Entwurf und Implementierung der ModUro-Toolbox zur Analyse von Urothelsimulationen

Mathuraa Pathmanathan

Bachelor-Thesis

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Studiengang Medizinische Informatik

Fakultät für Informatik

Hochschule Mannheim

25.02.2016

Durchgeführt bei der Hochschule Mannheim

Betreuer

Prof. Markus Gumbel, Hochschule Mannheim

Zweitkorrektor: Angelo Torelli

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 25.02.2016

Mathuraa Pathmanathan

Abstract

Im Projekt „Modellierung und Simulation in der Urologie“ (ModUro) wird mittels softwarebasierter Simulationen untersucht, wie Krebszellen im Gewebe des Urotheliums entstehen [1]. Hierzu wird die Software „CompuCell3D“ (CC3D) verwendet, mit der verschiedene Hypothesen getestet werden. Analog zu anderen Experimenten, stellt sich auch bei Computersimulationen die Notwendigkeit auf, Dokumentation darüber zu führen, mit welchen Parametern, Startwerten und grundlegenden Modellen ein Experiment durchgeführt wurde. Die Ergebnisse der Experimente werden in der Folge statistisch ausgewertet. CC3D bietet diese Funktionalität nicht, weshalb in dieser Arbeit die ModUro-Toolbox entwickelt wurde. Mit Hilfe von Szenarien werden die möglichen Interaktionen mit der Anwendung dargestellt und darauf basierend die Anforderungen definiert. Die Anforderungen an die grafische Benutzeroberfläche werden mittels Prototypen erhoben und validiert. Die umgesetzte ModUro-Toolbox basiert auf das MVC-Pattern. Sie ermöglicht es, die Ergebnisse der rechenintensiven CC3D-Simulationen in einer textuellen und graphischen Übersicht benutzerfreundlich darzustellen und statistisch zu analysieren. Somit können Projekte konfiguriert werden, die später auf der grafischen Benutzeroberfläche untersucht werden können.

Inhaltsverzeichnis

1 Einleitung

Biologische Systeme weisen einen hohen Komplexitätsgrad auf [2]. Aufgrund dieser Komplexität stößt die in-vitro und in-vivo Forschung auf technische Barrieren und biologische Beschränkungen [3, p. 198]. Die Nachbildung biologischer Prozesse im Computer bietet daher eine neue Möglichkeit, ergänzend zu bio- und medizintechnologischen Methoden, Krankheiten und Therapien weiter zu erforschen [4]. In der heutigen Zeit sind etwa 51% der Männer und 43% der Frauen an Krebs betroffen [5]. Der Blasenkrebs zählt zu den häufigeren Krebsarten, mit jährlich etwa 16.000 Neuerkrankungen in Deutschland [6]. Die häufigste Form von Blasenkrebs sind Urothelkarzinome, die innerhalb des Urotheliums entstehen. Abbildung 1, eine Kopie aus der Wikipedia, zeigt Krebszellen, die in das umliegende Gewebe reinwachsen.

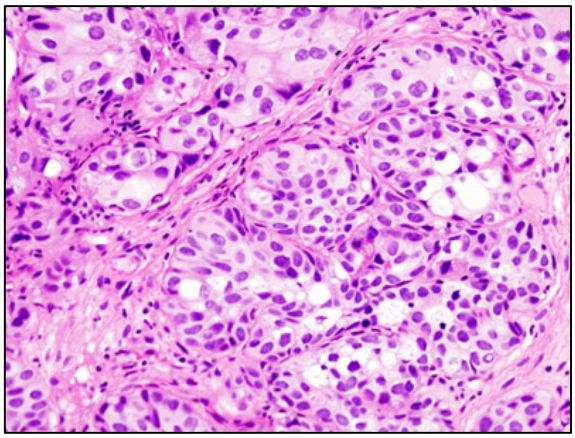

Abbildung 1: aus Wikipedia - Urothelkarzinom der Blase [7]

Im Projekt „Modellierung und Simulation in der Urologie“ (ModUro) wird in Kooperation mit der Medizinischen Fakultät Mannheim der Universität Heidelberg am Uniklinikum Mannheim mittels Computermodellen untersucht, wie Zellen im Gewebe des Urotheliums entstehen und differenzieren. Mittels Computermodellen können komplexe Zellinteraktionen, Lage und Haftungseigenschaften, die Entwicklung von Zelltypen in Abhängigkeit von der Zeit und weitere Parameter untersucht werden [4]. Das Ziel von ModUro ist es, das Urothelium sowohl in einen gesunden Zustand zu simulieren, als auch die Stabilität des Urotheliums im gestressten Zustand zu testen [1]. „Gestresst“ bedeutet hier eine ungewöhnliche Abweichung der biochemischen Parameter von einer gesunden Norm. Hierzu wird eine Software namens „CompuCell3D“ (CC3D) verwendet, mit der verschiedenen Hypothesen getestet werden können. CC3D ist ein Open-Source Softwareprodukt der Indiana University in

Bloomington (USA). Das Framework stellt eine Simulationsumgebung für die Modellierung von zellbasierten Geweben, Organen und Organismen bereit [8]. In Abbildung 2 zeigt der CC3D-Player die Entstehung der Zellen im Verlaufe der Zeit.

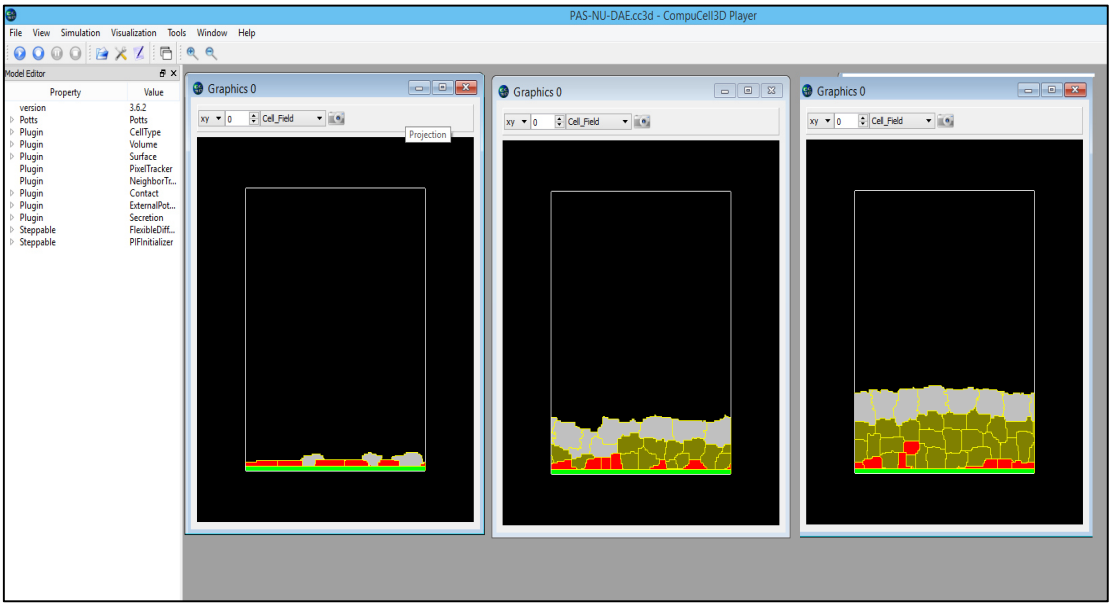

Abbildung 2: Simulationsdarstellung im CompuCell3D-Player

1.1 Ziel der Arbeit

Wie bei anderen Experimenten ist es auch bei Computersimulationen nötig, Dokumentation darüber zu führen mit welchen Parametern, Startwerten und Modellen ein Experiment durchgeführt wurde. Die entstandenen Ergebnisse werden später zur statistischen Analysen verwendet. Das Framework CC3D bietet keine Funktionalität um die Ergebnisse zu analysieren. Daher wird in dieser Arbeit die ModUro-Toolbox entwickelt. Die Toolbox soll es ermöglichen, die Ergebnisse der rechenintensiven CC3D-Simulationen in einer textuellen und graphischen Übersicht benutzerfreundlich darzustellen. Somit können Hypothesen über Zelldifferenzierung oder Zellzykluszeiten einfach verglichen und ausgewertet werden.

1.2 Aufbau der Arbeit

Der Aufbau der Bachelorarbeit „Entwurf und Implementierung der ModUro-Toolbox zur Analyse von Urothelsimulationen“ wird in folgende Kapitel gegliedert:

Der theoretische Teil der Bachelorarbeit befasst sich mit den biologischen und informationstechnischen Grundlagen der Modellbildung und Zellsimulation. Des Weiteren werden grundlegende Aspekte zur Erstellung von grafischen Benutzeroberflächen behandelt. Im Abschnitt der Analyse werden anhand von Szenarien Anforderungen erhoben, die die Funktionalitäten der Software definieren. Darauf basierend werden Konzepte beschrieben, wie die grafische Benutzeroberfläche aufgebaut werden kann. Im weiteren Verlauf der Arbeit wird das Design der Benutzeroberfläche erklärt. Anschließend wird im Kapitel „Umsetzung“ die praktische Entwicklung der ModUro-Toolbox anhand eines Projektbeispiels beschrieben. Im Kapitel „Zusammenfassung und Ausblick“ werden die umgesetzten Funktionalitäten zusammengefasst. Abschließend werden mögliche Funktionen zur Erarbeitung in weiterführenden Projekten aufgezeigt, die eine sinnvolle Erweiterung der ModUro-Toolbox darstellen.

1.3 Voraussetzungen

Folgende Dokumente und Prototypen können nützlich sein, um die in dieser Arbeit erläuterten Zusammenhänge besser verstehen zu können:

- A.Torelli et al. : Modeling of the Urothelium with an Agent Based Approach [1]
- A.Torelli: Computer Simulation of Urothelial Cells (Masterarbeit) [9]
- CC3D Quick Start Guide [10]
- A. Roch: ModUro Toolbox (Studienarbeit) [11]
- M. Riedel: ModUro Toolbox - Anbindung einer Persistenzkomponente (Studienarbeit) [12]
- M. Gumbel: Q&D-Prototyp[1]

[1] Der Q&D-Prototyp kann unter folgendem Link heruntergeladen werden: //141.19.146.103/Moduro/src/cc3d/QDToolbox

2 Grundlagen

Im ersten Abschnitt dieser Kapitel wird auf die Grundlagen der Modellbildung und Zellsimulation eingegangen. Im Projekt ModUro wird die zelluläre Modellbildung mittels CC3D durchgeführt. Hierzu wird die Funktionalität dieser Software erklärt. Anschließend werden Definitionen erläutert, die für diese Arbeit relevant sind. Abschließend werden grundlegende Aspekte zur grafischen Benutzeroberfläche erläutert.

2.1 Modellbildung und Zellsimulation

Das Urothelium kleidet als spezialisiertes, mehrschichtiges Epithel die Blase aus und verhindert den Austausch von Wasser und Toxinen zwischen Blut und Urin [4]. Die äußerste Zellstruktur des Urotheliums ist die Basalmembran. Sie stabilisiert das Gewebe und gibt der Blase ihre Form. Auf der Basalmembran befinden sich Stammzellen. Diese erschaffen mittels Zellteilung und Differenzierung zwei weitere Zelltypen: Die Basalzellen und die Intermedialzellen. Dabei stabilisieren die Basalzellen die Intermedialzellen, die den größten Anteil in der Gesamtstruktur haben. Aufgrund biochemischer Prozesse – vermutlich ausgelöst durch dem verstärkten Kontakt mit Urin in der Blase – differenzieren einige Intermedialzellen weiter in widerstandsfähigere Schirmzellen. Sie schirmen das Gewebe gegen die schädigende Säure des Urins ab. Abbildung 3 zeigt ein gesundes Urothelium sowohl im histologischen Schnitt (a) als auch in der computerbasierten Simulation (b).

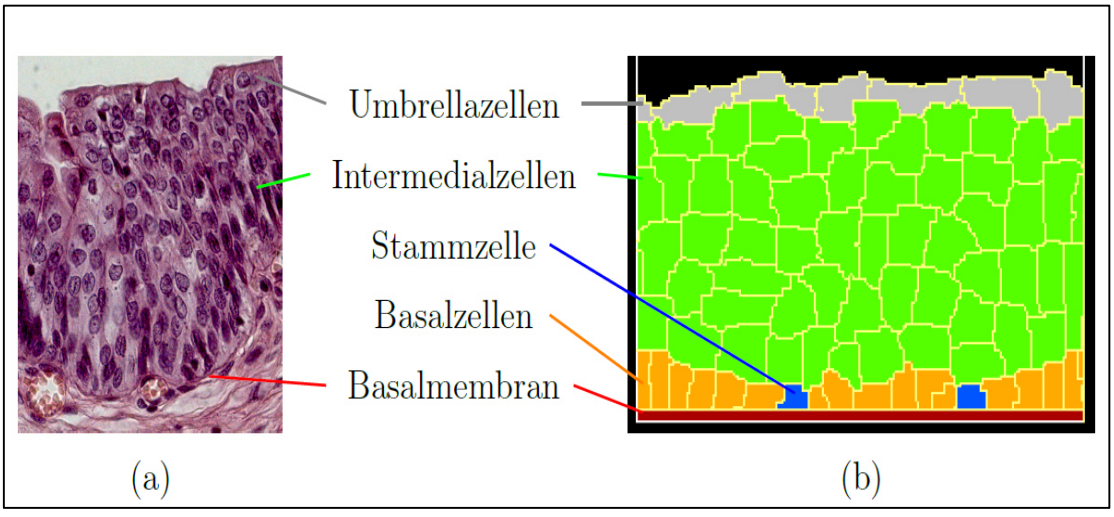

Abbildung 3: aus Paper „Modeling of the Urothelium with an Agent Based Approach“ - Gesundes Urothelium [1]

Obwohl die oben genannten Informationen wohlbekannt sind, besteht weiterhin Unsicherheit darüber, wie Krebs im Urothelium entsteht. Die aktuelle Forschung hat daher die Aufgabe, die tumorbildenden Prozesse besser zu verstehen. Das Projekt ModUro befasst sich mit Computersimulationen, mit deren Hilfe die Krebsentstehung und Progression des Blasenkarzinoms erforscht werden kann. Hierzu werden Hypothesen in mathematische Modelle übertragen und diese Modelle innerhalb einer Software umgesetzt. Anschließend werden deren Ergebnisse verglichen und ausgewertet. Die Modellierung basiert auf der „Glazier-Graner-Hogeweg“ (GGH) Methode. Der GGH-Ansatz ist ein gitterbasiertes Monte-Carlo[2] Simulationsverfahren, das eine Energiefunktion minimiert. Objekte in einer Simulation werden durch diese Energiefunktion beschrieben und beeinflusst. Darüber hinaus müssen im Rahmen von ModUro die folgenden biologischen Prozesse berücksichtigt werden:

- **Entstehung von neuen Zellen**
  Neue Zellen entstehen ausschließlich durch die Zellteilung. Die Zellteilung ist immer einem Differenzierungsprozess unterworfen und bei nährstoffabhängigen Zellen (NU = nutrient-dependent) an ein Minimum verfügbarer Nährstoffe gebunden.
- **Sterben von vorhandenen Zellen**
  Eine Zelle stirbt entweder durch Apoptose[3], oder durch einen mechanischen Prozess der Entleerung.
- **Wachstum von Zellen**
  Das Wachstum einer Zelle ist entweder unendlich (IN = infinite growth), oder nährstoffabhängig (NU = nutrient-dependent).
- **Differenzierungsprozess**
  Abbildung 4 zeigt den möglichen Differenzierungsprozess für jeden Zelltyp des Urotheliums.

[2] Der Monte-Carlo-Step („MCS“) ist eine diskrete Zeiteinheit. 1440 Monte-Carlo-Steps entsprechen dem GGH-Ansatz nach für das ModUro-Projekt einem Tag [1].

[3] Die Apoptose ist ein programmierter Selbstmord einer Zelle.

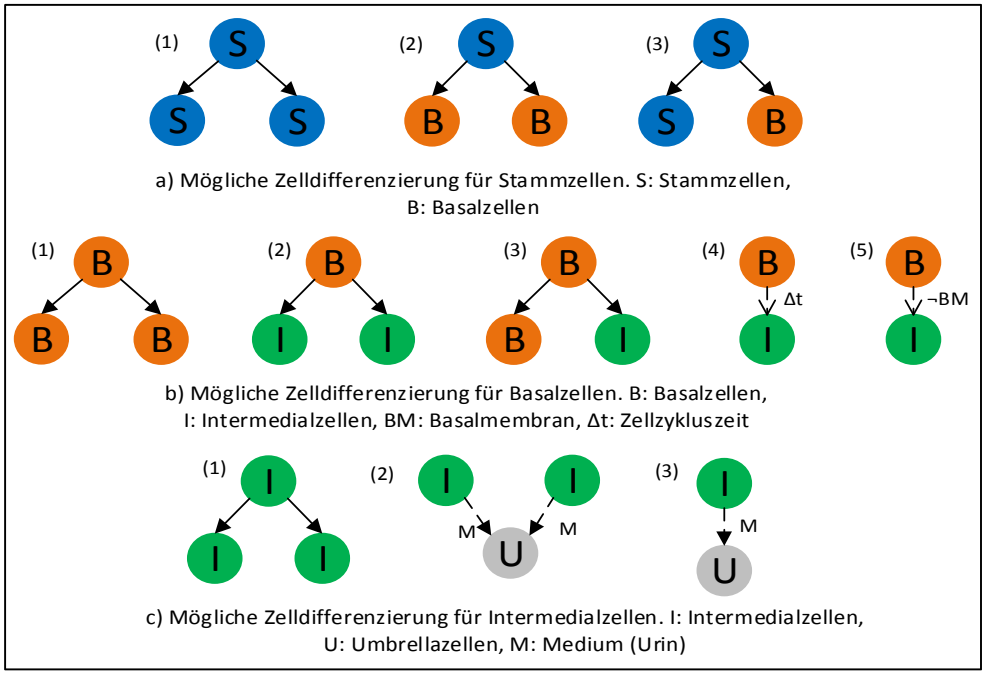

Abbildung 4: nach Paper „Modeling of the Urothelium with an Agent Based Approach“ - Zelldifferenzierung [1]

- **Sortierungsprozess**
  Das Gewebe kann sich entweder zufällig (RA = random), oder auf Grundlage der Differenzhaftungs-Hypothese (DAE = differential adhesion) sortieren.

Somit ergeben sich vier Variationen von Modellen im ModUro-Projekt:

1. NU-RA: nährstoffabhängiges Wachstum und zufälliger Sortierungsprozess
2. NU-DAE: nährstoffabhängiges Wachstum und Sortierungsprozess nach Differenzhaftungs-Hypothese
3. IN-RA: unendliches Wachstum und zufälliger Sortierungsprozess
4. IN-DAE: unendliches Wachstum und Sortierungsprozess nach Differenzhaftungs-Hypothese

Um die Stabilität und Lebensfähigkeit eines Modells zu messen wurde für das ModUro-Projekt eine Fitnessfunktion berechnet. Diese Funktion basiert auf einem Fitnesswert der Anordnung („Arrangement“) und dem Volumen („Volume“) des simulierten Gewebes. Die Fitnessfunktion lautet

$$f = \frac{f_a + f_v}{2},$$

wobei f_a „Fitness der Anordnung“ und f_v „Fitness des Volumens“ entspricht.

Die im Kapitel „Modellbildung und Zellsimulationen“ beschriebenen Hintergrundinformationen beruhen vorwiegend auf das Paper „Modeling of the Urothelium with an Agent Based Approach“ [1].

2.2 CompuCell3D

CC3D ist ein Open-Source Softwareprodukt der Indiana University in Bloomington (USA). Das Framework stellt eine Simulationsumgebung für die Modellierung komplexer virtueller Gewebe, Organe und Organismen bereit, mit der verschiedene Hypothesen getestet werden können [8]. CC3D ist in C/C++ mit Python-Erweiterungen geschrieben und bietet einfache Erweiterungsmöglichkeiten durch Python Skripte an, die in ModUro genutzt werden. Für jede CC3D-Simulation muss zum einen die Simulationsumgebung konfiguriert und zum anderen der Ablauf der Simulation implementiert werden. Die Einstellungen für die Simulationsumgebung können in Extensible Markup Language (XML), oder Python erfolgen. Der Ablauf einer Simulation wird mittels Python-Plug-Ins definiert [10]. Die Simulationsdatei CM-IN-DAE.cc3d (Code 1) zeigt ein Beispiel, wie eine Simulation in XML definiert wird.

```
1   <Simulation version="3.7.3">
2    <PythonScript Type="Python">Simulation/Urothel.py</PythonScript>
3    <Resource Type="Python">Simulation/UrotheliumConfig.py</Resource>
4    <Resource Type="Python">Simulation/UrothelSteppables.py</Resource>
5   </Simulation>
```

Code 1: Simulationsdatei CM-IN-DAE.cc3d

Eine CC3D-Simulation kann über den CC3D-Player abgespielt werden. Dabei werden in einem Verzeichnis die Ergebnisse der Simulation protokolliert, um sie später analysieren zu können. Die folgende Abbildung 5 zeigt die dabei entstehenden Ergebnisdateien aus einer Beispielsimulation.

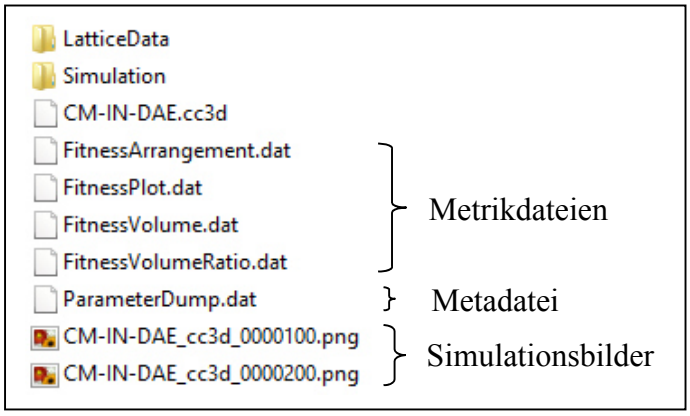

Abbildung 5: Ergebnisdateien der Simulation

2.3 ModUro

CC3D dient ausschließlich zum Ausführen und Abspielen der konfigurierten Modelle und Simulationen. Die Ergebnisdateien zu einer Simulation werden im Projekt ModUro definiert. Im Folgenden werden die Ergebnisdateien erläutert.

Metrikdateien

Eine Metrik ist hier eine Einheit, mit der eine Eigenschaft über einen Zahlenwert ausgedrückt wird. Die gemessenen Eigenschaften sind die Anordnung von Zellen, ihre Größe und der daraus errechnete Gesamtfitnesswert. Die entsprechenden Metrikdateien sind

- FitnessArrangement.dat,
- FitnessVolume.dat,
- und FitnessPlot.dat.

Hierbei handelt es sich um reine Textdateien. Beim Ausführen der Simulation über CC3D werden zwei Werte gespeichert. Zum einen der Zeitpunkt t und zum anderen der Wert der Metrik zu diesem Zeitpunkt. Zum Beispiel beinhaltet die folgende FitnessArrangement.dat (Abbildung 6) Werte für den Zeitpunkt von 0.0 bis 10.5. Die Fitness der Anordnung zum Zeitpunkt 3.0 beträgt 0.5. Die beiden anderen Metrikdateien sind analog aufgebaut.

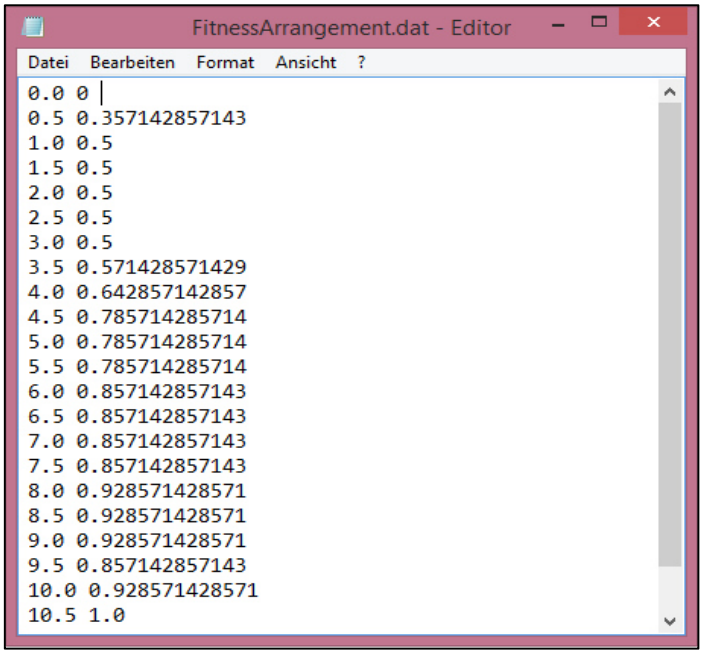

Abbildung 6: FitnessArrangement.dat

Metadatei

Die ParameterDump.dat-Datei enthält weitere Metadaten über die Simulation, wie zum Beispiel den Startzeitpunkt „startTime“ der Simulation (Abbildung 7).

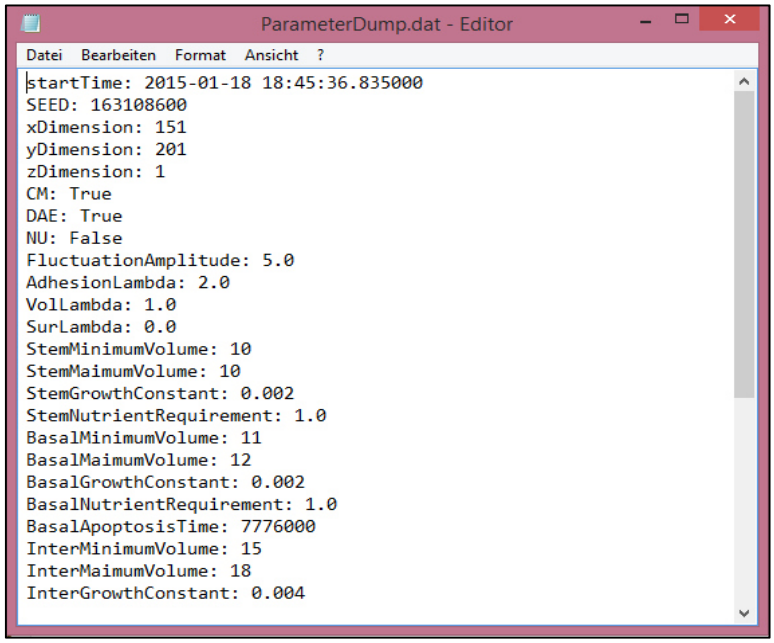

Abbildung 7: ParameterDump.dat

Simulationsbilder

CC3D speichert ebenfalls Simulationsbilder. Es kann festgelegt werden, zu welchen definierten Zeitpunkten ein Screenshot gemacht werden soll. Die gespeicherten Bilder zeigen die Zellen im Gewebe des simulierten Urotheliums. Im Folgenden sind Beispielbilder zu der Simulation zu sehen (Abbildung 8).

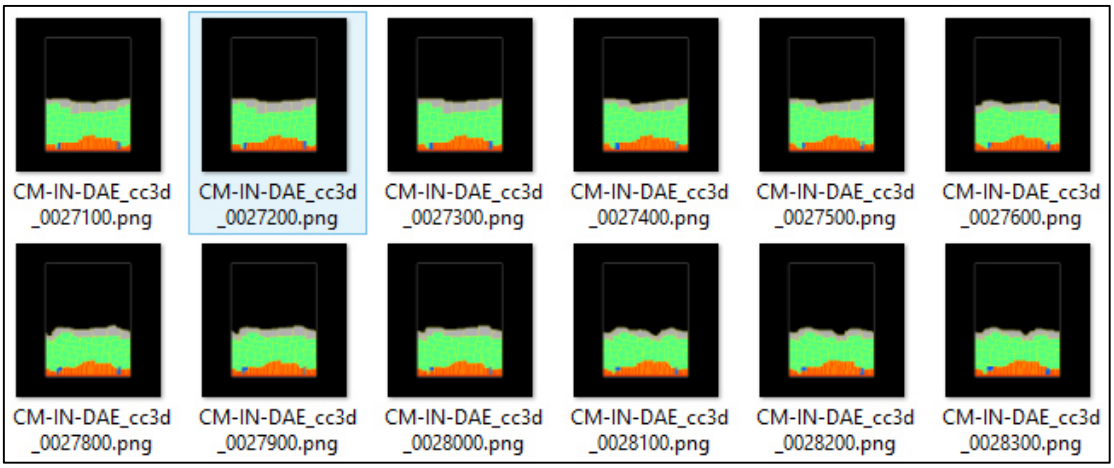

Abbildung 8: Simulationsbilder im Simulationsverzeichnis

2.4 Definitionen

Die Folgende Tabelle 1 mit Definitionen dient dazu, die in dieser Arbeit erläuterten Zusammenhänge besser verstehen zu können.

Tabelle 1: Definitionen

Begriff	**Definition**
minTime	„MinTime“ ist ein definierter Zeitpunkt.
maxTime	„MaxTime“ ist ein definierter Zeitpunkt.
toTime	„ToTime“ entspricht die Dauer einer Simulation. In einer Metrikdatei ist dieser der letzte Zeitpunktwert.
lastFitness	„LastFitness“ entspricht dem Fitnesswert vom letzten Zeitpunkt in der Gesamtfitness.
isInSteadyState	„SteadyState“ bedeutet, dass sich die Simulation in einen stabilen Zustand befindet. „IsInSteadyState“ ist definiert durch `toTime >= minTime`.
abgeschlossen	Eine Simulation ist abgeschlossen wenn `toTime >= maxTime` gilt.
abgebrochen	Eine Simulation ist abgebrochen wenn `lastFitness < 0.05 && isInSteadyState` gilt.
Projekt	Ein „Projekt“ beinhaltet Verzeichnisse („Nodes“), welche die Modellklassen und dazugehörige Simulationen enthält.
Verzeichnis/Node	„Verzeichnisse“ enthalten Simulationsverzeichnisse mit deren Metrik-, Metadateien und Simulationsbilder.
ModelType	„ModelType“ beschreibt eine Modellklasse.
MetricType	„MetricType“ beschreibt eine Metrik.

Die Definitionen „minTime“, „maxTime“, „toTime“, „lastFitness“, „inSteadyState“, „abgeschlossen“ und „abgebrochen“ wurden im ModUro-Projekt festgelegt [4].

2.5 Grafische Benutzeroberfläche

Wie es der Name bereits suggeriert, ist eine grafische Benutzeroberfläche (oder kurz GUI für Graphical User Interface) eine Software-Komponente, die es dem Benutzer ermöglicht, mit einer Anwendung in Interaktion zu treten [13, p. 255]. Diese Interaktion wird durch grafische Bedienelemente erlaubt, die zum Beispiel per Mausklick ausgewählt werden können. Auf diese Weise kann der Benutzer Funktionalitäten des Programms ausführen [14]. Jedes Bedienelement, welches grafisch dargestellt wird, ist für eine bestimmte Bedienaufgabe zuständig. Eine grafische Benutzeroberfläche wird durch geeignete Erstellung von Prototypen entwickelt. Somit können schon in früheren Phasen der Entwicklung erste Fehler erkannt, und Maßnahmen eingeleitet werden, um diese zu korrigieren [15, p. 272]. Dazu werden zu Beginn Prototypen entwickelt, die ohne Funktionalität dem Anwender vorgeführt werden. Der Anwender wird in den Entwicklungsprozess integriert und kann so Rückmeldungen zum Produkt geben [16, p. 4]. Auf die Rückmeldungen basierend werden die Anforderungen entsprechend angepasst. Für die Konzeption von Prototypen können unterschiedliche Techniken angewandt werden. In dieser Arbeit werden zunächst Entwürfe und Skizzen mit Papier und Bleistift erstellt. Anschließend wird ein GUI Prototyping Tool namens „Balsamiq“ [17] eingesetzt. Sie dient zur Erstellung von Prototypen grafischer Benutzeroberflächen am Computer.

Werkzeuge zur GUI - Erstellung

Es existieren zahlreiche Werkzeuge zur Erstellung von grafischen Benutzeroberflächen. Als Entwicklungsumgebung für ModUro-Toolbox wird „IntelliJ IDEA“ (Abbildung 9) von Jet Brains verwendet. IntelliJ bietet die Möglichkeit, externe Tools zur visuellen Erstellung von grafischen Benutzeroberflächen einzubinden. Der „JavaFX Scene Builder“ (Abbildung 10) ermöglicht Benutzeroberflächen in JavaFX ohne Codierung zu entwerfen.

[4] Definitionen wurden aus der QDT-Toolbox (//141.19.146.103/Moduro/src/cc3d/QDToolbox) entnommen.

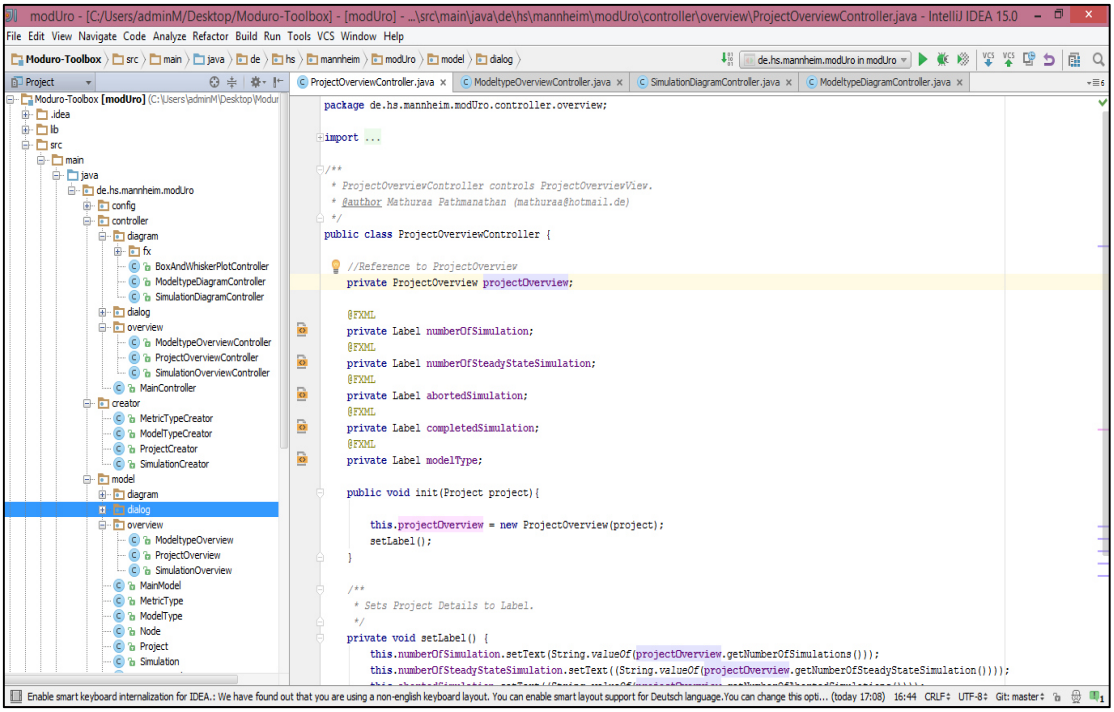

Abbildung 9: IntelliJ IDEA - Entwicklungsumgebung

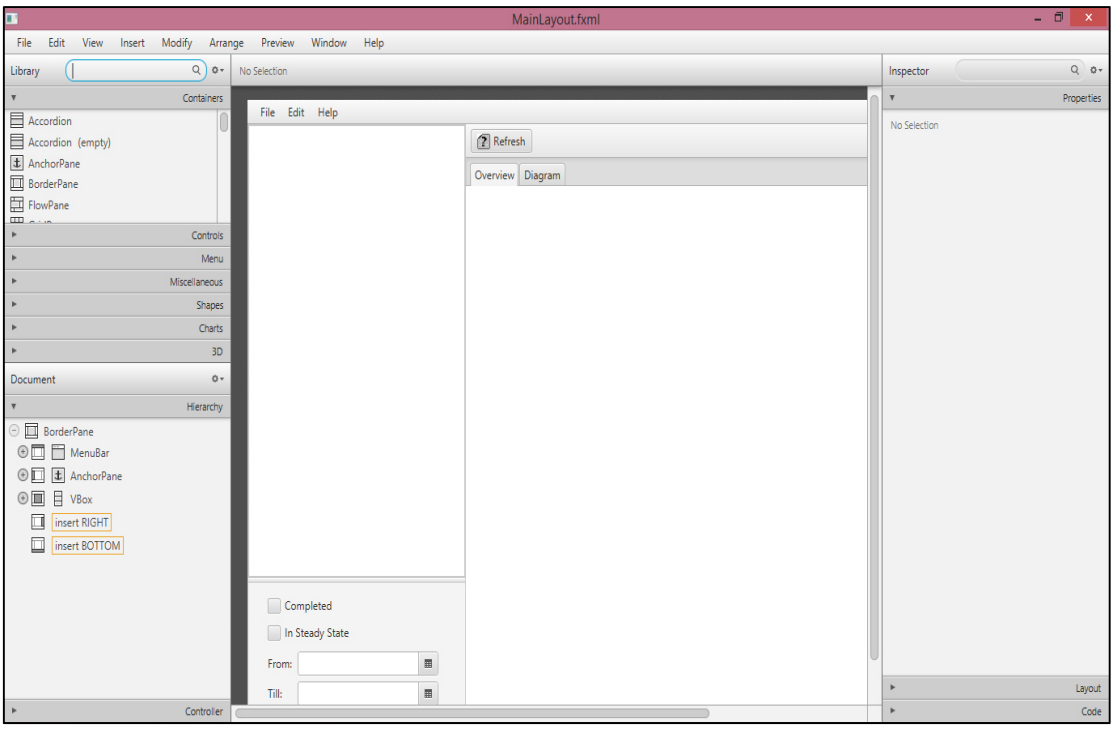

Abbildung 10: JavaFX Scene Builder

3 Analyse

Ziel der folgenden Analyse ist es, die Anforderungen und Interaktionen der Anwendung zu identifizieren und zu untersuchen [18, p. 92]. Im Kontextdiagramm (Abbildung 11) wird veranschaulicht, in welchem Kontext die ModUro-Toolbox eingesetzt wird. Des Weiteren werden zunächst Beispielszenarien vorgestellt und anschließend Anforderungen formuliert. Die Szenarien entstanden sowohl aus Gesprächen mit den Endnutzern der Anwendung, als auch aus der Evaluierung der vorherigen Studienarbeiten [11] [12] heraus.

3.1 Kontextdiagramm

„Knoten“ sind im Folgenden verschiedene Rechner, auf denen Simulationsdateien vorhanden sind. Die Simulationsdateien werden durch CC3D erzeugt und in einem Verzeichnis („Node“) abgelegt. Die ModUro-Toolbox soll lokal auf dem Rechner des Nutzers ausgeführt werden. Hierbei soll der Benutzer die zu verwendenden Verzeichnisse in die ModUro-Toolbox eintragen. Über eine Netzwerkverbindung soll die Toolbox auf die entsprechenden Verzeichnisse zugreifen und die Dateien einlesen.

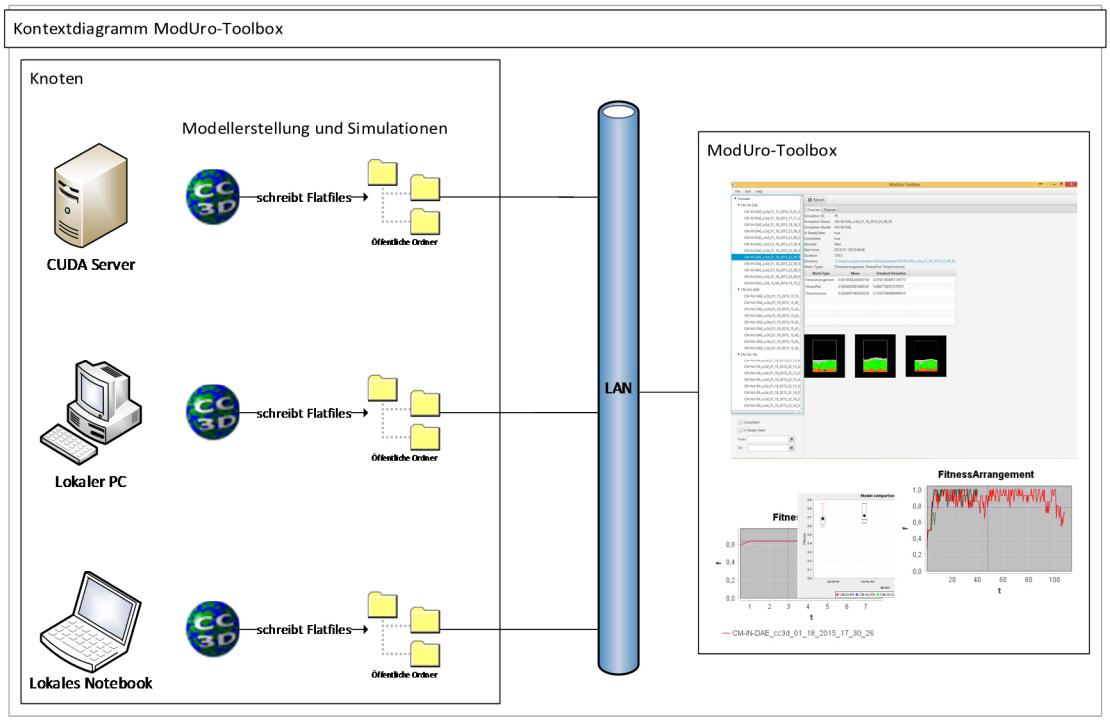

Abbildung 11: Kontextdiagramm ModUro-Toolbox

3.2 Szenarien

In diesem Kapitel werden mögliche Szenarien anhand von Anwendungsfall-diagrammen erläutert. Sie dienen dazu, unterschiedlichste Kommunikation zwischen Benutzer und Anwendung und der Interaktionen innerhalb der Anwendung erkennbar zu machen.

3.2.1 Projekt konfigurieren

Eine Computersimulation kann mit unterschiedlichen Parametern durchgeführt werden. Das sind beispielsweise Startwerte oder Modellklassen. Bestimmte Simulationsläufe mit ähnlichen Parametern können als ein Projekt oder eine Einheit gruppiert werden. Dies erfolgt durch die Konfiguration von Projekten. Die Konfiguration wird folgendermaßen vorgenommen (Abbildung 12):

- Der Benutzer vergibt einen Namen für das jeweilige Projekt.
- Zum jeweiligen Projekt fügt er Verzeichnisse („Nodes“) mit Simulationsläufen hinzu.

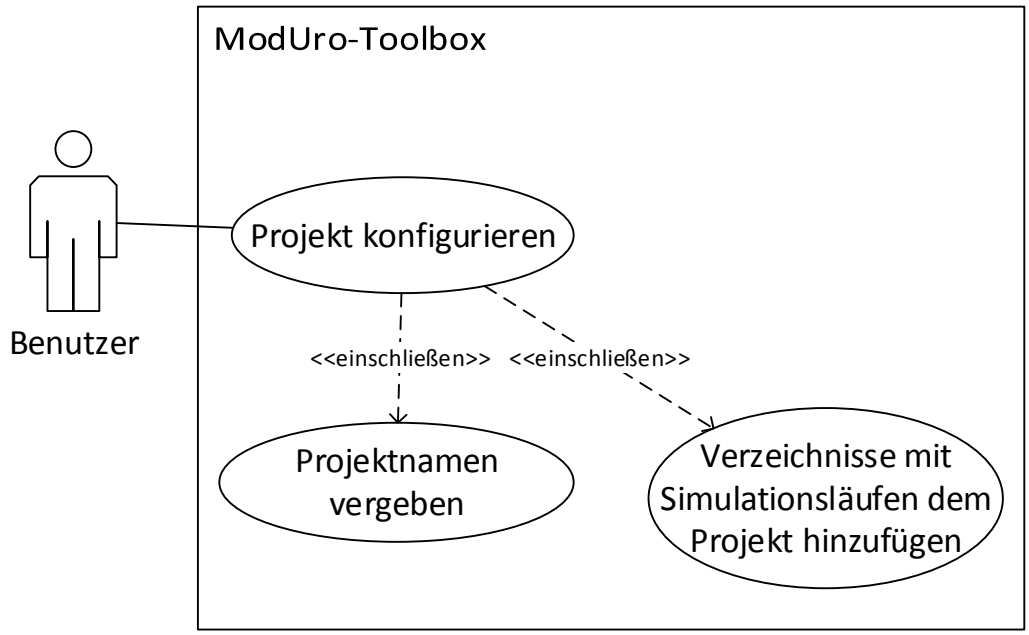

Abbildung 12: Szenario - Projekt konfigurieren

3.2.2 Projekt analysieren

Der Entwickler einer Computersimulation hat ein Interesse daran zu erfahren, wie viele Simulationsläufe im gesamten Projekt fehlgeschlagen beziehungsweise erfolgreich sind. Er benötigt eine Übersicht über die jeweiligen Simulationsläufe im Projekt, sowohl in einer textuellen, als auch einer graphenbasierten Übersicht. Die graphenbasierte Übersicht soll es ihm ermöglichen, die verschiedenen Modelle auf ihren Fitnesswert hin zu analysieren. Das Anwendungsfalldiagramm (Abbildung 13) zeigt die Anwendungsfälle beim Projekt Analysieren.

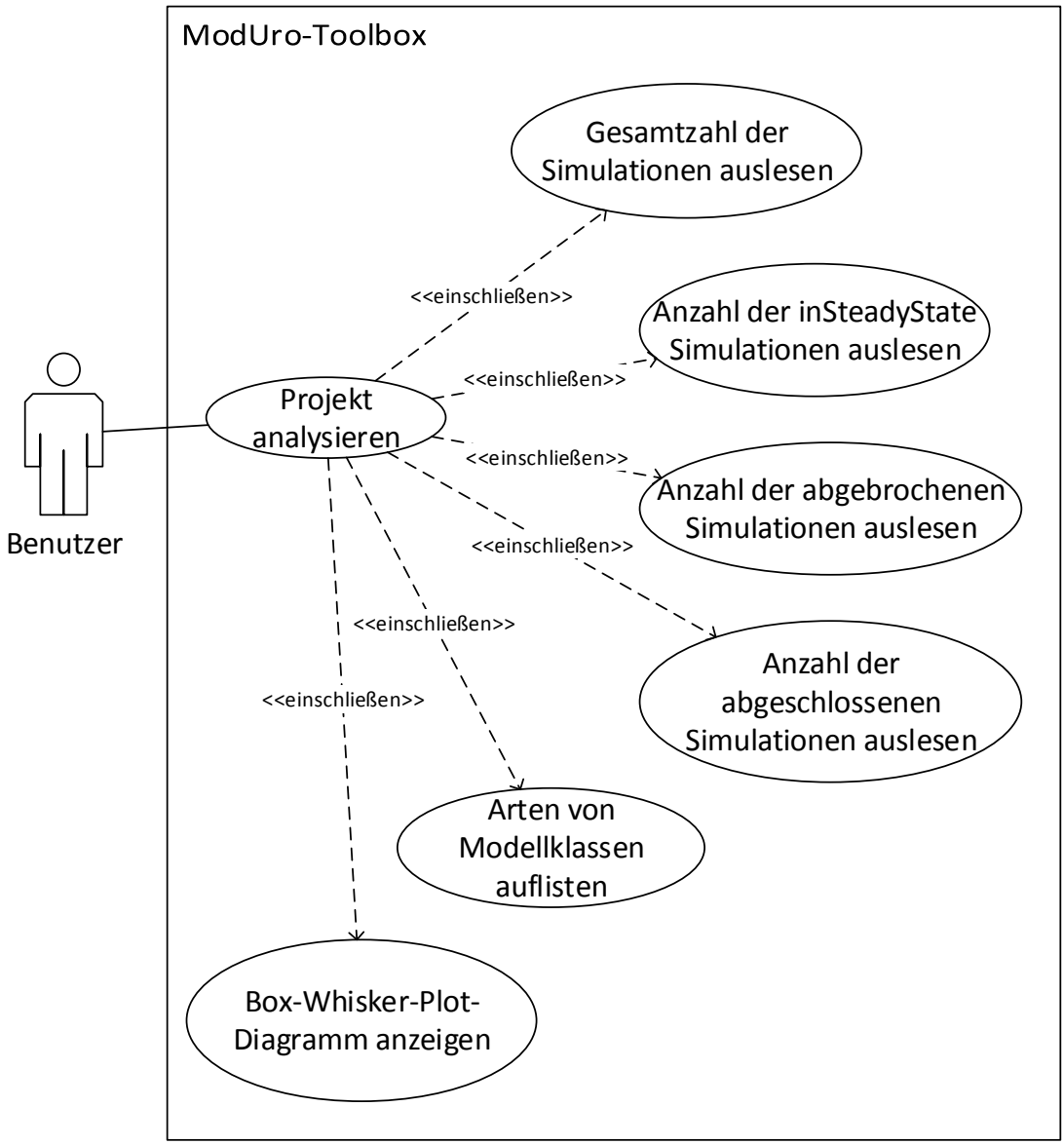

Abbildung 13: Szenario - Projekt analysieren

3.2.3 Modell analysieren

Simulationsläufe, die zu einer bestimmten Modellklasse gehören, werden zu dieser Modellklasse zusammengefasst. Ein Modell enthält somit mehrere Simulationsläufe, die in der Summe einen Fitnesswert aufweisen. Der Benutzer kann anhand einer Übersicht die erforderlichen Informationen zu einem bestimmten Modell erhalten und diesen analysieren (Abbildung 14). Die Metriken einzelner Simulationen sollen über eine graphenbasierte Darstellung angezeigt werden.

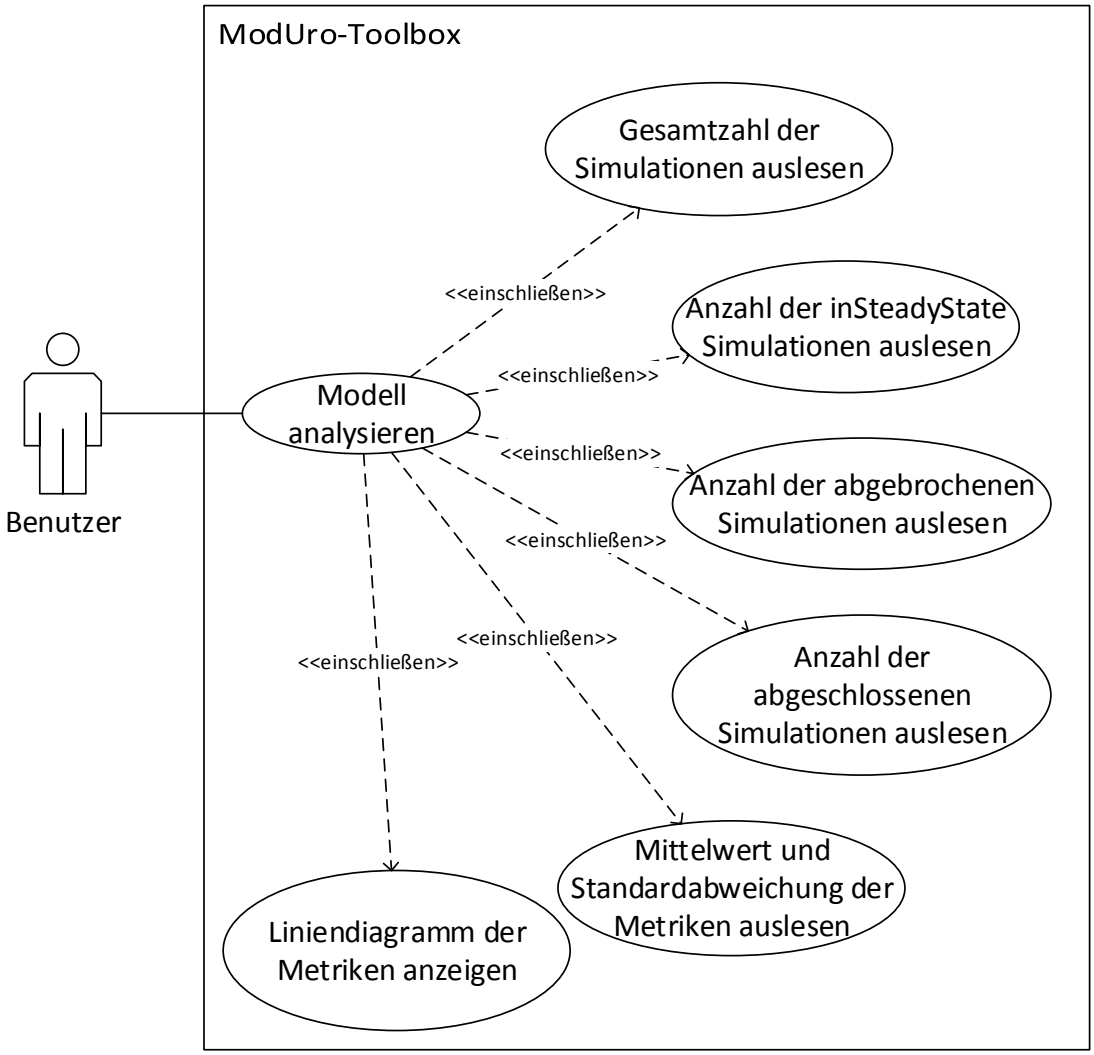

Abbildung 14: Szenario - Modell analysieren

3.2.4 Simulation analysieren

Zu einem Simulationslauf gibt es weitere Metadaten, zum Beispiel Startpunkt oder Dauer, die den Zustand zu einem gegebenen Zeitpunkt einer Simulation wiederspiegeln. Über einer textuellen Übersicht sollen diese Daten angezeigt werden. Die Simulationsbilder sollen ebenfalls zum Vergleich mit den Daten zur Analyse angezeigt werden. Das Diagramm zum Szenario „Simulation analysieren“ (Abbildung 15) stellt die möglichen Anwendungsfälle dar.

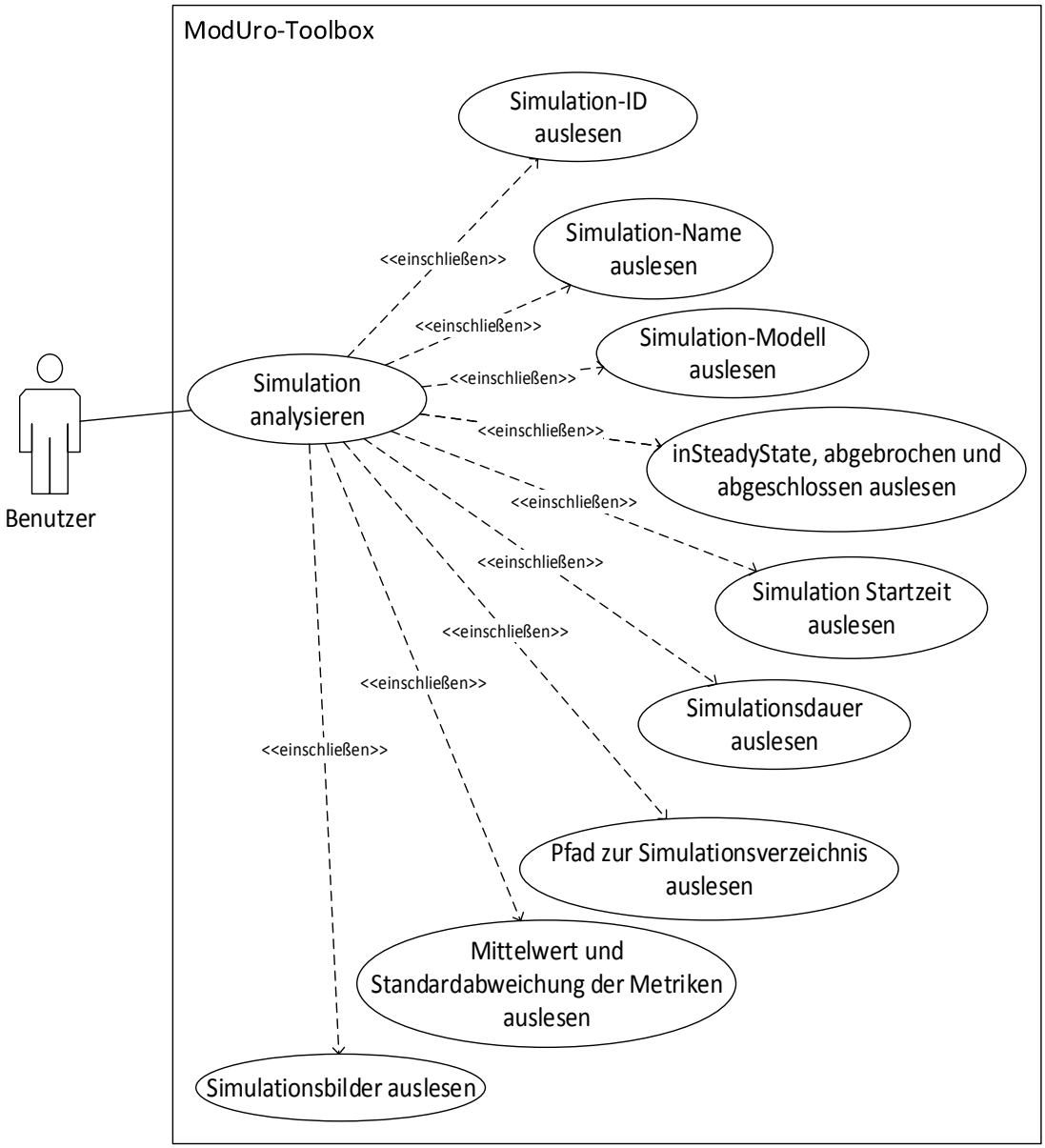

Abbildung 15: Szenario - Simulation analysieren

3.2.5 Filtern nach Modellen und Simulationen

Der Benutzer filtert bei seiner Analyse nach Simulationen oder Modellen, die für ihn von Interesse sind (Abbildung 16). Hierbei erwartet er folgende Kriterien zum Filtern:

- Zeitpunkt
- abgeschlossene Simulationen
- inSteadyState Simulationen (siehe Tabelle 1: Definitionen)

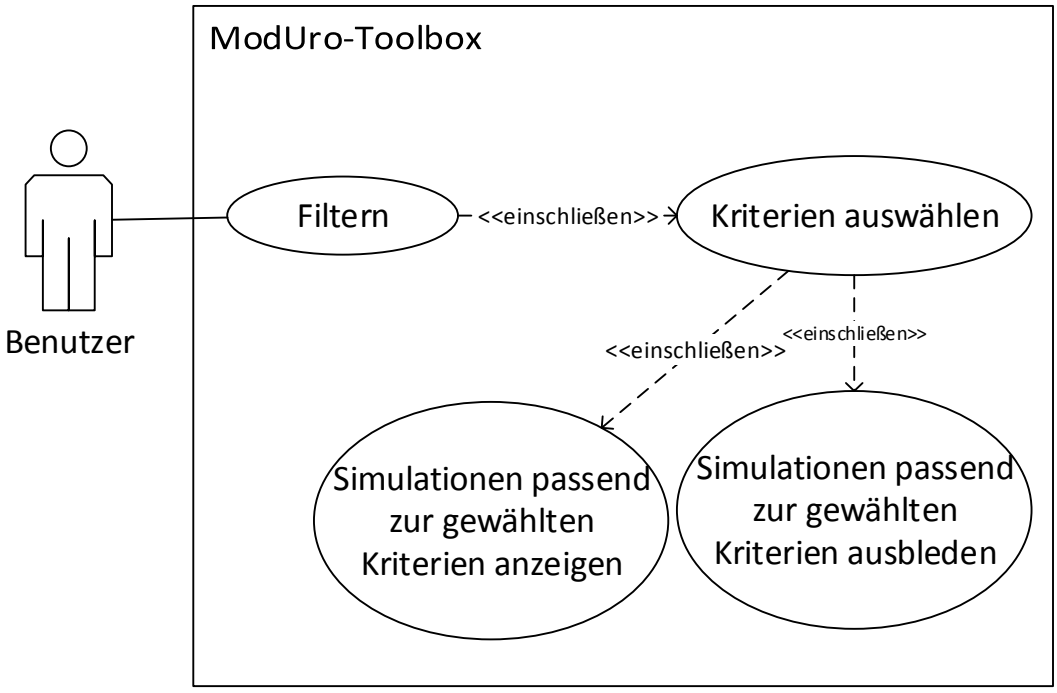

Abbildung 16: Szenario - Filtern

3.2.6 Vergleich von Metriken

Der Benutzer möchte die Metriken einer Simulation über eine graphenbasierte Darstellung miteinander vergleichen. Hierbei will er entweder Metriken von einer Simulation miteinander vergleichen, oder Metriken einer Modellklasse.

3.2.7 Exportieren von Graphen

In manchen Fällen werden Graphen von Simulationen für wissenschaftliche Arbeiten benötigt. Der Benutzer möchte über die Diagramme Graphen exportieren und weiterverwenden.

3.3 Anforderungen

Anhand der erstellten Szenarien ergeben sich funktionale- und nichtfunktionale Anforderungen, die hier aufgeführt werden. Gefordert war eine lauffähige Toolbox, mit der sich die Ergebnisse der rechenintensiven CC3D-Simulationen in einer grafischen Übersicht, zur einfachen Analyse der Ergebnisse, darstellen lassen.

3.3.1 Funktionale Anforderungen (FA)

Die folgenden funktionalen Anforderungen beschreiben die gewünschten Funktionalitäten der Anwendung. Einige der Anforderungen wurden bereits in der Studienarbeit von Andreas Roch herausgearbeitet [11].

#FA0001:	Die Toolbox soll in Java 8 mit einer JavaFX GUI realisiert werden.
#FA0010:	Die Toolbox muss Simulationen von verschiedenen lokalen Netzwerk-Dateisystemen („Knoten“) sammeln und auswerten können.
#FA0020:	Die Toolbox muss mehrere Projekte verwalten können.
#FA0021:	Die Toolbox muss mehrere Knoten verwalten können.
#FA0022:	Ein Projekt muss mehrere Nodes beinhalten können.
#FA0030:	Ein Projekt muss in einer Baumansicht mit der grafischen Oberfläche angezeigt werden können.
#FA0040:	Die Toolbox muss ermöglichen bestimmte Modellklassen als ein Projekt zu gruppieren.
#FA0041:	Die Toolbox muss ermöglichen bestimmte Simulationsläufe als eine Modellklasse zu gruppieren.
#FA0042:	Die Toolbox muss ermöglichen verschiedene Metriken zu einer Simulation zu zuordnen.
#FA0050:	In einer Projektübersicht müssen Angaben über Gesamtzahl der Simulationen im Projekt, Anzahl der abgebrochenen Simulationen, Anzahl der steady-state Simulationen, Anzahl der abgeschlossenen Simulationen und Arten der Modellklassen angezeigt werden.

#FA0051: In einer Modellklassenübersicht müssen Angaben über Gesamtzahl der Simulationen im Modell, Anzahl der abgebrochenen Simulationen, Anzahl der steady-state Simulationen, Anzahl der abgeschlossenen Simulationen und Arten der Metriken und deren Gesamtmetrik (Mittelwert und Standardabweichung) angezeigt werden.

#FA0052: In einer Simulationsübersicht müssen Angaben zur Simulation angezeigt werden:
- Simulation-ID, SimulationName, Anfangszeit, Modellklasse, Abgebrochen, Abgeschlossen, InSteadyState, Dauer, Pfad zur Simulation
- Arten der Metriken mit deren Mittelwerte und Standard-abweichungen
- Simulationsbilder (20%, 50% und 80%)

#FA0060: In einer Projektdiagrammübersicht muss ein Box-Whisker-Plot Diagramm angezeigt werden, welches die Fitness der einzelnen Modellklassen graphenbasiert darstellt.

#FA0061: In einer Modellklassendiagrammübersicht müssen die Simulationen der ausgewählten Modellklasse in einem Liniendiagramm angezeigt werden.

#FA0062: In der Modellklassendiagrammübersicht muss es möglich sein, zwischen den verschiedenen Metriken zu navigieren und deren Fitnessverlauf in einem Liniendiagramm anzuzeigen.

#FA0063: In einer Simulationsdiagrammübersicht muss zu der ausgewählten Simulation ein Liniendiagramm angezeigt werden.

#FA0064: In der Simulationsdiagrammübersicht muss es möglich sein, zwischen den verschiedenen Metriken zu navigieren und die entsprechenden Liniendiagramme anzuzeigen.

#FA0070: Über einen Filter soll das Filtern von einzelnen Simulationen, Modellen und Projekten möglich sein.

#FA0080: Eine Exportfunktion muss das Exportieren von ausgewählten Graphen ermöglichen (nach Tikz-Format).

#FA0090: Über eine Importfunktion soll das automatische Aktualisieren und Hinzufügen nicht vorhandener Simulationen erfolgen.

3.3.2 Nichtfunktionale Anforderungen (NA)

Die nichtfunktionalen Anforderungen beschreiben, in welcher Qualität die genannten Funktionen umgesetzt werden sollen.

Funktionalität

Die Toolbox soll Hauptfunktionen zum Lesen und Analysieren von Daten der Simulationen beinhalten. Diese sollen eine effektive Analyse von Simulationen ermöglichen.

Korrektheit

Daten von Simulationsmodellen können erst richtig analysiert werden, wenn sie von der Toolbox fehlerfrei eingelesen und angezeigt werden. Daher ist die Korrektheit eine Voraussetzung für den Einsatz dieser Anwendung.

Erweiterbarkeit

Die Entwicklung von Computermodellen ist einem ständigen Wandel ausgesetzt. Es kann unter Umständen der Fall sein, dass sich die Datenformate der Ergebnisdateien ändern, oder neue Metriken und Modelle hinzukommen, die die Fitness einer Simulation ausmachen. Um dies zu gewährleisten, müssen die Funktionalitäten generisch sein, damit eine einfache Integration möglich ist.

Benutzbarkeit

Da es sich bei der Anwendung um eine grafische Benutzeroberfläche handelt, spielt die Benutzbarkeit eine sehr wichtige Rolle. Die Anwendung muss den Benutzer dabei helfen seine Ziele effektiv, effizient und zufriedenstellend zu erreichen [19]. Hierbei sollte das Design der grafischen Benutzeroberfläche den Dialogprinzipien [20] der Benutzbarkeit entsprechen. Somit soll die Anwendung für den Benutzer leicht erlernbar und einfach benutzbar sein.

3.4 Anwendungsfälle

Das Anwendungsfalldiagramm ModUro-Toolbox (Abbildung 17) stellt die identifizierten Anwendungsfälle nochmals übersichtlich dar. Über die Anwendungsfälle wird genau festgelegt, welche Interaktionen zwischen Nutzer und Anwendung auftreten können und welche Aktionen das System ausführen muss.

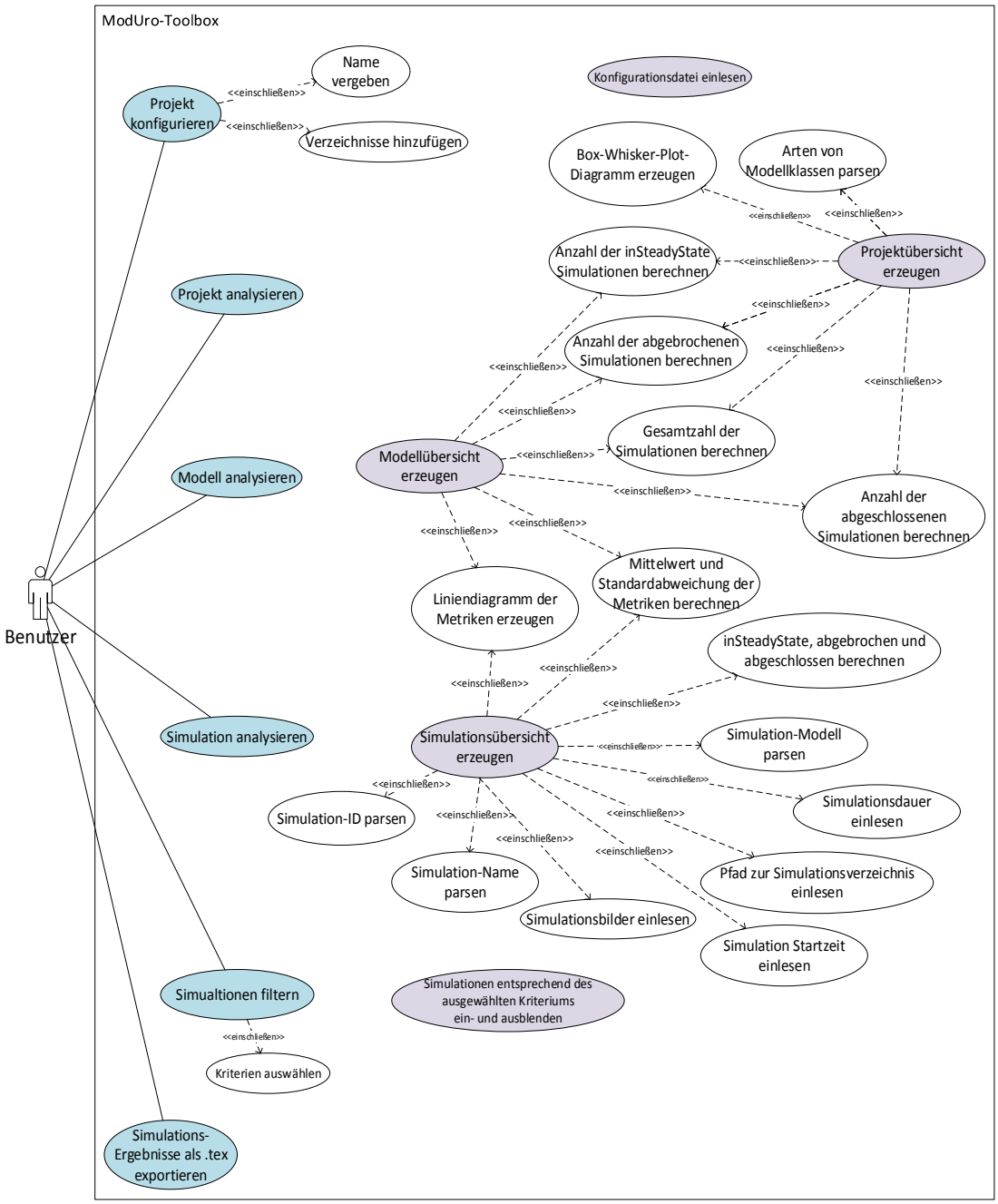

Abbildung 17: Anwendungsfalldiagramm ModUro-Toolbox

4 Design

In diesem Kapitel wird die Architektur der Anwendung erläutert. Das Paket- und Klassendiagramm beschreiben, wie die Anwendung aufgeteilt ist und in welchen Beziehungen die einzelnen Klassen zueinander stehen. Im letzten Unterkapitel wird der Prototyp von der grafischen Benutzeroberfläche erläutert.

4.1 Paketdiagramm

Die Toolbox wurde nach dem Model-View-Controller (MVC) Pattern konzipiert. Durch die klare Trennung zwischen Model, Controller und View weisen die einzelnen Komponenten eine hohe Kapselung auf [21, pp. 520-521]. Somit können die Komponenten bei Bedarf gegen andere Komponente einfacher ausgetauscht werden. Jeder View besitzt seinen eigenen Controller, der die Kommunikation zwischen View und Model steuert (Abbildung 18 (1) und (2)). Alle Views in der Anwendung werden in einem Gesamtview („MainLayout.fxml“) vereint, der die einzelnen Teil-Views beinhaltet. Der zum Gesamtview gehörende Controller ist der „Main-Controller“. Dieser nimmt die Aktionen vom Gesamtview entgegen und leitet sie an die entsprechenden Komponenten weiter oder holt Daten aus dem Model (1). Er ist ebenfalls für die Generierung der Controller der Teil-Views zuständig.

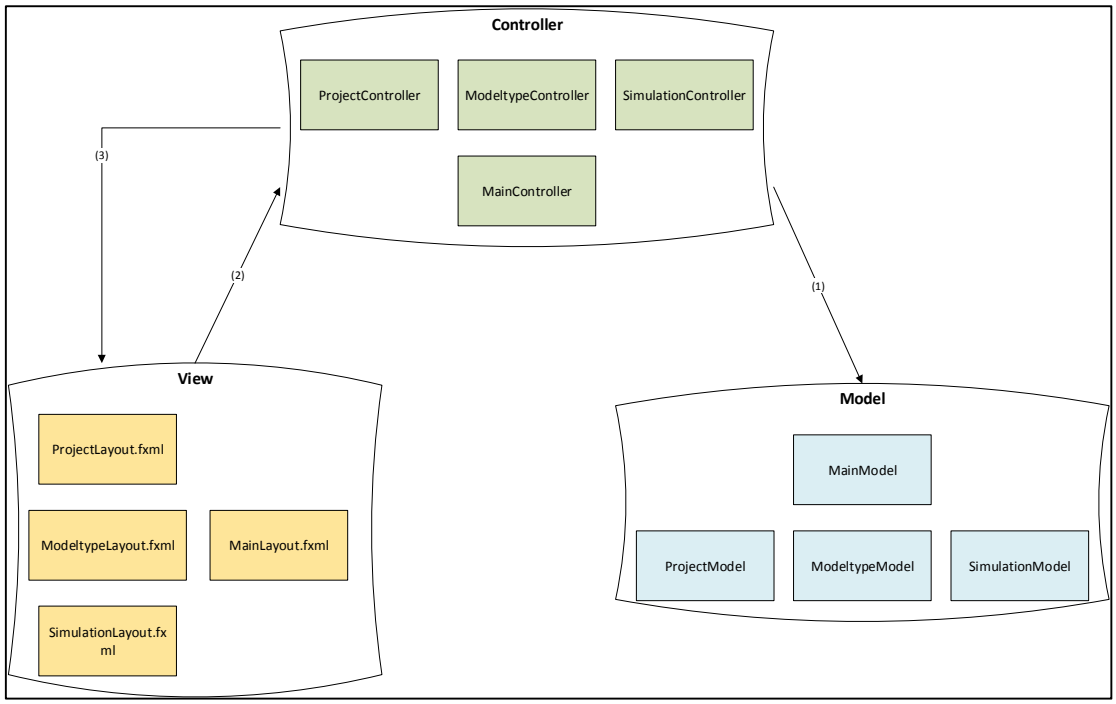

Abbildung 18: Abstrakte Darstellung des MVC-Patterns der ModUro-Toolbox

de.hs.mannheim.modUro

Das nachfolgende Paketdiagramm (Abbildung 19) stellt die grundlegende Unterteilung der Anwendung dar.

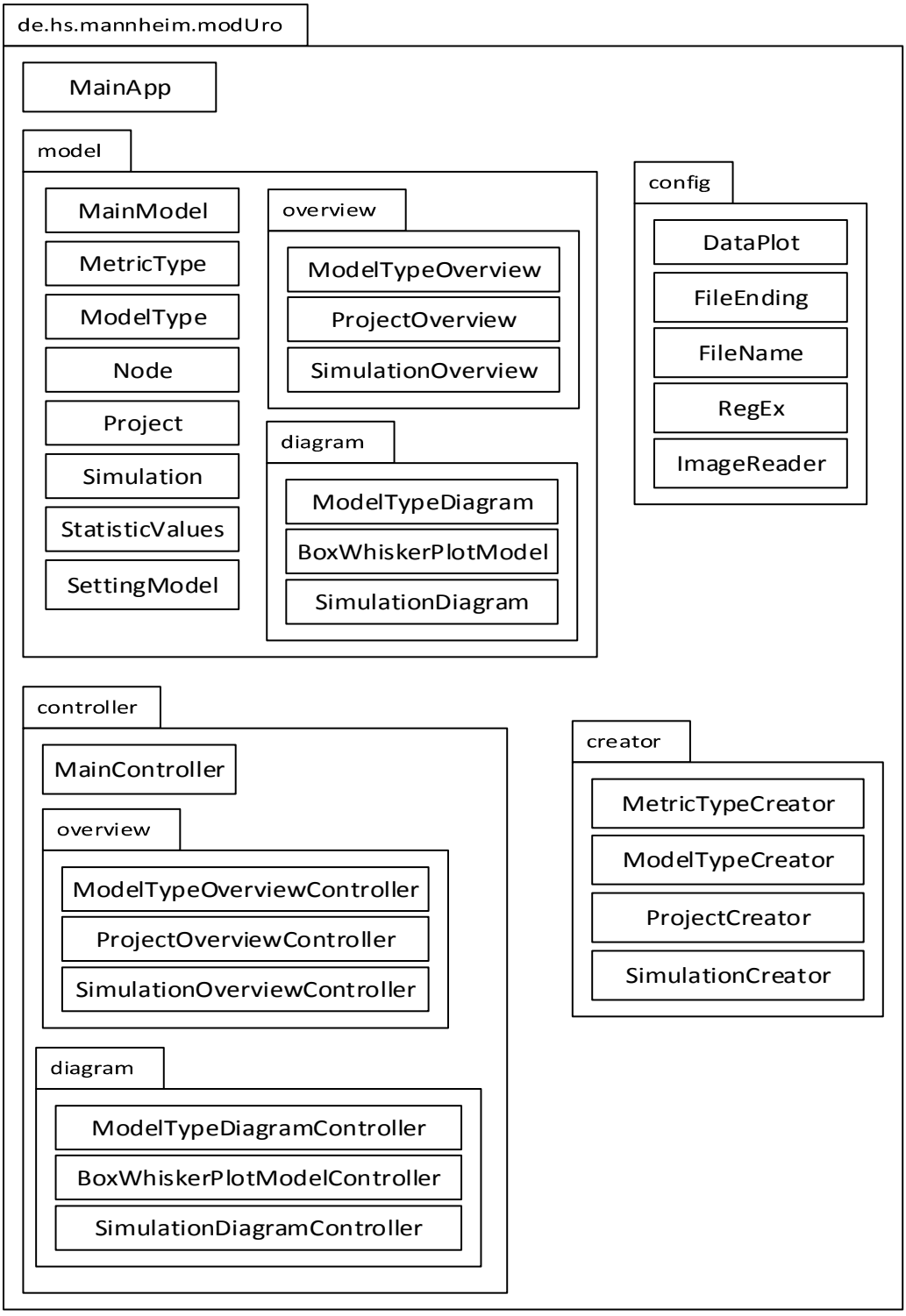

Abbildung 19: Paketdiagramm ModUro-Toolbox

Das Hauptpaket „de.hs.mannheim.modUro“ beinhaltet die Klasse „MainApp“ und die Unterpakete „model“, „controller“, „creator“ und „config“. Die FXML-Dateien für den View werden als Ressourcen eingebunden.

model

Im Paket „model“ befinden sich Klassen für die Datenhaltung. Das „MainModel“ ist das Hauptmodel, welches eine Liste der konfigurierten Projekte mit den Simulationsläufen enthält. Mittels „MainModel“ wird später eine Baumansicht mit den Projekten erzeugt (siehe Kapitel „Umsetzung“). Die Unterpakete „overview“ und „diagram“ halten die Daten für die textuelle Übersicht und graphischen Diagramme vor.

controller

Das „controller“-Paket enthält die Controller für die Kommunikation zwischen dem View und dem Model. Der Hauptcontroller „MainController“ bearbeitet die wesentlichen Aktionen des Benutzers. Im Unterpaket „overview“ und „diagram“ enthaltene Controller werden erzeugt, sobald die dazugehörigen Views aufgerufen werden. Die Controller übernehmen für die jeweiligen Teil-Views die Kommunikation mit dem Model.

creator

Klassen im Paket „creator“ sind sogenannte „Hilfsklassen“, die bei der Erzeugung von Objekte wie „Project“, „Node“, „ModelType“, „Simulation“ und „MetricType“ behilflich sind.

config

Im „config“ vorhandene Klassen dienen der Konfiguration. Hier können wesentliche Einstellungen konfiguriert werden.

resources/FXML

FXML ist eine XML-basierte Markup-Sprache, die verwendet wird, um den Objektbaum von mit JavaFX erstellten Benutzeroberflächen zu spezifizieren [22, p. 491]. Die Views von ModUro-Toolbox werden mittels FXML beschrieben und als

Ressourcen zur Laufzeit geladen. In der folgenden Abbildung 20 werden die FXML-Dateien zu den verschiedenen Teil-Views aufgelistet.

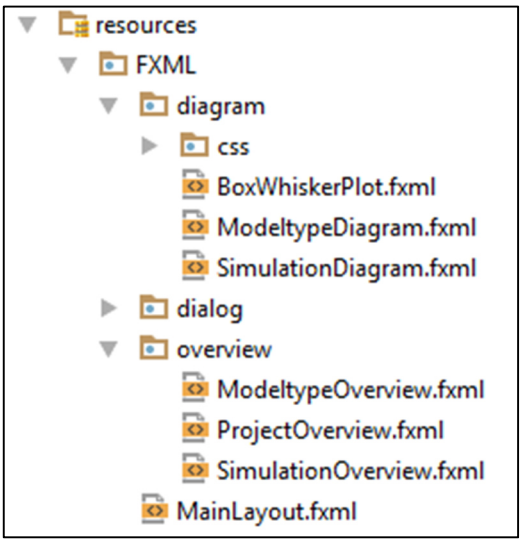

Abbildung 20: resources/FXML

Der Gesamtview, das Hauptfenster der Anwendung, wird durch MainLayout.fxml definiert. In ihr werden zur Laufzeit die weiteren Views geladen. Im „overview“ enthaltene Teil-Views stellen textuelle Informationen dar. Diagram-Views erzeugen statistische Diagramme zur graphischen Analyse der Simulationen. Der folgende Code 2 zeigt wie ein View in FXML aufgebaut ist.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <?import javafx.scene.control.*?>
3   <?import javafx.scene.layout.*?>
4   <?import java.lang.*?>
5   <?import javafx.scene.chart.*?>
6
7   <ScrollPane xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
8   fx:controller="de.hs.mannheim.modUro.controller.diagram.BoxAndWhiskerPlotController">
9      <content>
10           <TitledPane fx:id="boxWhiskerPane"/>
11     </content>
12  </ScrollPane>
```

Code 2: BoxWhiskerPlot.fxml

Über das Attribut `fx:controller` ist die Kommunikation zwischen dem View und dem Controller sichergestellt.

4.2 Klassendiagramm

Im Folgenden werden die Beziehungen zwischen den Klassen schrittweise erläutert. Im Anhang befindet sich das vollständige Klassendiagramm zur ModUro-Toolbox.

4.2.1 Model

Das Klassendiagramm - Model (Abbildung 21) beschreibt die Beziehungen der Modelklassen untereinander. In der Tabelle 2 werden die Klassen genauer erläutert.

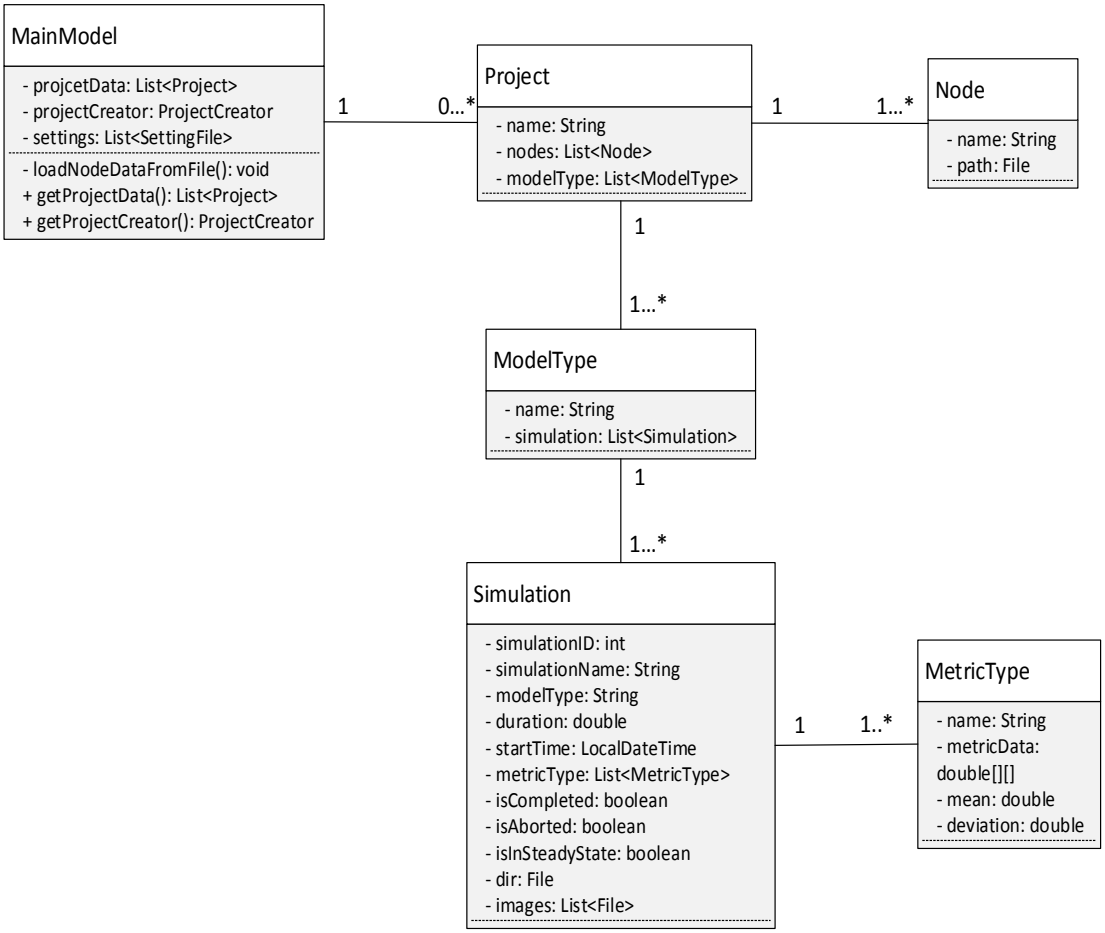

Abbildung 21: Klassendiagramm - Model

Tabelle 2: Beschreibung der Modellklassen

Klasse	Beschreibung
MainModel	Die Klasse „MainModel“ enthält eine Liste mit „SettingFiles“. In den „SettingFiles“ befinden sich Konfigurationsdaten zum Projekt, wie Projektname, Verzeichnisname und Pfade. Über die „loadNodeData-FromFile()“-Methode werden die Konfigurationen in die Toolbox geladen. Das „MainModel“ beinhaltet ebenfalls eine Liste mit Projekten, die später in der Baumansicht angezeigt werden.
Project	Das „Project“-Objekt repräsentiert eine Projektinstanz. Das Objekt besteht aus einen Namen, welches der Name vom Projekt ist. Es beinhaltet ebenfalls eine Liste von Verzeichnissen „Nodes“ und „ModelTypes“. Im Projekt muss immer eine „Node“-Instanz vorhanden sein.
Node	„Node“ beschreibt einen Pfad zu einem Verzeichnis. Über das Attribut „name“ kann ein sprechender Name für das Verzeichnis übergeben werden.
ModelType	Eine „ModelType“-Instanz beinhaltet den Namen der entsprechenden Modellklasse sowie eine Liste von Simulationen, die der Modellklasse gehören. Jede Modellklasse mit deren Simulationen gehört genau zu einem Projekt.
Simulation	Die „Simulation“-Klasse dient zur Erzeugung von einer „Simulation“-Instanz. In ihr sind die Informationen über eine Simulation vorhanden. Jede Simulation gehört genau einer Modellklasse an und beinhaltet mindestens eine Metrik. Ohne eine Metrik ist die Simulation nutzlos und wird in die Toolbox nicht eingelesen.
MetricType	„MetricType“ repräsentiert eine Metrik, welche die Fitnessdaten beinhaltet.

4.2.2 MainModel, MainController und MainLayout

In JavaFX kann ein View entweder in einer Java Klasse erzeugt oder in der sogenannten „FXML“-Format definiert werden [23, p. 9]. Alle Views in ModUro-Toolbox wurden wie im Kapitel „Paketdiagramm“ >> „resources/FXML“ beschrieben in FXML definiert. Der „MainController“ greift auf die Benutzerelemente in dem View über sogenannten „Annotationen“ (`@FXML`) zu. Hierzu werden in der FXML-Datei IDs vergeben, die der „MainController“ anspricht [24, p. 122].

Beispielweise wird im „MainLayout“ das Steuerelement „TreeView“ folgendermaßen mit der ID `projectTree` definiert:

```
1   <TreeView fx:id="projectTree" prefHeight="184.0"/>
```

Code 3: TreeView Definition in FXML

Im „MainController“ wird auf das „Tree-View“ Element über `@FXML` zugegriffen.

```
1   @FXML
2   private TreeView<String> projectTree;
```

Code 4: @FXML Annotation

Im folgenden Klassendiagramm (Abbildung 22) wird auf die Beziehung zwischen MainModel, MainController und MainLayout eingegangen.

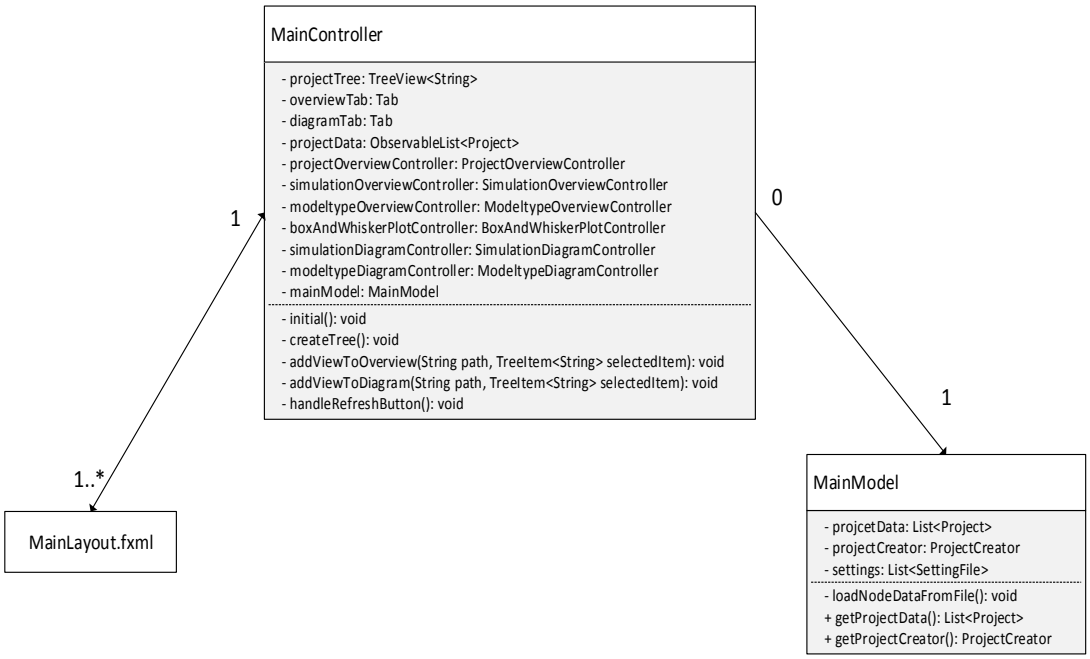

Abbildung 22: MainModel, MainController und Mainlayout

Tabelle 3: Beschreibung von MainModel und MainController

Klasse	Beschreibung
MainModel	Das „MainModel“ steht für sich allein und hat keine Referenzen zum „MainController“ und zum „MainLayout“.
MainController	Der „MainController“ enthält das „MainModel“. Er greift auf die Projektdaten im „MainModel“ zu und stellt diese für die View zur Verfügung. Über die FXML-Annotation kommuniziert er mit den Steuerelementen in der View. Der „MainController“ erzeugt ebenfalls die Controller von den Teil-Views. Die „createTree()“-Methode erzeugt anhand der Projektdaten die Baumansicht, in der die Projekte aufgelistet werden.

Analog zu „MainModel“, „MainController“ und „MainLayout“ entsprechen die Beziehungen der Teil-Views mit deren Controller und Modelle.

4.3 Sequenzdiagramm

Das folgende Sequenzdiagramm (Abbildung 23) gibt eine Übersicht über die Interaktionen zwischen den Klassen im Paket „creator“.

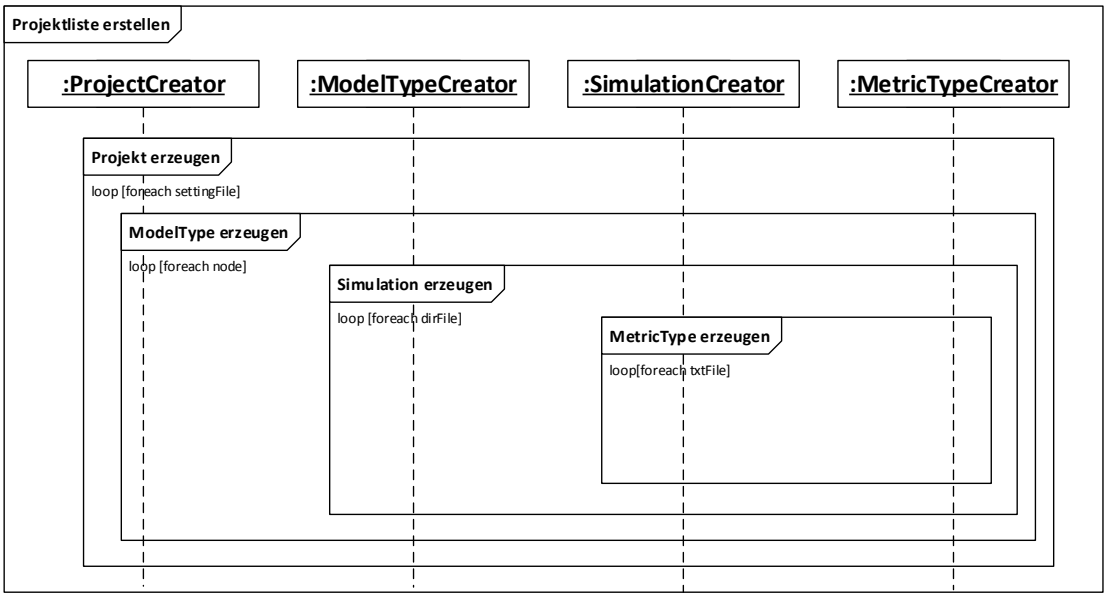

Abbildung 23: Überblick - Projektliste erstellen

Das Sequenzdiagramm „Projekt erzeugen“ (Abbildung 24) beschreibt den Ablauf, wie eine Projektinstanz erzeugt wird. Die Klasse „ProjectCreator“ enthält eine Methode „createProject()“. Diese bekommt ein „SettingFile“ übergeben. „SettingFile“ beinhaltet Konfigurationsdaten, wie Projektname und Verzeichnisse, zu einem Projekt. Anhand der Konfigurationsdaten wird der Projektname und die Verzeichnisse („Nodes“) segmentiert („geparst“). Die „ModelTypes“ werden vom „ModelType-Creator“ erstellt.

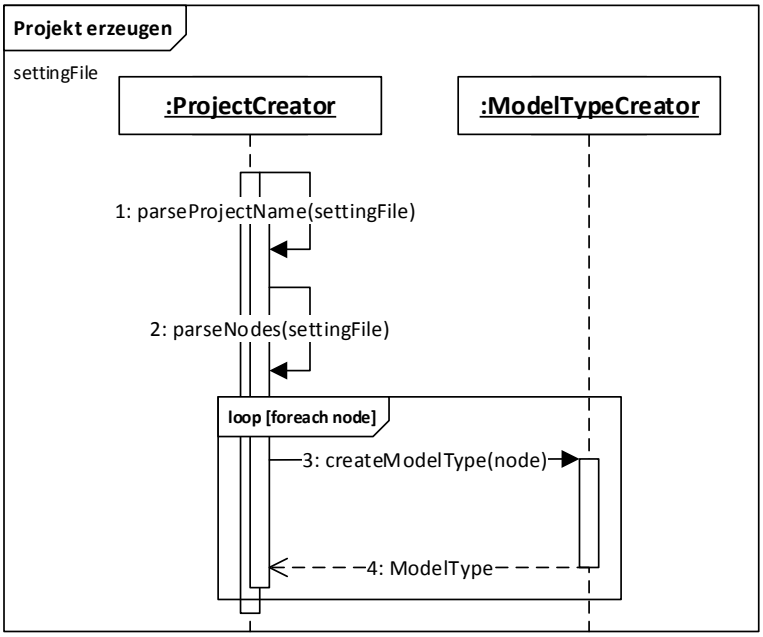

Abbildung 24: Projekt erzeugen

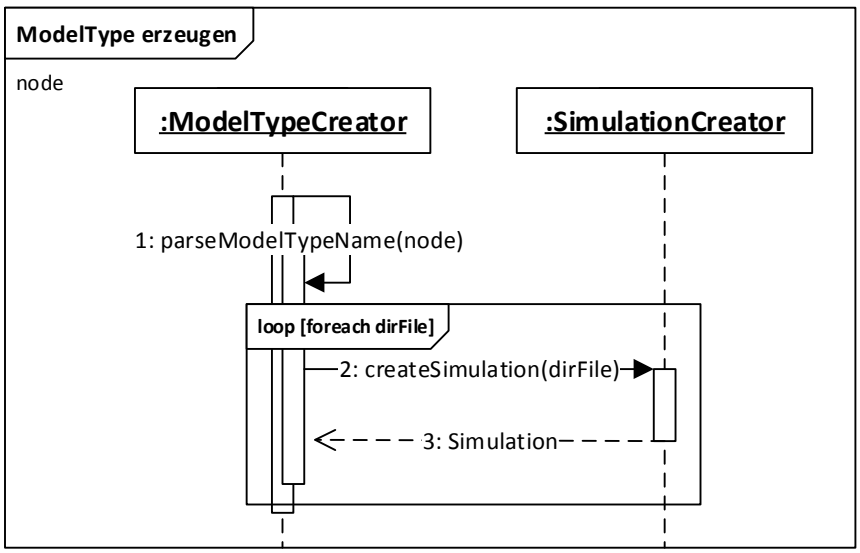

Abbildung 25: Modeltype erzeugen

Der „ModelTypeCreator“ bekommt eine Liste mit Verzeichnisse („Nodes“) übergeben. Ein Verzeichnis enthält wiederum Simulationsverzeichnisse von unterschiedlichen „ModelTypes“. Um ein „ModelyType“ zu erzeugen (Abbildung 25), muss das Ausgangsverzeichnis nacheinander abgearbeitet werden. Verzeichnisse mit ähnlichen Namensbenennung werden als ein „ModelType“ zusammengefasst. Diese zusammengefassten Verzeichnisse werden zur Erstellung von Simulationen an die Klasse „SimulationCreator“ übergeben.

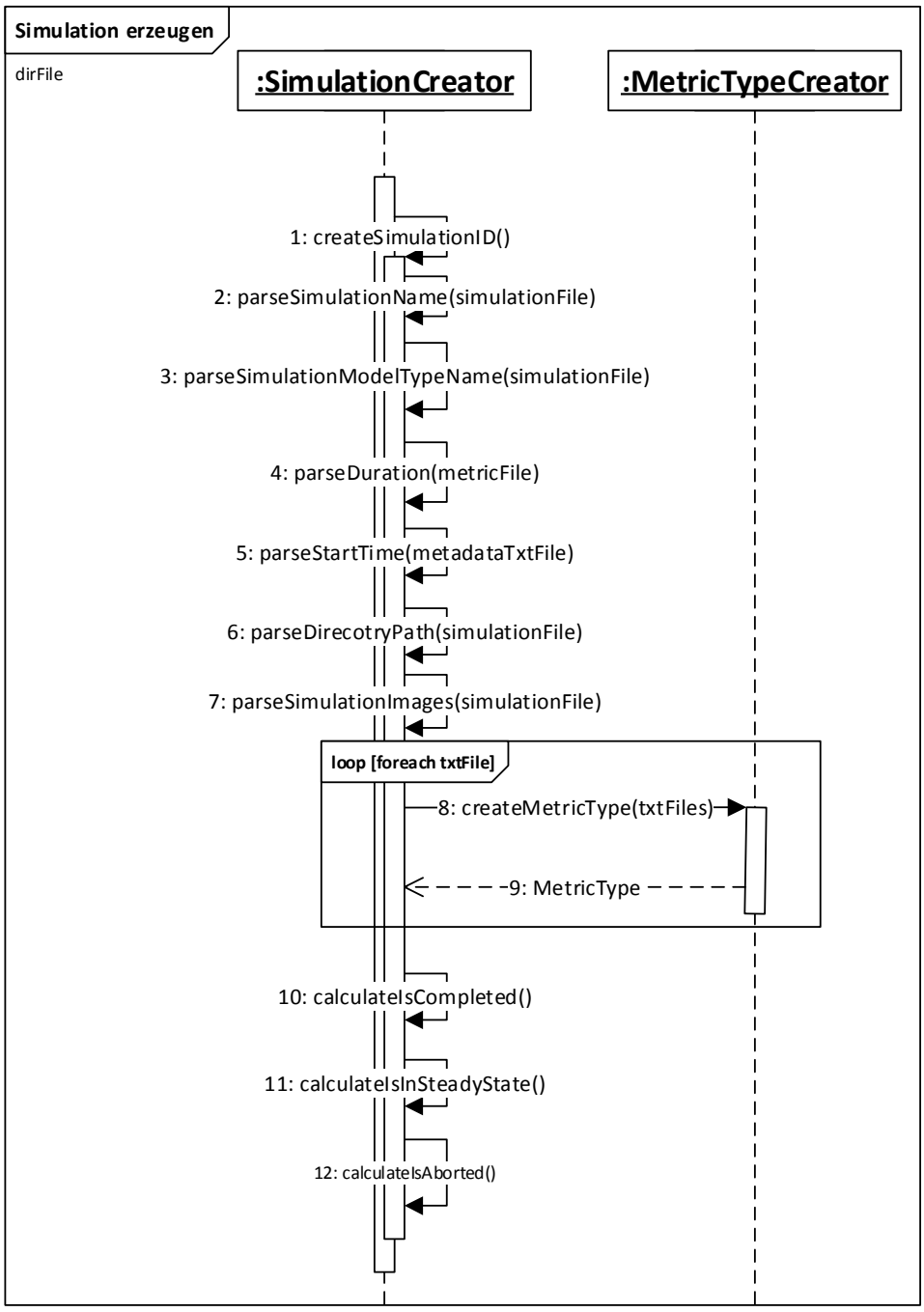

Abbildung 26: Simulation erzeugen

Analog zu „Projekt erzeugen“ und „ ModelType erzeugen“ werden bei „Simulation erzeugen“ (Abbildung 26) und „MertricType erzeugen“ (Abbildung 27) die Daten eingelesen und segmentiert (geparst).

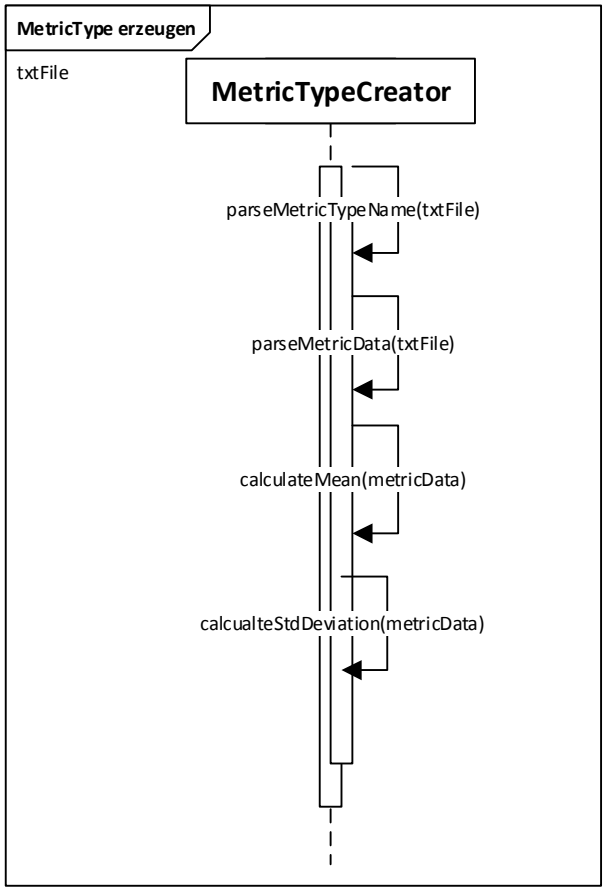

Abbildung 27: MetricType erzeugen

4.4 GUI Prototyp

Für die ersten Gestaltungsideen der grafischen Benutzeroberfläche wurden zunächst Skizzen mit Papier und Bleistift angefertigt und anschließend ein Mock-up[5] mit einem geeigneten Programm erstellt.

Mittels folgenden Entwurfs (Abbildung 28) hatte der Endanwender einen ersten Eindruck, wie die Anwendung später aussehen könnte. Bei diesem frühen Entwurf wurde festgestellt, dass Elemente nach der Evaluierung verändert werden müssten. Es gab zu viele Steuerelemente, die überflüssig waren. Diese wurden im späteren Design zusammengefasst.

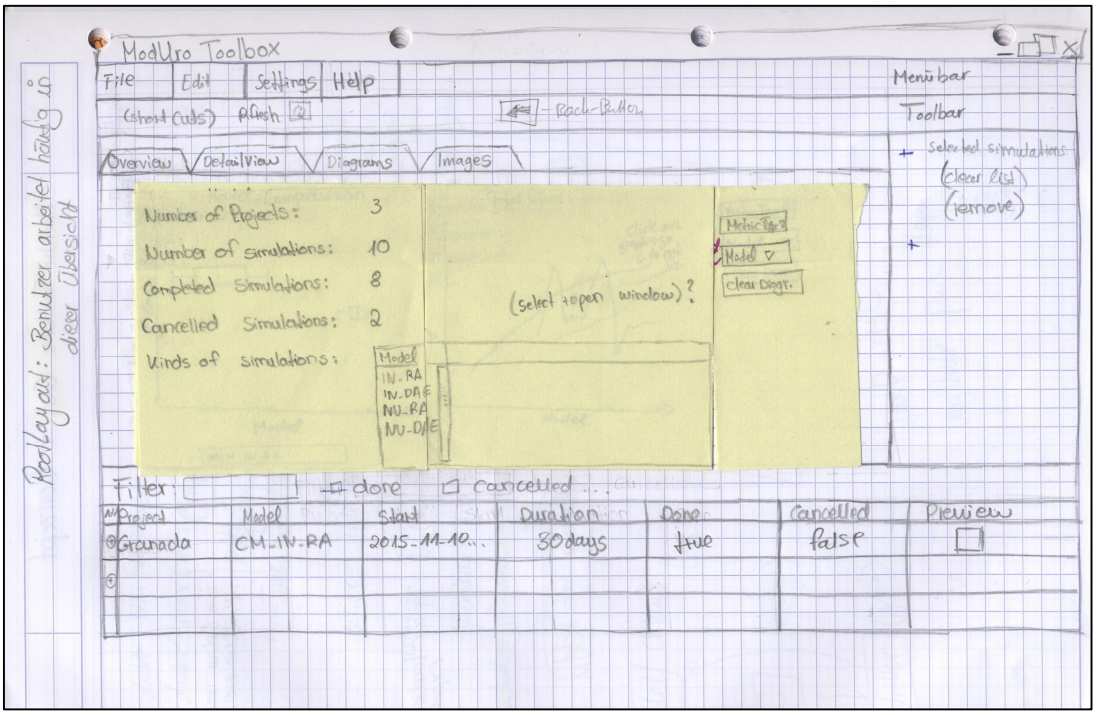

Abbildung 28: Papier Prototyp

Nach der Evaluierung des Papier-Prototyps wurde mittels der Software „Balsamiq“ [17] computerbasierte Mock-ups erstellt. Die in der ersten Evaluierung ergebenen Unstimmigkeiten und Verbesserungen wurden hier angepasst. Im Folgenden sind zwei erstellte Mock-ups (Abbildung 29 und Abbildung 30) zu sehen.

[5] Reines Grundgerüst der Bedienelemente ohne weitere Funktionalität. Mit Hilfe von Mock-ups können Anforderungen an die Benutzeroberfläche in Zusammenarbeit mit Endanwender besser ermittelt werden.

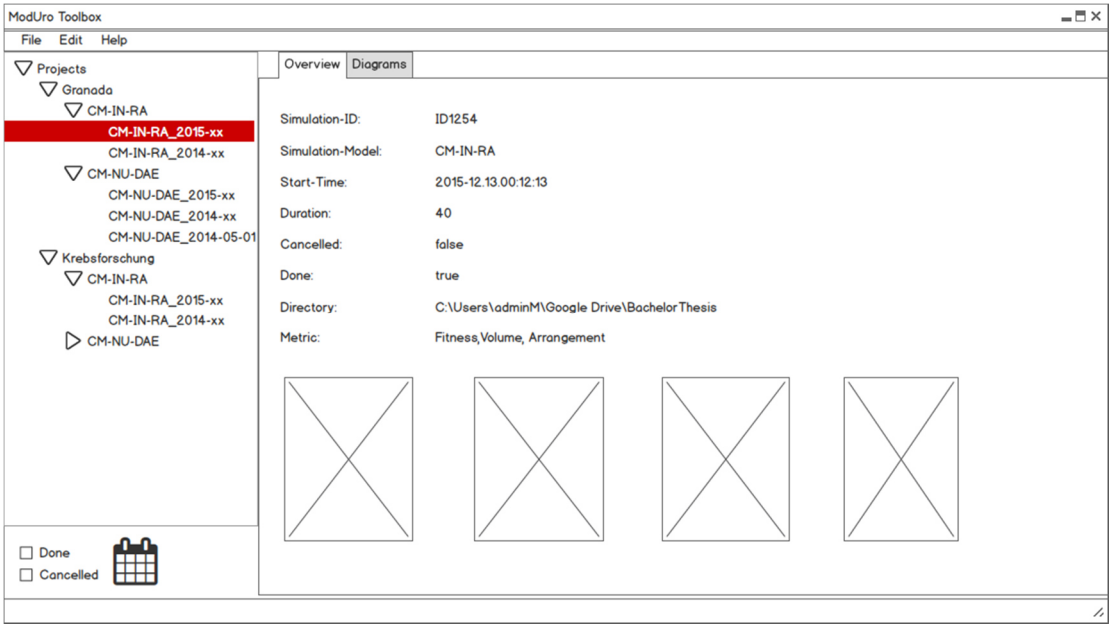

Abbildung 29: Overview Mock-up

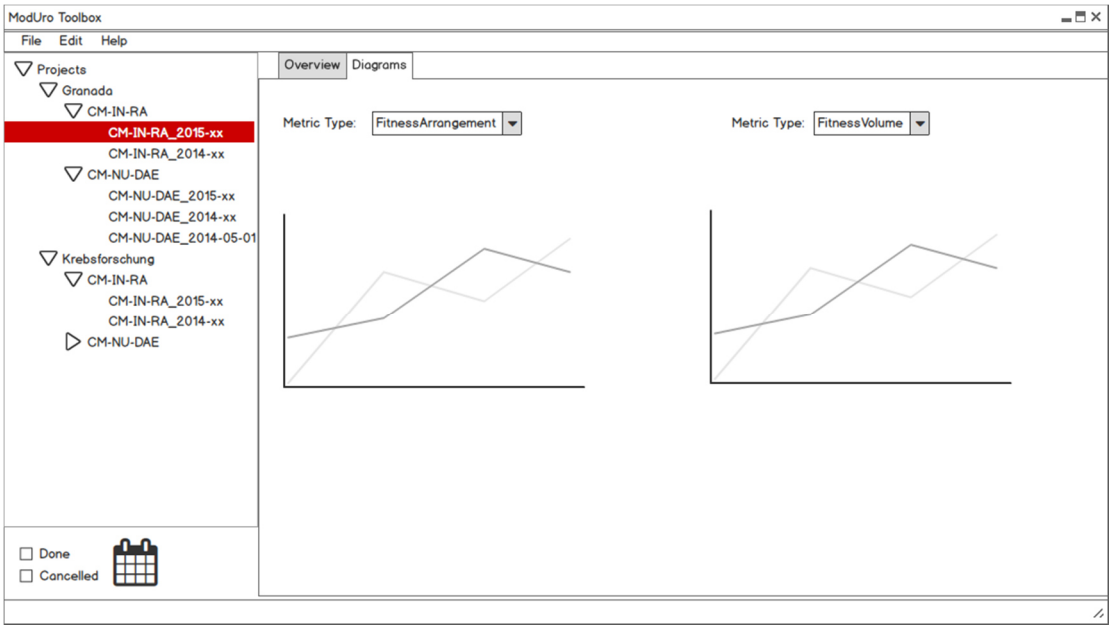

Abbildung 30: Diagram Mock-up

Strukturierung der grafischen Benutzeroberfläche

Es gibt mehrere Möglichkeiten um Elemente einer grafischen Benutzeroberfläche zu gruppieren. Eine Möglichkeit ist „das Gesetz der Nähe und der Ähnlichkeit“ aus der Gestaltpsychologie. Es besagt, dass Elemente zusammengehöhrend wahrgenommen werden, wenn sie räumlich nah beieinander liegen, von Linien oder Kästen umrahmt

werden, sich gemeinsam bewegen und verändern oder sich aufgrund von Farbe, Form oder Größe gleichen [25].

Die klare räumliche Trennung in der ModUro-Toolbox erfolgt im „BoderPane“-Layout. Ein „BorderPane“ besteht aus fünf Bereiche dem „Top“, „Bottom“, „Left“, „Right“ und „Center“ (Abbildung 31).

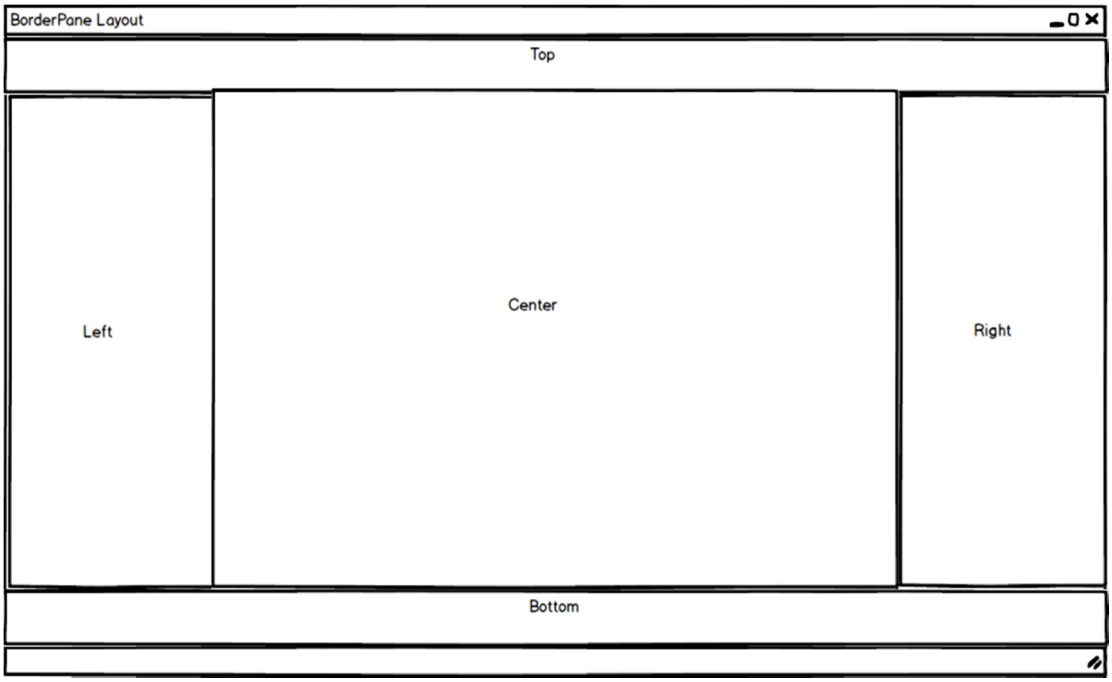

Abbildung 31: BorderPane Layout Beispiel

Im Bereich „Top“ befindet sich später die Menüleiste. In der Menüleiste werden Dialoge zum Speichern und Laden von Einstellungen aufgerufen. Der „Left“-Bereich ist wiederum in zwei Bereiche getrennt. Im oberen Bereich wird der Projektüberblick angezeigt und im unteren Bereich werden Steuerelemente zur Filterung von Projekte angeboten. Der „Center“ ist der eigentliche Informationsansicht. Hier werden alle Informationen der aktuell ausgewählten Projekte, Modellklassen oder Simulationen visualisiert. Die Bereiche „Bottom“ und „Right“ werden für ModUro-Toolbox nicht verwendet.

5 Umsetzung

Dieser Abschnitt beschäftigt sich mit der Umsetzung der ModUro-Toolbox. Im Rahmen dieser Arbeit wurde die Toolbox soweit entwickelt, dass Simulationen von verschiedenen lokalen und Netzwerk-Dateisystemen eingelesen werden und sich in einer grafischen Übersicht darstellen lassen. Die Umsetzung wird anhand eines Beispielprojekts namens „Granada“ gezeigt.

5.1 Eingesetzte Technologien

JavaFX

JavaFX ist ein GUI-Framework von Oracle, um grafische Benutzeroberflächen zu erstellen. Es setzt auf SceneGraph[6], deklaratives Layoutformat, Properties[7] und Bindings[8] für die einfache Anbindung des User Interfaces (UI) an das Datenmodell und verwendet Cascading Style Sheets (CSS) für das Gestalten der Oberfläche der Anwendung [24].

JFreeChart

JFreeChart ist eine in Java entwickelte Open-Source Bibliothek, die innerhalb von Java-basierten Anwendungen verwendet werden kann. Das Framework unterstützt verschiedene statistische Diagrammtypen, wie Säulendiagramme, Balkendiagramme, Tortendiagramme, Histogramme und Box-Whisker-Plot [26]. Mittels JFreeChart können dynamische Daten in einer Anwendung in Form von Diagrammen mit wenig Entwicklungsaufwand angezeigt werden.

[6] Eine objektorientierte Datenstruktur, die bei der Entwicklung computergrafischer Anwendungen eingesetzt wird. Mit ihr kann man logische und räumliche Anordnung der darzustellenden zwei- oder dreidimensionalen Szene beschreiben.

[7] Eigenschaft

[8] Bindings umfassen einen oder mehrere Werte (dependencies), deren Zustand überwacht und bei Änderungen automatisch angepasst wird.

5.2 Aufbau der Anwendung

Die Abbildung 32 zeigt die umgesetzte ModUro-Toolbox zur Laufzeit. Die Anwendung ist in folgenden Bereichen aufgeteilt:

A. Menüleiste
B. Projektüberblick
C. Filtermenü
D. Übersicht
E. Diagramm
F. Toolbar

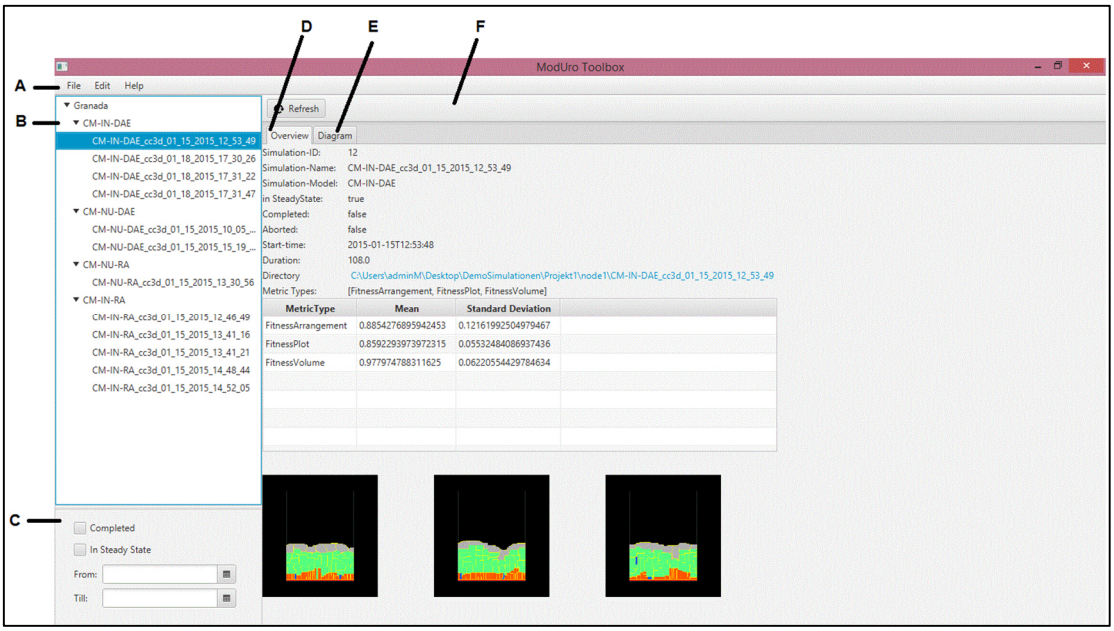

Abbildung 32: ModUro-Toolbox zur Laufzeit

In den nachfolgenden Kapiteln werden die einzelnen Bereiche genauer erläutert und auf deren Umsetzung eingegangen.

5.3 Projekte und Nodes verwalten

Der Benutzer kann in der setting.xml-Datei Projektnamen und Verzeichnisse für das entsprechende Projekt verwalten. Bei dieser Datei handelt es sich um eine XML-Datei. Der Code 5 zeigt ein Beispiel, wie eine setting.xml-Datei aufgebaut ist. Im Beispiel wird ein Projekt namens „Granada“ konfiguriert. Dieses Projekt beinhaltet Simulationen von dem Verzeichnis „DesktopWorkSpace“, dessen Pfad

„C:\Users\adminM\Desktop\CC3DWorkspace“ ist. Beim Starten der Anwendung wird diese Datei eingelesen.

```
1   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2   <projects>
3       <project>
4         <name>Granada</name>
5         <node>
6            <name>DesktopWorkSpace</name>
7            <path>C:\Users\adminM\Desktop\CC3DWorkspace</path>
8         </node>
9       </project>
10  </projects>
```

Code 5: Beispiel einer setting.xml-Datei

Die folgende „Document Type Definition“ (DTD) (Code 6) für die Setting.xml-Datei definiert die Elemente, Attribute und ihr Verhalten zueinander:

```
1   <?xml version="1.0"/>
2   <!DOCTYPE projects[
3   <!ELEMENT projects(project+)>
4   <!ELEMENT project(name, node)>
5   <!ELEMENT node(name, path)>
6   <!ELEMENT name (#PCDATA)>
7   <!ELEMENT path (#PCDATA)>
8   ]>
```

Code 6: DTD der setting.xml-Datei

- Zeile 2: Die DTD wird eingeleitet von `<!DOCTYPE>` und dem Namen des Wurzelelements. Das Wurzelelement enthält alle anderen Kinderelemente.
- Zeile 3: Hier wird das Wurzelelement `projects` deklariert. Es enthält ein Kindelement `project`, das einmal oder mehrmals auftreten kann. Es muss mindestens einmal auftreten.
- Zeile 4: Das im `project`-Element enthaltene Kinderelemente `name` und `node` müssen in der gleichen Reihenfolge exakt einmal erscheinen.
- Zeile 5: Die im `node`-Element enthaltenen Kinderelemente `name` und `path` müssen ebenfalls in der gleichen Reihenfolge exakt einmal erscheinen.
- Das Schlüsselwort `#PCDATA` (Parsed Character Data) gibt an, dass der Inhalt des Elements aus Zeichendaten (Text) bestehen kann.

JAXB

Das Einlesen der Setting.xml-Datei erfolgt über „Java Architecture for XML Binding“ (JAXB). Die Programmierschnittstelle ermöglicht es, Daten aus einem XML-Schema an Java-Klassen zu binden und Instanzen zu generieren [27, p. 434]. Dieses Verfahren wird als XML-Datenbindung bezeichnet. Der folgende Code 7 zeigt, wie die Setting.xml-Datei in der Anwendung eingelesen wird.

```
1   /**
2    * Loads configuration data from setting.xml
3    */
4   private void loadNodeDataFromFile() {
5     try {
6         JAXBContext context = JAXBContext.newInstance(SettingFileWrapper.class);
7         Unmarshaller um = context.createUnmarshaller();
8
9         // Reading XML from the file and unmarshalling.
10        SettingFileWrapper wrapper = (SettingFileWrapper) um.unmarshal(SETTINGXML);
11
12        settings.clear();
13        settings.addAll(wrapper.getProject());
14
15    } catch (Exception e) { // catches ANY exception
16        Alert alert = new Alert(Alert.AlertType.ERROR);
17        alert.setTitle("Error");
18        alert.setHeaderText("Could not load data");
19        alert.setContentText("Could not load data from file:\n" + SETTINGXML.getPath());
20
21        alert.showAndWait();
22    }
23  }
```

Code 7: Setting.xml-Datei einlesen

Das Objekt `Unmarshaller` übernimmt den Prozess der Umwandlung eines XML-Dokuments in die Instanz einer Java-Klasse.

5.4 Baumansicht

Durch die Verwendung der Baumansicht als Projektüberblick ist die Anforderung #FA0030 erfüllt. Dargestellt ist eine Baumstruktur (Abbildung 33) mit dem Beispielprojekt Granada, welche die dazugehörigen Modellklassen (CM-IN-DAE, CM-NU-DAE, CM-NU-RA, und CM-IN-RA) und deren Simulationen enthält. Die Baumansicht besteht aus einer Hierarchie mit drei Stufen:

Hierarchiestufe 1 → Projekt

Hierarchiestufe 2 → Modellklasse

Hierarchiestufe 3 → Simulation

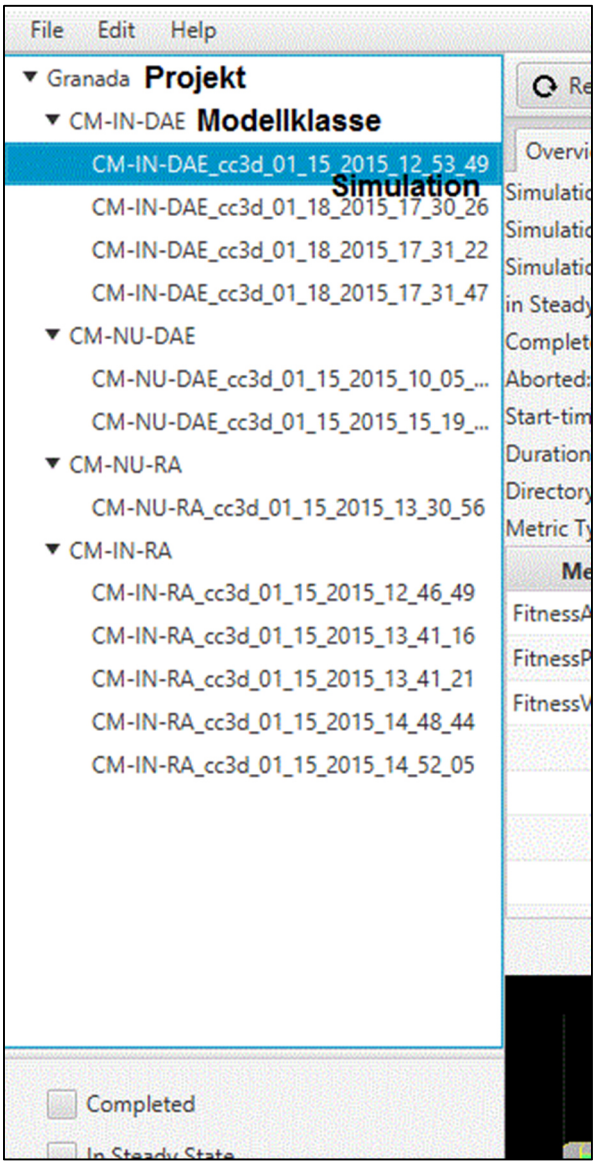

Abbildung 33: Projektüberblick in der Baumansicht

TreeView

Die Baumansicht wird in JavaFX über das Steuerelement „TreeView“ realisiert. Diese Tree-Komponente dient zum Anzeigen von hierarchischen Daten. Da eine „ObservableList[9]“ nicht geeignet ist, um hierarchische Daten zu repräsentieren, werden die Datenobjekte, indem Fall die Projekte, in ein „TreeItem“ eingepackt. [24]

SelectionModel

Im Übersicht - und Diagrammbereich werden diese Informationen angezeigt, die zur Laufzeit in der Baumansicht selektiert sind. Das heißt, wenn eine Simulation selektiert wird, müssen Informationen zu dieser Simulation angezeigt werden. Hierzu wird eine Funktion benötigt, die das Selektieren in der Baumansicht überwacht. Das sogenannte „SelectionModel“ übernimmt diese Aufgabe. Es wird an das „TreeView“-Element gebunden. Wird ein Projekt, Modellklasse oder eine Simulation selektiert, so übergibt das „SelectionModel“ das selektierte „TreeItem“ an eine Methode weiter. Diese Methode zeigt anschließend die Informationen in der Übersicht an.

5.5 Projektübersicht

Um in die Projektübersicht zu gelangen, muss das Projekt (hier Granada) und der „Overview“-Tab mittels Mausklick ausgewählt werden. Die Projektübersicht zeigt eine textuelle Übersicht über alle geladenen Modelklassen mit deren Simulationen:

- Anzahl aller geladenen Simulationen
- Anzahl aller in steadyState befindliche Simulationen
- Anzahl aller abgebrochenen Simulationen
- Anzahl aller abgeschlossenen Simulationen
- Arten von Modellklassen

Abbildung 34 zeigt eine Projektübersicht. Im Projekt „Granada“ gibt es 12 Simulationen. Davon befinden sich zwei Simulationen im stabilen Zustand. Keine Simulationen sind weder abgebrochen noch abgeschlossen. Das Projekt hat folgende Arten von Modellklassen: CM-IN-DAE, CM-NU-DAE, CM-NU-RA, CM-IN-RA.

[9] ObservableLists verfügen über Methoden, welche die Liste bei Änderungen überwachen.

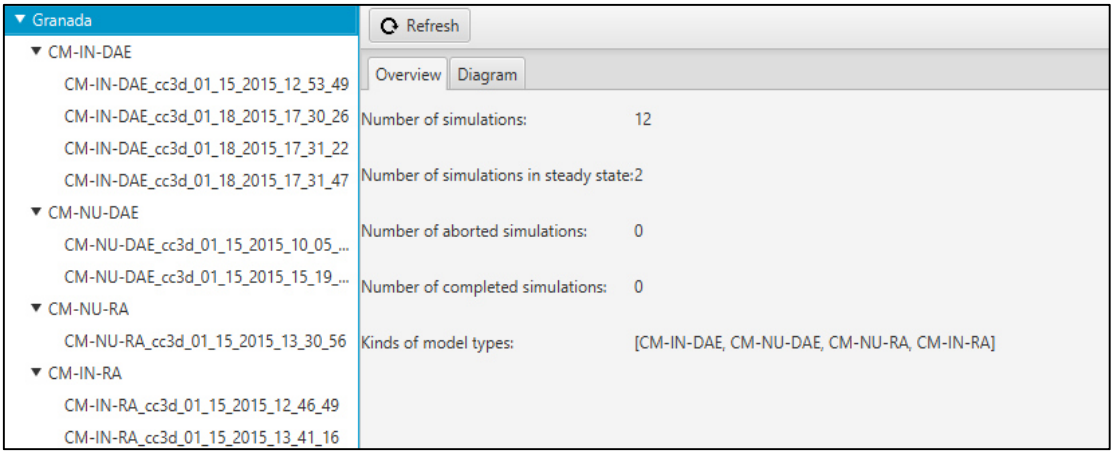

Abbildung 34: Projektübersicht

5.6 Modellübersicht

Wird eine Modellklasse (hier CM-IN-DAE) ausgewählt, so wird die Modellübersicht angezeigt (Abbildung 35), welche alle geladenen Simulationen dieser Modellklasse anzeigt. Zusätzlich werden in einer Tabellenansicht die Mittelwerte und Standardabweichung aller Metriken in diesem Modell aufgelistet.

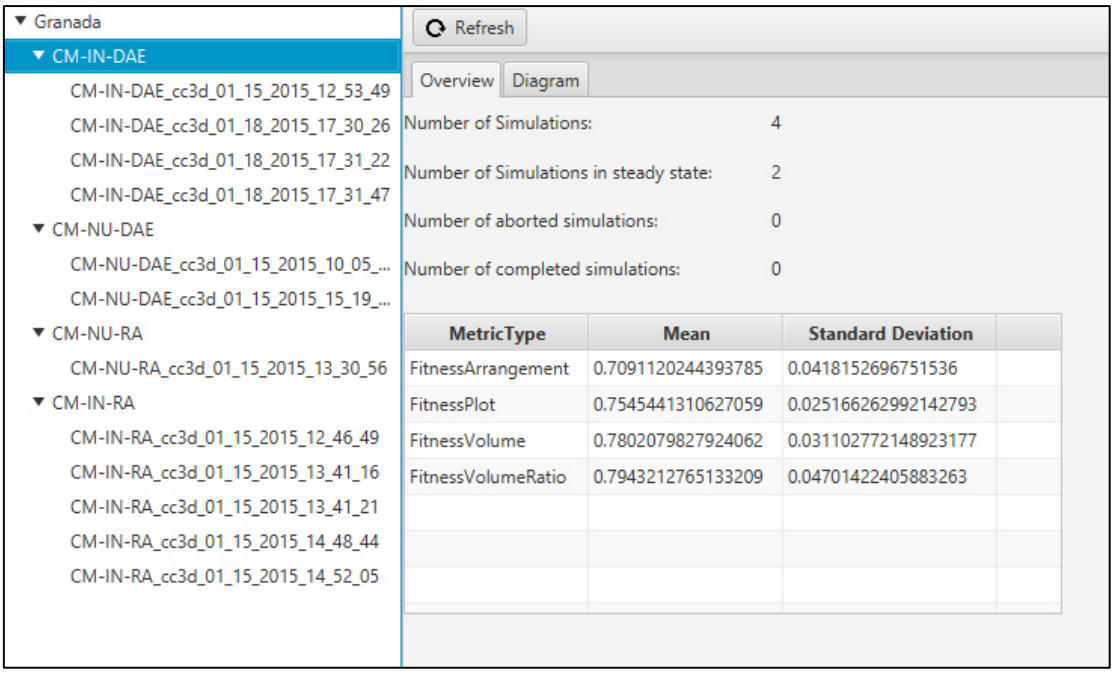

Abbildung 35: Modellübersicht

TreeTableView

Für die Tabellenansicht der Metriken wird das Steuerelement „TableView“ verwendet. Die „TableView“ verwendet als Modell eine „ObservableList“. In der „ObservableList“ sind „MetricType“-Objekte vorhanden. Jedes Element in der Liste entspricht einer Zeile in der „TableView“. Die Werte für die Spalten entsprechen Eigenschaften des Objektes. Das heißt, das Objekt „MetricType“ hat die Eigenschaften „Name“, „Mittelwert“ und „Standardabweichung“, welche die Spalten bilden.

5.7 Simulationsübersicht

Die Simulationsübersicht (Abbildung 36) enthält relevante Metadaten über eine Simulation. Diese sind

- Simulation-ID,
- Simulation-Name,
- Simulation-Model,
- IstInSteadyState,
- IstAbgeschlossen,
- IstAbgebrochen,
- Startzeitpunkt,
- Dauer,
- Verzeichnis, in dem sich die Simulation befindet (mit Mausklick auf den Pfad kann das entsprechende Verzeichnis geöffnet werden),
- alle zur Simulation gehörenden Metriken mit deren Mittelwerten und Standardabweichungen,
- und eine Serie von Zustandsbildern.

Die „Simulation-ID“ wird zufällig vergeben. Der „Simulation-Name“ entspricht der Name des Ordners, in dem die Simulationsdateien vorhanden sind. Der Name des Modells wird aus den Namen des Ordners segmentiert. Die ParameterDump.dat-Datei enthält den Startzeitpunkt. Dieser wird eingelesen. Die Dauer einer Simulation wird aus der Metrikdatei eingelesen. Hierbei entspricht die Dauer den letzten Zeitpunkt in der Metrikdatei. Die Zustände „istInSteadyState“, „istAbgeschlossen“ und „istAbgebrochen“ werden aus der Gesamtmetrik, die FitnessPlot.dat-Datei, berechnet.

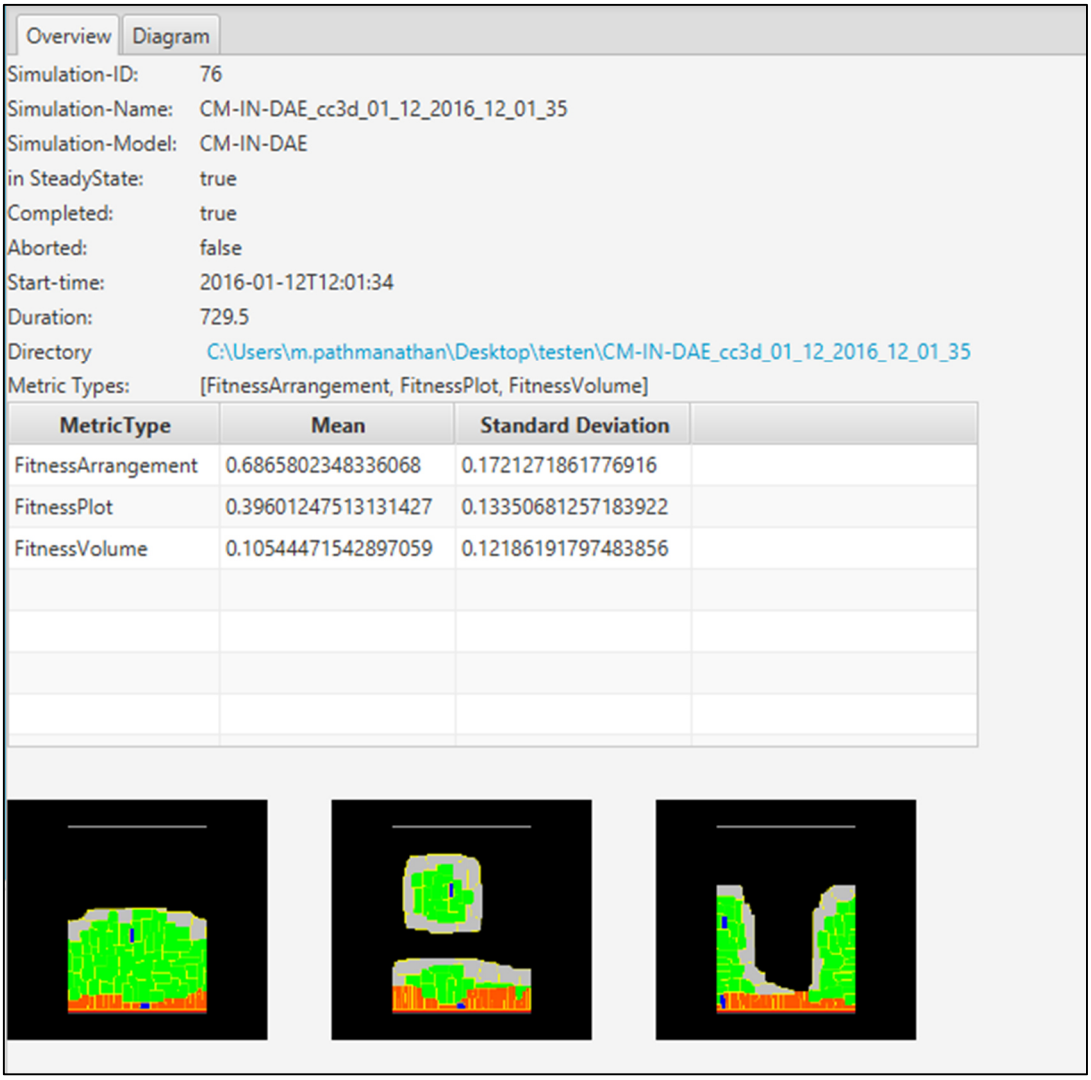

Abbildung 36: Simulationsübersicht

Metriken einlesen

Die Metrikdateien sind reine Textdateien, die in Java über „BufferedReader“ eingelesen werden können. In der Metrikdatei sind zwei Werte, der Zeitpunkt und der Fitnesswert, gespeichert. Diese werden in der Toolbox in einer zweidimensionalen Array gespeichert.

Bilder einlesen

In der Simulationsübersicht wird zur einfachen Analyse Simulationsbilder angezeigt. Hierzu werden die Bilder von drei Zeitpunkten genommen. Das erste Bild entspricht 20% von der Gesamtdauer, das zweite Bild 50% und das letzte 80%.

5.8 Box-Whisker-Plot Diagramm

Die Box-Whisker-Plot Diagrammansicht (Abbildung 37) zeigt eine graphenbasierte Übersicht über die Fitness der verschiedenen Modellklassen. Auf der x-Achse werden die unter-schiedlichen Modellklassen aufgelistet. Die y-Achse enthält die Fitnesswerte.

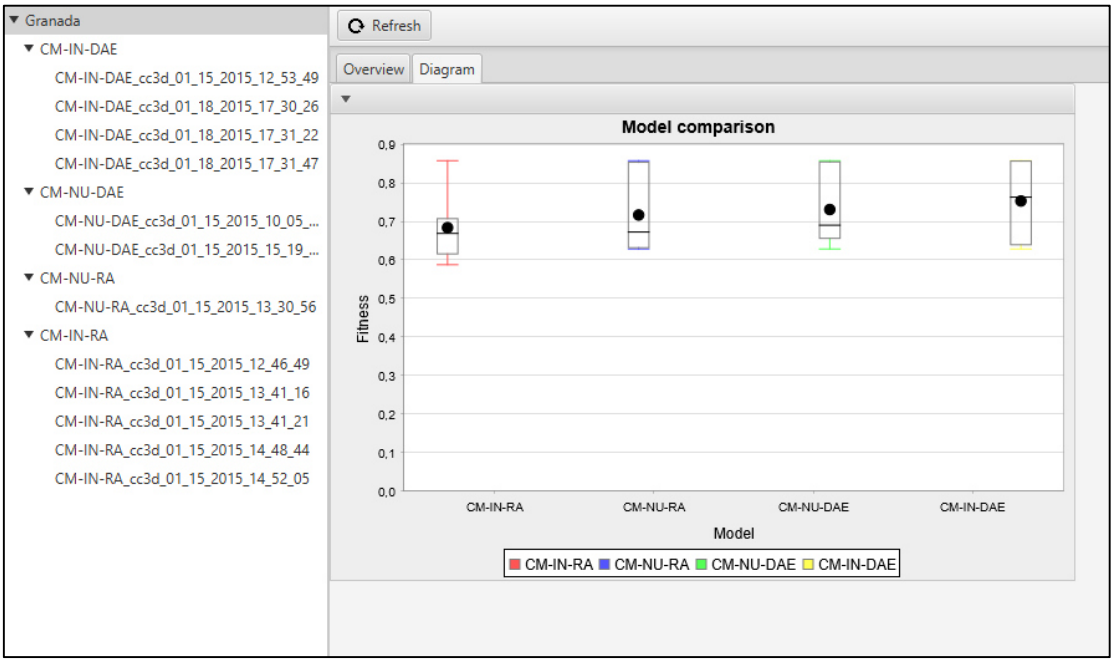

Abbildung 37: Box-Whisker-Plot Diagrammansicht

BoxAndWhiskerCategoryDataset

Der Box-Whisker-Plot wird mit Hilfe von JFreeChart erstellt. Hierzu wird eine „BoxAndWhiskerCategoryDataset“ verwendet. Diese ist eine Datenmenge, welche den Median, Minimalwert, Maximalwert und die Quantile zu jeder Modellklasse enthält.

5.9 Liniendiagramm

Die Metriken werden über Liniendiagramme analysiert. Hierzu gibt es eine Übersicht, die zwei Liniendiagramme gegenüberstellt (Abbildung 38). Ein Liniendiagramm wird in JFreeChart mittels „XYSeries“ erzeugt. „XYSeries“ repräsentiert Daten in der Form (x,y). In einer Metrik ist x der Zeitpunkt und y der Fitnesswert.

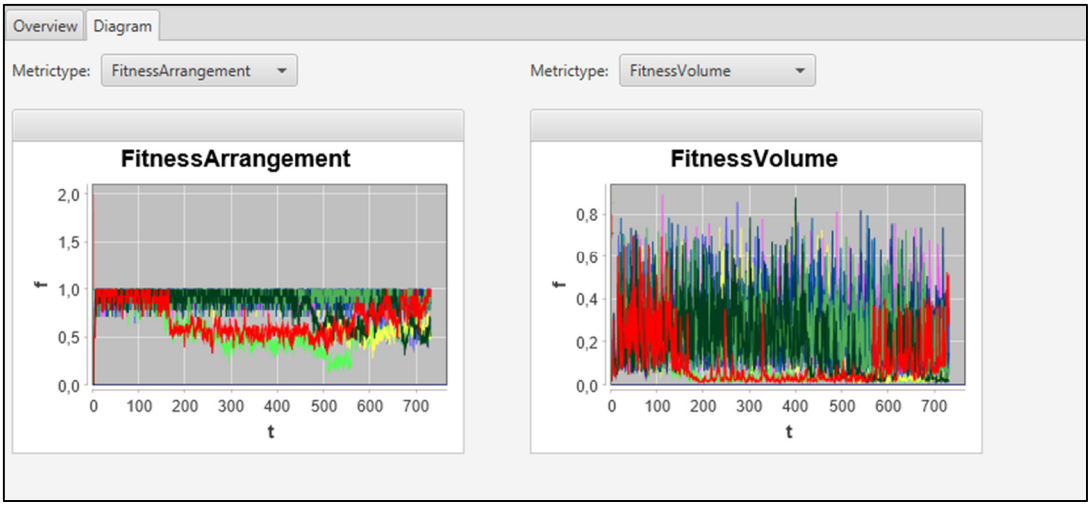

Abbildung 38: Liniendiagramm zur Analyse von Metriken

Auswahl der Metriken

Standardmäßig werden im Liniendiagramm die Metriken „FitnessArrangement“ und „FitnessVolume“ gegenübergestellt. Über das Steuerelement „ChoiceBox“ (Abbildung 39) kann eine andere Metrik ausgewählt werden.

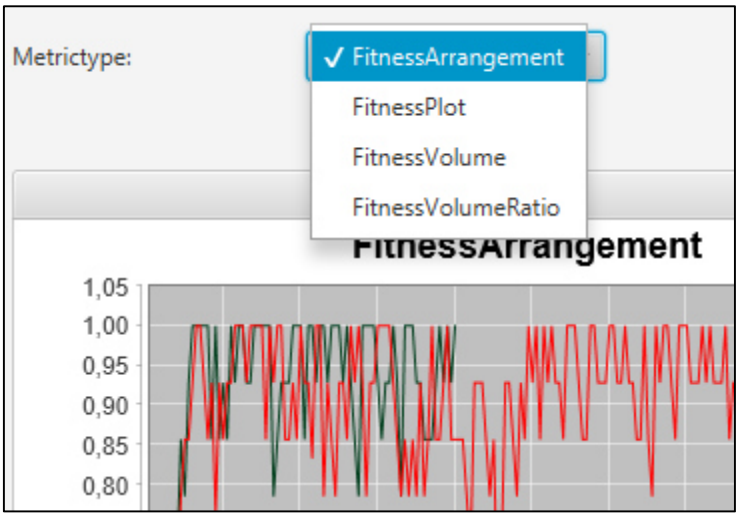

Abbildung 39: Auswählen von Metriken

5.10 Export von Grafiken

In wissenschaftliche Arbeiten werden Grafiken viel verwendet. Über eine Export-Funktion kann ein Diagramm über die Toolbox einfach exportiert werden. Mittels rechter Mausklick auf ein Diagramm öffnet sich ein Menü (Abbildung 40). Über das

Menü kann gewählt werden als welches Dateiformat das Diagramm abgespeichert werden soll. In dieser Arbeit wurde das Exportieren des Diagramms als PNG und JPEG realisiert. Das Exportieren nach TIKZ muss noch umgesetzt werden.

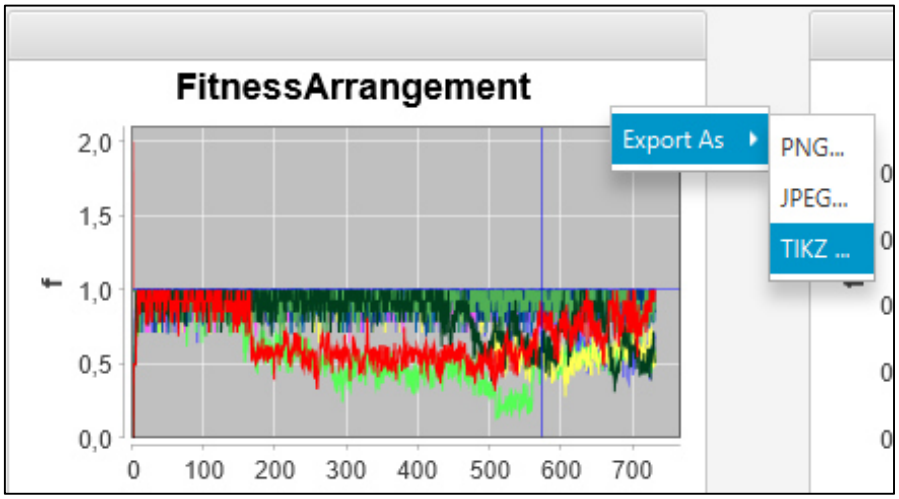

Abbildung 40: Export von Diagramme

5.11 Filter

Im Filtermenü (Abbildung 41) können Simulationen nach Kriterien gefiltert werden. Es sind bereits Steuerelemente für das Filtern nach Zuständen (abgeschlossen und stabiler Zustand) und Zeitpunkt gegeben. Die Logik hinter den Steuerelementen muss noch komplett realisiert werden.

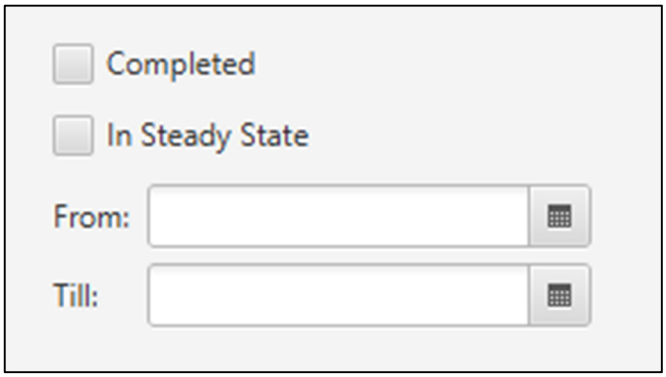

Abbildung 41: Filterung nach Simulationen

5.12 Verwendete Bibliotheken in ModUro-Toolbox.

Die Bibliotheken die in der ModUro-Toolbox verwendet werden sind

- commons-math3-5.5.jar [28],
- jcommon-1.0.23.jar [29],
- jfreechart-1.0.19.jar [30],
- jfreesvg-2.0.jar [31],
- und junit-4.11.jar [32].

Commons-Math ist eine Bibliothek, die Methoden zur mathematischen und statistischen Berechnungen bereithält. Hierüber werden die statistischen Werte, wie Mittelwerte und Standardabweichungen der Metriken berechnet. Die Bibliotheken jcommon, jfreechart und jfreesvg dienen zu Erstellung von graphenbasierte Übersichten. Die JUnit Bibliothek wird für das Erstellen von automatisierten Testfällen benutzt.

6 Zusammenfassung und Ausblick

Im Folgenden werden umgesetzte, teilweise umgesetzte und nicht umgesetzte Funktionalitäten beschrieben. Abschließend erfolgt ein Ausblick auf Erweiterungsmöglichkeiten nach Abschluss dieser Arbeit.

6.1 Zusammenfassung

Im Rahmen dieser Arbeit zum Projekt „Entwurf und Implementierung der ModUro-Toolbox zur Analyse von Urothelsimulationen“ wurde die ModUro-Toolbox entwickelt und für die statistische Analyse von Zellsimulationen verwendet. Die in Java 8 entwickelte grafische Toolbox (#FA0001) sammelt Ergebnisse der rechenintensiven CC3D-Simulationen von verschiedenen lokalen und Netzwerk-Dateisystemen und wertet sie aus. Die konfigurierten Projekte mit deren Modellklassen und Simulationen können über verschiedene Ansichten wie Projektübersicht, Modellübersicht und Simulationsübersicht angesehen werden. Die Ansichten liefern relevante Informationen über die jeweiligen Simulationen. Die statistischen Diagramme wie Box-Whisker-Plot und Liniendiagramm ermöglichen es, die Metriken miteinander zu vergleichen. Der Quellcode zur ModUro-Toolbox ist auf GitHub als Open-Source-Projekt veröffentlicht worden. Er kann unter den folgenden Link heruntergeladen werden: https://github.com/informatik-mannheim/Moduro-Toolbox.git.

In dieser Arbeit wurden folgende Funktionalitäten umgesetzt:

- Verwaltung von Projekte und Knoten (lokal und Netzwerk): #FA0010, #FA0020, #FA0021, #FA0022
- Projektüberblick als Baumansicht: #FA0030, #FA0040, #FA0041, #FA0042
- Projektübersicht #FA0050, Modellklassenübersicht #FA0051, Simulationsübersicht #FA0052
- Box-Whisker-Plot #FA0060, Liniendiagramm #FA0061, #FA0062, #FA0063, #FA0064
- Import von nicht vorhandenen Simulationen (Aktualisieren): #FA0090

Die Filterfunktion definiert durch Anforderung #FA0070 und die Exportfunktion #FA0080 wurden nur teilweise umgesetzt. Es sind aktuell nur Steuerelemente in den View definiert. Die Logik dieser Funktionen muss vollständig realisiert werden.

Die Korrektheit der Anwendung wurde mittels JUnit-Testfällen und der Behebung vorhandener Fehler erreicht. Im Anhang werden diese Testfälle zu ModUro-Toolbox

genauer erläutert. Parallel zur Entwicklung wurde die Toolbox von Endanwendern benutzt und getestet. Hierbei wurden ebenfalls einige Verbesserungen in der Benutzbarkeit vorgenommen. Somit ist die Anforderung an die Benutzbarkeit erfüllt.

6.2 Ausblick

In dieser Arbeit wurden die wesentlichen Hauptfunktionalitäten für die statistische Analyse von Zellsimulationen mittels CC3D realisiert. Während der Entwicklung entstanden einige Ideen, wie die Anwendung weiter entwickelt und so die Benutzerfreundlichkeit verbessert werden könnte. Diese Erweiterungen wurden noch nicht implementiert. Die folgenden Ideen könnten in einem Folgeprojekt umgesetzt werden.

Verwaltung von Projekten und Knoten über GUI

Bisher werden Projekte und Knoten mittels einer XML-Datei (setting.xml) verwaltet. Eine benutzerfreundliche Erweiterung wäre es, die Konfiguration über die GUI anzubieten. Siehe hierzu den Quellcode unter: \src\main\java\de\hs\mannheim\modUro \model\dialog.

Vergabe der ID für eine Simulation

Die „Simulation-ID“ einer Simulation wird über eine zufallsgenerierte Zahl vergeben. Diese sollte geändert werden, sodass die ID aus dem „Seed“-Wert und dem Zeitpunkt der Simulation besteht. Die beiden Werte sind im ParameterDump.dat enthalten.

Verzeichnisse ausschließen

In einem konfigurierten Projekt gibt es oft Verzeichnisse, die für die Analyse irrelevant sind. Es könnte eine Funktionalität realisiert werden, die das Ausschließen von Ordnern behandelt.

Übersicht über fehlende Metriken und Dateien

Nicht alle Simulationen enthalten alle geforderten Dateien für die Toolbox. Bei manchen Simulationen fehlt beispielsweise die ParameterDump.dat-Datei, weil sie aufgrund von Fehlern im CC3D-Player nicht erzeugt wurde. Ohne diese Datei existiert diese Simulation nicht im Projekt. Eine Übersicht über diese fehlerhafte Simulationen wäre für den Anwender bei der Suche hilfreich.

Abkürzungsverzeichnis

ModUro	Modellierung und Simulation in der Urologie
CC3D	CompuCell3D
GGH	Glazier-Graner-Hogeweg
MCS	Monte-Carlo-Step
XML	Extensible Markup Language
GUI	Graphical User Interface
FA	Funktionale Anforderung
NA	Nicht funktionale Anforderung
UI	User Interface
CSS	Cascading Style Sheets
DTD	Document Type Definition
JAXB	Java Architecture for XML Binding

Abbildungsverzeichnis

Quellcodeverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] A. Torelli, F. Siegel, P. Erben und M. Gumbel, „Modeling of the Urothelium with an Agent Based Approach," *Bioinformatics and Biomedical Engineering - Lecture Notes in Computer Science, Volume 9044,* pp. 375-385, 2015.

[2] H. Dickhaus und P. Knaup-Gregori, Biomedizinische Technik - Medizinische Informatik (Band 6), Deutsche Gesellschaft für Biomedizinische Technik im VDE ISBN: 978-3-11-025222-4.

[3] M. Gumbel, R. Grebe, M. Knapp-Mohammady, G. Ullman und J. Langowski, Handbuch der Medizinischen Informatik (2. Auflage) - Modellierung biologischer Prozesse, Carl Hanser Verlag München, 2005, ISBN: 3-446-22701-6.

[4] M. Gumbel und P. Erben, *ModUro: Ein Modellierungstool für die translationale Medizin im Urothelkarzinom,* Hochschule Mannheim, DFG-Antrag.

[5] Zentrum für Krebsregisterdaten, Krebsarten - Krebs gesamt, 2015, URL: http://www.krebsdaten.de/Krebs/DE/Content/Krebsarten/Krebs_gesamt/krebs_gesamt_node.html [Stand: 17.02.2016].

[6] Onko Internetportal, Blasenkrebs, Harnblasenkrebs, Blasenkarzinom, URL: http://www.krebsgesellschaft.de/onko-internetportal/basis-informationen-krebs/krebsarten/andere-krebsarten/blasenkrebs/definition-und-haeufigkeit.html [Stand: 11.02.2016].

[7] Wikipedia, Bladder urothelial carcinoma histopathology (2) at trigone.jpg, URL: https://upload.wikimedia.org/wikipedia/commons/7/74/Bladder_urothelial_carcinoma_histopathology_%282%29_at_trigone.jpg [Stand: 17.02.2016].

[8] J. A. Izaguirre, R. Chaturvedi, C. Huang, T. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. Alber, G. Hentschel und S. A. G. J. A. Newman, CompuCell, a multi-model framework for simulation of morphogenesis, URL: https://bioinformatics.oxfordjournals.org/content/20/7/1129.full.pdf [Stand: 21.01.2016].

[9] A. Torelli, *Computer Simulation of Urothelial Cells - Master Thesis,* Hochschule Mannheim, September 2014.

[10] M. Swat, A. Shirinifard, A. Balter, N. Poplawski und J. A. Glazier, CompuCell3D 3.6.0 Quick Start Guide, URL: http://compucell3d.org/BinDoc/cc3d_binaries/Manuals/compucell3d_quickstartguide_1.5.pdf [Stand: 11.02.2016].

[11] A. Roch, *ModUro Toolbox,* Hochschule Mannheim, Studienarbeit.

[12] M. Riedel, *ModUro Toolbox - Anbindung einer Persistenzkomponente,* Hochschule Mannheim, Studienarbeit, 2015.

[13] G. Starke, Effektive Softwarearchitekturen 7. Auflage, Hanser Verlag, ISBN: 978-3-446-44361-7.

[14] ITWissen/Das große Online-Lexikon für Informationstechnologie, GUI (graphical user interface) - Grafische Benutzeroberfläche, URL: http://www.itwissen.info/definition/lexikon/graphical-user-interface-GUI-Grafische-Benutzeroberflaeche.html [Stand: 18.02.2016].

[15] S. Kleuker, Grundkurs Software-Engineering mit UML 3. Auflage, Springer Vieweg, ISBN: 978-3-658-00641-9.

[16] C. Abras, D. Maloney-Krichmar und J. Preece, „User-Centered Design," in *Bainbridge, W. Encyclopedia of Human-Computer Interaction*, Sage Publications.

[17] balsamiq, balsamiq Mockups v.3.3.9, URL: https://balsamiq.com/ [Stand: 24.02.2016].

[18] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, U. Mehlig, T. Neumann, M. Völter und U. Zdun, Software-Architektur: Grundlagen - Konzepte - Praxis, Spektrum Akademischer Verlag, ISBN: 3-8274-1534-9.

[19] Handbuch Usability, Definition Usability: ISO 9241 Software Ergonomie, URL: http://www.handbuch-usability.de/begriffsdefinition.html [Stand: 17.02.2016].

[20] K.-H. HDoz. Dr. Rödiger, Software-Ergonomie '93 - Von der Benutzungsoberfläche zur Arbeitsgestaltung, Fachbereich Mathematik/Informatik Universität Bremen: Vieweg + Teubner, ISBN: 978-3-519-02680-8 , 1993.

[21] M. Geirhos, Entwurfsmuster - Das umfassende Handbuch, Rheinwerk Computing, ISBN: 978-3-8362-2762-9.

[22] K. Günster, Einführung in Java, Unterkapitel: GUIs mit JavaFX, Rheinwerk Verlag, ISBN: 978-3-8362-2867-1.

[23] R. Steyer, Einführung in JavaFX: Moderne GUIs für RIAs und Java-Applikationen, Springer Vieweg, ISBN: 978-3-658-02835-0.

[24] A. Epple, JavaFX 8 Grundlagen und fortgeschrittene Techniken, dpunkt.verlag, ISBN: 978-3-86490-169-0, 2015.

[25] T. Dr. Wirth, Die Gesetze der Nähe und Ähnlichkeit, URL: http://www.kommdesign.de/texte/gestaltpsychologie1.htm [Stand: 18.02.2016].

[26] Tutorials Point, JFreeChart Tutorial - Overview, URL: http://www.tutorialspoint.com/jfreechart/ [Stand: 01.02.2016].

[27] B. McLaughlin, Java & XML 2. Auflage, O'Reilly Verlag, ISBN: 3-89721-296-X.

[28] Commons Apache, commons-math3-5.5.jar, URL: https://commons.apache.org/proper/commons-math/ [Stand: 24.02.2016].

[29] JFree, jcommon-1.0.23.jar, URL: http://www.jfree.org/jcommon/ [Stand: 24.02.2016].

[30] JFreeChart, jfreechart-1.0.19.jar, URL: http://www.jfree.org/jfreechart/ [Stand: 24.02.2016].

[31] JFree, jfreesvg-2.0.jar, URL: http://www.jfree.org/jfreesvg/ [Stand: 24.02.2016].

[32] JUnit, junit-4.11.jar, URL: http://junit.org/ [Stand: 24.02.2016].

[33] S. Kleuker, Qualitätssicherung durch Softwaretests - Vorgehensweisen und Werkzeuge zum Test von Java-Programmen, Springer Vieweg, ISBN: 978-3-8348-0929-2.

[34] A. Spillner und T. Linz, Basiswissen Softwaretest, dpunkt.verlag, ISBN: 978-3-89864-642-0.

Anhang

Anhang 1: Klassendiagramm - Gesamtübersicht

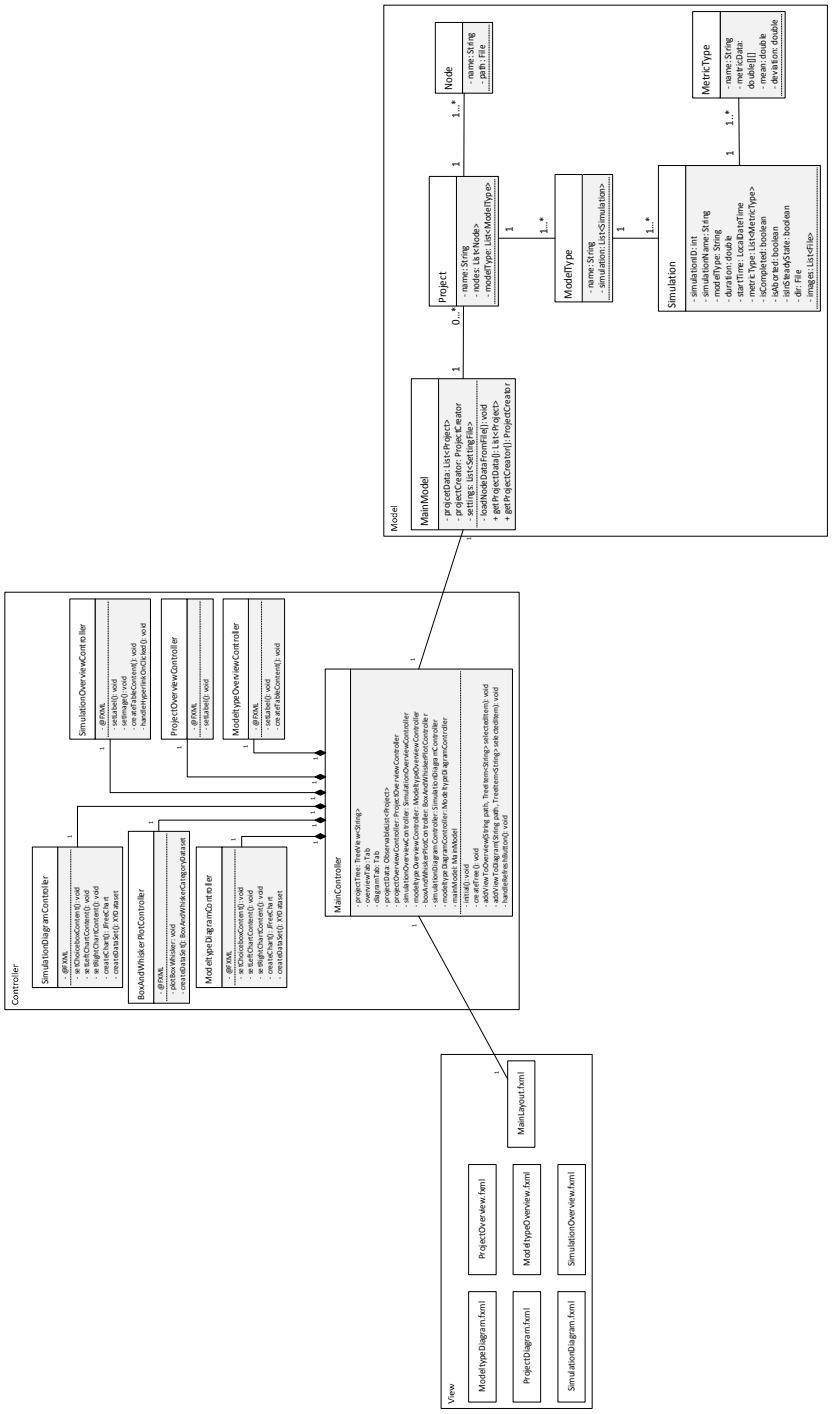

Abbildung 42: Klassendiagramm - Gesamtübersicht

Anhang 2 : Klassendiagramm - Controller

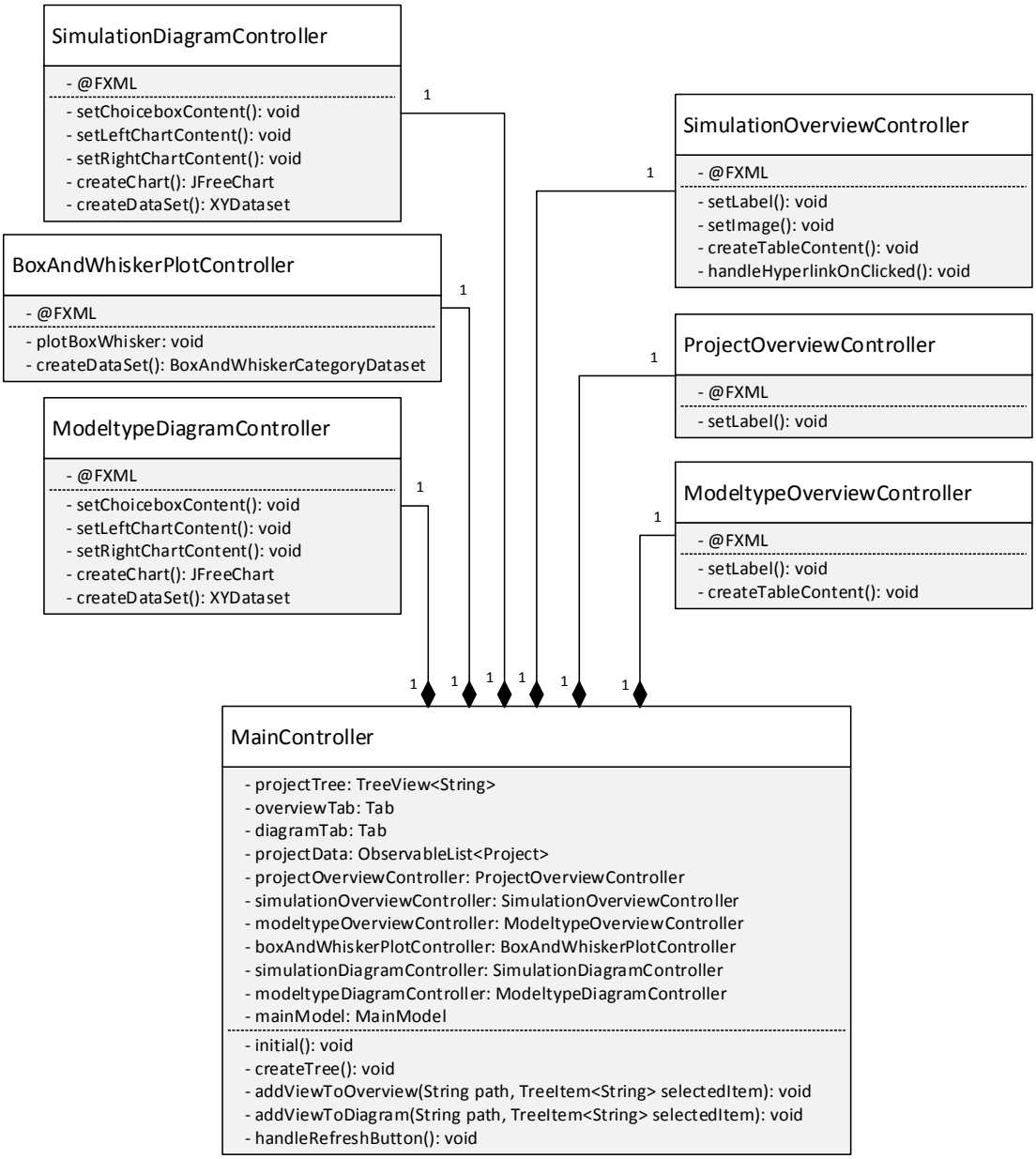

Abbildung 43: Klassendiagramm - Controller

Anhang 2: JUnit Test

Wird eine Anwendung manuell getestet, so kann es der Fall sein, dass einige Funktionalitäten übersehen werden, oder das häufige Wiederholen von Testfällen zu Fehlern führen. Daher sollten Tests einfach und sicher wiederholbar, und schnell zu erstellen sein. JUnit ist ein Test-Framework für klassische Java-Programme. Es bietet die Möglichkeit automatisierte Testfälle zu schreiben und diese auszuführen. [33] Für die Qualitätssicherung der ModUro-Toolbox wurde das Whitebox-Verfahren gewählt. Die Idee des Whitebox-Verfahrens ist es, dass alle Quellcodeteile mindestens einmal getestet werden. Der sogenannte „Anweisungstest“ ist eine Whitebox-Testfallentwurfsmethode. Im Mittelpunkt stehen die einzelnen Anweisungen des Testobjekts. [34]

Paketstruktur der Testklassen

Für die Testklassen wurde dieselbe Paketstruktur gewählt wie beim Java Quellcode der ModUro-Toolbox. Somit ist eine klare Trennung zwischen produktivem Code und dem Test-Code gegeben. Da das Paket „model“ Klassen für die Datenhaltung enthält, wurden nur diese Klassen ausführlich getestet. Der View und der zugehörige Controller wurden manuell zur Laufzeit der ModUro-Toolbox getestet. Für alle Methoden in den Modellklassen wurden Testfälle geschrieben. Die Validierung von privaten Methoden erfolgte durch das indirekte Testen über „getter“- und „setter“- Methoden.

Anmerkung

Bevor dem Ausführen der JUnit Tests müssen die Verzeichnispfade in der setting.xml-Datei angepasst werden. Einige Testmethoden enthalten absolute Pfade, die ebenfalls angepasst werden müssen. Beispielsweise der erwartete Simulationspfad.

Simulationsdaten für den Test

Die Testfälle basieren auf echte Simulationsdaten. Code 8 zeigt die Konfigurationsdatei setting.xml.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
    <project>
        <name>Project1</name>
        <node>
            <name>Project1Node1</name>
            <path>C:\Users\adminM\Desktop\Moduro-Toolbox\src\test\resources\Simulationdata\Projekt1/node1</path>
        </node>
        <node>
            <name>Project1Node2</name>
            <path>C:\Users\adminM\Desktop\Moduro-Toolbox\src\test\resources\Simulationdata\Projekt1/node2</path>
        </node>
    </project>
    <project>
        <name>Project2</name>
        <node>
            <name>Project2Node1</name>
            <path>C:\Users\adminM\Desktop\Moduro-Toolbox\src\test\resources\Simulationdata\Projekt2\node1</path>
        </node>
    </project>
</root>
```

Code 8: Setting.xml für JUnit

Im Folgenden ist die Ordnerstruktur der Simulationsdateien dargestellt.

- Projekt1
  - Node1
    - CM-IN-DAE
      - CM-IN-DAE_cc3d_01_15_2015_12_53_49
    - CM-NU-DAE
      - CM-NU-DAE_cc3d_01_15_2015_10_05_02
  - Node2
    - CM-IN-DAE
      - CM-IN-DAE_cc3d_01_18_2015_17_30_26
- Projekt2
  - Node1
    - PAS-IN-DAE
      - PAS-IN-DAE_cc3d_01_15_2015_12_46_01
    - PAS-IN-RA
      - PAS-IN-RA_cc3d_12_04_2014_11_03_08

Testfälle

Testklasse	Testmethode	Eingabe	Ausgabe
MainModelTest	countProject()	Konfiguration mit 2 Projekte	2
SimulationOverviewTest	nameOfFirstSimulationInSimulationList()	Projekt2, Modellklasse2, Simulation1	PAS-IN-RA_cc3d_12_04_2014_11_03_08
	modeltypeOfSimulation()	Projekt2, Modellklasse2, Simulation1	PAS-IN-RA
	durationOfSimulation()	Projekt2, Modellklasse2, Simulation1	702.0
	startTimeOfSimulation()	Projekt2, Modellklasse2, Simulation1	2014-12-04 11:03:08
	listOfMetricTypesInSimulation()	Projekt2, Modellklasse2, Simulation1	FitnessArrangement, FitnessPlot, FitnessVolume
	simulationIsCompleted()	Projekt2, Modellklasse2, Simulation1	true
	simulationIsInSteadyState()	Projekt2, Modellklasse2, Simulation1	True
	simulationIsAborted()	Projekt2, Modellklasse2, Simulation1	true
	imagesOfSimulation()	Projekt2, Modellklasse2, Simulation1	PAS-IN-RA_cc3d_0009000.png PAS-IN-RA_cc3d_0009600.png PAS-IN-RA_cc3d_0010100.png
	filePathOfSimulation()	Projekt2, Modellklasse2, Simulation1	<>\\Simulationdata\\Projekt2\\node1\\PAS-IN-RA_cc3d_12_04_2014_11_03_08
ProjectOverviewTest	numberOfSimulationsTest()	Projekt2	2
	numberOfSteadyStateSimulation()	Projekt2	1
	numberOfAbortedSimulations()	Projekt2	1
	numberOfCompletedSimulations()	Projekt2	1
	listOfModeltypesInProject()	Projekt2	PAS-IN-DAE, PAS-IN-RA
ModeltypeOverviewTest	numberOfSimulationsTest()	Projekt2, Modellklasse2	1
	numberOfSteadyStateSimulation()	Projekt2, Modellklasse2	1
	numberOfAbortedSimulations()	Projekt2, Modellklasse2	1
	numberOfCompletedSimulations()	Projekt2, Modellklasse2	1
	statisticValuesOfFitnessPlot()	Projekt2, Modellklasse2	Name: FitnessPlot Mean: 0,37 StdDev: 0,00

Testklasse	Testmethode	Eingabe	Ausgabe
SimulationCreatorTest	nameOfFirstSimulationInSimulationList()	Projekt1, Modellklasse1, Simulation1	CM-IN-DAE cc3d 01 15 2015 12 53 49
	modeltypeOfSimulation()	Projekt1, Modellklasse1, Simulation1	CM-IN-DAE
	durationOfSimulation()	Projekt1, Modellklasse1, Simulation1	108.0
	startTimeOfSimulation()	Projekt1, Modellklasse1, Simulation1	2015-01-15 12:53:48
	countMetricTypeOfSimulation()	Projekt1, Modellklasse1, Simulation1	3
	simulationIsNotCompleted()	Projekt1, Modellklasse1, Simulation1	false
	simulationIsInSteadyState()	Projekt1, Modellklasse1, Simulation1	True
	simulationIsNotAborted()	Projekt1, Modellklasse1, Simulation1	false
ProjectCreatorTest	projectName()	Projekt1	Project1
	countProjectNodes()	Projekt1	2
	countModelTypes()	Projekt1	2
ModeltypeCreatorTest	nameOfFirstModeltypeInModeltypeList()	Projekt1, Modellklasse1	CM-IN-DAE
	countSimulationInSpecificModeltype()	Projekt1, Modellklasse1	2
MetrictypeCreatorTest	metricTypeName()	Projekt1, Modellklasse1, Simulation1, Metrik1	FitnessArrangement
	metricData()	Projekt1, Modellklasse1, Simulation1, Metrik1	0.5, 3.5 1.0, 0.5 1.5, 0.5
	meanOfSimulation()	Projekt1, Modellklasse1, Simulation1, Metrik1	1.5
	stdDevOfSimulation()	Projekt1, Modellklasse1, Simulation1, Metrik1	1.7320508075688772

Anhang 3: ModUro-Toolbox Quellcode

Der Quellcode der ModUro-Toolbox ist ebenfalls ein Teil der Bachelorarbeit. Da der Quellcode weiterentwickelt wird, wurde ein Tag mit dem Namen „MPathmanathan2016“ erstellt, welcher nicht verändert werden kann. Dieser Stand (Version) zählt zur Bachelorarbeit.

Link zum Quellcode: https://github.com/informatik-mannheim/Moduro-Toolbox.git.