhalb der Ressourcen des MARS-Explorer kopiert werden (FA 16: Beispielmodelle einstellen). Die Struktur und weitere Vorraussetzungen wurden in einer Markdown-Datei festgehalten.

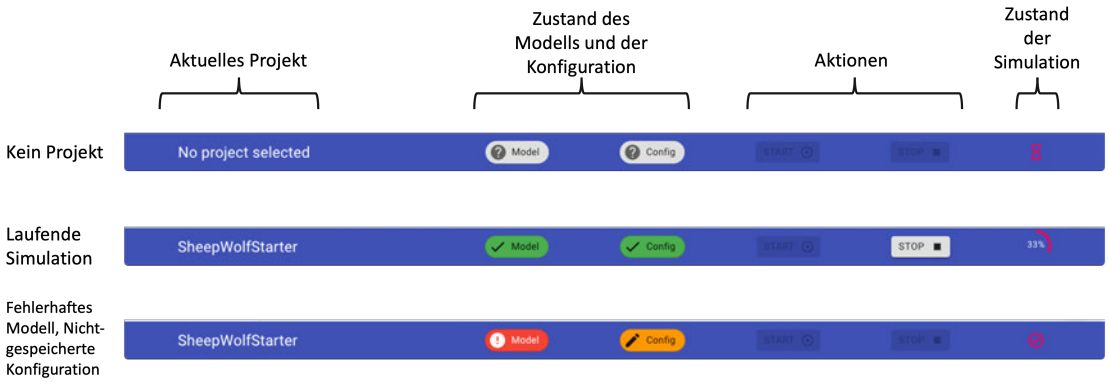


Abbildung 7.4: Zustandsmatrix der Toolbar innerhalb des MARS-Explorer

7.1.2 Ausführung der Simulation

In Abbildung 7.4 ist eine Zustandsmatrix der Toolbar dargestellt, die jederzeit am unteren Rand des MARS-Explorer zu sehen ist. Die linke Seite der Toolbar zeigt das aktuelle Projekt. In der Mitte der Leiste werden zwei sog. *Chip*-Elemente angezeigt, welche zum einen den Zustand des Modells (FA 14: Status anzeigen) und zum anderen den Zustand der Konfiguration anzeigen (FA 29: Status der Konfiguration darstellen). Auf der rechten Seite der Leiste ist ein Button *START* zum Starten einer Simulation dargestellt (FA 30: Simulation starten). Analog dazu kann der nebenstehende Button *STOP* zum Stoppen einer Simulation verwendet werden (FA 32: Simulation stoppen). Am rechten Rand der Leiste ist jederzeit ein Symbol zu sehen, welches den Zustand der Simulation anzeigt (FA 33: Simulationsstatus anzeigen).

Das Schaubild zeigt neben dem grundlegenden Aufbau verschiedene beispielhafte Zustände der Toolbar. Im ersten Zustand ist kein Projekt ausgewählt, was durch den dargestellten Schriftzug *No project selected* verdeutlicht wird. Ohne ein ausgewähltes Projekt, kann keine Validierung stattfinden, weswegen die Zustände des Modells und der Konfiguration mit einem Fragezeichen-Symbol versehen sind. Außerdem kann ohne ein Projekt keine Simulation ausgeführt werden, weswegen der *Start*-Knopf deaktiviert ist. Der zweite Zustand zeigt die Toolbar während einer laufenden Simulation. Eine Simulation kann nur angestoßen werden, wenn Modell und Konfiguration erfolgreich validiert worden sind. Dies wird mit der grünen Farbe und mit den Haken-Symbolen gekennzeichnet. Wurde eine Simulation gestartet, kann diese mithilfe des aktiven Buttons *STOP* angehalten werden. Der Zustand der Simulation wird in Form eines Kreises, der sich je nach Simulationsfortschritt weiter füllt, dargestellt (FA 34: Fortschritt anzeigen). Die unterste Toolbar

Abbildung 7.5: Bildschirmfoto MARS-Explorer: Darstellung des Output-Dialogs nach Klick auf das Simulationszustands-Symbol

in Abbildung 7.4 zeigt unterschiedliche Zustände für das Modell und die Konfiguration an. Das Modell wurde nicht erfolgreich validiert, weswegen es durch die rote Farbe und das Ausrufezeichen-Symbol als fehlerhaft gekennzeichnet wird. Die Konfiguration enthält nicht gespeicherte Änderungen, was durch die Farbe Orange und das Stift-Symbol dargestellt wird. Mit dem Mauszeiger kann über das rot gefärbte *Chip*-Element gefahren werden, um zu sehen, welche Klassen innerhalb des Modells einen Fehler enthalten. Da beide Zustände nicht valide sind, kann die Simulation nicht gestartet werden (FA 31: Simulationsstart überprüfen). Das Symbol rechts steht für den erfolgreichen Abschluss einer vorherigen Simulation.

Mit einem Klick auf das Symbol, welches den Zustand der Simulation anzeigt, wird der in Abbildung 7.5 dargestellte Dialog geöffnet. Dieser zeigt je nach ausgewähltem *CONSOLE*- oder *ERROR*-Tab entweder die Ausgabe der Simulation (FA 35: Ausgabe einsehen) – welche bspw. mit `Console.WriteLine` getätigt wurde – oder die Fehlermeldung, die während der Simulation aufgetreten ist (FA 36: Fehler einsehen). Treten innerhalb einer Simulation Fehler auf, wird dieser Dialog mit dem Tab *ERROR* automatisch geöffnet, sodass der Benutzer direkt eine Begründung für das Fehlschlagen der Simulation bekommt.

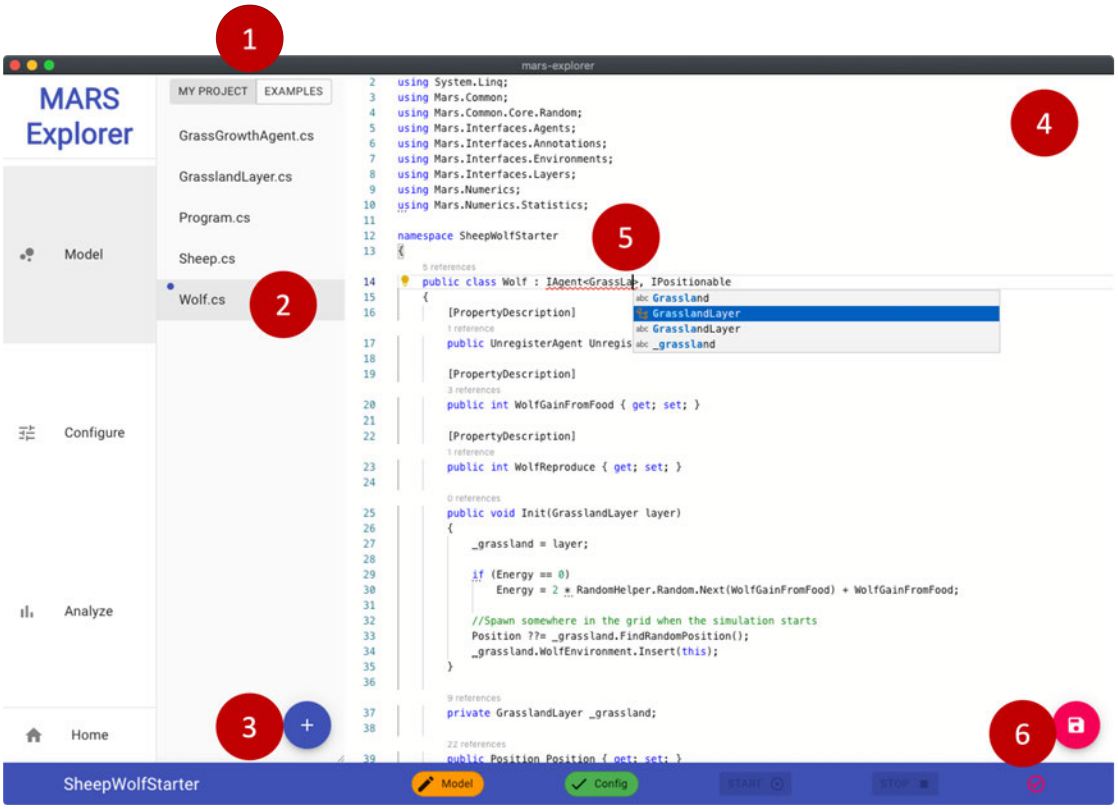


Abbildung 7.6: Bildschirmfoto MARS-Explorer: Bearbeiten des Quellcodes einer Klasse

7.1.3 Modellierung

Abbildung 7.6 zeigt den MARS-Explorer während der Modellierung. Punkt 1 stellt eine Unterscheidung zwischen der Ansicht des eigenen Projekts (*MY PROJECT*) und der Beispielprojekte (*EXAMPLES*) dar. In diesem Fall sind die Klassen des eigenen Projekts über Punkt 2 zu sehen (FA 5: Klassen ansehen). Diese können mit einem Klick ausgewählt werden, sodass der Quellcode der jeweiligen Klasse im rechten Fenster bei Punkt 4 angezeigt wird (FA 6: Klasse ansehen). Mit einem Rechtsklick auf die Einträge in der Liste öffnet sich ein Menü, welches die Möglichkeit zum Löschen (FA 9: Klasse löschen) und Umbenennen (FA 8: Klasse umbenennen) der jeweiligen Klasse bietet. Das Umbenennen ist in der implementierten Version allerdings – aufgrund des zuvor erwähnten Problems mit der möglichen Vielzahl an Referenzen – deaktiviert. Der blaue Punkt, welcher links über dem Listeneintrag *Wolf.cs* bei Punkt 2 zu sehen ist, dient als Indikator dafür, dass die Klasse bearbeitet, aber noch nicht gespeichert wurde. Es kann ebenso ein roter Punkt dargestellt werden, wenn sich innerhalb der jeweiligen Klasse ein

Fehler befindet. Im Fenster bei Punkt 4 befindet sich der Quelltext-Editor, der auf dem selben Editor wie die bekannte IDE *Visual Studio Code* basiert. Er kann dazu verwendet werden, den Quelltext einer Klasse zu bearbeiten (FA 7: Klasse bearbeiten).

In der Abbildung ist unter Punkt 5 außerdem eine automatische Vervollständigung zu sehen, die den Benutzer dabei unterstützt, Code zu schreiben (FA 12: Eingabe automatisch vervollständigen). Die Eingaben des Benutzers werden mit jeder Eingabe validiert (FA 13: Stetige Validierung des Modells). Aus diesem Grund ist eine rote Markierung unter dem geschriebenen Code zu sehen, die aufzeigt, dass an dieser Stelle ein Fehler vorliegt. Die Fehlermeldungen können durch Hovern über die jeweilige Markierung eingesehen werden (FA 15: Fehler ansehen). In diesem Fall ist der Fehler irrelevant, da der Benutzer die Eingabe offensichtlich noch nicht abgeschlossen hat. Das *Chip*-Element der Toolbar im unteren Bereich ist in diesem Fall noch nicht fehlerhaft markiert, da die Änderungen des Nutzers noch nicht gespeichert wurden. Erst nach dem Speichern, wird der *Chip* – je nach Ergebnis der Validierung – entweder grün oder rot eingefärbt. Punkt 6 zeigt ein Disketten-Symbol, welches zum Speichern der Änderungen betätigt werden kann. Getätigte Eingaben können mit den bekannten Tastenkombinationen (unter Windows bspw. `STRG + Z`) oder dem hier nicht dargestellten Anwendungsmenü des jeweiligen Betriebssystems rückgängig gemacht (FA 10: Eingabe rückgängig machen) oder wiederholt (FA 11: Eingabe wiederholen) werden. Des Weiteren werden dem Benutzer IDE-typische Funktionen über ein Kontextmenü zur Verfügung gestellt. Dabei handelt es sich um die selben Funktionen, die auch von *Visual Studio Code* unterstützt werden, wie bspw. *Definitionen aufzeigen*, *Umbenennung von Variablen* oder *Formatierung*.

Punkt 3 zeigt einen Button mit dem neue Klassen erzeugt werden können (FA 1: Klasse erstellen). Der Dialog der sich daraufhin öffnet, ist in Abbildung 7.7 dargestellt. Dieser gibt dem Benutzer die Möglichkeit, zwischen den drei Typen *AGENT*, *ENTITY* und Layer auszuwählen, wobei Layer zusätzlich in die drei Kategorien *BASIC LAYER*, *RASTER LAYER* und *VECTOR LAYER* unterteilt ist. Unterhalb der Buttons wird ein Informationstext angezeigt, der über die unterschiedlichen Typen informiert, um den Benutzer bei der Auswahl zu unterstützen (FA 2: Hilfestellung zur Auswahl der Objekte erhalten). Nachdem die notwendigen Daten für die Erzeugung der jeweiligen Objekte im darunterliegenden Textfeld eingegeben wurden, kann der Benutzer durch einen Klick auf *CREATE* seine Auswahl bestätigen. Dadurch wird eine für den jeweiligen Typen angefertigte Vorlage verwendet, die bereits das Klassenkonstrukt mit implementierten Schnittstellen beinhaltet (FA 3: Objekt-Vorlagen verwenden). Eine beispielhafte Vorlage wurde bereits in Kapitel 4.3 vorgestellt. Nach einer entsprechenden Erzeugung ist

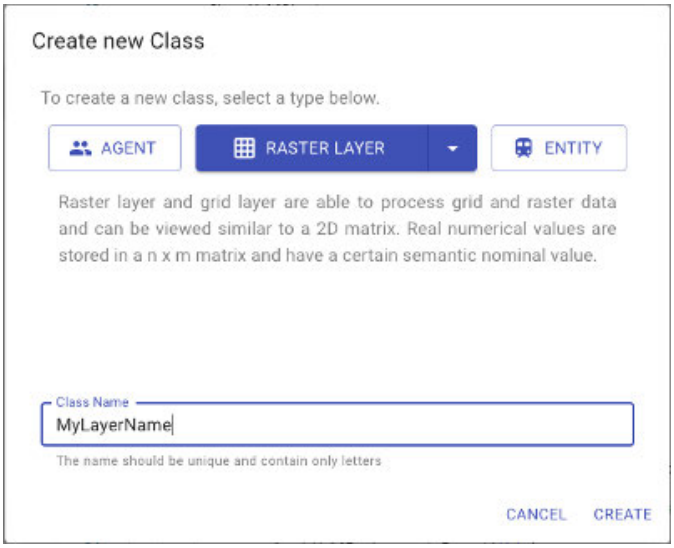


Abbildung 7.7: Bildschirmfoto MARS-Explorer: Dialog zum Erstellen einer neuen Klasse

das neue Objekt der Simulation allerdings noch nicht bekannt. Erst wenn dieses Objekt wie in Kapitel 4.3 der `ModelDescription` hinzugefügt worden ist, wird es innerhalb der Simulation verwendet. Dies ist ein Schritt, welcher theoretisch von der Anwendung übernommen werden könnte. Da die Syntax der Programmiersprache jedoch sehr komplex ist, kann das fehlerfreie Einfügen von Code-Abschnitten – wie die die Registrierung eines Objektes – ohne den Einsatz eines Compilers nicht garantiert werden. Die Umsetzung der Anforderung *Neues Objekt registrieren* (FA 4) wurde aufgrund des überproportionalen Aufwands im Rahmen dieser Arbeit nicht durchgeführt. Um den Benutzer dennoch in dieser Hinsicht zu unterstützen, wird zu jeder neuen Klasse der in Zeile 5 des Listings 7.1 dargestellte Kommentar generiert. Dieser kann vom Modellierenden kopiert und in den Einstiegspunkt des Modells eingefügt werden. Außerdem dient er als Gedächtnisstütze dafür, das Objekt zu registrieren.

Listing 7.1: Quellcode einer erstellten Layer-Klasse innerhalb des MARS-Explorer

```
1 using Mars.Components.Layers;
2
3 namespace SheepWolfStarter
4 {
5     // TODO: add "description.AddLayer<MyLayerName>();" to
          your Program.cs
6     public class MyLayerName : RasterLayer
```

```
7       {
8       }
9   }
```

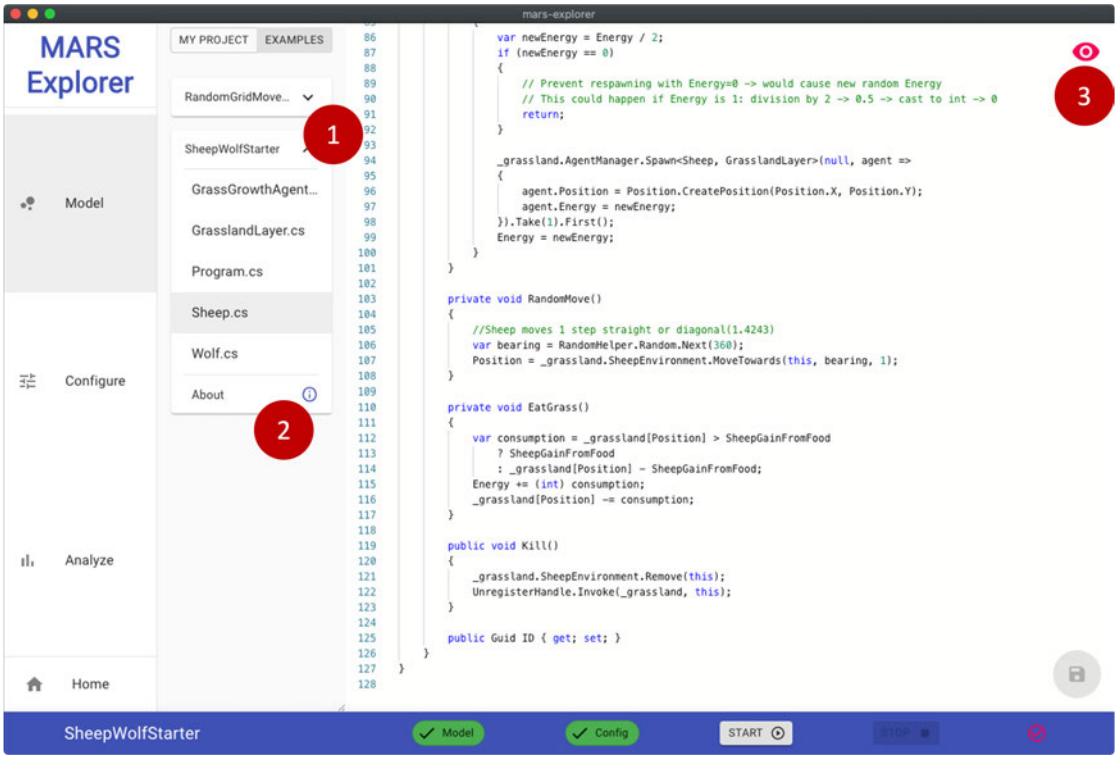


Abbildung 7.8: Bildschirmfoto MARS-Explorer: Darstellung der Beispielprojekte innerhalb des Tabs *Model*

Da die Studierenden noch kein ausgeprägtes Fachwissen über die vielzähligen Funktionen und Methoden des MARS-Framework besitzen, bieten die in Abbildung 7.8 dargestellten Beispielprojekte eine Art Nachschlagewerk. Diese können angesehen werden, indem *EXAMPLES* in der oberen Leiste über Punkt 1 ausgewählt wird. Die Auswahl kann jederzeit stattfinden, ohne, dass ggf. nicht-gespeicherte Änderungen des Benutzers verloren gehen. Die Liste bei Punkt 1 zeigt die verschiedenen Beispielprojekte, die von den Lehrenden hinterlegt worden sind. Im aktuellen Beispiel sind das lediglich zwei. Die Projekte können mit einem Klick auf die Pfeile auf- bzw. zugeklappt werden, um die beinhalteten Klassen zu offenbaren bzw. zu verstecken. Die Klassen können daraufhin wie im anderen Tab ausgewählt und auf der rechten Seite eingesehen werden (FA 17: Beispielmodelle ansehen). Der unterste Eintrag *About* bei Punkt 2 zeigt einen Dialog mit Informationen zu dem jeweiligen Projekt nach einem Klick. Dabei handelt es sich um die selben Information

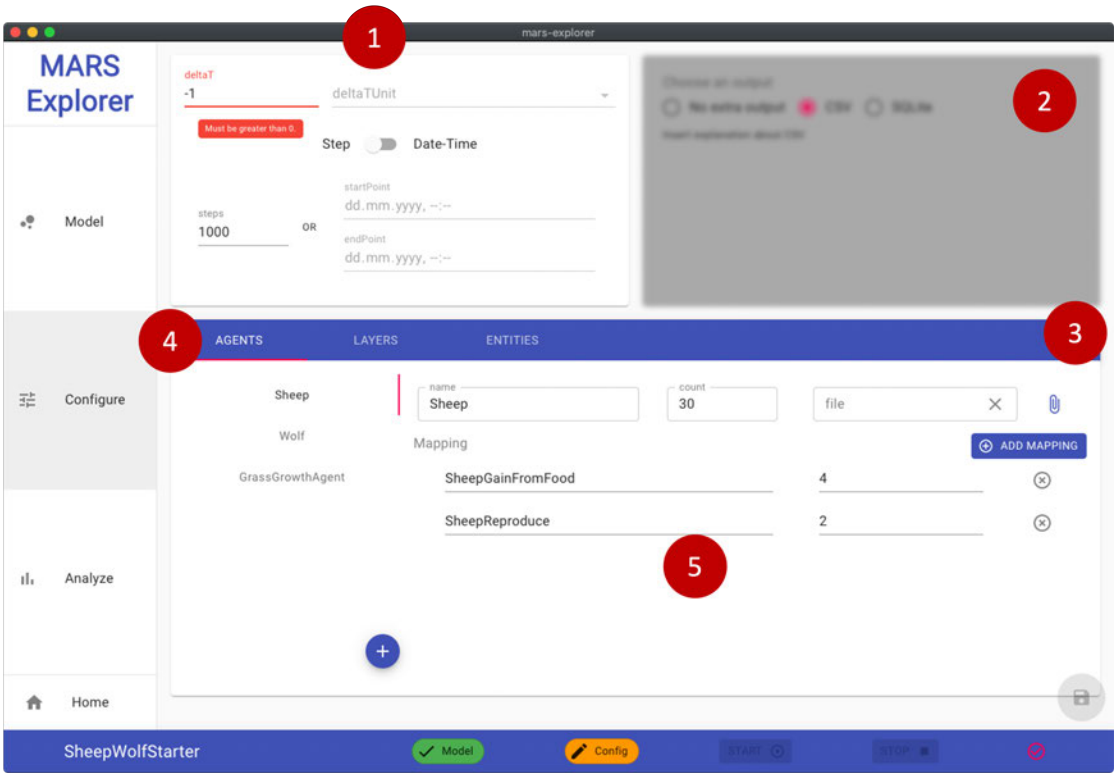


Abbildung 7.9: Bildschirmfoto MARS-Explorer: Konfiguration der Simulation

wie in der Projektverwaltung dargestellt (FA 18: Informationen zu Beispielmodellen einsehen). Das Auge-Symbol bei Punkt 3 zeigt dem Benutzer, dass die geöffnete Datei nur gelesen, aber nicht bearbeitet werden kann. So können die Benutzer die Inhalte einsehen und kopieren, ohne dass sie befürchten müssen, das funktionierende Modell in einen invaliden Zustand zu bringen.

7.1.4 Konfiguration

Die Abbildung 7.9 zeigt die Seite zur Konfiguration der Simulation (FA 20: Konfiguration darstellen). Dieses entspricht vom Aufbau her den angefertigten Mockups:

- Bei Punkt 1 befinden sich alle Felder, welche die globalen Attribute betreffen (FA 21: Globale Einstellungen bearbeiten)
- Punkt 2 zeigt die verborgenen Felder bzgl. der Konfiguration der Ausgabe der Simulationsergebnisse

- Punkt 3 zeigt das Mapping der verschiedenen Typen (*Agenten-Mappings bearbeiten* (FA 22), *Layer-Mappings bearbeiten* (FA 23), *Entity-Mappings bearbeiten* (FA 24))

Die drei Bereiche wurden nach logischen Zusammenhängen gruppiert, sodass voneinander abhängige Felder innerhalb eines Kastens zu sehen sind (FA 19: Konfigurationsmöglichkeiten logisch anordnen). Des Weiteren wurde für jedes Feld ein geeignetes Eingabeelement festgelegt, welches Fehleingaben erschwert. Zur Eingabe des Start- und Enddatums der Simulation kann bspw. nur das vorgegebene Format `dd.mm.yyyy` oder ein *Datepicker*-Dialog verwendet werden. Ein weiteres Beispiel sind die Zeiteinheiten (*deltaTUnit*), die aus einer Menge von vorgefertigten Optionen mithilfe eines *Select*-Elements ausgewählt werden können. In der Abbildung 7.9 wird am Beispiel des Textfeldes von *deltaT* gezeigt, wie Fehlermeldungen dargestellt werden. Das jeweilige Feld wird bei syntaktischen oder semantischen Fehlern rot markiert (FA 27: Fehler in der Konfiguration anzeigen) und eine Fehlermeldung angezeigt (FA 28: Grund für Fehler anzeigen). Eine Validierung wird nach dem Verlassen eines Eingabeelements – auch *blur* genannt – durchgeführt, um sicherzustellen, dass die Konfiguration valide ist. Der Bereich aus Punkt 2 wurde unkenntlich gemacht, da die Konfiguration der Ausgabe der Simulationsergebnisse ein Thema ist, welches nachträglich den fortgeschrittenen Modellierenden zugeordnet wurde. Eine ausführliche Erläuterung dazu ist in Kapitel 4.3 zu finden.

Durch das Mapping können den Attributen der Objekte verschiedene Werte zugeordnet werden. Dazu wird im ersten Schritt der Typ des entsprechenden Objekts in der Leiste bei Punkt 4 ausgewählt (*AGENTS*, *LAYERS*, *ENTITIES*). Die bisher angelegten Mappings des ausgewählten Typs werden daraufhin in der linken darunterliegenden Liste angezeigt. Mit einem Klick auf das Plus-Symbol kann ein neues Mapping angelegt werden. Wird ein Mapping aus der Liste ausgewählt, werden dessen Werte im rechten Bereich bei Punkt 5 dargestellt. Neben dem Namen (*name*) und der Möglichkeit, die Anzahl der Agenten zu bearbeiten (*count*), kann eine Eingabedatei verlinkt werden (*file*), dessen Daten dann als Mapping für das jeweilige Objekt verwendet werden. In der darunterliegenden Tabelle bei Punkt 5 findet die Zuweisung der im Modell festgelegten Attribute statt. In diesem Fall existiert innerhalb des Agenten *Sheep* ein gekennzeichnetes Attribut *SheepGainFromFood*, dem der Wert 4 zugewiesen wird. Neue Zuweisungen können mit einem Klick auf den Button *ADD MAPPING* hinzugefügt und bestehende mit einem Klick auf das jeweilige Kreuz-Symbol entfernt werden.

Genau wie in der Modellierung existiert ein Disketten-Button, welcher die getätigten Eingaben speichert. Außerdem können Eingaben mithilfe der Betriebssystem-abhängigen

Abbildung 7.10: Bildschirmfoto MARS-Explorer: Darstellung der letzten Simulationsergebnisse in Form eines Linien-Diagramms zur Bestimmung der Anzahl der Agenten pro Simulationsfortschritt

Tastenkombination rückgängig gemacht (FA 25: Eingaben in der Konfiguration rückgängig machen) oder wiederholt werden (FA 26: Eingaben in der Konfiguration wiederholen).

7.1.5 Analyse

Der letzte Navigationspunkt, der innerhalb des MARS-Explorer existiert, ist die Analyse. Dieser Tab ist in Abbildung 7.10 dargestellt. Auch dieses Fenster ist in drei Abschnitte aufgeteilt. In der Liste bei Punkt 1 werden alle Agenten angezeigt, über die innerhalb der Simulation Daten gesammelt wurden. In diesem Fall sind das die Agenten *GrassGrowthAgent*, *Sheep* und *Wolf*, wovon lediglich die letzten beiden zur Darstellung ausgewählt worden sind (FA 39: Ergebnisse filtern). Die Auswahl bei Punkt 2 wird dazu genutzt, um zwischen den Darstellungsarten zu unterscheiden. Durch die Auswahl der Option *Number of agents per Progress(Line-Chart)* lässt sich der Modellierende bei Punkt 3 die absolute

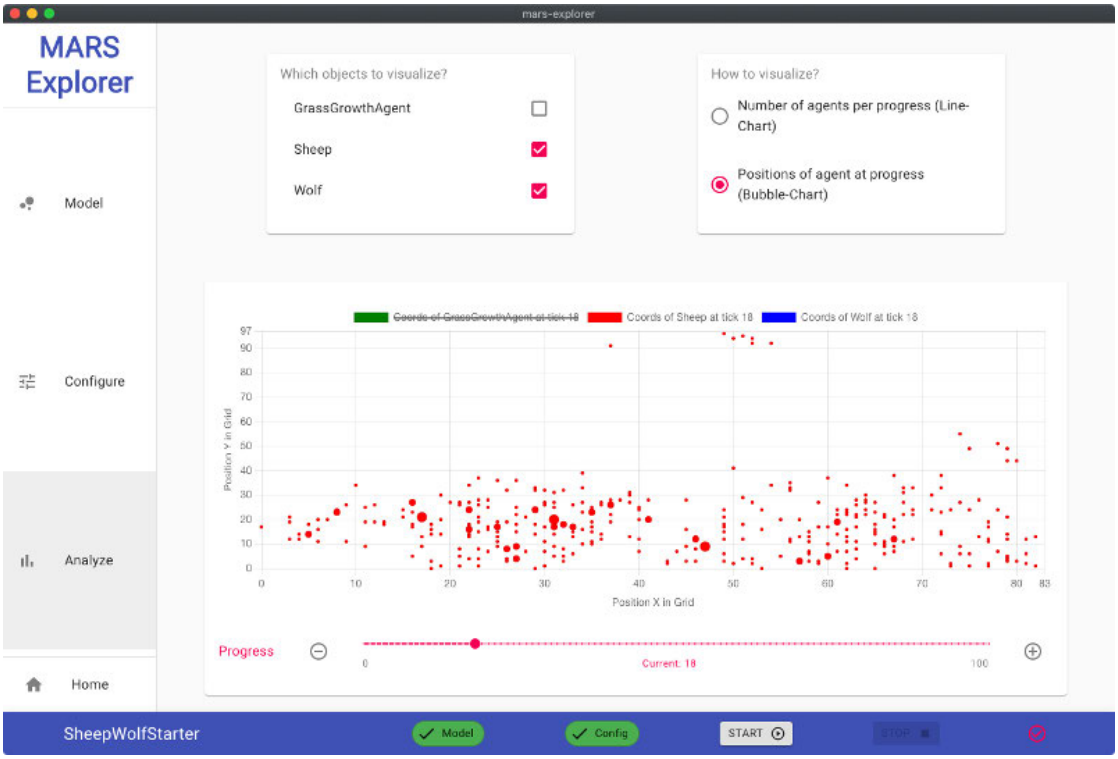


Abbildung 7.11: Bildschirmfoto MARS-Explorer: Darstellung der letzten Simulationsergebnisse in Form eines Blasen-Diagramms zur Bestimmung der Positionen der Agenten pro Simulationsfortschritt

Anzahl der ausgewählten Agenten zu den jeweiligen Simulationsfortschritten der Simulation anzeigen (FA 37: Agentenanzahl einsehen). In diesem Fall starten die Agenten *Sheep* und *Wolf* mit einer vergleichbar hohen Anzahl an Agenten bis sich letztendlich bei ca. 10% Simulationsfortschritt die *Sheep*-Agents um ein Vielfaches vermehren.

Als zweite Darstellungsmöglichkeit können die Positionen der Agenten angezeigt werden. Eine solche Visualisierung ist in Abbildung 7.11 zu sehen. Mit Hilfe des unterhalb des Diagramms dargestellten *Slider*-Elements kann der zu untersuchende Zeitpunkt der Simulation bestimmt werden. Die Blasen innerhalb des Diagramms stellen die Positionen der Agenten zu dem ausgewählten Zeitpunkt dar (FA 38: Agentenpositionen einsehen). Dabei gilt: je größer die Blase, desto mehr Agenten befinden sich an dieser Position. In dem gezeigten Beispiel befinden sich bei einem Simulationsfortschritt von 18% die meisten *Sheep*-Agenten innerhalb des vorstellbaren Rechtecks, welches durch die Koordinaten $(0, 0)$ und $(83, 60)$ gespannt wird.

Die Graphen zeigen immer die Daten der zuletzt gestarteten Simulation. Ebenso ist es möglich, die Live-Simulationsergebnisse einer laufenden Simulation zu betrachten.

7.1.6 Zusammenfassung

Zusammenfassend kann über die funktionalen Anforderungen gesagt werden, dass 41 von 44 Anforderungen umgesetzt worden sind. Für eine gesammelte Übersicht der Anforderungen und dessen jeweiligen Zustand der Umsetzung, wurde die im Anhang dargestellte Tabelle A.3 angefertigt. Die drei nicht umgesetzten Anforderungen sind:

- *Neues Objekt registrieren* (FA 4)
- *Klasse umbenennen* (FA 8)
- *Projekte umbenennen* (FA 43)

Für die Registrierung der Objekte werden jedoch Hilfestellungen im MARS-Explorer geboten. Das Umbenennen von Klassen kann umgangen werden, indem der Quelltext der jeweiligen Klasse kopiert, eine neue Klasse angelegt und der kopierte Inhalt in die neue Klasse eingefügt wird. Lediglich zur Umbenennung eines Projekts existiert innerhalb des MARS-Explorer aktuell noch keine alternative Möglichkeit. Dies ist jedoch eine Funktion, die im Nachhinein – ohne Einschränkungen der Architektur oder Entwurfsmuster – implementiert werden kann.

7.2 Nicht-funktionale Anforderungen

Innerhalb dieses Abschnitts werden die nicht-funktionalen Anforderungen mithilfe der in Kapitel 4.4 aufgestellten Test-Kriterien auf ihre Umsetzung überprüft. Jedes Unterkapitel behandelt eine Anforderung. Im Rahmen dessen werden die Test-Kriterien der jeweiligen Anforderung genannt. Bei erfolgreicher Umsetzung werden die Kriterien mit einem ✓ und bei nicht erfolgreicher Umsetzung mit einem ✗ markiert. Des Weiteren wird erläutert, warum die Test-Kriterien der jeweiligen Anforderung als umgesetzt bzw. nicht umgesetzt bewertet wurden.

7.2.1 Funktionale Tauglichkeit

- ✗ Es werden alle funktionalen Anforderungen aus Kapitel 4.3 umgesetzt
- ✓ Es existieren Tests, die Teile der Funktionalitäten testen
- ✓ Es werden Nutzertests durchgeführt, die ebenfalls Aufschluss über die korrekte Funktionsweise geben

Wie zuvor bereits erwähnt, wurden die meisten Funktionalitäten implementiert. Zu zwei von drei unerfüllten Anforderungen existiert ein Workaround und die verbleibende Anforderung steht der korrekten Funktionsweise des MARS-Explorer nicht im Weg. Um die korrekte Funktionalität sicherzustellen, wurden Tests angefertigt, welche vor allem die zuvor erläuterten *Reducer* testen, da sie einen Großteil der Logik beinhalten. Des Weiteren wurden Nutzertests sowohl mit den Lehrenden als auch mit einer Versuchsgruppe durchgeführt (siehe Kapitel 8), um die korrekte Funktionsweise sicherzustellen.

7.2.2 Kompatibilität

- ✓ Externe Systeme, die im Zusammenhang mit dem MARS-Explorer stehen, werden von der Anwendung auf ihre Existenz geprüft
- ✓ Sollten Abhängigkeiten nicht installiert oder fehlerhaft sein, sodass sie nicht von der Anwendung verwendet werden können, wird der Benutzer darauf hingewiesen

Um Fehler mit externen Systemen möglichst frühzeitig auszuschließen, wird zum Start des MARS-Explorer das System des Modellierenden auf eine Installation des .NET-SDK überprüft. Kann keine Installation gefunden werden, wird der Nutzer auf die Abhängigkeit hingewiesen und um eine Installation gebeten. Die Anwendung wird in diesem Fall wieder geschlossen. Neben dem .NET-SDK verwendet der MARS-Explorer das externe System *Omnisharp-Roslyn* als Languageserver. Dieses wird von der Anwendung mit ausgeliefert und dementsprechend während der Entwicklung überprüft.

7.2.3 Performanz

- ✓ Es werden Nutzertests auf verschiedenen Systemen durchgeführt, um eine ausreichende Performanz zu verifizieren
- ✓ Es werden bei den Nutzertests keine Auffälligkeiten im Bezug auf die Performanz festgestellt

Die Performanz wurde sowohl durch Tests auf verschiedenen Systemen des Autors als auch auf Systemen der Lehrenden sichergestellt. Während der Versuchsdurchführung der Testgruppe gab es keine Auffälligkeiten, die vom Autor beobachtet oder von den Testern gemeldet worden sind.

Des Weiteren wurden Maßnahmen ergriffen, wodurch die Performanz sichergestellt wird. Ein Beispiel für eine dieser Maßnahmen ist die Darstellung des Simulationsfortschritts innerhalb der Analyse. Während der Simulation werden vom MARS-Framework große Datenmengen übermittelt, die sich je nach Größe der Simulation in ihrem Umfang stark unterscheiden können. An dieser Stelle wird vom MARS-Explorer nach relevanten Daten gefiltert. Die Daten sind relevant, wenn folgende Bedingung zutrifft: $(receivedProgress - lastProgress) \geq 1$. Trifft die Bedingung nicht zu, werden die Daten vom MARS-Explorer nicht weiter verarbeitet. Dies sorgt für eine Reduzierung der Datenmenge um ein Vielfaches und somit für eine flüssige und sinnvolle Darstellung der Datenpunkte. Aufgrund dieser Unvollständigkeit der Daten ist eine solche Reduzierung jedoch nicht für detaillierte Forschungsfragen geeignet.

Trotz der getroffenen Maßnahmen kann eine Simulation mit hinreichend großer Anzahl an Agenten zu Performanz-Problemen führen. Dies ist aber nicht unbedingt der Implementierung des MARS-Explorer geschuldet, sondern der Ressourcen-intensiven Natur von MAS und der Performanz des MARS-Framework, welches als Ausführungsumgebung verwendet wird.

7.2.4 Usability

- ✓ Es werden Design-Entwürfe erstellt, die die Usability berücksichtigen
- ✓ Es werden während der Entwicklung Nutzertests mithilfe der Forschungsgruppe durchgeführt, um die Usability kontinuierlich sicherzustellen und ggf. Änderungen rechtzeitig durchführen zu können

- ✓ Es wird eine Umfrage an den Nutzern durchgeführt, wovon ein Teilaspekt der Usability dediziert wird

Alle oben genannten Punkte wurden ausnahmslos eingehalten. Zu Anfang wurden Design-Entwürfe erstellt, die kontinuierlich in Absprache mit Mitgliedern der MARS-Forschungsgruppe verbessert worden sind. Die Mockups wurden in Kapitel 4.7 vorgestellt. Wie bereits während der Performanz erwähnt, wurde die Anwendung mehrere Male von Mitgliedern der Forschungsgruppe getestet. Verbesserungsmöglichkeiten, die während der Tests aufgefallen sind, wurden entweder in die Mockups oder in einer späteren Phase der Entwicklung direkt in die Anwendung eingebaut.

Die Usability wurde außerdem durch Orientierung an den in Kapitel 2.2 und 3.3 ausgearbeiteten Aspekten verbessert. Dies beinhaltet bspw. den Grundaufbau der Navigation, das Ausblenden von irrelevanten Aktionen für die aktuelle Aufgabe, die Unterstützung der üblichen Tastenkombinationen, die Verwendung von Bestätigungs-Dialogen bei kritischen Funktionen oder die Einführung von Beispielen innerhalb der Anwendung. Eine weitere Maßnahme zur Sicherstellung der Usability ist der durchgeführte Nutzertest, welcher in Kapitel 8 behandelt wird.

7.2.5 Sicherheit

- ✓ Sicherheitsaspekte werden während des Architekturdesigns berücksichtigt
- ✓ Es werden nachweisliche Maßnahmen getroffen, um den Zugriff auf die Daten des Anwenders zu verhindern

Die Erklärung aus Kapitel 5.3 zeigt, dass das Thema der Sicherheit ausschlaggebend für die grundlegende Architektur des MARS-Explorer war. Um in jedem Fall zu verhindern, dass Angreifer Zugriff auf Schnittstellen des Betriebssystems des Anwenders bekommen, wurde das Frontend des MARS-Explorer von diesen Schnittstellen isoliert. Die Kommunikation, um auf Funktionen des Betriebssystems zugreifen zu können, findet lediglich über vordefinierte und kontrollierte Kanäle statt.

7.2.6 Wartbarkeit

- ✓ Es werden Werkzeuge zur statischen Code-Analyse eingesetzt

- ✓ Die Architektur der Anwendung folgt einem detailliert beschriebenen Konzept und/oder bekannten Entwurfsmustern

Um die Wartbarkeit der Anwendung in Zukunft zu gewährleisten, wurden verschiedene Maßnahmen getroffen. Zum einen wurde bei der Einbindung von Technologien bzw. Frameworks auf deren Langlebigkeit geachtet. Zum anderen wurde die umgesetzte Architektur und die verwendeten Entwurfsmuster in Kapitel 5 ausführlich erläutert und begründet. So wurde auf bekannte Entwurfsmuster wie bspw. MVP und auch auf Verhaltensmuster wie *Observer* und *Mediator* aufgebaut. Semantische, syntaktische oder Fehler in der Formatierung wurden durch die Einführung von der Programmiersprache *TypeScript* und *ESLint*, einem Werkzeug zur statischen Code-Analyse, vermieden.

7.2.7 Portabilität

- ✓ Es werden mögliche Technologien abgewägt und der Faktor der Langlebigkeit mit hohem Gewicht bewertet
- ✓ Es existiert eine installierbare Datei für die aktuellste Version von Windows
- ✓ Es existiert eine installierbare Datei für mind. eine aktuelle Linux-Distribution
- ✓ Es existiert eine installierbare Datei für die aktuellste Version von MacOS

Um eine plattformübergreifende Anwendung zu entwickeln, wurde in Kapitel 6.1.1 eine Abwägung verschiedener Framework durchgeführt. Aufgrund der Langlebigkeit, der Vielzahl an existierenden Ressourcen und der Vielzahl der Konsumenten hat sich der Autor für das JavaScript-Framework *Electron* entschieden. Mithilfe von Electron können diverse Versionen für die verschiedenen gängigen Betriebssysteme gebaut werden. Für den Nutzertest konnten Installationen für die Betriebssysteme MacOS, Windows, sowie Redhead- oder Debian-basierte Linux-Distributionen bereitgestellt werden.

8 Evaluation des MARS-Explorer

Um die zu Beginn der Arbeit aufgestellte Forschungsfrage zu beantworten, wird in diesem Kapitel eine umfassende Evaluation des MARS-Explorer durchgeführt. Dazu werden die verwendeten Methoden zur Evaluation in einem ersten Schritt erklärt und der Versuchsaufbau beschrieben. Die aus dem Versuch resultierenden Ergebnisse werden mithilfe von Diagrammen vorgestellt, woraufhin sie im Anschluss diskutiert werden. Zu der Diskussion gehört zum einen die Interpretation der Ergebnisse, sowie die damit einhergehende Untersuchung der aufgestellten Hypothesen. Am Ende dieses Kapitels werden die Ergebnisse zusammengefasst und auf Basis dessen die Forschungsfrage dieser Arbeit beantwortet.

8.1 Evaluationsrahmen

Innerhalb dieses Abschnitts wird das grundlegende Konzept, welches für die vorliegende Evaluation verwendet worden ist, erläutert und der allgemeine Aufbau, sowie die Durchführung des Versuchs erklärt.

8.1.1 Konzept

Diese Evaluation stützt sich auf die Arbeit von Alomari u. a. (2020) in welcher ein Evaluations-Framework für Cyberlearning Environments vorgestellt wurde. Zur Definition des Begriffs Cyberlearning greifen die Autoren auf die folgende Formulierung von Borgman u. a. zurück:

Definition 8.1.1 (Cyberlearning nach Borgman u. a. (2008))**.** [. . .] the use of networked computing and communications technologies to support learning.

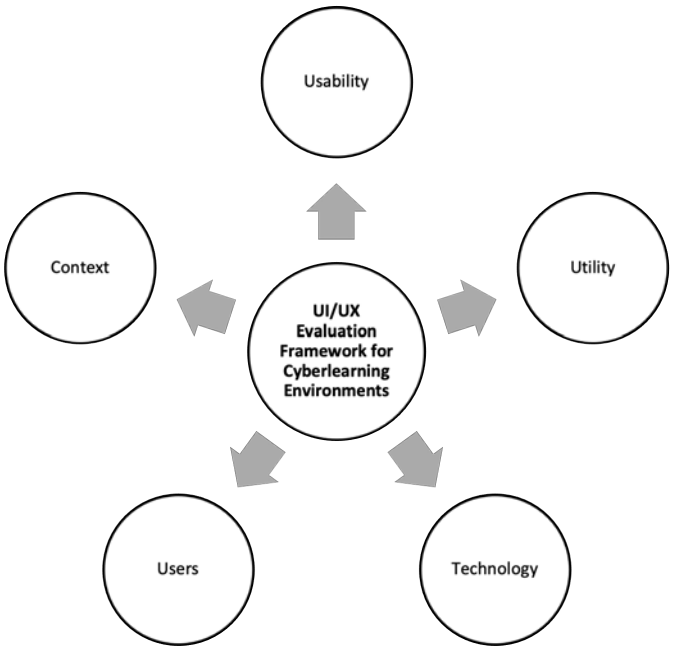


Abbildung 8.1: Aspekte des Evaluation-Framework in Anlehnung an Alomari u. a. (2020, S. 5)

Das Framework definiert fünf Aspekte, die für die Evaluation der Nützlichkeit einer Lernanwendung von Bedeutung sind. Diese sind in Abbildung 8.1 dargestellt. Im Folgenden werden diese Aspekte erklärt und ihnen jeweils die spezifischen Attribute dieser Arbeit zugeordnet.

Context

Der Aspekt *Context* beschäftigt sich mit der Definition der Fachdomäne. Die Fachdomäne des MARS-Explorer wurde bereits ausgiebig im Kapitel *Grundlagen* und *Verwandte Arbeiten* behandelt. Es sollte an dieser Stelle dennoch erwähnt werden, dass die Evaluation im Rahmen eines Versuchs innerhalb der Lehrveranstaltungen *Artificial Intelligence & Software Agents* und *Artificial Intelligence & Software Agents – The Project* an der HAW Hamburg durchgeführt wurde.

Users

Die *Users* bzw. Teilnehmer dieses Versuchs sind Studierende zwischen dem zweiten und dem fünften Semester im Bachelor-Studiengang *Informatik* an der HAW Hamburg. Dementsprechend verfügen die Teilnehmer i.d.R. über Grundkenntnisse der Programmierung, da innerhalb des ersten und zweiten Semesters die Grundlage der Programmierung bzw. der Objekt-Orientierten Programmierung gelehrt werden. Bei den im *Context* genannten Modulen handelt es sich um eine Einführung in das Thema von AI und MAS, weshalb die Teilnehmer voraussichtlich über keine oder nur wenige Kenntnisse verfügen. Da die beiden Kurse zum Teil von den gleichen Studierenden besucht worden sind, betrug die Anzahl der individuellen Teilnehmer 18.

Technology

Der Aspekt *Technology* beschreibt die Lehrstrategien, die von der technischen Anwendung verfolgt werden. Eine Erläuterung der Strategien hat bereits im Kapitel *Lehrstrategien und -komponenten* stattgefunden. Es wird die Strategie verfolgt, den individuellen Lernprozess des Modellierenden mithilfe der Anwendung zu unterstützen. Dabei soll der Lehrende jedoch nicht ersetzt, sondern lediglich ergänzt werden. Es kommen neben der Modellierung und Simulation verschiedene Lehrkomponenten wie Ressourcen in Form von Vorlesungsunterlagen oder der Dokumentation des Framework zum Einsatz.

Utility und Usability

Um die Nützlichkeit der Anwendung mithilfe des Framework bewerten zu können, gilt es, Methoden bzw. Instrumente zur Bemessung der Aspekte *Utility* und *Usability* einzusetzen. In Summe bilden beide Aspekte die Nützlichkeit (*Usefulness*) der Anwendung. Die *Utility* beschäftigt sich mit der Frage, ob die vom Benutzer benötigten Funktionen vom System angeboten werden. Nach Nielsen (1994b, S. 25) besitzen Lernsysteme im Unterschied zu Unterhaltungsanwendungen eine äußerst hohe Utility. Die Grundlagen und der Einflussfaktor der *Usability* auf den Lernerfolg wurden bereits umfassend in Kapitel 2.2 bzw. 3.3 erläutert. Auch Alomari u. a. (2020) und Beale und Sharples (2002) sind der Auffassung, dass die Usability einen entscheidenen Teil zum Lernerfolg eines System beiträgt. Die Methoden zur Bemessung beider Aspekte werden im folgenden Abschnitt vorgestellt.

8.1.2 Aufbau

Die Evaluation wird im Rahmen eines Nutzertests durch die zuvor erwähnte Benutzergruppe durchgeführt. Die Studierenden bekommen neben der installierbaren Datei des MARS-Explorer eine Liste mit Aufgaben ausgehändigt, welche der letzten Seite des Handouts im Anhang A.1.1 entnommen werden kann. Diese ist dazu gedacht, den Studierenden den gesamten Funktionsumfang der Anwendung aufzuzeigen. Nach der Bearbeitung der gestellten Aufgaben, sind die Studierenden dazu angeleitet eine Umfrage durchzuführen, die darauf abzielt, die *Usefulness* der Anwendung zu bemessen. Die Fragen bzw. Aufgaben wurden innerhalb des Fragebogens in die Kategorien *Utility* und *Usability* aufgeteilt. Die Ergebnisse der Umfrage werden dazu verwendet, die aufgestellten Hypothesen verifizieren bzw. falsifizieren zu können. Die Umfrage kann im Anhang A.1.2 eingesehen werden.

Um neben fachlichen und eindeutig zu beantwortenden Fragen auch subjektive Einschätzungen der Tester erfassen zu können, wird die folgende fünf-stufige Likert-Skala verwendet:

- Strongly agree
- Agree
- Neither agree nor disagree
- Disagree
- Strongly disagree

Im Folgenden werden die Methoden zur Bemessung der Utility und der Usability näher erläutert.

Bewertung der Utility

Um die Utility des Systems zu evaluieren, wurden die fachbezogenen Hypothesen herangezogen. Aus ihnen wurden Fragen abgeleitet, die zur Untersuchung der jeweiligen Hypothesen verwendet wurden. Die abgeleiteten Fragen sind in der Tabelle 8.1 zusammen mit der jeweiligen Hypothese dargestellt.

Tabelle 8.1: Abbildung der Hypothesen auf Fragen für den Nutzertest

Hypothese	Die Verwendung einer Lernplattform fördert das Verständnis von den verschiedenen Objekten (Agent, Layer, Entität). (H1)
Fragestellung	• What are the core elements of a MARS framework model? (Mehrfach-Auswahl aus: Objects of user-defined class hierarchy, Agents, Maps, Layers, Entities, Environments, Lists) • The system helps me to better understand the different objects that exist within a simulation. (Likert-Skala)
Hypothese	Die Verwendung einer Lernplattform fördert das Verständnis von den verschiedenen Phasen der Multi-Agenten-Simulation (Modellierung, Konfiguration, Simulation, Analyse). (H2)
Fragestellung	• Put the following phases within a MARS model in the order in which they are usually performed. (Sortierung der Phasen: Modeling, Configuration, Simulation, Analysis; anfängliche Phase zuerst, letzte Phase zuletzt) • The system helps me to improve my understanding of the different phases within a project. (Likert-Skala)
Hypothese	Die Verwendung einer Lernplattform führt dazu, dass Fehler innerhalb der Modellierung oder Parametrisierung vom Benutzer schneller identifiziert und behoben werden. (H3)

Fragestellung

- In the configuration, is it possible to ... ? (Auswahl zwischen True und False; *Possible* bedeutet in diesem Kontext, dass eine Konfiguration theoretisch gespeichert und eine Simulation fehlerfrei ausgeführt werden kann)
  - ... use steps and time ranges at the same time?
  - ... specify a startPoint but no endPoint?
  - ... specify a negative value for deltaT?
  - ... specify a negative value for step?
  - ... set the number of agents?
  - ... set the number of layers?
  - ... initialise agents with an input file?
  - ... initialise entities with an input file?
- The system helps me to identify my errors in the modeling process. (Likert-Skala)
- The system helps me to identify my errors within the configuration. (Likert-Skala)
- The system helps me to better understand the API of the MARS framework and thus make fewer mistakes. (Likert-Skala)

Hypothese Die Verwendung einer Lernplattform führt dazu, dass seltener Experten zur Unterstützung benötigt werden. (H4)

Fragestellung

- As a result of using the system, I need less support from the experts. (Likert-Skala)

Bewertung der Usability

Da die Usability für den Lernerfolg bei Nutzung einer Lernanwendung ausschlaggebend ist, soll mithilfe der hier verwendeten Methoden ein Meinungsbild geschaffen werden. Zu diesem Zweck wurden zwei Instrumente verwendet: Zum einen die von Lewis (2002) entworfene Umfrage und zum anderen die von Nielsen formulierten Heuristiken. Beide Aussagemengen werden von den Teilnehmenden mithilfe der fünf-stufigen Likert-Skala bewertet.

Um einen allgemeinen Eindruck über die Zufriedenheit der Benutzer mit der Anwendung zu erhalten, wurde die 3. Version des von Lewis (2002) aufgestellten *Computer System Usability Questionnaire* (CSUQ), wie im weiteren Verlauf des Kapitels erläutert, in etwas gekürzter Form herangezogen. Nach Lewis sollten den Teilnehmern die Fragen im Anschluss an die Bearbeitung von beispielhaften Aufgaben gestellt werden. Konkret wurden folgende Aussagen verwendet:

1. Overall, I am satisfied with how easy it is to use the System.
2. I feel comfortable using the System.
3. It was easy to learn to use the System.
4. I believe I became productive quickly using the System.
5. It is easy to find the information I need.
6. The interface of the System is pleasant.
7. I like using the interface of this System.
8. I would recommend the System to fellow students.
9. Overall, I am satisfied with the Application.

Clarke u. a. (2014) evaluierten ebenfalls eine Lernanwendung mithilfe des CSUQ. Von ihnen wurde zusätzlich noch die Aussage 8 (*I would recommend the System to fellow students*) ergänzt, die ursprünglich nicht Teil der Umfrage von Lewis war.

Eine andere weit verbreitete Methode zur Evaluation ist die heuristische Evaluation, welche von Nielsen und Molich (1990) formuliert worden ist. Mit ihr soll ein Großteil der

häufigsten Fehler eines Designs identifiziert werden können, indem eine Reihe von Faktoren und dazugehörige Leitfragen aufgestellt werden. Im Folgenden werden die Heuristiken von Nielsen kurz erläutert:

Visibility of system status Der Benutzer sollte zur jederzeit wissen, in welchem Zustand sich das System gerade befindet und in einem akzeptablen Zeitrahmen über neue Zustände informiert werden.

Match between system and the real world Das verwendete Vokabular des Systems sollte sich am Wissen des Benutzers orientieren und dementsprechend typische Ausdrücke verwenden.

User control and freedom Für den Benutzer sollte jederzeit eine Möglichkeit existieren, zum Startbildschirm der Anwendung zurückzukehren. Außerdem sollten sowohl *Undo*- als auch *Redo*-Aktionen unterstützt werden.

Consistency and standards Der Benutzer sollte nicht überlegen müssen, ob Begriffe, die innerhalb der Anwendung verwendet werden, Synonyme zu zuvor verwendeten Wörtern sind. Es sollte eine konsistente Sprache verwendet werden, die ebenso den Standards der jeweiligen Plattform folgt.

Error prevention Anstatt dem Benutzer Fehler zu präsentieren, sollten diese frühzeitig verhindert werden. Falls dies nicht möglich ist, sollten bei Aktionen, die potentiell Fehler zur Folge haben können, Dialoge verwendet werden, um sicherzustellen, dass der Benutzer sich des möglichen Auftretens von Fehlern bewusst ist.

Recognition rather than recall Der Benutzer sollte immer genau die Informationen angezeigt bekommen, die er für seine jeweilige Aufgabe benötigt. Dabei sollte besonders darauf geachtet werden, dass der Benutzer sich keine Informationen aus vorherigen Schritten merken muss.

Flexibility and efficiency of use Um gleichwohl Anfänger als auch Experten der Anwendung zu anzusprechen, sollten fortgeschrittene Funktionen vorhanden sein, aber versteckt werden. So profitieren sowohl die Anfänger als auch die Experten von der Anwendung.

Aesthetic and minimalist design Es sollten nur die Informationen angezeigt werden, die für die aktuelle Aufgabe relevant sind. Werden unnötige Informationen dargestellt, konkurrieren diese mit den wichtigen Informationen um die Aufmerksamkeit des Benutzers.

Help users recognize, diagnose and recover from errors Fehlermeldungen sollten in natürlicher Sprache präzise verfasst werden, sodass der Benutzer die Gründe für den Fehler nachvollziehen und entsprechend reagieren kann.

Help and documentation Allgemein sollte eine Dokumentation vermieden werden. Falls in Ausnahmefällen aber dennoch eine notwendig ist, sollte diese einfach aufzufinden, zu durchsuchen und auf die jeweiligen Aufgaben des Benutzers zugeschnitten sein.

Diese Heuristiken wurden als Basis für die in Tabelle 8.2 formulierten Aussagen verwendet. Die Aussagen wurden von den Teilnehmer der Umfrage mithilfe einer Likert-Skala bewertet. Zwar ist diese Art der Evaluation optimalerweise mit Experten aus dem UI/UX Bereich durchzuführen (Nielsen, 1994a), jedoch liefern auch die Ergebnisse aus den Nutzertests mit Nicht-Experten einen hohen Mehrwert und können bei der Identifikation der großen Mehrheit von Fehlern innerhalb des User Interfaces helfen (Ssemugabi und de Villiers, 2007). Um Überschneidungen mit den Heuristiken von Nielsen und Molich zu vermeiden, wurden – wie oben bereits erwähnt – nicht alle Fragen des CSUQ verwendet.

8.1.3 Durchführung

Der Versuch wurde im Rahmen der Lehrveranstaltungen *Artificial Intelligence & Software Agents* und *Artificial Intelligence & Software Agents – The Project* an der HAW Hamburg durchgeführt. Zu Beginn wurde der Testgruppe eine kurze Einführung zu dem Autor selber und zu dieser Arbeit gegeben. Im Anschluss wurden die Gruppen dazu aufgefordert, sich ein Dokument herunterzuladen, welches alle Informationen zum durchzuführenden Experiment beinhaltet. Das Dokument kann im Anhang in Abbildung A.1.1 eingesehen werden. Da nicht alle Teilnehmenden deutsch gesprochen haben, wurde das ausgeteilte Informationsblatt auf englisch formuliert.

Das Dokument beinhaltet eine Anleitung zur Durchführung des Experiments. Wie bereits mehrfach erwähnt, basiert das MARS-Framework auf .NET und benötigt zur Entwicklung von Modellen das .NET-SDK. Die dazugehörige Installationsanleitung wurde innerhalb des Dokuments verlinkt. Um den MARS-Explorer zu installieren, wurde für jedes gängige Betriebssystem ein Link bereitgestellt, der die jeweilige Installationsdatei beinhaltete. Im Rahmen der Lehrveranstaltungen wurde ein Zeitraum von ca. 60 bis 90 Minuten zur Verfügung gestellt. In dieser Zeit wurde das Experiment – je nach Verfügbarkeit von Computern – in Einzel- oder Partnerarbeit durchgeführt. Falls es zu schwerwiegenden Problemen mit der Anwendung auf den Geräten der Testenden kam,

Tabelle 8.2: Abbildung der Heuristiken von Nielsen und Molich (1990) auf konkrete Aussagen zur Bewertung der Usability

Heuristik	**Abgeleitete Aussagen**
Visibility of system status	The system keeps me informed about its current status.
	I receive feedback from the system within a reasonable period of time.
Match between system and the real world	All texts of the system are worded in an understandable way.
User control and freedom	I can cancel any of my actions at any time.
	At any time I know how to return to the home page.
Consistency and standards	The system uses common vocabulary for applications.
Error prevention	No unhandled errors have occurred while using the system.
Recognition rather than recall	All information I need for my tasks is visible.
	The system helps me to understand all its features.
Flexibility and efficiency of use	The most common key combinations have the expected effect (e.g. CMD/STR + S = Save).
Aesthetic and minimalist design	The user interface does not show more information than necessary.
Help users recognize, diagnose and recover from errors	If errors occur, they are phrased in an understandable way.
	When errors occur, the error messages help me understand why they occurred.

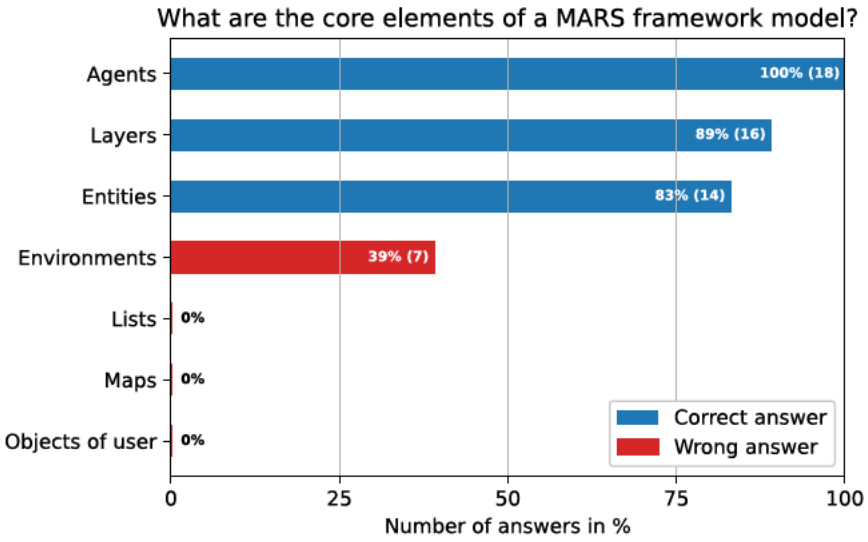


Abbildung 8.2: Ergebnisse zu den Kernelementen des MARS-Framework bzgl. der Hypothese 1

wurde ein alternatives Gerät zur Verfügung gestellt, sodass die Anwendung ausprobiert, begutachtet und die Umfrage durchgeführt werden konnte.

8.2 Vorstellung der Ergebnisse

Die Ergebnisse des Nutzertests werden innerhalb dieses Kapitels vorgestellt. Gruppiert werden die Ergebnisse nach ihrem Bezug zu den Hypothesen und der Usability. Eine ausführliche Interpretation der Ergebnis findet in Kapitel 8.3 statt.

8.2.1 Ergebnisse bzgl. der Hypothesen

Die Ergebnisse, die für die Untersuchung der Hypothesen verwendet werden, beinhalten sowohl Antworten auf fachbezogene Fragen als auch Meinungen der Teilnehmenden. Zur Darstellung der Diagramme wurde die Farbe Blau für korrekte und Rot für inkorrekte Antworten auf fachbezogene Fragen verwendet.

Tabelle 8.3: Anteile und absolute Anzahl der Antworten in einer Likert-Skala zur Aussage *The system helps me to better understand the different objects that exist within a simulation.*

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
0%	0%	11% (2)	61% (11)	28% (5)

Hypothese 1

Die erste Frage zu den Kernelemente des Framework wurde von dem größten Teil der Teilnehmer korrekt beantwortet, wie die Abbildung 8.2 zeigt. Die *Agents* wurden von allen Teilnehmern als Kernelemente des MARS-Framework identifiziert. Layer und Entities hingegen wurden von 89% und 83% der Teilnehmenden richtig erkannt. Als einzige falsche Antwort wurde dagegen *Environments* mit 39% genannt.

Ebenfalls der Hypothese 1 zugeordnet, ist die Aussage: *The system helps me to better understand the different objects that exist within a simulation.* Die Meinungen der Teilnehmenden sind in Tabelle 8.3 dargestellt. 16 Personen haben der Aussage mindestens zugestimmt, die verbleibenden zwei Teilnehmenden dagegen eine neutrale Aussage getroffen.

Hypothese 2

Um Hypothese 2 zu beantworten, wurden die Teilnehmenden dazu aufgefordert, die vier folgenden Phasen absteigend in die richtige Reihenfolge zu bringen, wie sie durchlaufen werden.

1. *Modeling*
2. *Configuration*
3. *Simulation*
4. *Analysis*

Die vier Kuchendiagramme in Abbildung 8.3 zeigen die Ergebnisse dieser Aufgabe. Das Diagramm mit der Beschriftung *First Step* beschreibt, dass *Modeling* (*M*) von 72% der Teilnehmenden korrekterweise als erster Schritt angesehen wird. Abgesehen vom ersten

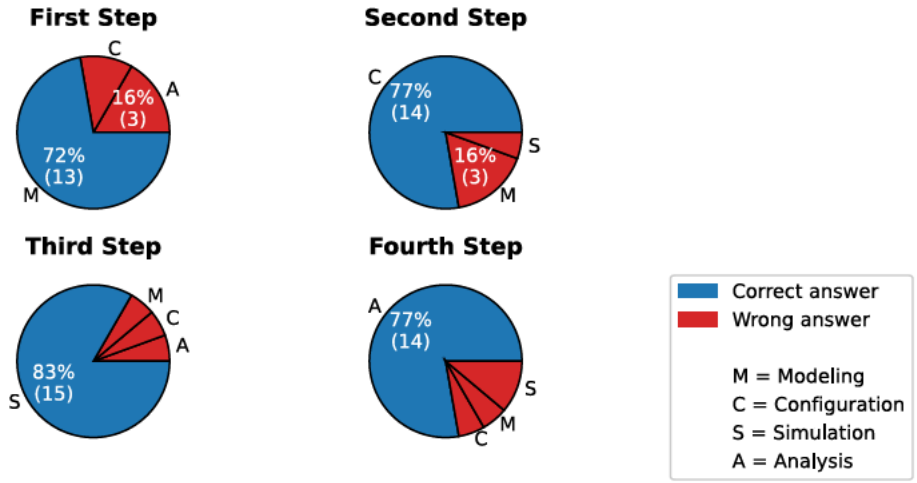


Abbildung 8.3: Ergebnisse der Anordnung der verschiedenen Phasen innerhalb des MARS-Framework bzgl. Hypothese 2

Tabelle 8.4: Anteile und absolute Anzahl der Antworten auf einer Likert-Skala zur Aussage *The system helps me to improve my understanding of the different phases within a project.*

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
0%	0%	17% (3)	50% (9)	33% (6)

Schritt wurden die Phasen von mindestens drei Viertel der Teilnehmer korrekt angeordnet. Insgesamt sind durchschnittlich 77% der Antworten korrekt. Wie die Ergebnisse in Tabelle 8.4 zeigen, stimmen 50% der Teilnehmenden der Aussage zu und 33% sogar stark zu, dass sie das System beim Verständnis der verschiedenen Phasen unterstützt.

Hypothese 3

Um zu überprüfen, inwieweit der MARS-Explorer die Teilnehmenden bei der Identifikation von Fehlern unterstützt, wurden mehrere Fragen formuliert, die von den Teilnehmenden mit Wahr (*True*) oder Falsch (*False*) beantwortet werden sollten. Die Ergebnisse in Abbildung 8.4 offenbaren mit einem Mittelwert der korrekten Antworten von ca. 70% die meisten Verständnisprobleme. Besonders auffällig sind die Antworten zu den Fragestellungen bzgl. der Eingabedatei einer Entity (*initialise entities with an input file?*) mit

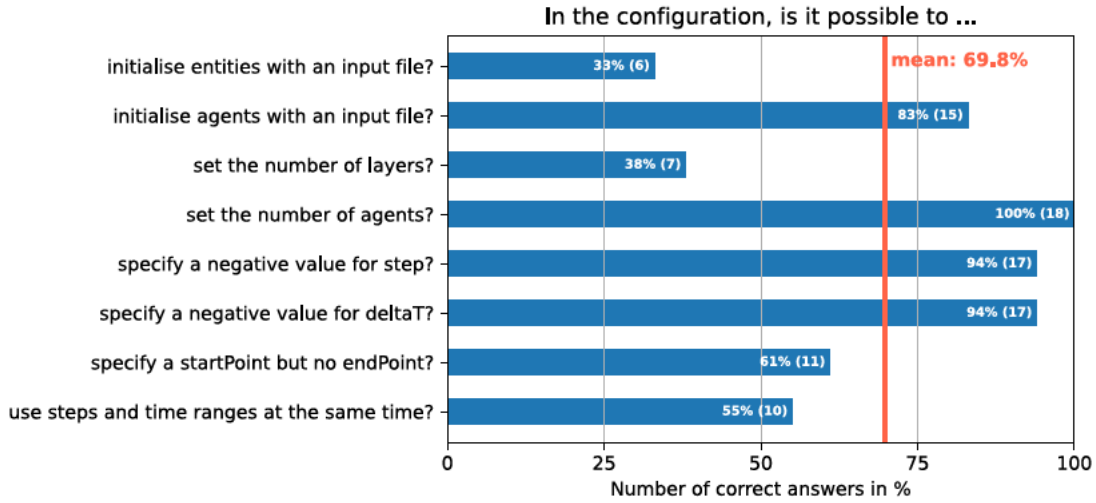


Abbildung 8.4: Ergebnisse zu Wahr- und Falsch-Aussagen bzgl. der Konfiguration einer Simulation

Tabelle 8.5: Anteile und absolute Anzahl der Antworten in einer Likert-Skala zur Aussage *As a result of using the system, I need less support from the experts.*

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
0%	0%	33% (6)	44% (8)	22% (4)

33% richtigen Antworten, sowie der Anzahl der Layer (*set number of layers?*) mit einem Anteil von 37% an korrekten Antworten.

Die Ergebnisse in Abbildung 8.5 zeigen, dass den Aussagen bzgl. der Fehleridentifikation mit einem Durchschnitt von ca. 73% mindestens zugestimmt wurde. Auffällig ist in diesem Fall der hohe Anteil an neutralen Stimmen in Bezug auf die Identifikation von Fehlern im Modellierungs- bzw. Konfigurations-Prozess.

Hypothese 4

Die Tabelle 8.5 zeigt die Meinungen der Teilnehmenden gegenüber der Notwendigkeit von Experten. Mit einem Anteil von einem Drittel wurde die Aussage *As a result of using the system, I need less support from the experts* neutral bewertet. 44% hingegen stimmen der Aussage zu, während 22% der Aussage sogar stark zustimmen.

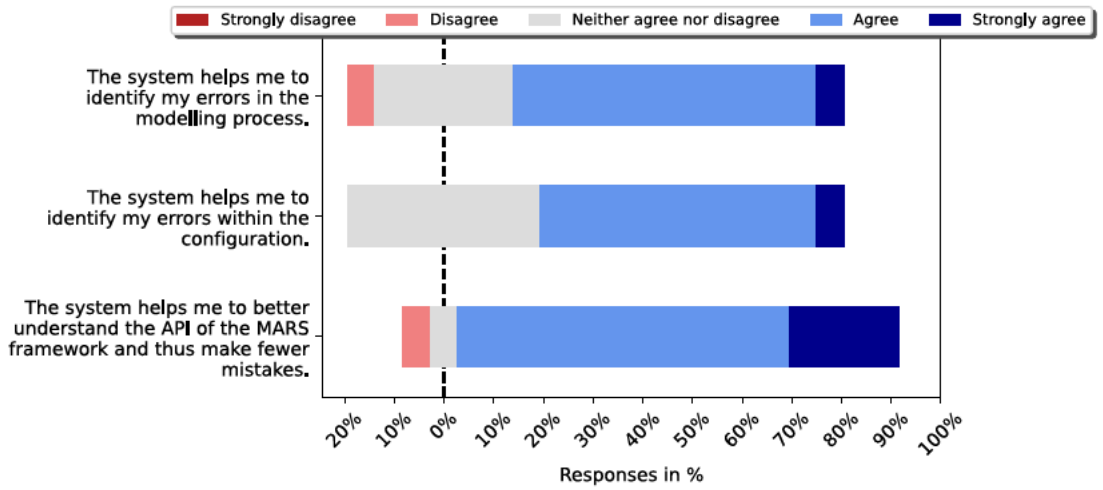


Abbildung 8.5: Ergebnisse zu den Aussagen bzgl. der Hypothese 3

8.2.2 Aussagen über die Utility

Wie die Meinungen zu den Aussagen des CSUQ in Abbildung 8.6 zeigen, wurde die Usability des MARS-Explorer von den Teilnehmern überwiegend positiv bewertet. Auffällige Aussagen sind die schnelle Erhöhung der Produktivität (*I believe I became productive quickly using the System*) mit einem Anteil an neutralen Aussagen von 33%, sowie das einfache Finden von Informationen (*It is easy to find the information I need*) mit einem Widerspruch und einem Drittel an neutralen Stimmen.

In Abbildung 8.7 ist das Feedback der Teilnehmer zu verschiedenen Aspekten der Usability-Heuristiken von Nielsen und Molich (1990) zu sehen. Neben durchaus positiven Rückmeldungen zeigen sich hier verschiedene Faktoren, die genauer untersucht werden müssen.

Zu den auffälligen Aussagen gehören bspw. die Statusanzeige (*The system keeps me informed about its current status*) und die Performanz (*I receive feedback from the system within a reasonable period of time*), die von ca. 30% bis 40% der Teilnehmer als neutral bewertet worden ist. Außerdem erscheint der Anteil an neutralen und widersprechenden Stimmen, bzgl. der Aussage zu den unbehandelten Fehlern (*No unhandled errors have occurred while using the system*), mit 44% relativ hoch. Hier ist auffällig, dass fast die Hälfte der Teilnehmer der Aussage neutral gegenüberstehen. Weitere Auffälligkeiten zeigen sich ebenfalls in den beiden letzten Aussagen zum Verständnis der auftretenden Fehler. Auch hier haben sich ca. 40% bzw. 50% für eine neutrale Aussage entschieden.

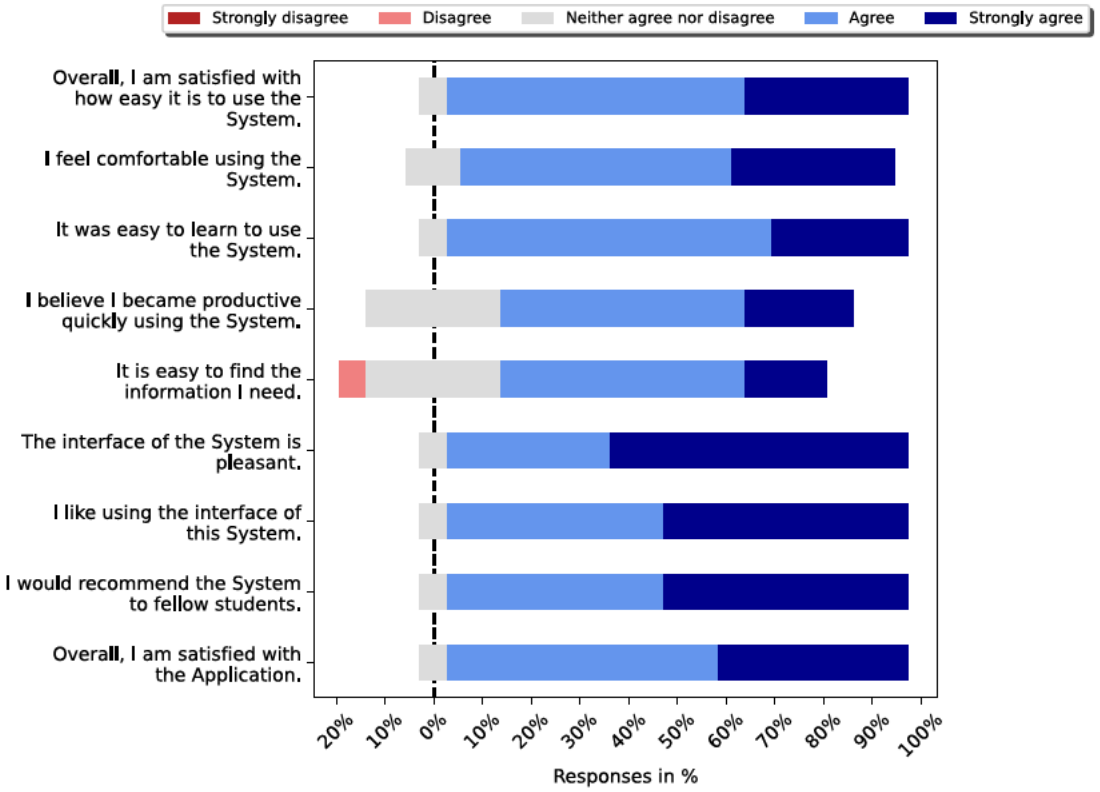


Abbildung 8.6: Feedback der Teilnehmer zur allgemeinen Zufriedenheit mit dem System in Form einer Likert-Skala

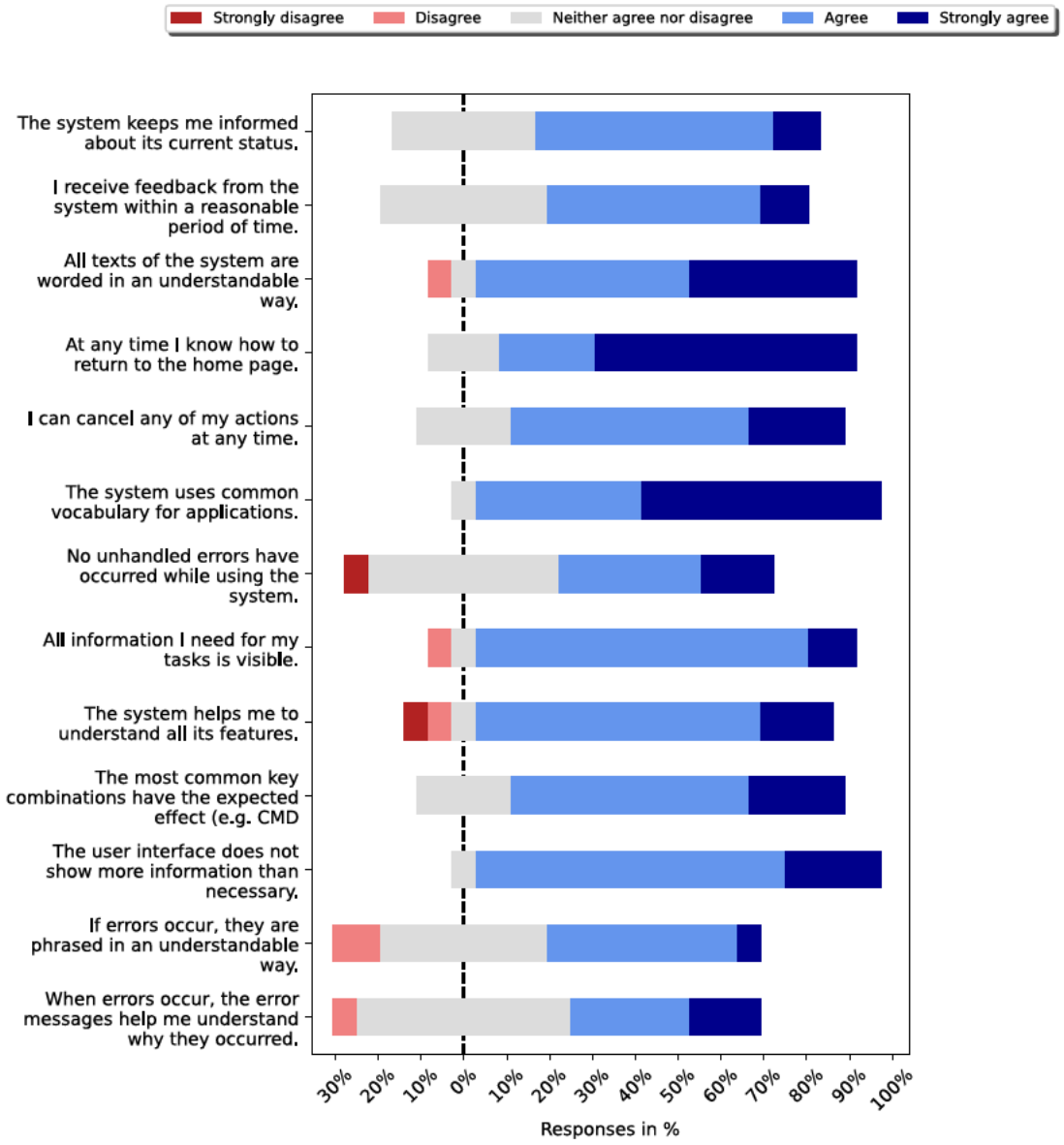


Abbildung 8.7: Feedback der Teilnehmer zu verschiedenen Aspekten der Usability des Systems in Form einer Likert-Skala

8.3 Diskussion

Zu Beginn dieses Kapitels werden die Ergebnisse des Nutzertests interpretiert und die aufgestellten Hypothesen daraufhin entweder verifiziert oder falsifiziert. Im Anschluss wird diese Untersuchung in Hinblick auf die Validität kritisch reflektiert.

8.3.1 Interpretation der Ergebnisse

Hypothese 1

Die Aufgaben zu den Kernelementen des MARS-Modells wurden von dem Großteil der Teilnehmenden korrekt beantwortet. Lediglich die *Environments* wurden von sieben Teilnehmern fälschlicherweise als ein Kernelement genannt.

Innerhalb des Framework existiert das Konzept der *Environments*. Dieses wird dazu verwendet, um Agenten oder Entitäten innerhalb eines Objekts anzuordnen. Für die Umsetzung werden jedoch nur vorgefertigte Klassen bereitgestellt, die zur Ergänzung der Funktionalitäten verwendet werden können und über keine eigenen Ein- oder Ausgabewerte innerhalb einer Simulation verfügen. Aus diesem Grund zählt es nicht zu den Kernelementen eines MARS-Modells. Des Weiteren kann ein Teil der inkorrekten Antworten evtl. der semantischen Ähnlichkeit zum Begriff *Layer* zugeschrieben werden, welcher im weiten Sinne ebenfalls die Umwelt der Agenten beschreibt.

Dass *Entities* von den drei Kernelementen am wenigstens genannt worden ist, könnte daran liegen, dass die in der Anwendung beinhalteten Beispielmodelle keine Entity-Elemente verwenden. Außerdem wird eine *Entity* im Vergleich zu *Layer* und *Agent* seltener verwendet.

Da der Großteil der Teilnehmer die Objekte eines MARS-Modell benennen konnte und der Aussage über das geförderte Verständnis gegenüber der verschiedenen Objekte mit einer Mehrheit von 89% mindestens zugestimmt wurde, kann die aufgestellte Hypothese 1 als vorläufig bestätigt betrachtet werden: Die Verwendung einer Lernplattform fördert das Verständnis von den verschiedenen Objekten (Agent, Layer, Entität).

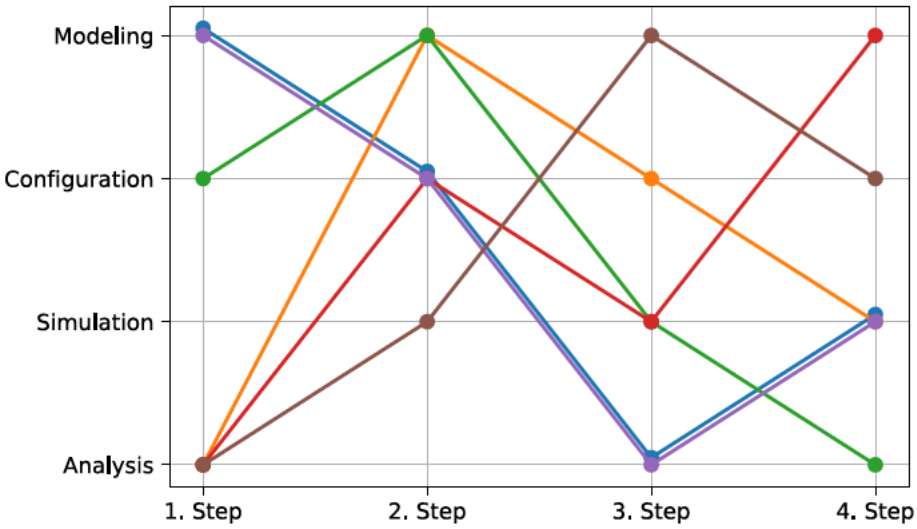


Abbildung 8.8: Detaillierte Darstellung der inkorrekten Antworten zur Anordnung der verschiedenen Phasen innerhalb des MARS-Framework in der jede Linie für die Antwort eines Teilnehmers steht

Hypothese 2

Aus den Ergebnissen bzgl. der Reihenfolge der verschiedenen Phasen lässt sich schließen, dass der Großteil der Studierenden ein Verständnis für die korrekte Vorgehensweise innerhalb des MARS-Framework entwickelt hat. Die größten Unsicherheiten gab es bei der Zuordnung von Schritt 1. Zwar haben 72% der Teilnehmer den Schritt korrekt identifiziert, doch zeigt sich auch, dass 16% die Analyse fälschlicherweise als ersten Schritt definiert haben.

Denkbar ist, dass der Schritt *Analyse* als Untersuchung der Forschungsfrage der Studierenden selbst interpretiert wurde. Basierend auf dem in Kapitel 4.3 vorgestellten Ablauf der Modellierung würde dies den vom MARS-Explorer vernachlässigten Schritten *Daten vorbereiten*, *Konzeptuelles Modell erstellen* und *Modell mit Framework-Experten besprechen* entsprechen. Diese Frage bezog sich allerdings ausschließlich auf das MARS-Framework in dem die Analyse, die Betrachtung und Interpretation der Simulationsergebnisse beschreibt.

Des Weiteren zeigen der erste und der zweite Schritt eine leichte Tendenz zur Verwechslung, da dem ersten Schritt in zwei Fällen die *Konfiguration* und dem zweiten Schritt in drei Fällen die *Modellierung* zugeordnet worden ist. Eine solche Verwechslung kann aller-

dings bei genauerer Betrachtung der Antworten in Abbildung 8.8 ausgeschlossen werden. Der Graph zeigt die Antworten der Teilnehmer, die mindestens eine inkorrekte Antwort bzgl. der Aufgabe *Put the following phases within a MARS model in the order in which they are usually performed* gegeben haben. Jede farbige Linie stellt die Antworten einer Person dar. Da die Verteilung der fehlerhaften Antworten sehr willkürlich erscheint, ist davon auszugehen, dass sie sich nicht auf Schwachstellen des MARS-Explorer zurückführen lassen. Wenn das der Fall wäre, müsste sich bei der Betrachtung der fehlerhaften Antworten ein einheitlicheres Muster ergeben.

Da auch bei diesem Aspekt die korrekte Antwort im Durchschnitt von mehr als drei Vierteln der Befragten gegeben worden ist, wird auch in diesem Fall die Hypothese 2 vorläufig bestätigt: Die Verwendung einer Lernplattform führt dazu, dass Fehler innerhalb der Modellierung oder Parametrisierung vom Benutzer schneller identifiziert und behoben werden.

Hypothese 3

Auffällige Schwierigkeiten hatten die Befragten bei den Fragen zur Konfigurationsmöglichkeit. Dies lässt sich möglicherweise auf deren kurze Formulierung zurückführen, da sie keine Definition der jeweils verwendeten Fachbegriffe beinhalteten.

Der MARS-Explorer gibt dem Nutzer bspw. die Möglichkeit, innerhalb der Konfiguration für jeden Layer ein eigenes Mapping zu erstellen. Dieses Feature wurde möglicherweise als Antwort auf die Frage *... set the number of layers?* interpretiert. Mit der Frage sollte aber ursprünglich auf die Möglichkeit eingegangen werden, ein Attribut *count* innerhalb des Mappings eines Layers setzen zu können. Im Mapping eines Agenten befindet sich ein Eingabefeld, um den *Count* – also die Anzahl der Agenten innerhalb der Simulation – anzugeben. Im Mapping der Layer existiert ein solches Eingabefeld nicht, da ein Layer i.d.R. ein *Singleton* ist, also nur einmal instanziiert werden kann. Für die Entities existiert ebenfalls keine Möglichkeit einen *count* anzugeben, da die Verwendung einer Vielzahl an Entities als fortgeschrittenes Feature angesehen wird und aus diesem Grund nicht für die Zielgruppe geeignet ist.

Die Frage bzgl. der Initialisierung der Entities mithilfe einer Eingabedatei wurde lediglich von sechs Teilnehmern korrekt beantwortet. Für die Mappings von Agenten und Layern existiert ein Eingabefeld zur Verlinkung einer entsprechenden Datei. Aus dem oben genannten Grund, existiert diese Möglichkeit für Entities nicht.

Insgesamt zeigen die Ergebnisse aus Abbildung 8.4 einen Bedarf an mehr Informationen oder Erklärungen bzgl. der Konfiguration innerhalb des MARS-Explorer. Den drei Aussagen, die sich darauf beziehen, dass die Anwendung bei der Fehleridentifikation in der *Modellierung* und *Konfiguration* hilft, wurde allerdings von der großen Mehrheit mit einem Durchschnitt von 73% mindestens zugestimmt. Dennoch konnte ein relativ großer Anteil der Studierenden zwei Aussagen weder zustimmen noch widersprechen (28% bzw. 39%). Dies weist darauf hin, dass die Teilnehmer wahrscheinlich noch kein ausgeprägtes Empfinden für Fehler innerhalb der Modellierung oder der Konfiguration entwickelt haben.

Trotz der im Vergleich zu den anderen fachlichen Fragen hohen Anzahl an falschen Antworten, wurden im Durchschnitt ca. 70% der Fragen bzgl. Hypothese 3 korrekt beantwortet. Hervorzuheben sind daraus bspw. die Fragen *... set the number of agents?*, *... specify a negative value for deltaT?* oder *... specify a negative value for step?*, die von fast allen Teilnehmern korrekt beantwortet wurden (100%, 94%, 94%). Da in Summe also sowohl die fachlichen Fragen als auch die subjektiven Einschätzungen positiv ausfallen, wird die dazugehörige Hypothese 3 ebenfalls vorläufig bestätigt: Die Verwendung einer Lernplattform führt dazu, dass Fehler innerhalb der Modellierung oder Parametrisierung vom Benutzer schneller identifiziert und behoben werden.

Hypothese 4

Mit Hypothese 4 soll überprüft werden, ob die Studierenden durch die Verwendung der Lernplattform weniger Unterstützung von Experten benötigen. Die Rückmeldungen bzgl. der Aussage *As a result of using the system, I need less support from the experts* sind mit einer einfachen Zustimmung von 44% und einer starken Zustimmung von 22% zum größten Teil positiv. Jedoch zeigt sich auch, dass dieser Aspekt von einem Drittel der Befragten nicht eingeschätzt werden konnte. Zwei Gründe sind für die neutrale Haltung denkbar:

- Die Anwendung war für die meisten der erste praktische Berührungspunkt mit dem Thema MAS bzw. dem MARS-Framework, weshalb ein Vergleichswert fehlte
- Experten mussten trotz der Verwendung der Lernplattform zur Unterstützung herangezogen werden

Da die Aussage eine Zustimmung von insgesamt zwei Drittel der Teilnehmer erhielt, wird die Hypothese 4 vorläufig bestätigt: Die Verwendung einer Lernplattform führt dazu, dass seltener Experten zur Unterstützung benötigt werden.

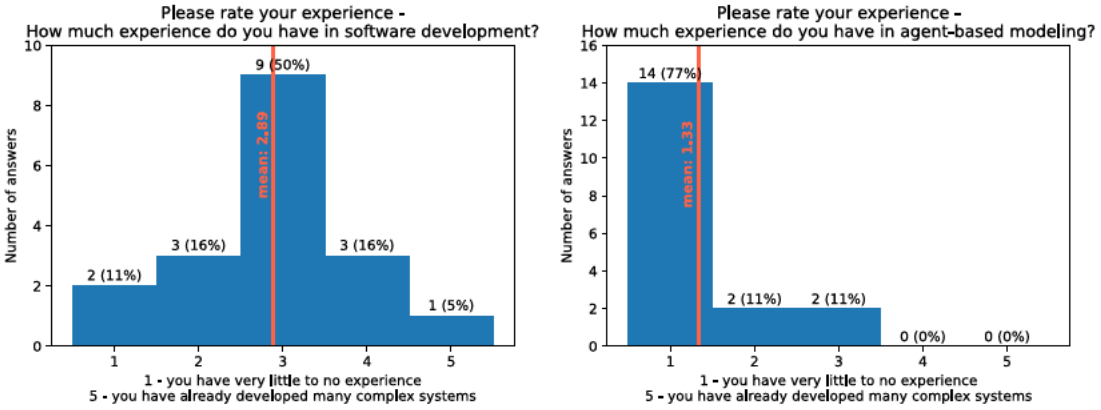
Usability

Bei Betrachtung der Ergebnisse bzgl. der Usability der Anwendung kann insgesamt ein deutlich positiver Trend beobachtet werden. Dennoch gibt es einige Aspekte, die geringere Zustimmung finden, als Andere.

Einer dieser Aspekte ist der Produktivitätsgewinn, der von ca. einem Drittel der Teilnehmenden als neutral empfunden worden ist. Dadurch, dass die Teilnehmer lediglich 60 bis 90 Minuten Zeit für die Installation, das Testen und die Bewertung hatten, bestand für die Teilnehmer vermutlich ein gewisser Zeitdruck unter dem die Aufgaben abgearbeitet wurden und das System nicht ausgiebig auf den Produktivitätsgewinn getestet werden konnte. Da diese Untersuchung für viele Teilnehmenden der erste praktische Berührungspunkt mit dem Thema MAS war, existiert für eine Vielzahl der Teilnehmer außerdem kein Vergleichswert.

Es zeigt sich aber gleichzeitig auch, dass das einfache Finden von Informationen von einem Drittel der Teilnehmenden neutral bewertet worden ist. Da die Teilnehmer im Bereich von ABM bzw. MAS noch gar keine bis wenig Expertise besitzen, erscheint eine Einschätzung über den allgemeinen Informationsgehalt wahrscheinlich schwierig. Unterstrichen wird diese Vermutung von den Ergebnissen der heuristischen Evaluation. Der Aussage *All information I need for my task is visible* wurde von insgesamt 89% der Teilnehmenden zugestimmt. Dies zeigt, dass der gegebene Informationsgehalt für die jeweilige Aufgabe dennoch ausreichend ist.

Auffällig sind auch alle Aussagen in Bezug auf Fehler innerhalb der Anwendung. Eine mögliche Erklärung für die hohe Anzahl an neutralen bzw. widersprechenden Antworten ist der Begriff von *unbehandelten* Fehlern, der innerhalb der Aussagen verwendet, aber nicht genauer definiert wurde. Gemeint waren Fehler, die den Teilnehmern unerwartet erscheinen, wie bspw. Programmabstürze oder sonstige überraschende Verhaltensweisen der Anwendung. Da das System für die Teilnehmer vollständig unbekannt war und die Teilnehmenden keine Experten für UI- oder UX-Design sind, erschien die Unterscheidung über behandelte und unbehandelte Fehler vermutlich zusätzlich schwerer. Eine weitere



(a) In der Softwareentwicklung (b) In der Agenten-basierten Modellierung

Abbildung 8.9: Erfahrungswerte der Teilnehmer

Erklärung für die hohe Anzahl an neutralen Stimmen ist, dass keine Fehler aufgetreten sind und die Aussagen deshalb neutral bewertet wurden.

Trotz der zuvor erläuterten Auffälligkeiten der Aussagen bzgl. der Usability zeichnen die Ergebnisse mit einer einfachen bzw. starken Zustimmung von durchschnittlich 74% ein deutlich positives Bild der Teilnehmer gegenüber der Usability des MARS-Explorer.

8.3.2 Reflexion in Hinblick auf die Validität

Zu beachten ist bei den vorgestellten Ergebnissen, dass es sich bei den Teilnehmenden, nicht um Experten im Bereich HCI bzw. Usability handelt. Aus diesem Grund ist zumindest die Evaluation der Usability stark vom individuellen Kenntnisstand und Empfinden des Teilnehmenden abhängig. Des Weiteren spiegeln diese Ergebnisse nur einen ersten Eindruck der Zielgruppe über eine erste funktionsfähige Version des MARS-Explorer wider, mit der sie sich lediglich maximal 90 Minuten befasst haben.

Aus zeitlichen und organisatorischen Gründen konnte im Rahmen dieser Ausarbeitung keine vorherige Abfrage des Kenntnisstandes der Teilnehmenden über das Thema MAS oder das MARS-Framework durchgeführt werden. Daher ist es möglich, dass einige Aspekte den Teilnehmenden bereits vertraut waren und die fachbezogenen Fragen aus diesem Grund korrekt beantwortet worden sind. Um diese Möglichkeit auszuschließen, wurden Fragen bzgl. der Erfahrungen der Teilnehmenden formuliert. Diese wurden zu Beginn des Fragebogens beantwortet. Die Ergebnisse dieser Fragen sind in Abbildung 8.9

dargestellt. Sie zeigen mit einem arithmetischen Mittelwert von 2.89 (Abbildung 8.9a) bzw. 1.33 (Abbildung 8.9b), dass die Teilnehmenden im Durchschnitt über mittelmäßige Erfahrung in der Softwareentwicklung und über sehr wenig Erfahrung in der ABM verfügen. Die Ergebnisse belegen, dass es sich bei den Teilnehmenden höchstwahrscheinlich um die in dieser Arbeit angestrebte Zielgruppe handelt.

8.4 Beantwortung der Forschungsfrage

Zusammenfassend kann gesagt werden, dass die Ergebnisse der Umfrage einen sehr guten ersten Eindruck der Teilnehmer gegenüber des MARS-Explorer vermitteln. Die Fragen zur Fachlichkeit des MARS-Framework wurden vom Großteil der Teilnehmer korrekt beantwortet und auch mit Blick auf die Usability zeigen sich die Teilnehmer mit stark überwiegender Mehrheit positiv gestimmt. Die Teilnehmer haben allerdings auch Schwächen des MARS-Explorer identifizieren können. Die Konfiguration des Simulationsszenarios sollte umfassendere Informationen über die jeweiligen Eingabefelder liefern. Außerdem sollte innerhalb der Beispielprojekte der Typ *Entity* behandelt werden. Um den angehenden Modellierenden mehr Informationen zu liefern, würde es sich anbieten, die Dokumentation des MARS-Framework stärker einzubinden, bspw. in Form von Verlinkungen oder kurzen Auszügen.

Die zu Beginn formulierte Forschungsfrage lautete: *Wie können fachfremde Benutzer des MARS-Framework ganzheitlich in der Simulation von Modellen unterstützt werden?*. Zuvor angestellte Untersuchungen wie von Serenko und Detlor (2002) oder Systeme wie NetLogo von Tisue und Wilensky (2004) zeigen, dass solche Plattformen zur Modellierung und Simulation von Modellen einen großen Beitrag zum Lernerfolg leisten können. Anhand der Untersuchungen, die in Kapitel 3 vorgestellt worden sind und auf Basis der Beobachtungen der MARS-Forschungsgruppe wurden Hypothesen abgeleitet, deren Untersuchung zeigen sollte, ob eine entsprechende Anwendung fachfremde Nutzer in der Simulation von Modellen innerhalb des MARS-Framework unterstützen kann.

Da alle Hypothesen vorläufig bestätigt werden konnten, lautet die Antwort auf die Forschungsfrage: Fachfremde Benutzer können ganzheitlich in der Simulation von Modellen unterstützt werden, indem ihnen eine Lernplattform zur Verfügung gestellt wird, die die Lernkomplexität reduziert. Eine solche Plattform ist der im Rahmen dieser Arbeit entwickelte MARS-Explorer. Dieser vereint die durchlaufenen Phasen des Modellierenden in einer Benutzeroberfläche, vereinfacht die Generierung neuer Modelle, sorgt für

eine einfache Konfiguration des zu simulierenden Szenarios und bietet aufschlussreiche Visualisierungen für vergangene und sogar laufende Simulationen.

9 Zusammenfassung und Ausblick

Multiagentensysteme sind ein komplexes Forschungsgebiet, dessen Umsetzung auf die Unterstützung von einfach zu verstehenden Werkzeugen angewiesen ist. Einsteiger sehen sich zum einen mit der Komplexität des Themas MAS, sowie zum anderen mit der Komplexität der damit einhergehenden Werkzeuge konfrontiert. Das Ziel dieser Arbeit war es daher, zu untersuchen, wie Einsteiger bestmöglich in der Modellierung innerhalb des MARS-Framework, welches von der Forschungsgruppe MARS an der HAW Hamburg entwickelt worden ist, unterstützt werden können. Eine Literaturrecherche hat gezeigt, dass auf das Thema spezialisierte Lernanwendungen bei der Modellierung unterstützen können. Um die Forschungsfrage zu beantworten, wurden auf Basis dieser Erkenntnis vier Hypothesen aufgestellt, die jeweils auf einen Kernaspekt des MARS-Framework abzielen.

Ein weiterer Aspekt dieser Arbeit beschäftigte sich mit der Konzeption und Entwicklung einer entsprechenden Lernplattform, die vor allem mit Fokus auf die Vermittlung von Lehrinhalten und auf die Langlebigkeit umgesetzt werden sollte, da sie fester Bestandteil zukünftiger Lehrmodule mit Bezug auf das Themengebiet MAS und das MARS-Framework werden soll. Aus diesem Grund wurden die nötigen Anforderungen an die Lernplattform sorgfältig anhand der Literatur und den Erfahrungswerten der Forschungsgruppe MARS herausgearbeitet, die es in den darauffolgenden Schritten umzusetzen galt. Das Ergebnis ist der sog. MARS-Explorer, welcher die verschiedenen Phasen, die ein Modellierender durchläuft, in einer übersichtlichen Benutzeroberfläche abbildet. Dadurch wird die Anzahl der Werkzeuge, die ein Einsteiger für die praktische Umsetzung eines MAS benötigt, deutlich reduziert und die verschiedenen Funktionen in einer einheitlichen Oberfläche präsentiert. Dies führt zu einer deutlich reduzierten Komplexität, da der Einsteiger zu jeglichen Aufgaben innerhalb des Arbeitsflusses Hilfestellungen und Informationen bekommt.

Die genannten Vorteile der entsprechenden Anwendung bestätigten sich in der Evaluation des MARS-Explorer, die mithilfe eines Nutzertests durchgeführt worden ist. Achtzehn

Studierende, die eine einführende Vorlesung in das Thema MAS besuchten, bekamen den MARS-Explorer mitsamt einer Liste von Aufgaben und einem anschließenden Fragebogen ausgeteilt, der von ihnen nach der Bewältigung der Aufgaben ausgefüllt worden ist. Die Ergebnisse des Fragebogens wurden dazu verwendet, um zum einen ein allgemeines Stimmungsbild über den MARS-Explorer einzufangen und zum anderen die aufgestellten Hypothesen zu überprüfen und die dazugehörige Forschungsfrage zu beantworten. Alle Hypothesen konnten aufgrund der deutlich positiven Rückmeldungen vorläufig verifiziert und die Forschungsfrage beantwortet werden: Fachfremde Benutzer können ganzheitlich in der Simulation von Modellen unterstützt werden, indem ihnen eine Lernplattform zur Verfügung gestellt wird, die die Lernkomplexität reduziert. Eine solche Plattform ist der im Rahmen dieser Arbeit entwickelte MARS-Explorer. Dieser vereint die durchlaufenen Phasen des Modellierenden in einer Benutzeroberfläche, vereinfacht die Generierung neuer Modelle, sorgt für eine einfache Konfiguration des zu simulierenden Szenarios und bietet aufschlussreiche Visualisierungen für vergangene und sogar laufende Simulationen.

Neben den überwiegend positiven Rückmeldungen hat die Umfrage aber auch gezeigt, dass Verbesserungspotentiale für den MARS-Explorer bestehen. Dazu gehört zum einen der Informationsgehalt innerhalb der Szenariokonfiguration, in der die Studierenden anhand eines Formulars verschiedene Einstellungen bzgl. ihrer durchzuführenden Simulation treffen können. Die Informationen zu den jeweiligen Feldern sollten direkt in die Anwendung integriert werden, anstatt erst bei falscher Eingabe auf Fehler hinzuweisen. Darüber hinaus sollte die Rolle des Typen *Entity* innerhalb einer Simulation weiter ausgearbeitet werden, indem bspw. die Beispielprojekte um die Verwendung von Entities erweitert werden. Neben den von den Testern identifizierten Problemen existieren noch drei Anforderungen, die aufgrund einer negativen Kosten-Nutzen-Abwägung innerhalb des Entwicklungszeitraums nicht implementiert werden konnten. Da sich diese Anforderungen lediglich mit der Umbenennung bzw. der automatischen Registrierung neuer Objekte beschäftigen, welche ebenso durch das Einfügen einer einzelnen Code-Zeile erreicht werden kann, haben sie keinen ausschlaggebenden Einfluss auf den Lernprozess der Studierenden. Dennoch sollten sie aus Gründen der angenehmeren Handhabung und der daraus resultierenden Zufriedenheit nachgeliefert werden.

Die Ergebnisse der Umfrage sollten z.T. unter Vorbehalt betrachtet werden, da es sich bei den Teilnehmern der Umfrage nicht um Experten des Forschungsgebiets Human Computer Interaction handelte. Innerhalb der Umfrage wurde eine heuristische Evaluation durchgeführt, die ursprünglich für Experten aus diesem Bereich konzipiert wurde. Hier

würde es sich also anbieten, eine entsprechende Evaluation auch von einer kleinen Gruppe von Experten durchführen zu lassen, um noch detailliertere Erkenntnisse über Verbesserungspotentiale des MARS-Explorer zu sammeln.

Literaturverzeichnis

[Abar u. a. 2017] Abar, Sameera ; Theodoropoulos, Georgios K. ; Lemarinier, Pierre ; O'Hare, Gregory M.: *Agent Based Modelling and Simulation tools: A review of the state-of-art software.* 2017

[Abramov 2021] Abramov, Dan: *Redux - A predictable state container for JavaScript apps.* 2021. – URL https://redux.js.org/. – [Online; aufgerufen am 11.12.2021]

[Ainsworth u. a. 2011] Ainsworth, Shaaron ; Prain, Vaughan ; Tytler, Russell: Drawing to learn in science. In: *Science* 333 (2011), Nr. 6046, S. 1096–1097

[Alomari u. a. 2020] Alomari, Hakam W. ; Ramasamy, Vijayalakshmi ; Kiper, James D. ; Potvin, Geoff: A User Interface (UI) and User eXperience (UX) evaluation framework for cyberlearning environments in computer science and software engineering education. In: *Heliyon* 6 (2020), Nr. 5, S. e03917. – URL https://doi.org/10.1016/j.heliyon.2020.e03917. – ISSN 24058440

[Argonne National Laboratory 2021] Argonne National Laboratory: *Repast Suite Documentation.* 2021. – URL https://repast.github.io/. – [Online; aufgerufen am 27.12.2021]

[AvaloniaUI OÜ 2021] AvaloniaUI OÜ: *Avalonia Docs.* 2021. – URL https://avaloniaui.net/. – [Online; aufgerufen am 10.12.2021]

[Baddeley 1981] Baddeley, Alan: The concept of working memory: a view of its current state and probable future development. In: *Cognition* (1981)

[Beale und Sharples 2002] Beale, Russell ; Sharples, Mike: Design Guide for Developers of Educational Software. (2002), S. 1–34

[Biswanger und Muehsig 2021] Biswanger, Gregor ; Muehsig, Robert: *GitHub - ElectronNET/Electron.NET: Build cross platform desktop apps with ASP.NET Core (Razor Pages, MVC, Blazor).* 2021. – URL https://github.com/ElectronNET/Electron.NET. – [Online; aufgerufen am 10.12.2021]

[Bodine u. a. 2020] Bodine, Erin N. ; Panoff, Robert M. ; Voit, Eberhard O. ; Weisstein, Anton E.: Agent-Based Modeling and Simulation in Mathematics and Biology Education. In: *Bulletin of Mathematical Biology* 82 (2020), Nr. 8. – URL https://doi.org/10.1007/s11538-020-00778-z. – ISBN 0123456789

[Bollen und Van Joolingen 2013] Bollen, Lars ; Van Joolingen, Wouter R.: SimSketch: Multiagent simulations based on learner-created sketches for early science education. In: *IEEE Transactions on Learning Technologies* 6 (2013), Nr. 3, S. 208–216. – ISSN 19391382

[Borgman u. a. 2008] Borgman, Christine L. ; Abelson, Hal ; Dirks, Lee ; Johnson, Roberta ; Koedinger, Kenneth R. ; Linn, Marcia C. ; Lynch, Clifford A. ; Oblinger, Diana G. ; Pea, Roy D. ; Salen, Katie u. a.: Fostering learning in the networked world: The cyberlearning opportunity and challenge. A 21st century agenda for the National Science Foundation. (2008)

[Bressan u. a. 2017] Bressan, Paulo A. ; Reis, Thiago H. dos ; Roberto, Artur J. ; de Paiva Guimarães, Marcelo: Considerations for designing educational software for different technological devices and pedagogical approaches. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10279 LNCS (2017), S. 143–154. – ISBN 9783319586991

[Clarke u. a. 2014] Clarke, Peter J. ; Davis, Debra ; King, Tariq M. ; Pava, Jairo ; Jones, Edward L.: Integrating testing into software engineering courses supported by a collaborative learning environment. In: *ACM Transactions on Computing Education* 14 (2014), Nr. 3. – ISSN 19466226

[Cohn 2004] Cohn, Mike: *User stories applied: For agile software development.* Addison-Wesley Professional, 2004

[Dorri u. a. 2018] Dorri, Ali ; Kanhere, Salil S. ; Jurdak, Raja: Multi-Agent Systems: A Survey. In: *IEEE Access* 6 (2018), S. 28573–28593. – ISSN 21693536

[Facebook Inc. 2021a] Facebook Inc.: *Introducing Hooks – React.* 2021. – URL https://reactjs.org/docs/hooks-intro.html. – [Online; aufgerufen am 18.11.2021]

[Facebook Inc. 2021b] Facebook Inc.: *React – A JavaScript library for building user interfaces.* 2021. – URL https://reactjs.org/. – [Online; aufgerufen am 10.12.2021]

[Fowler 2006] Fowler, Martin: *GUI Architectures.* 2006. – URL https://martinfowler.com/eaaDev/uiArchs.html. – [Online; aufgerufen am 08.12.2021]

[Gamma 2004] Gamma, Erich: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, 2004

[Ginot und Le Page 1998] Ginot, Vincent ; Le Page, Christophe: Mobidyc, a generic multi-agents simulator for modeling populations dynamics. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* Springer (Veranst.), 1998, S. 805–814

[Ginot und Souissi 2003] Ginot, Vincent ; Souissi, Sami: Mobidyc tutorial. (2003), S. 1–63. – URL http://link.springer.com/10.1007/3-540-64574-8{_}467

[Glake u. a. 2017] Glake, Daniel ; Weyl, Julius ; Dohmen, Carolin ; Hüning, Christian ; Clemen, Thomas: Modeling through model transformation with MARS 2.0. In: *Simulation Series* 49 (2017), Nr. 5, S. 13–24. – ISBN 9781510838246

[Grudin 1992] Grudin, Jonathan: Utility and usability: research issues and development contexts. In: *Interacting with computers* 4 (1992), Nr. 2, S. 209–217

[Hüning u. a. 2016] Hüning, Christian ; Dalski, Jan ; Adebahr, Mitja ; Lenfers, Ulfia ; Thiel-Clemen, Thomas ; Grundmann, Lukas: Modeling & simulation as a service with the massive multi-agent system MARS. In: *Simulation Series* 48 (2016), Nr. 1, S. 1–8. – ISBN 9781510823150

[Hüning u. a. 2014] Hüning, Christian ; Wilmans, Jason ; Feyerabend, Nils ; Thiel-Clemen, Thomas: MARS - A next-gen multi-agent simulation framework. In: *Simulation in Umwelt- und Geowissenschaften, Workshop Osnabr{ü}ck* (2014), Nr. 2008, S. 1–14

[ISO 9241:11 2018] Ergonomics of human-system interaction – Usability: Definitions and concepts / International Organization for Standardization. Geneva, CH, 2018. – Standard. – URL https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:en

[ISO/IEC 25010 2011] Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models / International Organization for Standardization. Geneva, CH, 2011. – Standard. – URL https://www.iso.org/standard/35733.html

[Janssen u. a. 2008] Janssen, Marco A. ; Alessa, Lilian N. ia ; Barton, Michael ; Bergin, Sean ; Lee, Allen: Towards a community framework for agent-based modelling. In: *Jasss* 11 (2008), Nr. 2. – ISSN 14607425

[Lewis 2002] Lewis, James R.: Psychometric evaluation of the PSSUQ using data from five years of usability studies. In: *International Journal of Human-Computer Interaction* 14 (2002), Nr. 3-4, S. 463–488

[Liu 2021] Liu, Shanhong: *Most used web frameworks among developers worldwide, as of 2021*. 2021. – URL https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. – [Online; aufgerufen am 10.12.2021]

[Louca und Zacharia 2012] Louca, Loucas T. ; Zacharia, Zacharias C.: Modeling-based learning in science education: cognitive, metacognitive, social, material and epistemological contributions. In: *Educational Review* 64 (2012), Nr. 4, S. 471–492

[Louca u. a. 2011] Louca, Loucas T. ; Zacharia, Zacharias C. ; Constantinou, Constantinos P.: In Quest of productive modeling-based learning discourse in elementary school science. In: *Journal of Research in Science Teaching* 48 (2011), Nr. 8, S. 919–951

[Macal 2016] Macal, C. M.: Everything you need to know about agent-based modelling and simulation. In: *Journal of Simulation* 10 (2016), Nr. 2, S. 144–156. – ISSN 17477786

[Macal und North 2013] Macal, Charles M. ; North, Michael J.: Successful approaches for teaching agent-based simulation. In: *Journal of Simulation* 7 (2013), Nr. 1, S. 1–11. – URL http://dx.doi.org/10.1057/jos.2012.1. – ISSN 17477786

[MARS Group 2021a] MARS Group: *Multi-Agent Research and Simulation | MARS | MARS LIFE Runtime System*. 2021. – URL https://mars.haw-hamburg.de/index.html. – [Online; aufgerufen am 23.11.2021]

[MARS Group 2021b] MARS Group: *Projects – MARS Group*. 2021. – URL https://mars-group.org/projects-using-mars/. – [Online; aufgerufen am 23.11.2021]

[MARS Group 2021c] MARS Group: *Publications – MARS Group*. 2021. – URL https://mars-group.org/publications/. – [Online; aufgerufen am 23.11.2021]

[Material-UI SAS 2021] Material-UI SAS: *MUI: The React component library you always wanted*. 2021. – URL https://mui.com/. – [Online; aufgerufen am 12.12.2021]

[Microsoft 2021a] Microsoft: *Monaco Editor*. 2021. – URL https://microsoft.github.io/monaco-editor/index.html. – [Online; aufgerufen am 17.11.2021]

[Microsoft 2021b] Microsoft: *Das .NET Compiler Platform SDK (Roslyn APIs).* 2021. – URL https://docs.microsoft.com/de-de/dotnet/csharp/roslyn-sdk/. – [Online; aufgerufen am 12.11.2021]

[Microsoft 2021c] Microsoft: *TypeScript: JavaScript With Syntax For Types.* 2021. – URL https://www.typescriptlang.org/. – [Online; aufgerufen am 10.12.2021]

[Microsoft 2021d] Microsoft: *What is the Language Server Protocol?* 2021. – URL https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/. – [Online; aufgerufen am 15.11.2021]

[Moore und Redmond-Pyle 1995] Moore, A. ; Redmond-Pyle, D.: Graphical User Interface Design and Evaluation: A Practical Process, 1995

[.NET Foundation 2021] .NET Foundation: *omnisharp-roslyn.* 2021. – URL https://github.com/OmniSharp/omnisharp-roslyn. – [Online; aufgerufen am 10.12.2021; Verwendeter Commit: fa8eb43b6345cf6e34a4c963d5952a87dfd00e88]

[NetLogo 2021] NetLogo: *NetLogo Web: Rabbits Grass Weeds.* 2021. – URL http://www.netlogoweb.org/launch#http://www.netlogoweb.org/assets/modelslib/Sample%20Models/Biology/Rabbits%20Grass%20Weeds.nlogo. – [Online; aufgerufen am 05.11.2021]

[Niazi und Hussain 2011] Niazi, Muaz ; Hussain, Amir: Agent-based computing from multi-agent systems to agent-based models: A visual survey. In: *Scientometrics* 89 (2011), Nr. 2, S. 479–499. – ISSN 01389130

[Nielsen 1994a] Nielsen, Jakob: Heuristic evaluation. In: *Usability inspection methods* (1994)

[Nielsen 1994b] Nielsen, Jakob: *Usability engineering.* Morgan Kaufmann, 1994

[Nielsen 2012] Nielsen, Jakob: Usability 101: Introduction to usability. (2012)

[Nielsen und Molich 1990] Nielsen, Jakob ; Molich, Rolf: Heuristic evaluation of user interfaces. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1990, S. 249–256

[OpenJS Foundation 2021a] OpenJS Foundation: *Electron | Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML und CSS entwickeln.* 2021. – URL https://www.electronjs.org/. – [Online; aufgerufen am 15.11.2021]

[OpenJS Foundation 2021b] OpenJS Foundation: *Electron Support | Electron*. 2021. – URL https://www.electronjs.org/docs/latest/tutorial/support. – [Online; aufgerufen am 08.12.2021]

[OpenJS Foundation 2021c] OpenJS Foundation: *ESLint - Find and fix problems in your JavaScript code*. 2021. – URL https://eslint.org/. – [Online; aufgerufen am 11.12.2021]

[OpenJS Foundation 2021d] OpenJS Foundation: *Process Model*. 2021. – URL https://www.electronjs.org/docs/v14-x-y/tutorial/process-model. – [Online; aufgerufen am 15.11.2021]

[OpenJS Foundation 2021e] OpenJS Foundation: *Security, Native Capabilities, and Your Responsibility | Electron*. 2021. – URL https://www.electronjs.org/docs/latest/tutorial/support. – [Online; aufgerufen am 08.12.2021]

[Parlangeli u. a. 1999] Parlangeli, Oronzo ; Marchigiani, Enrica ; Bagnara, Sebastiano: Multimedia systems in distance education: Effects of usability on learning. In: *Interacting with Computers* 12 (1999), Nr. 1, S. 37–49. – ISSN 09535438

[Phillips 1997] Phillips, Robin: *The Developer's Handbook of Interactive Multimedia*. Routledge, 1997

[Prokhorov 2021] Prokhorov, Alexey: *GitHub - megahertz/electron-log: Just a simple logging module for your Electron application*. 2021. – URL https://github.com/megahertz/electron-log. – [Online; aufgerufen am 06.01.2022]

[Sandars und Lafferty 2010] Sandars, John ; Lafferty, Natalie: Twelve tips on usability testing to develop effective e-learning in medical education. In: *Medical teacher* 32 (2010), Nr. 12, S. 956–960

[Serenko und Detlor 2002] Serenko, Alexander ; Detlor, Brian: Agent Toolkits: A General Overview of the Market and an Assessment of Instructor Satisfaction with Utilizing Toolkits in the Classroom. In: *Technical Report 455* (2002), S. 49

[Ssemugabi und de Villiers 2007] Ssemugabi, Samuel ; Villiers, Ruth de: A Comparative Study of Two Usability Evaluation Methods Using a Web-Based E-Learning Application. (2007), S. 204. ISBN 9781595937759

[Stack Exchange Inc. 2021] Stack Exchange Inc.: *Stack Overflow Developer Survey 2021*. 2021. – URL https://insights.stackoverflow.com/survey/2021. – [Online; aufgerufen am 10.12.2021]

[Starke 2015] Starke, Gernot: *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. Carl Hanser Verlag GmbH Co KG, 2015

[Sycara 1998] Sycara, Katia P.: Multiagent Systems. In: *AI Magazine* 19 (1998), Jun., Nr. 2, S. 79. – URL https://ojs.aaai.org/index.php/aimagazine/article/view/1370

[Tisue und Wilensky 2004] Tisue, Seth ; Wilensky, Uri: Netlogo: A simple environment for modeling complexity. In: *Conference on Complex Systems* (2004), S. 1–10. – URL http://ccl.sesp.northwestern.edu/papers/netlogo-iccs2004.pdf. – ISBN 0769520723

[TypeFox GmbH 2021] TypeFox GmbH: *monaco-languageclient*. 2021. – URL https://github.com/TypeFox/monaco-languageclient. – [Online; aufgerufen am 10.12.2021; Verwendeter Commit: 8a9ebf9add881767fbcf8d2f9ac87bfa6559a96b]

[Van Nuland u. a. 2017] Van Nuland, Sonya E. ; Eagleson, Roy ; Rogers, Kem A.: Educational software usability: Artifact or Design? In: *Anatomical Sciences Education* 10 (2017), Nr. 2, S. 190–199. – ISSN 19359780

[Wooldridge 2009] Wooldridge, Michael: *An introduction to multiagent systems*. John wiley & sons, 2009

[Wooldridge und Jennings 1995] Wooldridge, Michael ; Jennings, Nicholas R.: Intelligent agents: Theory and practice. In: *The Knowledge Engineering Review* 10 (1995), Nr. 2, S. 115–152. – ISSN 14698005

A Anhang

A.1 Material zur Versuchsdurchführung

A.1.1 Handout

Evaluation of the MARS-Explorer

Thank you for participating in this experiment. Below you will find some information in brief.

Who? 👨‍🎓

- Jan-Niklas Voß, 25 years old
- Studying computer science at HAW since 2019
- Currently writing my master thesis within the MARS Group
- Focused on application development with a special focus on frontend development

What? ✍🏻

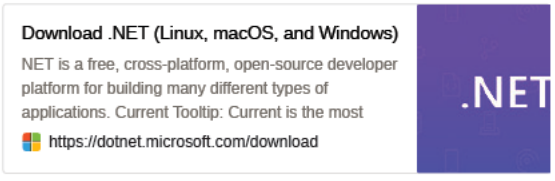
- Within my thesis I developed the **MARS-Explorer**
- This is an application that is intended to be used in the future as a learning platform for multi-agent systems and the MARS framework
- I would like to evaluate whether the application is suitable to assist non-experts comprehensively in the process of modelling agent-based systems

Why? 🤷🏻‍♂️

- We have noticed that the large number of tools is an unpleasant hurdle to get into the topic of Multi-Agent-Systems
- You should not be spending much time on the tools and more on the modelling/learning process itself

How? 💡

- To be able to make full use of the MARS Explorer, you need to have the .NET Core SDK installed



- Download the MARS-Explorer for your current OS *(links are below)*
- Install the application
- Below is a checklist of things for you to do to get familiar with the application
- Feel free to continue experimenting without any instructions
- Once you have worked through the checklist and you feel like you are ready, please fill out the evaluation questionnaire (*link is below*)

Where? 🔗

The password for the questionnaire and for the MARS-Explorer is: **mars**

MARS-Explorer

MacOS

https://cloud.haw-hamburg.de/index.php/s/zH7VmKnACF4lKXq

Windows

https://cloud.haw-hamburg.de/index.php/s/iv8OKR18zKeQQcZ **(x64)**

https://cloud.haw-hamburg.de/index.php/s/9uZaQJl39m2oNLw **(x86)**

Linux

https://cloud.haw-hamburg.de/index.php/s/Ri1pwD49ltavcgs **(Debian/.deb)**

https://cloud.haw-hamburg.de/index.php/s/zLexUlGXLLSN0NM **(Redhead/.rpm)**

Questionnaire

Evaluation of the MARS Explorer (LamaPoll - Online Survey)

https://campus.lamapoll.de/Evaluation-of-the-MARS-Explorer/

To-Do ☑️

Below you will find a few tasks that are intentionally not described in detail.

In case you cannot complete a task, please feel free to ask me. If you have any problems, please state them in the questionnaire in its designated section.

It is recommended to work through them from top to bottom.

- [ ] Read the description of an example project
- [ ] Identify the number of example projects
- [ ] Open a copy of the SheepWolfStarter project
- [ ] Identify the number of models (incl. Program.cs)
- [ ] Make valid changes to any model
- [ ] Save your changes
- [ ] Have a look at a file from the examples
- [ ] Create ...
  - [ ] ... a new vector layer
  - [ ] ... a new agent
  - [ ] ... a new entity
- [ ] Delete one of the created objects
- [ ] Increase the number of `steps` in the configuration to 500
- [ ] Try to set `deltaT` to a negative number
- [ ] Increase the number of sheep agents in the configuration to 50
- [ ] Increase the property `SheepGainFromFood` in the configuration to 5
- [ ] Save the changes you made to the configuration
- [ ] Start a simulation
- [ ] Identify the number of Sheep agents in the simulation while it is running
- [ ] Check the console output of the simulation
- [ ] Identify the number of Sheep agents that exist at the 50% Simulation Progress
- [ ] Identify the coordinate with the maximum number of Sheep agents

A.1.2 Umfrage

Introduction

Hello and welcome to my survey!

I am Jan-Niklas and I am currently writing my master thesis on the topic of multi-agent simulations and how this topic can be taught more easily within the context of the MARS framework.

Before you do this survey, you should already have experimented some time with the MARS-Explorer (hereafter only referred to as "the system").

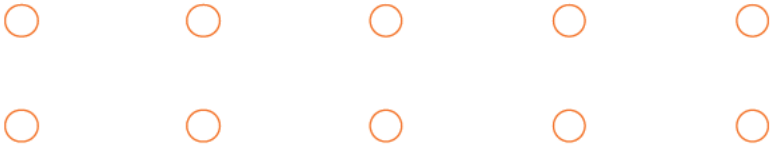
I would like to ask you a few simple questions regarding your satisfaction with the tool and its usefulness. In order for me to evaluate whether the tool supports you in your learning process, there are a few subject-related questions. Please answer them honestly and without looking in other sources than the MARS Explorer.

This survey is completely anonymous and will not affect your grade in any way.

Your experience

Please rate your experience

Where:
0 = you have very little to no experience
5 = you have already developed many complex systems.



Survey created with
LamaPoll

Satisfaction with the System

Please rate your satisfaction with the system

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree
Overall, I am satisfied with how easy it is to use the System.	○	○	○	○	○
I feel comfortable using the System.	○	○	○	○	○
It was easy to learn to use the System.	○	○	○	○	○
I believe I became productive quickly using the System.	○	○	○	○	○
It is easy to find the information I need.	○	○	○	○	○
The interface of the System is pleasant.	○	○	○	○	○
I like using the interface of this System.	○	○	○	○	○
I would recommend the System to fellow students.	○	○	○	○	○
Overall, I am satisfied with the Application.	○	○	○	○	○

Usability of the System

Please rate your satisfaction with the system

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree
The system keeps me informed about its current status.	○	○	○	○	○
I receive feedback from the system within a reasonable period of time.	○	○	○	○	○
All texts of the system are worded in an understandable way.	○	○	○	○	○
At any time I know how to return to the home page.	○	○	○	○	○
I can cancel any of my actions at any time.	○	○	○	○	○
The system uses common vocabulary for applications.	○	○	○	○	○
No unhandled errors have occurred while using the system.	○	○	○	○	○
All information I need for my tasks is visible.	○	○	○	○	○
The system helps me to understand all its features.	○	○	○	○	○
The most common key combinations have the expected effect (e.g. CMD/STR + S = Save).	○	○	○	○	○
The user interface does not show more information than	○	○	○	○	○

Survey created with LamaPoll

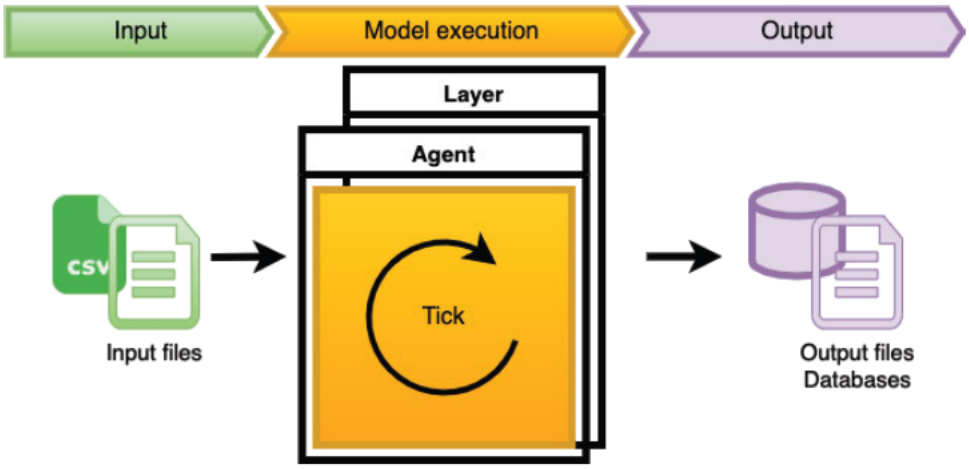
necessary.					
If errors occur, they are phrased in an understandable way.	○	○	○	○	○
When errors occur, the error messages help me understand why they occurred.	○	○	○	○	○

MARS related questions

What are the core elements of a MARS framework model?

- [ ] Objects of user-defined class hierarchy
- [ ] Environments
- [ ] Lists
- [ ] Entities
- [ ] Layers
- [ ] Maps
- [ ] Agents

Three-phase information system



Agents are the central part of the MARS simulation, which can be viewed as an information system with three phases. First, there is some kind of input data that is integrated in the model, either in layer or agent. The model execution is then performed and generates output data in different formats for analyzation.

To which phase of the previously shown process belong(s) ...

	Input	Model execution	Output
PropertyDescription?	◯	◯	◯
Method init?	◯	◯	◯
Method tick?	◯	◯	◯

Survey created with LamaPoll

Put the following phases within a MARS model in the order in which they are usually performed.

Drag and drop the first process step to the top and the last one to the bottom.

- Analysis
- Configuration
- Simulation
- Modeling

In the configuration, is it possible to ... ?

The term "possible" means that the configuration with the values can be saved and the simulation can theoretically be executed without errors.

	True	False
... use steps and time ranges at the same time?	○	○
... specify a startPoint but no endPoint?	○	○
... specify a negative value for deltaT?	○	○
... specify a negative value for step?	○	○
... set the number of agents?	○	○
... set the number of layers?	○	○
... initialise agents with an input file?	○	○
... initialise entities with an input file?	○	○

Usefulness of the System

Did you have any problems with the tasks you were given?

If this is the case, please mark the tasks you had problems with and briefly describe what caused it.

- [ ] Read the description of an example project
- [ ] Identify the number of examples projects
- [ ] Open a copy of the wolf sheep model
- [ ] Identify the number of models (incl. Program.cs)
- [ ] Make valid changes to any model
- [ ] Save your changes

- [ ] Have a look at a file from the examples
- [ ] Create a new vector layer
- [ ] Create a new agent
- [ ] Create a new entity
- [ ] Delete one of the created objects
- [ ] Increase the number of `steps` in the configuration to 500
- [ ] Try to set `deltaT` to a negative number
- [ ] Increase the number of sheep agents in the configuration to 50

- [ ] Increase the property `SheepGainFromFood` in the configuration to 5
- [ ] Save the changes you made to the configuration
- [ ] Start a simulation
- [ ] Identify the number of sheep agents in the simulation while it is running
- [ ] Check the console output of the simulation
- [ ] Identify the number of Sheep agents that exist at the 50% Simulation Progress
- [ ] Identify the coordinate with the maximum number of Sheep agents

Please rate the usefulness of the system

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree
The system helps me to better understand the different objects that exist within a simulation.	○	○	○	○	○
The system makes it easier for me to create new simulation objects.	○	○	○	○	○
The system helps me to improve my understanding of the different phases within a project.	○	○	○	○	○
The system helps me to identify my errors in the modelling process.	○	○	○	○	○
The system helps me to identify my errors within the configuration.	○	○	○	○	○
The system helps me to better understand the API of the MARS framework and thus make fewer mistakes.	○	○	○	○	○
As a result of using the system, I need less support from the experts.	○	○	○	○	○

Feedback

Do you have further feedback?

A.2 Zustand der Umsetzung der funktionalen Anforderungen

Tabelle A.1: Übersicht über den Zustand der Umsetzung der funktionalen Anforderung in der ✓ für umgesetzt und ✗ für nicht umgesetzt steht

Modul	Anforderung	Umgesetzt?
Modellierung	*Klasse erstellen* (FA 1)	✓
	Hilfestellung zur Auswahl der Objekte erhalten (FA 2)	✓
	Objekt-Vorlagen verwenden (FA 3)	✓
	Neues Objekt registrieren (FA 4)	✗
	Klassen ansehen (FA 5)	✓
	Klasse ansehen (FA 6)	✓
	Klasse bearbeiten (FA 7)	✓
	Klasse umbenennen (FA 8)	✗
	Klasse löschen (FA 9)	✓
	Eingabe rückgängig machen (FA 10)	✓
	Eingabe wiederholen (FA 11)	✓
	Stetige Validierung des Modells (FA 13)	✓
	Status anzeigen (FA 14)	✓
	Fehler ansehen (FA 15)	✓
	Beispielmodelle einstellen (FA 16)	✓
	Beispielmodelle ansehen (FA 17)	✓
	Informationen zu Beispielmodellen einsehen (FA 18)	✓

Konfiguration	*Konfiguration darstellen* (FA 20)	✓
	Konfigurationsmöglichkeiten logisch anordnen (FA 19)	✓
	Globale Einstellungen bearbeiten (FA 21)	✓
	Agenten-Mappings bearbeiten (FA 22)	✓
	Layer-Mappings bearbeiten (FA 23)	✓
	Entity-Mappings bearbeiten (FA 24)	✓
	Eingaben in der Konfiguration rückgängig machen (FA 25)	✓
	Eingaben in der Konfiguration wiederholen (FA 26)	✓
	Fehler in der Konfiguration anzeigen (FA 27)	✓
	Grund für Fehler anzeigen (FA 28)	✓
	Status der Konfiguration darstellen (FA 29)	✓
Simulation	*Simulation starten* (FA 30)	✓
	Simulationsstart überprüfen (FA 31)	✓
	Simulation stoppen (FA 32)	✓
	Simulationsstatus anzeigen (FA 33)	✓
	Fortschritt anzeigen (FA 34)	✓
	Ausgabe einsehen (FA 35)	✓
	Fehler einsehen (FA 36)	✓
Analyse	*Agentenanzahl einsehen* (FA 37)	✓
	Agentenpositionen einsehen (FA 38)	✓
	Ergebnisse filtern (FA 39)	✓

Projektverwaltung	*Projekte einsehen* (FA 40)	✓
	Projekte umbenennen (FA 43)	✗
	Projekte löschen (FA 44)	✓
	Projekt erstellen (FA 41)	✓
	Beispielmodelle kopieren (FA 42)	✓

A.3 Umfrageergebnisse

Tabelle A.2: Antworten der Teilnehmer bzgl. der Aufgabe: *Please rate your experience.*, wo der Wert 1 für wenig bis keine Erfahrung und der Wert 5 für Erfahrung in der Entwicklung von mehreren komplexen Systeme steht.

Häufigkeit Anzahl	**Wert 1**	**Wert 2**	**Wert** 3	**Wert 4**	**Wert 5**
How much experience do you have in software development?	2	3	9	3	1
How much experience do you have in agent-based modeling?	14	2	2	0	0

Tabelle A.3: Antworten der Teilnehmer bzgl. der Aufgabe: *Please rate your satisfaction with the system.*

Anzahl Antworten	**Strongly agree**	**Agree**	**Neither agree nor disagree**	**Disagree**	**Strongly disagree**
Overall, I am satisfied with how easy it is to use the System.	6	11	1	0	0
I feel comfortable using the System.	6	10	2	0	0
It was easy to learn to use the System.	5	12	1	0	0
I believe I became productive quickly using the System.	4	9	5	0	0
It is easy to find the information I need.	3	9	5	1	0
The interface of the System is pleasant.	11	6	1	0	0

I like using the interface of this System.	9	8	1	0	0
I would recommend the System to fellow students.	9	8	1	0	0
Overall, I am satisfied with the Application.	7	10	1	0	0
The system keeps me informed about its current status.	2	10	6	0	0
I receive feedback from the system within a reasonable period of time.	2	9	7	0	0
All texts of the system are worded in an understandable way.	7	9	1	1	0
At any time I know how to return to the home page.	11	4	3	0	0
I can cancel any of my actions at any time.	4	10	4	0	0
The system uses common vocabulary for applications.	10	7	1	0	0
No unhandled errors have occurred while using the system.	3	6	8	0	1
All information I need for my tasks is visible.	2	14	1	1	0

The system helps me to understand all its features.	3	12	1	1	1
The most common key combinations have the expected effect (e.g. CMD/STR + S = Save).	4	10	4	0	0
The user interface does not show more information than necessary.	4	13	1	0	0
If errors occur, they are phrased in an understandable way.	1	8	7	2	0
When errors occur, the error messages help me understand why they occurred.	3	5	9	1	0

Tabelle A.4: Antworten der Teilnehmer bzgl. der Frage: *What are the core elements of a MARS framework model?*

Optionen	**Anzahl**
Agents	18
Layers	16
Entities	15
Environments	7
Lists	0
Maps	0
Objects of user-defined class hierarchy	0

Tabelle A.5: Antworten der Teilnehmer bzgl. der Aufgabe: *To which phase of the previously shown process belong(s) ...*

Anzahl Antworten	**Input**	**Model execution**	**Output**
PropertyDescription?	10	3	5
Method init?	10	8	0
Method tick?	0	14	4

Tabelle A.6: Antworten der Teilnehmer bzgl. der Aufgabe: *Put the following phases within a MARS model in the order in which they are usually performed.*

Anzahl Antworten	**Position**	**Platz 1**	**Platz 2**	**Platz 3**	**Platz 4**
Modeling	1	13	3	1	1
Configuration	2	2	14	1	1
Simulation	3	0	1	15	2
Analysis	4	3	0	1	14

Tabelle A.7: Antworten der Teilnehmer bzgl. der Frage(n): *In the configuration, is it possible to ... ?*

Anzahl Antworten	**True**	**False**
... use steps and time ranges at the same time?	8	10
... specify a startPoint but no endPoint?	7	11
... specify a negative value for deltaT?	1	17
... specify a negative value for step?	1	17
... set the number of agents?	18	0
... set the number of layers?	11	7
... initialise agents with an input file?	15	3
... initialise entities with an input file?	12	6

Tabelle A.8: Antworten der Teilnehmer bzgl. der Frage: *Did you have any problems with the tasks you were given?*

Optionen	**Anzahl**
Read the description of an example project	0
Identify the number of examples projects	0
Open a copy of the wolf sheep model	0
Identify the number of models (incl. Program.cs)	0
Make valid changes to any model	0
Save your changes	0
Have a look at a file from the examples	0
Create a new vector layer	1
Create a new agent	0
Create a new entity	0
Delete one of the created objects	0
Increase the number of ‘steps‘ in the configuration to 500	0
Try to set ‘deltaT‘ to a negative number	0
Increase the number of sheep agents in the configuration to 50	0
Increase the property ‘SheepGainFromFood‘ in the configuration to 5	0
Save the changes you made to the configuration	0
Start a simulation	6
Identify the number of sheep agents in the simulation while it is running	0
Check the console output of the simulation	0
Identify the number of Sheep agents that exist at the 50% Simulation Progress	0
Identify the coordinate with the maximum number of Sheep agents	0

Tabelle A.9: Antworten der Teilnehmer bzgl. der Aufgabe: *Please rate the usefulness of the system.*

Anzahl Antworten	**Strongly agree**	**Agree**	**Neither agree nor disagree**	**Disagree**	**Strongly disagree**
The system helps me to better understand the different objects that exist within a simulation.	5	11	2	0	0
The system makes it easier for me to create new simulation objects.	8	9	1	0	0
The system helps me to improve my understanding of the different phases within a project.	6	9	3	0	0
The system helps me to identify my errors in the modelling process.	1	11	5	1	0
The system helps me to identify my errors within the configuration.	1	10	7	0	0
The system helps me to better understand the API of the MARS framework and thus make fewer mistakes.	4	12	1	1	0
As a result of using the system, I need less support from the experts.	4	8	6	0	0

Tabelle A.10: Antworten der Teilnehmer bzgl. der Frage: *Do you have further feedback?*

Wert/Antwort
Ich wünschte ich hätte dieses Tool bereits letztes Semester im K.I. WP gehabt. Sehr nützlich
Help tab is empty :(
really good software
Whenever looking at the simulation data you always need to select the desired agents, would be nice to have them selected over multiple simulation runs, so you dont have to check the checkboxes everytime you look at the data

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original