

Simulieren und Testen von Operational Transform Algorithmen

Masterarbeit

*Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik*

Vorgelegt von Marcus Sümnick am 21. April 2014

Betreuer: Prof. Dr. rer. nat. habil. Clemens Cap
Erstgutachter: Prof. Dr. rer. nat. habil. Clemens Cap
Zweitgutachter: Prof. Dr.-Ing. habil. Gero Mühl

Zusammenfassung

Die universelle Verfügbarkeit von breitbandigen Netzzugängen hat das Zusammenarbeiten von Menschen in den letzten 15 Jahren nachhaltig verändert. Sie stellen einen Grundstein für die Zusammenarbeit in Echtzeit dar. Ein weiteres grundlegendes Element, ohne das die diese Zusammenarbeit in Echtzeit nicht möglich wäre, sind Algorithmen. Diese schaffen die Grundlage dafür, dass Daten ohne spürbare Verzögerung ausgetauscht und verarbeitet werden können. Ein Verfahren für die Verarbeitung dieser Daten ist die Operational Transformation (OT). Dieses ermöglicht es mehreren Nutzern, ein Dokument gleichzeitig zu bearbeiten und die Änderungen der jeweils anderen Nutzer zu sehen. Die hohe Komplexität der OT-Algorithmen führt dazu, dass einerseits das Nachvollziehen und andererseits die testweise Implementation schwierig und aufwendig ist.

Diese Arbeit nimmt sich diesem Problem an und unterbreitet mit der Simulationsumgebung *Simone* auf konzeptioneller Ebene einen umfangreichen Lösungsansatz. Es werden unterschiedliche Teilkonzepte ausgearbeitet und vorgestellt, bei deren Umsetzung die Simulationsumgebung die Arbeit mit OT-Algorithmen erleichtern kann. In einer Fallstudie wird gezeigt, wie sich die Auseinandersetzung mit OT-Algorithmen ohne und mit der Simulationsumgebung gestaltet.

Im Ergebnis zeigt die Arbeit, dass ein an die Erfordernisse von OT-Algorithmen angepasstes Interface die Arbeit mit eben diesen unterstützen und erleichtern kann. Durch die Bereitstellung eines Rahmens für die Ausführung kann der Fokus von der Schaffung einer Bedienungsschnittstelle auf die Auseinandersetzung mit dem eigentlichen Algorithmus verlagert werden.

CR-Klassifikation, Key Word

I.6.7 Simulation Support Systems

Operational Transformation, Discrete-event simulation (DES), Userinterface

Abstract

The ubiquitous access to broadband internet has changed the collaboration over the last 15 years strongly. It represents a cornerstone for the real time collaboration. Another basic element which is crucial for real time collaboration are the algorithms. Both provide the fundament to exchange and process the data in real time. A particular method is Operational Transformation (OT). It allows multiple users to edit a shared document in real time and to see changes by all other users instantly. The high complexity of OT algorithms makes it difficult and costly to retrace the exact work and to implement an optimal execution environment, especially because the OT algorithms work distributed.

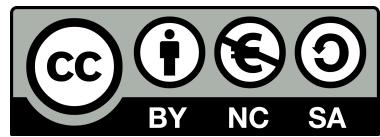
This Thesis investigates this problem and proposes the simulation environment *Simone* as a conceptual approach. The different parts of the approach will be elaborated and discussed. The case study compares the work with OT algorithms by using *Simone* and an alternative approach.

It has been shown that interface which is customized by the requirements of an OT algorithm can facilitate the work. By providing a framework the focus can be shifted from the programming parts to the algorithms itself.

CR-Klassifikation, Key Word

I.6.7 Simulation Support Systems

Operational Transformation, Discrete-event simulation (DES), Userinterface



Dieses Werk ist lizenziert unter einer Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz.

<http://creativecommons.org/licenses/by-nc-sa/4.0/>

This work is licensed under a Creative Commons Attribution 4.0 International License.

<http://creativecommons.org/licenses/by-nc-sa/4.0/>

Inhaltsverzeichnis

1. Einleitung.....	1
2. Grundlagen.....	4
2.1 Operational Transformation	4
3. Gegenstand der Arbeit.....	9
3.1 Herleitung.....	9
3.2 Ansatz.....	11
3.3 Simulationsstrategie	14
3.4 Alternative Ansätze	16
4. Grundgerüst der Simulationsumgebung	18
4.1 Discrete-Event Simulation	19
4.2 Client-Server-Modell.....	23
4.3 Backend	24
4.4 Frontend.....	27
5. Userinterface-fokussierte Konzepte.....	30
5.1 Definition eines Ausgangszustands	30
5.2 Traces	34
5.3 Semiautomatische Ausführung	41
5.4 Verteilte Bedienung.....	44
5.5 Highlighting.....	46
6. Backend-fokussierte Konzepte	52
6.1 Import und Export von Traces	52
6.2 Snapshots des Simulators.....	55
6.3 Break Points.....	58
6.4 Erweiterte Break Points.....	62
6.5 Protokolltreue	63
6.6 Externe Schnittstellen.....	68
7. Fallstudie	73
7.1 Simulationsumgebung.....	73
7.2 Eigenständiger Ansatz.....	81
7.3 Auswertung.....	84
8. Ausblick.....	86
I. Appendix.....	87
A. Verwendete Techniken.....	87

II. Literaturverzeichnis.....	90
III. Abbildungsverzeichnis.....	92

1. Einleitung

In diesem Jahr feiert Apple, der US-amerikanische Hard- und Softwarehersteller, den 30. Geburtstag seiner Personal-Computer-Serie Mac. Mit dem Macintosh Computer setzte Apple einen wichtigen Meilenstein in der Geschichte der Arbeitsplatzcomputer. Der relativ geringe Preis und das neue Bedienkonzept waren Grundlage dafür, dass Arbeiten, die zuvor mit analogen Mitteln erfolgten, nun zunehmend auch mit der Unterstützung von Computern durchgeführt wurden. Die Zusammenarbeit in Projekten geschah zu Beginn über unterschiedliche Wege, beispielsweise über das Austauschen von Datenträgern, auf denen die Dokumente gespeichert wurden, oder durch den direkten Ausdruck selbiger. Die Entwicklung setzte sich fort und so war es wenige Jahre später möglich, über ein Netzwerk Dokumente und Daten auszutauschen. Es wurde einfacher, Daten zu verteilen und an gemeinsamen Dokumenten zu arbeiten. Geändert hatte sich hingen nichts an der Tatsache, dass „Zusammen an einem Dokument arbeiten“ immer bedeutete: Arbeitet Alice an einem Dokument, auf das auch Bob zugreift, kann Bob mit diesem nicht arbeiten. Es war somit eine Serialisierung der Arbeitsschritte in Verbindung mit der gemeinsamen Abstimmung, wer wann am Dokument arbeitet, notwendig. Halbautomatische Lösungsansätze halfen, bekannten und häufig auftretenden Problemen zu entgegnen. Zu diesen Problemen zählte, dass nicht immer allen Beteiligten an einem Projekt zu jeder Zeit wussten, wer im Besitz des aktuellen Dokuments ist. Durch diese Unklarheiten konnte es passieren, dass Dokumentenversionen verloren gingen oder an einer älteren Version weitergearbeitet wurde, obwohl es bereits eine aktuellere bei einem der anderen Projektteilnehmer gab. Ein Lösungsansatz war die Verwendung von Versionierungssystemen. Jede Änderung führte zu einer neuen Version des Dokuments. Damit wurde über eine zentrale Stelle ausgehandelt, welche die aktuelle Version des Dokuments ist. Bekannte Anwendungen, die diese Funktionalität bieten, sind cvs, svn oder git. Dieser Lösungsansatz erforderte jedoch,

dass im Fall eines nicht automatisch auflösbarer Konflikts ein Nutzer manuell Eingreifen und diesen selbst auflösen musste. Nicht automatisch auflösbar Konflikte treten dann auf, wenn das System nicht selbst entscheiden kann, ob die Dokumentenänderung von Alice oder die von Bob die verbindliche ist.

Die Forschung an Lösungsansätzen für die geschilderte Problematik begann in den 1980er Jahren und brachte in Form von Spezialanwendungen auch die ersten Implementierungen hervor. Diese speziellen Ansätze eigneten sich jedoch nicht zu einer Generalisierung, wie Ellis und Gibbs [1] feststellten. Diese untersuchten vorangegangene Lösungsansätze und schlugen mit dem Distributed Operational Transformation (dOPT) in Verbindung mit GROVE einen Ansatz vor, der mit seinem Modell und Algorithmus universell nutzbar sein sollte. Folgende Veröffentlichungen [2] [3] [4] setzten sich mit dem Ansatz von Ellis und Gibbs auseinander und versuchten Probleme mit diesem aufzuzeigen und zu beheben. Daneben existieren noch unterschiedlichste weiterführende Ansätze auf welche an dieser Stelle nicht konkret verwiesen werden soll.

Zusammenfassend werden die vorgeschlagenen Ansätze unter dem Begriff Operational Transformation (OT) Algorithmen. Die Änderungen, Weiterentwicklungen und Alternativen kamen in unterschiedlichen Projekten zur Anwendung - häufig handelte es sich dabei um akademische Prototypen. Mit Etherpad [5] und Google Wave [6] wurden zwei Anwendungen veröffentlicht, die beide OT-Algorithmen verwenden und in der Öffentlichkeit einen großen Erfolg feierten.

Möchte man sich nun näher mit diesen konkreten Implementierungen im Speziellen oder aber OT-Algorithmen im Allgemeinen auseinander setzen, so werden Schwierigkeiten dieses Unterfangens dabei schnell gewahr, so zum Beispiel, dass es sich dabei um Anwendungen handelt, die mit verteilten Algorithmen arbeiten. Eine Ausführung und gleichzeitige Analyse durch einen Nutzer gestaltet sich ohne dezidierte Hilfsmittel schwierig. Oder sei es, gerade bei bestehenden Implementierungen wie Etherpad, dass die Anwendungen sehr komplex und umfangreich sind.

Diese enthalten neben dem Algorithmus in Code-Form auch eine große Menge an Anwendungscode, welcher für die Nutzung der Software selbst wichtig ist, jedoch zusätzlich zum OT-Algorithmus erschlossen werden muss. Dieser große Umfang erschwert das Verstehen der Funktionsweise. Die Hypothese lautet folglich: Die Möglichkeit, einen bestehenden Algorithmus, sei es in formaler oder in implementierter Form, in einer definierten Umgebung, die für eben jene Anforderungen konzipiert ist, auszuprobieren zu können, würde ein solches Vorhaben erleichtern. Die Untersuchung und Konzeption eben jener Möglichkeit ist Gegenstand dieser Arbeit. Eine allgemeine Simulationsumgebung für OT-Algorithmen soll es ermöglichen, bestehende oder in der Entwicklung befindliche Algorithmen möglichst einfach und effizient zu erproben. Dabei steht die Erarbeitung von Konzepten für die gezielte Interaktion mit der konkreten Implementation im Vordergrund.

Im Folgenden betrachtet die Arbeit im Kapitel 2 die Grundlagen von OT-Algorithmen und beleuchtet Problemfelder. Sie stellt den Lösungsansatz für die genannte Aufgabe in Kapitel 3 überblicksartig vor und diskutiert alternative Herangehensweisen. Im Detail wird der Lösungsansatz in Kapitel 4 beleuchtet. Im 5. Kapitel werden erweiterte Konzepte mit dem Fokus auf das Userinterface und im 6. Kapitel mit dem Fokus auf das Backend vorgestellt und diskutiert. Anhand einer kurzen und vergleichenden Fallstudie in Kapitel 7 wird der konzeptionelle Ansatz der Arbeit in Form der Simulationsumgebung *Simone* vorgestellt. Zum Abschluss wird in Kapitel 8 ein Ausblick darauf gegeben, welche weiterführenden Möglichkeiten sich als Nächstes zur Auseinandersetzung bieten und welche für sinnvoll erachtet werden. Im Appendix der Arbeit wird kurz auf die verwendeten Techniken bei der Umsetzung der Simulationsumgebung eingegangen.

2. Grundlagen

Gab das vorherige Kapitel einen Überblick darüber, für welche Problemstellungen sich OT-Algorithmen eignen und wo diese bereits zum Einsatz kommen, gibt das nun folgende Kapitel einen Einblick in die Forschung rund um das Thema Operational Transformation. Es erfolgt die Vorstellung ausgewählter Themenbereiche.

In Literatur und Praxis werden unterschiedliche Ansätze mit einem gemeinsamen Ziel verfolgt: Mehreren Teilnehmern soll es ermöglicht werden, gemeinsam und gleichzeitig an einem Dokument zu arbeiten. OT-Algorithmen bieten dabei einen Ansatz.

2.1 Operational Transformation

Eine kurze Erklärung soll die prinzipielle Funktionsweise eines OT-Algorithmus zusammenfassen. Gegeben sind die Autoren, welche die Rolle eines Senders und die des Empfängers von Dokumentenänderungen einnehmen können. Jeder Autor verfügt über eine lokale Kopie des gemeinsam bearbeiteten Dokuments. Der OT-Algorithmus übernimmt dabei drei Aufgaben. Erstens: Die lokalen Änderungen am Dokument auf der lokalen Dokumentenkopie vorzunehmen und danach an alle anderen Autoren zu versenden. Zweitens: Die Änderungen von anderen Autoren zu empfangen und auf der lokalen Dokumentenkopie einzupflegen, sodass diese zu dem vom Autor intendierten Ergebnis führen. Drittens: Stehen Änderungen an einer Dokumentenkopie in Konkurrenz zu anderen Änderungen, so muss der OT-Algorithmus diesen Konflikt automatisch und mit Hilfe einer Transformation der Änderungen so auflösen, dass das intendierte Ergebnis beider Änderungen erhalten bleibt.

Die zuletzt implizit formulierte Bedingung, dass das intendierte Ergebnis erhalten bleiben muss, ist eine, die einem konkreten Konsistenz-Modell zugeordnet werden kann. Durch das Konsistenz-Modell werden Bedingungen aufgestellt, welche nach der Ausführung aller Schritte des

OT-Algorithmus auf allen Dokumenten gelten müssen. Die Konsistenz-Modelle können je nach OT-Algorithmus variieren und müssen die Erfüllung dieser Bedingung nicht notwendig fordern.

Ellis und Gibbs

Das von den beiden Autoren Ellis und Gibbs 1989 veröffentlichte Paper [1] bildet einen wichtigen Punkt in der Forschung um das kollaborative Arbeiten mit Computersystemen. Die beiden Autoren zeigen mit dem dOPT-Ansatz, wie konkurrierende Änderungen von Autoren an einem Dokument in Echtzeit so verarbeitet werden können, dass Konflikte automatisch gelöst werden. Mit GROVE (GRoup Outline Viewing Editor) zeigen sie auch eine Anwendung, die den entwickelten OT-Algorithmus implementiert. Mehrere Nutzer konnten mit GROVE gemeinsam und gleichzeitig an einem Dokument arbeiten.

Mit der Herausarbeitung der Eigenschaften von „real-time groupware systems“ schufen sie die Grundlage für einen allgemeineren algorithmischen Ansatz. „highly interactive, real-time, distributed, volatile, ad hoc, focused, external channel“ zählen sie als eben solche Eigenschaften auf und verlangen, dass eine Softwarelösung für ein „real-time groupware system“ eben diese Eigenschaften erfüllen und zusichern muss.

Ellis und Gibbs definieren ein Modell, welches sich wie folgt umreißen lässt: Von mehreren unterschiedlichen Orten aus wird ein gemeinsames Dokument bearbeitet. An jedem Ort existiert eine Kopie des Dokuments. „Insert“ und „Delete“ sind Operationen, die auf jeder Kopie des Dokuments ausgeführt werden können und durch die das Dokument verändert wird. Die Autoren definieren zwei Konsistenzeigenschaften, die gemeinsam das Konsistenz-Modell des dOPT-Ansatzes bilden. Nach der Ausführung des Algorithmus muss die Dokumentenkopie an jedem Ort diese Eigenschaften erfüllen, damit es als korrekt gilt: Precedence Property und Convergence Property. Für die Precedence Property muss gelten, dass sich die Reihenfolge der Ausführung der Operationen an allen Orten gleicht. Für die Convergence Property muss gelten, dass nach Ausführung aller Operationen an allen Orten die Kopien aller Dokumente gleich sind. Neben dem Konsistenz-Modell gibt es noch eine Transformation Matrix und den Distributed Operational Transformation (dOPT) Algorithm. Die

Transformation Matrix dient dabei als „key to resolving conflicting operations“. Der dOPT Algorithm ist die Kernkomponente und beinhaltet die Ausführungslogik. Diese ist neben den Regeln, nach denen die Transformation vorgenommen wird, essentiell. Sie beschreibt, wann und wo Kopien von Operationen vorgehalten werden, welche Vergleiche mit welchen zwischengespeicherten Zuständen notwendig sind, sowie welche Entscheidungen und welche Zwischenschritte auf Basis dieser Erkenntnis zu treffen sind.

Seit der Veröffentlichung des dOPT Algorithm gab es unterschiedliche Publikationen, welche die veröffentlichten Erkenntnisse reflektierten. Schwierigkeiten mit dem Konsistenz-Modell und seiner Erfüllung wurden dabei als Kern der Probleme benannt.

Sun und Ellis

Die zuvor erwähnte Probleme des dOPT Algorithm in Verbindung mit der Implementation in GROVE zeigen Sun und Ellis in [4] auf. Sie benennen das divergence problem und die causality-violation als eine Ursache für auftretende Fehler bei der Ausführung – die schematische Aufbereitung und Erklärung findet sich auf Seite 7. Mit REDUCE und dem Generic Operational Transformation (GOT) control algorithm schlagen sie einen neuen und gegenüber dOPT deutlich komplexeren Ansatz vor. Dieser soll die zuvor aufgetretenen Probleme umgehen.

Durch die Einführung und Definition eines neuen Konvergenz-Modells verfolgen sie einen alternativen Ansatz zu dOPT. Mit convergence wird verlangt, dass sich nach der Ausführung der gleichen Menge von Operationen auch alle Dokumente gleichen. Mit causality-preservation wird verlangt, dass die Ausführungsreihenfolge der Operationen immer in der kausal gleichen Reihenfolge stattfindet. Mit intention-preservation wird gefordert, dass der Effekt der Ausführung einer Operation überall identisch sein muss und dieser unabhängige Operationen nicht beeinflussen darf.

Zur Erfüllung dieser drei Bedingungen werden eine Vielzahl von neuen Ausführungsschritten und Funktionen eingeführt. Ein zentrales Element ist der Generic Operational Transformation (GOT) control algorithm. Er stellt sicher, dass Vor- und Nachbedingungen, welche vor und nach der

Transformation erfüllt sein müssen, eingehalten werden. Eine weitere wichtige Neuerung ist die Einführung einer exclusion transformation (ET). Diese ist parallel zur inclusion transformation (IT) angesiedelt. Die ET hat die Aufgabe, den Effekt einer Operation O_B aus einer Operation O_A zu extrahieren. Sie macht also das Ergebnis der IT rückgängig. Weiterhin werden interne Operationen wie „do“, „undo“ und „redo“ eingeführt. Diese ermöglichen das Rückwärtsausführen von Operationen.

Im Ergebnis gelingt es mit dem vorgeschlagenen Ansatz die Fälle zu lösen, bei denen der dOPT- und auch der adOPTed-Algorithmus mit ihren Konsistenz-Modell nicht funktionieren.

Schwierigkeiten und Probleme

Durch unterschiedliche Autoren wurde in der Vergangenheit immer wieder Kritik an den veröffentlichten Operational Transformation Algorithmen geäußert. Randolph et al [7] untersuchten 2012 mehrere IT-Funktionen unterschiedlicher Algorithmen. Sie überprüften, ob diese die Kriterien TP1 und TP2 erfüllen. Ihre Erfüllung wurde als notwendig für ein korrekt arbeitendes groupware system postuliert [1] [8]. Zusammenfassend stellten die Autoren fest, dass die untersuchten Algorithmen diese Bedingungen nicht erfüllen. „[...] TP1 is satisfied by some IT functions based on the position and character parameters. Thus, it is impossible to meet TP2 with these simple signatures“ [7] lautet die abschließende Erkenntnis.

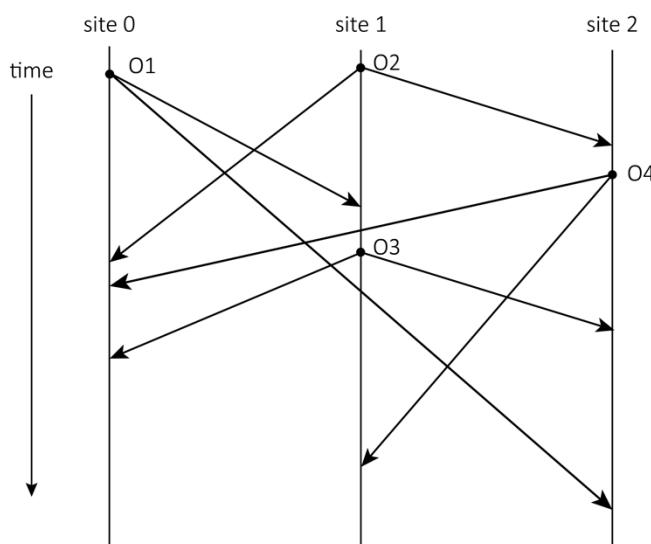


Abbildung 1 Beispielszenario, in dem es dem Algorithmus nicht möglich ist zu einem Ergebnis zu gelangen, dass alle geforderten Eigenschaften erfüllt. [4]

Doch schon früher wurden Schwierigkeiten mit veröffentlichten Algorithmen aufgezeigt. Detaillierter gehen Sun und Ellis [4] bereits 1998 auf die Veröffentlichungen von Ellis und Gibbs [1] ein. Die Verletzung der Konvergenz-Eigenschaft sowie die der Kausalität konnten auch hier als ein Problem aufgezeigt werden. Dies führt dazu, dass der dOPT Algorithm nicht korrekt funktioniert. Bereits ein einfaches Beispiel mit 4 Operationen, welche nicht die Kommutativitätseigenschaft erfüllen, führt demnach zu einer Verletzung der Konvergenz-Eigenschaft. Reduzieren und erkennen lässt sich das problematische Verhalten bei der genauen Betrachtung der Abbildung 1. Auf site1 wird die Operation O3 nach der Ankunft von Operation O1 ausgeführt. Auf site2 erfolgt die Ausführung in genau der anderen Reihenfolge. Dort wird erst Operation O3 empfangen und ausgeführt. Erst im Anschluss wird Operation O1 empfangen und verarbeitet. Wie von den Autoren zusammenfassend bemerkt, wird Operation O3 auf site2 in einem nicht existierenden Kontext ausgeführt und führt so zu einem falschen Ergebnis.

Anhand der beiden hier genannten Papiere lässt sich das Problemfeld von bestehenden OT-Algorithmen auf der algorithmisch Ebene gut erkennen. Zum einen sind es die Konsistenzmodelle, die mit ihren Eigenschaften den wichtigen Rahmen für die Algorithmen vorgeben. Verletzen die Operationen, die innerhalb eines Algorithmus arbeiten, diese Eigenschaften, kann dies dazu führen, dass Fehler bei der Ausführung auftreten. Zum anderen ist dies die Transformation zweier Operationen. Dies darf auch nicht dazu führen, dass sich die Eigenschaften der Operationen nachträglich in Bezug auf das Konsistenzmodell, verändern.

3. Gegenstand der Arbeit

Im vorherigen Kapitel wurde ein Überblick darüber gegeben, welchen Ansatz OT-Algorithmen verfolgen und welche Probleme im Zusammenhang mit diesen bereits bekannt sind. Im folgenden Kapitel wird der Ansatz dieser Arbeit vorgestellt. Nach einer Herleitung in Kapitel 3.1 erfolgt die überblickende Vorstellung des Ansatzes in Kapitel 3.2. Im anschließenden Kapitel 3.3 erfolgt die Erörterung des gewählten Vorgehens auf der Ebene der Simulationstechnik. Im letzten Kapitel 3.4 werden bekannte und mögliche Lösungsansätze zusammengefasst. Es erfolgt eine argumentative und mit Beispielen unterstützte Abgrenzung zu bereits existierenden Ansätzen und Möglichkeiten.

3.1 Herleitung

Die Auseinandersetzung mit bestehenden OT-Algorithmen kann aus unterschiedlichen Richtigen erfolgen. Eine solche Richtung kann sein, existierende Algorithmen und deren konkrete Funktionsweise nachzuvollziehen. Das Literaturstudium bietet dabei die Möglichkeit, einen Überblick über publizierte Ansätze zu erhalten. Auch für die Detailbetrachtung auf theoretischer Ebene eignet sich eine detaillierte und kritische Auseinandersetzung mit den Veröffentlichungen. Um das Verständnis für einen bestimmten OT-Algorithmus und dessen konkrete Arbeitsweise zu verbessern, ist es eine Option, die in der Literatur benannten Softwareumsetzungen – so vorhanden – für Versuche heranzuziehen und deren Funktionsweise genauer zu betrachten. Falls eine solche Anwendung zugänglich ist, gilt es abzuwägen, wie hoch der Einarbeitungsaufwand für den Code ist. Eine umfangreiche Quellcodebasis, eine Vermischung von Anwendungs- und Algorithmus-Code verbunden mit schlechtem oder unkommentiertem Code, können die Lösung dieser Aufgabe sehr schwierig und aufwendig werden lassen. Neue oder sehr alte, nicht mehr verwendete Programmiersprachen können den Einarbeitungsaufwand zudem erheblich steigern. Als

Schwierigkeit kann hinzukommen, dass die Anwendung für eine Untersuchung mit dem Ziel, ein besseres Verständnis über die Funktionsweise des OT-Algorithmus zu erlangen, ungeeignet ist. Dies kann beispielsweise dann der Fall sein, wenn die OT-Komponente nur ein sehr kleiner Teilaspekt der gesamten Anwendung ist und modular eingebunden wird. Der Aufwand, ein Gesamtverständnis für die Anwendung zu erarbeiten, um darauf aufbauend den OT-Algorithmus zu betrachten, ist schwer zu rechtfertigen. Ein allgemeineres Problem kann sein, dass ein punktueller Eingriff und die Manipulation von Werten während der Ausführung nicht möglich ist, was die experimentelle Auseinandersetzung sehr erschwert.

Eine weitere Richtung, aus der eine Auseinandersetzung mit Operational Transform erfolgen kann, ist die der (Weiter-)Entwicklung eines OT-Algorithmus. Zu Begin erfolgt das Nachvollziehen ausschließlich im Kopf, auf dem Papier oder dem Whiteboard. Viel wird im Gedankenexperiment ausprobiert, verworfen, überdacht und abgeändert. Dies passiert nicht immer im stillen Kämmerlein, sondern auch in Teamarbeit. Am Ende der ersten Iteration dieses Prozesses ist das Ergebnis im besten Fall ein Entwurf eines Algorithmus. Dieser kann, so der Autor die Möglichkeit dazu hat, im formalen Rahmen auf seine Richtigkeit überprüft werden. Ein formaler Beweis ist ein solcher möglicher Ansatz. Es kann unterschiedliche Gründe geben, warum zum aktuellen Zeitpunkt der Versuch eines Beweises viel zu früh oder gar nicht möglich ist. Hier kann zählen, dass der Algorithmus noch nicht ein einziges Mal an einem Beispiel erprobt wurde, dass dem, der den Algorithmus entwirft, die Erfahrung und die punktuellen Fähigkeiten fehlen, den Beweis korrekt zu führen. Oder, dass es zum Zeitpunkt des Entwicklungsstandes der formale Beweis in keiner Relation zum Aufwand steht, da das Ergebnis noch nicht stabil genug ist. Eine erste Implementierung des Algorithmus kann deshalb eine sinnvolle Möglichkeit darstellen. Für eine Erprobung, welche über das Abarbeiten einfachster Fälle hinaus geht, ist dies jedoch ungeeignet. Insbesondere die Tatsache, dass OT-Algorithmen verteilt arbeiten, ist hierfür verantwortlich. Eine Schnittstelle, die eine Bedienung des Algorithmus ermöglicht und unterstützt, ist hierbei ein wünschenswertes

Werkzeug. Diese muss neben einer validen Eingabe auch eine zulässige Ausführung sowie die Beobachtung der Abarbeitung ermöglichen. Der einmalige Konzeptions- und Entwicklungsaufwand hierfür ist im Kontext der dargelegten Bedingungen nicht zu unterschätzen.

Diese ausgewählten Beispiele zeigen anschaulich welche Fragen neben den eigentlichen und den Algorithmus betreffenden, noch zu beantworten sind, möchte man sich mit Operational Transformation praktisch auseinandersetzen. Beide Beispiele zeigen, dass es neben dem eigentlichen Forschungs- und Arbeitsgegenstand, dem Algorithmus, noch viele weitere Aufgaben gibt, welche mal im größeren, mal im geringeren Umfang anfallen. In jedem Fall müssen sie bewältigt werden und kosten möglicherweise wertvolle Zeit. Wären diese Fragen beantwortet und durch praktische Hilfestellungen unterfüttert, dann könnte eine Forschung und Entwicklung mit einem klareren Fokus auf den Algorithmus erfolgen.

3.2 Ansatz

Der im Folgenden dargelegte Ansatz hat das Ziel, die Forschung an und die Entwicklung von OT-Algorithmen zu unterstützen. Durch die Konzeption und Entwicklung der Simulationsumgebung *Simone* soll die Möglichkeit geschaffen werden, bereits existierende oder in einer frühen Entwicklungsphase befindliche Algorithmen gezielt ausprobieren zu können. Durch die Bereitstellung eines Grundgerüsts, dass auf die Erfordernisse von OT-Algorithmen sowohl auf Programmier- als auch auf Bedienebene zugeschnitten ist, wird eine Möglichkeit geschaffen, den zuvor beschriebenen Schwierigkeiten, welche die Konzeption und Entwicklung einer lauffähigen Umgebung eines OT-Algorithmus unterworfen sind, aus dem Weg zu gehen.

Insbesondere ist bei der Entwicklung von Anfang an interessant, bietet *Simone* die Möglichkeit, einen OT-Algorithmus zu implementieren, ohne dabei die Möglichkeit zur Bedienung des selbigen umsetzen zu müssen. Die Bereitstellung wichtiger Funktionen auf konzeptioneller und praktischer Ebene sind dabei wichtige Vorteile gegenüber einer eigenständigen Entwicklung.

Im Detail

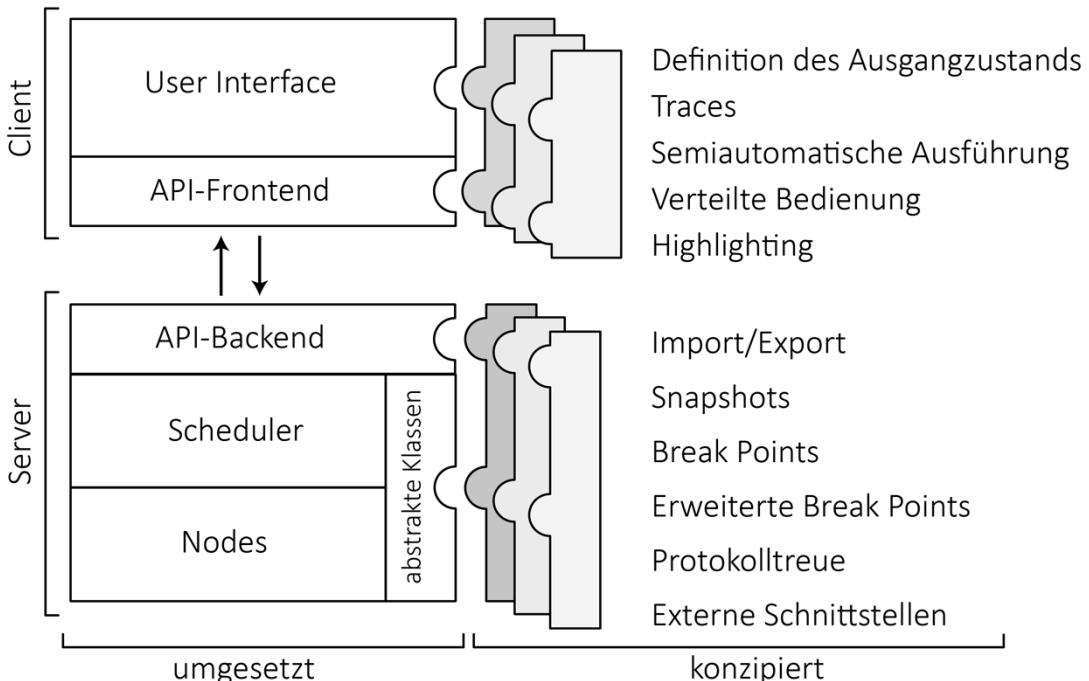


Abbildung 2 Strukturelle Gliederung des Ansatzes

Der Ansatz gliedert sich in zwei Teile. Der erste Teil befasst sich mit der Möglichkeit, einen Algorithmus in die Simulationsumgebung zu integrieren und auszuführen. Dies geschieht primär im Abschnitt des Servers. Eine Sammlung aus abstrakten Klassen ermöglicht es bei der Implementation des Algorithmus auf viele Grundfunktionen zurückzugreifen und mit wenigen Zeilen den eigentlichen Algorithmus sowie notwendige Hilfsfunktion umzusetzen. Die Steuerung erfolgt clientseitig durch eine externe Oberfläche, über welche die Nodes getrennt voneinander gesteuert werden können. Der Status jedes einzelnen Nodes ist jederzeit über das Interface einsehbar. Die Ausführung des Algorithmus erfolgt in einzelnen Schritten, so dass auch Zwischenzustände durch den Nutzer beobachtet und bewertet werden können.

Der zweite Teil des Ansatzes befasst sich mit Erweiterungen der Simulationsumgebung, sowohl auf Seiten des Servers als auch auf Seiten des Clients. Durch die Erweiterungen werden bereits existierende Mittel und Werkzeug, welche die Auseinandersetzung mit dem Algorithmus unterstützen, verbessert. Gewisse Erweiterungen schaffen auch komplett

neue Möglichkeiten. Es folgt eine kurze Motivation für die einzelnen später ausführlich vorgestellten Konzepte.

Definition eines Ausgangszustands - Bei der Erstellung einer Simulationsinstanz kann es gewünscht sein, unterschiedliche Nodes in einen anderen als den leeren Ausgangszustand zu versetzen. Dies kann durch das Definieren von Ausgangszuständen ermöglicht werden.

Traces - Die Aufzeichnung der Ausführung der Simulation – nachfolgendes als Traces bezeichnet – ermöglicht, das Abarbeitungsergebnis im Kontext der Ausführungsreihenfolge zu betrachten und zu bewerten. Ein erneutes Ausführen eines einmal aufgezeichneten Prozesses ermöglicht das Nachvollziehen der Ausführung zu einem späteren Zeitpunkt und im Detail.

Semiautomatische Ausführung - Das Ausführen von umfangreicheren Simulationen kann eine große Anzahl von Ausführungsschritten erfordern. Eine manuelle Ausführung ist in einem solchen Rahmen mühsam und langwierig. Die semiautomatische Ausführung bietet die Möglichkeit des kontinuierlichen und automatischen Fortschritts in der Simulation. Gleichzeitig ermöglicht es den sofortigen Eingriff, sollte dies gewünscht oder notwendig sein.

Verteilte Bedienung - Die verteilte Bedienung ermöglicht es mehreren Nutzer unterschiedliche Teile einer Simulationsinstanz unabhängig voneinander zu bedienen.

Highlighting - Durch die semiautomatische und vom Computer gesteuerte Ausführung ist dem Nutzer nicht immer direkt ersichtlich, wo in einem Augenblick überall eine Veränderung erfolgt. Dies soll dadurch ermöglicht und verbessert werden, dass ein Hervorheben der Änderungen im Interface sowie das Vor- und Zurückspringen innerhalb der einzelnen Veränderungen ermöglicht wird.

Import/Export - Der Export ermöglicht es die Ergebnisse der Ausführungen auch über die Laufzeit der Simulationsumgebung aufzubewahren. Auch können sie über den Import zu einem späteren Zeitpunkt erneut eingelesen und für eine erneute Ausführung verwendet werden.

Snapshots - In Situationen, in denen die Ausführung an einen Punkt gelangt, an der unterschiedliche Ausführungspfade möglich sind, kann es sinnvoll sein, den Zustand zu speichern um später einen alternativen Weg zu verfolgen. Diese Möglichkeit schaffen Snapshots.

Break Points - In einem komplexen Simulationsszenario kann es schwierig sein alle Nodes, ihre Zustände und andere Randbedingungen im Überblick zu behalten. Break Points mit bestimmten Bedingungen zu verknüpfen kann dem Nutzer bei genau dieser Herausforderung unterstützen. Sie bieten die Möglichkeit bestimmte Aktionen auszuführen oder das Ausführungslog um weitere Informationen anzureichern, wenn ein bestimmte Bedingung zutrifft.

Protokolltreue - Bei der Netzwerkkommunikation können verschiedene Netzwerkeigenschaften, welche für die Simulation bestimmter Eigenschaften eines OT-Algorithmus relevant sind, bei der Ausführung berücksichtigt und verwendet werden.

Externe Schnittstellen - Die Erprobung von bestehenden Algorithmen unter Verwendung der Simulationsumgebung kann eine interessante Erweiterung darstellen. Über eine externe Schnittstelle kann die Simulationsumgebung mit der bestehenden Anwendung kommunizieren und so den OT-Algorithmus ansprechen.

3.3 Simulationsstrategie

Beim Betrachten des Themas der Arbeit fällt die Wahl der Simulationsstrategie sehr schnell auf die ereignisorientiere Simulation – discret-event simulation (DES). Robinson [9] beschreibt die DES als „the system is modelled as a series of events, that is, instants in time when a state-change occurs“. Dies passt sehr gut zur Struktur der Arbeit von OT-Algorithmen: Eine Änderung an einem Dokument spiegelt ein Ereignis wider, dessen Ausführung zu einer Zustandsänderung des Dokuments führt. Zu Zustandsänderungen kommt es auch durch andere Aktionen innerhalb der Simulationsumgebung und der einzelnen Nodes, zum Beispiel durch das Versenden von Nachrichten.

Zu Beginn der Auseinandersetzung mit der Problemstellung stand die Frage, ob die Eigenentwicklung einer Simulationsumgebung der richtige Schritt sei. Die Entscheidung für die Entwicklung einer eigenen Softwarelösung schließt immer mit ein, dass dabei möglicherweise Fehler gemacht werden, die andere Ansätze mit ihrer Lösung bereits durchlebt und erfolgreich gelöst haben. Es wäre also in dieser Hinsicht eine gute Entscheidung, auf bereits existierende und erprobte Umsetzungen aufzubauen. Für die Ausführung einer DES-Simulation existiert eine Vielzahl von Simulatoren oder Frameworks, mit denen ein Simulator für einen bestimmten Algorithmus entwickelt werden kann. Es entsteht also die berechtigte Frage, warum der vorgeschlagene Ansatz auf die Eigenentwicklung setzt.

Für den gewählten Ansatz, selbst eine Simulationsumgebung zu entwickeln und nicht eine bestehende Lösung zu verwenden, sprechen mehreren Aspekte. Anders als bei Simulationen üblich, handelt es sich bei dem vorgeschlagenen Ansatz nicht nur um die Simulation eines bestimmten Algorithmus mit unterschiedlich parametrisierten Eingaben. Stattdessen sollen ganz unterschiedliche Algorithmen in ganz unterschiedlichen Szenarien ausgeführt und interaktiv erprobt werden können. Erprobung heißt dabei, dass der Nutzer über die Möglichkeit verfügen, soll auf experimentelle und spielerische Weise unterschiedliche Szenarien auszuprobieren und ihre Auswirkungen zu erfahren. Durch den weiten und schwer einzugrenzenden Rahmen ist es wichtig, dass in einem definierten und abgesteckten Kontext eine hohe Flexibilität besteht, um auf die gegebenen Erfordernisse entsprechend eingehen zu können. Das Anwendungsszenario der Simulationsumgebung beinhaltet explizit nicht die Überprüfung und Beantwortung einer konkreten Fragestellung auf Basis von Vergleichswerten zwischen unterschiedlichen Ausführungsszenarien. Stattdessen geht es darum, eine Bedienoberfläche für die Gruppe von OT-Algorithmen zu schaffen, um deren Ausführung beobachten und verfolgen zu können. Eine gebrauchstaugliche und für den Kontext der Nutzung angepasste Oberfläche ist in der hier konzeptionierten und vorgeschlagenen Lösung ein wichtiges Abgrenzungsmerkmal von anderen denkbaren Ansätzen. So hat die Ausführungsgeschwindigkeit

des Simulators keine bedeutende Rolle im Gesamtkonzept der Simulation. Je nach Simulationsaufgabe werden im Vergleich zu anderen Simulationen wenige Iterationsschritte benötigt, die zudem weder rechenintensiv noch zeitkritisch sind. Die Nutzung einer elaborierten Umgebung mit maximaler Ausführungsgeschwindigkeit ist aus dieser Perspektive somit nicht notwendig.

Betrachtet man die drei zuvor und in Abbildung 3 aufgeführten und zuvor erläuterten Kernpunkte und stellt diese in Beziehung zu der allgemeinen Vergleichstabelle von Robinson, so stellt sich die Umsetzung des benannten Konzepts als günstiger und möglicher Lösungsansatz heraus.

Feature	Spreadsheet	Programming language	Specialist simulation software
Range of application	Low	High	Medium
Modelling flexibility	Low	High	Medium
Duration of model build	Medium	Long	Short
Ease of use	Medium	Low	High
Ease of model validation	Medium	Low	High
Run-speed	Low	High	Medium
Time to obtain software skills	Short (medium for macro use)	Long	Medium
Price	Low	Low	High

Abbildung 3 Vergleichstabelle von Ansätzen für eine Simulation – Robinson [9] S. 42

3.4 Alternative Ansätze

Zur Lösung der Problemstellung sind unterschiedliche Ansätze neben dem oben aufgeführten denkbar. Mehrere dieser Ansätze wurden erprobt und sind auf Grund unterschiedlicher Unzulänglichkeiten keine vollwertige Alternative zu dem gewählten und in dieser Arbeit vorgestellten Ansatz. Nachfolgend wird auf zwei ausgewählte Alternativen eingegangen.

Aus technischer Sicht liegt die Verwendung von Unitests sehr nahe. Sie bieten die Möglichkeit, programmierte Funktionen mittels bestimmter Eingaben auszuführen und die Ausgaben auszuwerten. Dabei können unterschiedlichste Szenarien in Form von parametrisierten Eingaben definiert und in Folge ausgeführt und ausgewertet werden. Bei diesem Ansatz, der nur die äußere Ansteuerung der programmierten Umsetzung

einbezieht, fehlt die Berücksichtigung des Wunsches der aktiven Interaktion mit dem Algorithmus. Während der Ausführung der Unitests gibt es üblicherweise nicht mehr die Möglichkeit, in diese einzutreten und Änderungen vorzunehmen. Auch besteht nicht die direkte Möglichkeit, die durch die Ausführung geschehenden Veränderungen zu verfolgen und zu beobachten.

Als Ergänzung zu diesem Ansatz könnte die Nutzung einer IDE und der in ihr implementierten Debugging-Umgebung betrachtet werden. Diese kann genutzt werden, um den Mangel der Nachverfolgbarkeit der Ausführung zu kompensieren. Wie die Fallstudie im Kapitel 7 jedoch zeigt, bieten diese die IDE und die Debugging-Umgebung immer nur einen sehr fokussierten und technischen Blick auf den vom Algorithmus gerade ausgeführten Schritt. Eine detaillierte Auseinandersetzung erfolgt in der Fallstudie.

Als zweite Möglichkeit bietet sich die Verwendung einer bestehenden Simulationsumgebung als ein geeigneter Weg an. Dabei kann eine bereits gereifte und solide Umgebung genutzt werden, welche für die Simulation von Algorithmen geeignet ist. Konkret bedürfte es der Auswahl der entsprechenden Simulationsumgebung und der Implementation der entsprechenden Teile des ausgewählten Algorithmus in der Sprache, die durch Simulationsumgebung verwendet wird. Dabei hängt die Tatsache, ob die gesteckten Ziele erreicht werden, entscheidend von der verwendeten Simulationsumgebung und ihren Anpassungsmöglichkeiten im Verhältnis zur Komplexität ab. Als ein solcher Vertreter einer solchen Simulationsumgebung kann *Mathematica* [10] angesehen werden. Bei genauer Betrachtung besteht die berechtigte Annahme der Tauglichkeit für das Vorhaben. So bietet *Mathematica* unterschiedliche Möglichkeiten der Implementierung von Algorithmen. Zudem können über externe Schnittstellen Module in anderen Programmiersprachen entwickelt und mittels *Mathematica* angesprochen werden.

Dieser mögliche Ansatz wurde nicht weiter verfolgt, da wenig exakte Kenntnis über die genauen Details von *Mathematica* zu Beginn der Arbeit bestand, die eine Zeitabschätzung für eine erfolgreiche Bewältigung des Vorhabens erlaubt hätten.

4. Grundgerüst der Simulationsumgebung

Der zuvor allgemein umrissene Ansatz wird in den nächsten Kapiteln im Detail betrachtet. Dabei wird in diesem Kapitel auf die wesentlichen Konzepte und Funktionsweisen eingegangen. Diese bilden das Grundgerüst der Simulationsumgebung *Simone*. Sie wurden in einer ersten Umsetzung bereits prototypisch implementiert.

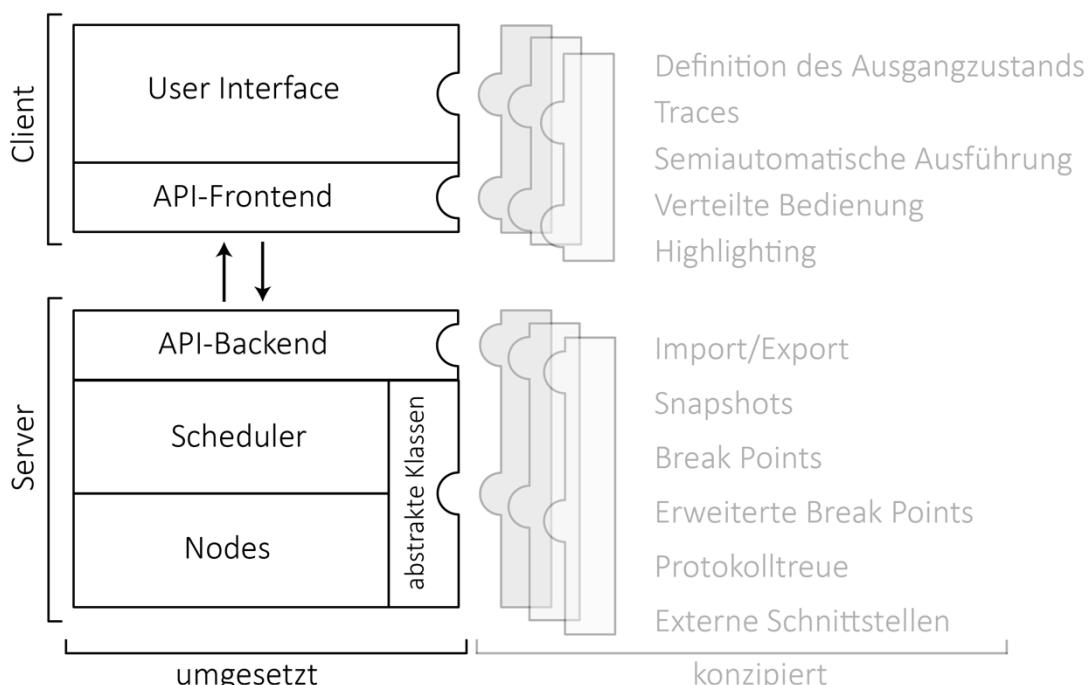


Abbildung 4 Auf der linken Seiten ist die Gliederung des Grundgerüsts der Simulationsumgebung hervorgehoben

In Abbildung 4 ist zu sehen, wie sich die Simulationsumgebung gliedert. Deutlich wird hierbei noch einmal hervorgehoben, wie sich das Grundgerüst von der Erweiterung strukturell abgrenzt.

Nachfolgend wird in Kapitel 4.1 auf das Simulationskonzept eingegangen. In Kapitel 4.2 erfolgt die Einordnung des Client-Server-Konzepts in den Kontext der Simulationsumgebung. Im folgenden Kapitel 4.3 wird die Konzeption und Struktur der Simulationsumgebung auf Seiten des Servers und in Kapitel 4.4 die Konzeption des Clients erörtert. Der Begriff Backend wird dabei synonym für den Teil der Simulationsumgebung verwendet, der auf dem Server ausgeführt wird. Die Verwendung von

Frontend erfolgt synonym für den Teil der Simulationsumgebung, der beim Nutzer im Browser ausgeführt wird.

4.1 Discrete-Event Simulation

Die Autoren Hartmann und Schwetman [11] benennen die Ziele einer Discrete-Event Simulation (DES) im Zusammenhang mit der Simulation eines Computersystems. Aufbauend auf der sich daraus ergebenden Gliederung wird diese nachvollzogen. Dies dient dazu, den besonderen Charakter der Simulationsumgebung einzuordnen. Wie bereits in der Beschreibung des Ansatzes erläutert, dient die Simulationsumgebung der Schaffung eines allgemeinen Ausführungsrahmens von OT-Algorithmen. Damit hat sie andere Aufgaben und Ziele, als eine übliche Simulationsumgebung, welche eine DES durchführen.

Im Folgenden findet eine Überprüfung sowie die Diskussion des Ergebnisses, ob *Simone* in die vorgeschlagenen Kategorien passt, statt.

Goals

Die Aufgabe der Simulationsumgebung ist es, dem Nutzer ein besseres Verständnis für die Funktionsweise eines bestimmten OT-Algorithmus zu ermöglichen. Es ist nicht das Ziel, die Ausführungsgeschwindigkeit des Algorithmus, die Nutzung von Ressourcen oder das Kommunikationsverhalten zum primären Zweck der vergleichenden Analyse zu erheben. Damit unterscheidet sich *Simone* bereits im ersten und einem sehr wesentlichen Punkt, nämlich ihrer Zielstellung, von herkömmlichen Simulatoren. Dieses veränderte Grundziel erlaubt es nun auch bei der gesamten Konzeption der Simulationsumgebung den Fokus eher in Richtung auf Interoperabilität und Steuerungsmöglichkeiten zu legen. Unberührt davon bleibt die Anforderung der Exaktheit und der Nachvollziehbarkeit in der Ausführung.

Resources and Entities

Nach Hartmann und Schwetman gibt es eine Unterteilung der Ressourcen in „Active Resources“ und „Passive Resources“. Active Resources werden als „typically used or occupied for specific interval of time“, Passive Resource werden als „obtained by an entity“ beschrieben. Entities sind demnach Teile

dieser Simulation, die Resources nutzen oder verarbeiten. Zudem kann es zu einer Konkurrenzsituation zwischen den Entities kommen.

In der Simulationsumgebung können nach diesen Kriterien Client- und Server-Nodes den Active Resources und Netzwerk-Nodes den Passiv Resources zugeordnet werden. Auf Programmierebene werden Clients- und Server-Nodes dabei intern auch als Active Nodes und Netzwerk-Nodes als Passive Nodes eingeordnet. Active Nodes führen Operationen aus und arbeiten mit diesen. Passive Nodes leiten Transport Capsules lediglich weiter. Die hierarchische Einordnung erfolgt dabei über die Vererbung von unterschiedlichen Eigenschaften, welche mit der jeweiligen Klasse verbunden sind. Bei Transport Capsules handelt es sich um Objekte, welche zur Übertragung von Nachrichten zwischen Nodes dienen. In ihnen können zur Übertragung bestimmte Daten gespeichert und dann über den Netzwerk-Node versendet werden.

Operations und Transport Capsules können nach der Einordnung von Hartmann und Schwetman den Entities zugeordnet werden. Beide werden von den Ressourcen verarbeitet.

Workload

Die Bestimmung des Workloads eines Systems wird von Hartmann und Schwetman als eine der wichtigsten Grundlagen für die Erstellung eines exakten und nützlichen Modells genannt. Es dient dazu, den Verbrauch von Ressourcen eines Systems zu beschreiben.

Wie bereits im Abschnitt Goals benannt, ist es das Ziel von *Simone*, die Nachvollziehbarkeit der Arbeit des Algorithmus zu ermöglichen. Explizit ausgeschlossen wird dabei die Messung des Ressourcenverbrauchs oder anderer, quantitativer Einordnungen. Eine Bestimmung des Workloads ist daher nicht notwendig und findet folglich nicht statt.

Output: Measures of Performance

Wie schon bei der Definition eines Workloads, so fällt es auch bei der von Hartmann und Schwetman angegebenen Kategorie der Leistungsmessung schwer, diese auf die Simulationsumgebung zu definieren und anzuwenden. Ein theoretisch denkbarer Ansatz wäre die Messung der Source Lines of Code, die es braucht, um einen OT-Algorithmus aus einem veröffent-

lichten Papier in einen lauffähigen und bedienbaren Zustand zu überführen. Hierzu muss eine vergleichende Untersuchung in einem abgesteckten Rahmen vorgenommen werden, um belastbare und vergleichbare Werte zu liefern. Dabei ist die Schwierigkeit der Verwendung von Source Lines of Code als Gütekriterium zu beachten und zu diskutieren. Dies erfolgt an dieser Stelle jedoch nicht weiter.

Ausführungsmodell

An diesem Punkt wird die Struktur von Hartmann und Schwetman kurz verlassen. Sie erörtern an dieser Stelle den Punkt „Duality of Models“, welche sich mit den unterschiedlichen „Simulation Entities“ in der Simulation von Computersystemen auseinandersetzt. In der behandelten Form ist dies für die Arbeit nicht relevant. Stattdessen wird betrachtet unter welchem Ausführungsmodell die Simulation erfolgt. Die Bestimmung der Ausführungsreihenfolge der Nodes erfolgt im manuellen Modus allein durch den Nutzer und seine Auswahl. Der Nutzer legt fest, welcher Node welche Operationen verarbeitet und ausführt. Ebenso kann er im Netzwerk die Reihenfolge der Nachrichten steuern. Dabei kann der Nutzer auf unterschiedliche Arten unterstützt werden. Das Konzept der Protokolltreue ermöglicht dabei sowohl im manuellen als auch im semiautomatischen Modus, das Einhalten bestimmter Eigenschaften des verwendeten Netzwerkprotokolls. Einer freien und davon unabhängigen Reihenfolge steht dabei jedoch nichts im Weg.

Darüber hinaus kann bei der semiautomatischen Ausführung die Ausführungsstrategie des Schedulers die Reihenfolge der Nodes beeinflussen. Die Nichtvorhersagbarkeit der Ausführungsreihenfolge wird durch die Verwendung einer randomisierten Auswahl und unter der Verwendung von Seeds, um eine Wiederholbarkeit in der Ausführung zu ermöglichen, modelliert. Bei anderen Strategien bedarf es der Modellierung dieser durch eigenen Code des Nutzers.

Object-Oriented Models of Systems

Wie von den Autoren empfohlen, nutzt auch *Simone* objektorientierte Programmietechniken für den Aufbau der Simulationsumgebung. Dies ermöglicht insbesondere die einfache Wiederverwendung von vorhande-

nen Schnittstellen bei der Neuimplementierung der OT-Algorithmen. Die Kapselung dient zur Abstraktion von Funktionalität, die nach außen und für den Nutzer nicht sichtbar sein muss.

Quasi-parallelism in System Models

Hartmann und Schwetman beschreiben quasi-parallelism als ein typisches Verhalten eines simulierten Systems: „[...] it is typical for the simulated execution of these multiple processes to be interleaved sequentially on a single processor“. Ein besonderes Erfordernis ist, dass die Ausführungsreihenfolge der Nodes in der Simulationsumgebung deterministisch erfolgen muss. Insbesondere ist dies auch dann der Fall, wenn es sich um ein tatsächlich parallel ausgeführtes System handelt. Die deterministische Reihenfolge der Ausführung bietet die Grundlage dafür, dass „[...] results are repeatable from simulation run to simulation run“.

Aus zwei Gründen wurde auch bei *Simone* der „quasi-parallelism“-Ansatz umgesetzt: Zum einen, weil die höhere Ausführungsgeschwindigkeit von einer echten parallelen Ausführung nicht benötigt wird und zum anderen weil der „quasi-parallelism“-Ansatz einfache zu implementieren ist.

Zusammenfassung

Simone lässt sich in Teilen sehr passend in das von Hartmann und Schwetman vorgegebene Raster einordnen. An den Punkten, an denen es nicht in das Raster passt lassen sich deutlich die Unterschiede zu einer klassischen Simulationsumgebung erkennen. Dies ist durch die Zielsetzung der Simulationsergebnung bedingt und eine bewusste Designentscheidung. Auswirkungen hat dies dadurch auch auf andere Charakteristika der Simulationsumgebung. Hierzu zählt unter anderem nicht notwendige Bewertung von „Workload“ und „Measurement of Performance“.

4.2 Client-Server-Modell

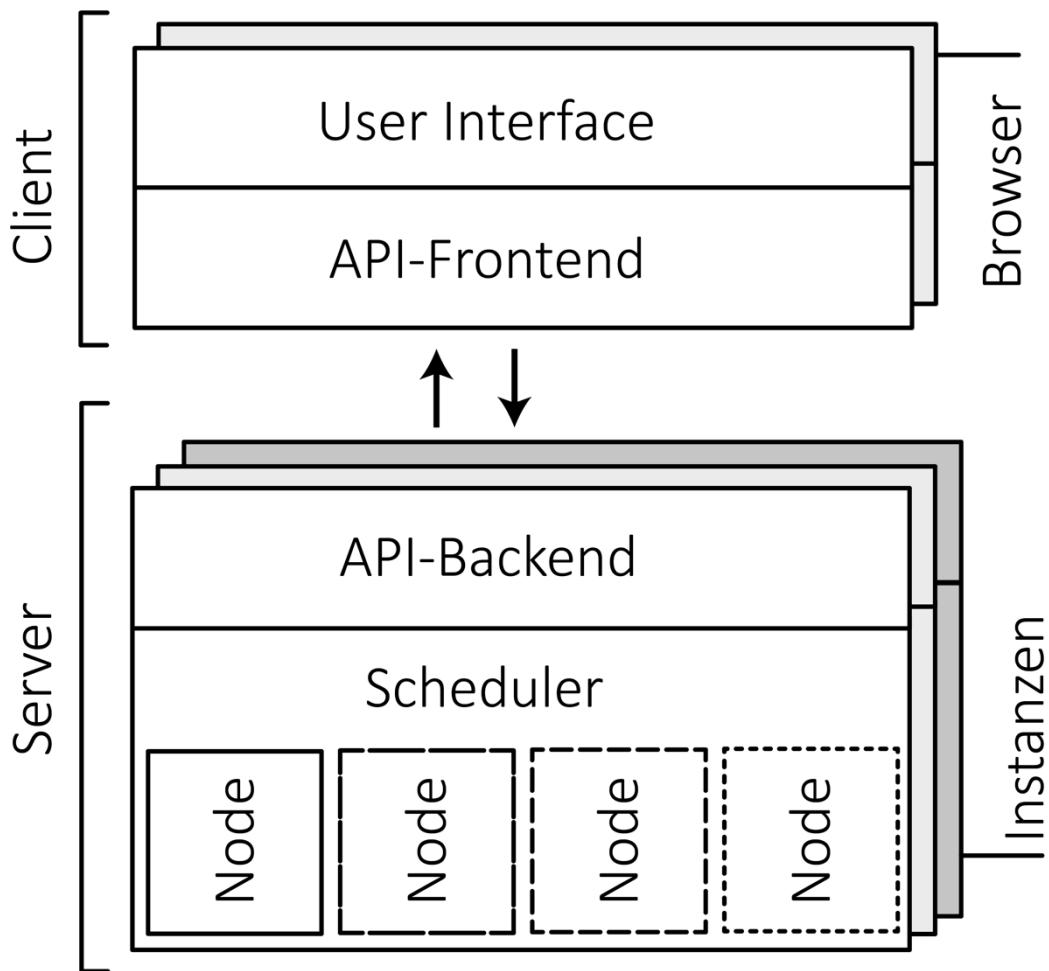


Abbildung 5 symbolische Darstellung eines ausgeführten Servers mit mehreren Instanzen

Unter der Verwendung des Client-Server-Modells ist im Kontext der Simulationsumgebung die in Abbildung 5 skizzierte Situation zu verstehen. Das Userinterface (UI) mit dem API-Frontend bildet den im Webbrowser des Nutzers ausgeführten Client. Die erweiterten Konzepte mit ihrem clientseitigen Schwerpunkt ermöglichen die Erweiterung des Clients. Die Nodes, der Scheduler und das API-Backend bilden den Server, welcher losgelöst vom Client ausgeführt wird. Auch hier bilden erweiterte Konzepte mit einem serverseitigen Schwerpunkt weitere Möglichkeiten, die Simulationsumgebung um Funktionen zu erweitern. Client und Server können nur zusammen zielführend betrieben werden. Eine voneinander losgelöste, das heißt alleinige Verwendung, ergibt sich,

anders als es das allgemeine Verständnis des Client-Server-Modells auch ermöglicht, nicht.

In einer Simulationsumgebung können unterschiedliche Instanzen gleichzeitig ausgeführt werden. Eine Instanz wird aus einem Scheduler und mehreren Nodes gebildet und kann über einen eindeutigen Bezeichner über den Client angesprochen werden.

Die Umsetzung des Client-Server-Modells erfolgte, um den Zugriff auf den Simulator dauerhaft von unterschiedlichen Orten aus zu ermöglichen. Dies ist in der Phase der Diskussion über die Ergebnisse der Simulation mit Dritten wichtiger als in jener, in der eine konzentrierte Entwicklung und Erprobung durch den Nutzer erfolgt. In der Phase des Austausches ist es hilfreich, Ergebnisse oder Logs mit anderen nicht nur zu besprechen sondern diese auch einsehen zu können. Die Entstehung des Ergebnisses in einer dafür zugeschnittenen Umgebung nachvollziehen zu können, ist dabei eine nicht zu unterschätzende zusätzliche Option. Mit dem Diskussionspartner einfach und ohne Schwierigkeiten bei der Ausführung der Simulationssoftware auf die gleichen Werkzeuge zurückzugreifen wird durch die Umsetzung des Client-Server-Modells ermöglicht.

4.3 Backend

Das Backend versammelt die Funktionalität des Simulators, die neben dem OT-Algorithmus und seinen Hilfsfunktionen zusätzlich für die Ausführung der Simulationsumgebung notwendig sind.

Das Backend lässt sich, wie in Abbildung 4 grob skizziert, in drei Bereiche unterteilt: die Nodes, den Scheduler und die API. Ein Node dient zur Ausführung und Verarbeitung von Operationen. Der Scheduler dient als Kommunikations- und Steuerschnittstelle zwischen den Nodes. Die API ermöglicht die Steuerung des Schedulers über das Frontend.

Konkrete Implementationen aus allen drei Bereichen ergeben sich durch Vererbung abstrakter Klassen. Abstrakt bezieht sich in diesem Rahmen auf die Tatsache, dass eine Instanziierung dieser Klassen zwar möglich, aber nicht intendiert ist. Die Programmiersprache python bietet keine

direkte Möglichkeit zur Definition einer abstrakten Klasse, die immer implementiert werden muss und nicht direkt instanziert werden kann, wie dies Beispielsweise in Java möglich ist.

Durch Vererbung und die vervollständigende Implementierung der abstrakten Klassen entstehen die nutzbaren Komponenten der Simulationsumgebung auf der Ebene der Programmierung. Zwei dieser Klassen ermöglichen es Nodes als aktiv und passiv einzuordnen und ihnen besondere Funktionalitäten zu geben. Client- und Server-Nodes sind aktive Nodes, der Netzwerk-Node ist ein passiver Node. Passive Nodes unterscheiden sich an zwei wesentlichen Stellen von aktiven Nodes: Sie verfügen nicht über ein Dokumentenmodell oder Operationen und leiten eingehende Nachrichten lediglich an die Empfänger weiter. Aktive Nodes instanzieren ein versionierendes Dokumentenmodell und verfügen über die Möglichkeit definierte Operationen auszuführen.

Das versionierende Dokumentenmodell ermöglicht das Nachvollziehen von Änderungen, die am lokalen Dokument vorgenommen wurden. Ein State des Dokumentenmodells wird durch eine Versionsnummer, die für das lokale Modell eindeutig ist, für die Operation und das Dokument, das durch die Anwendung der Operation entstanden ist, charakterisiert. Das Modell speichert alle States und referenziert auf den aktuellen State. Änderungen an einem realen Dokument werden in der Simulation dadurch abgebildet, dass diese operationalisiert werden. Infolgedessen besteht die Möglichkeit, Berechnungen auf und mit den Operationen vorzunehmen. Im konkreten Fall der aktiven Nodes werden Operationen bei der Implementation des OT-Algorithmus definiert. Übliche Operationen sind insert und delete. Darüber hinaus sind auch viele weitere Operationen denk- und umsetzbar und vom OT-Algorithmus abhängig. Durch das Bereitstellen der abstrakten Klasse für Operationen wird eine Standard-Schnittstelle für die Umsetzung nutzerspezifischer Operationen angeboten. Eine Erweiterung um eigene Funktionen ist somit einfach möglich. Durch die Nutzung eines Präfixes bei Funktionsnamen wird auf der Ausführungsebene von python definiert, dass es sich um Funktionen der Operational Transformation handelt. Die Nodes haben über den Aufruf dieses Funktionsnamens in Kombination mit den jeweiligen Präfix

die Möglichkeit, durch die Ausführung der Funktionen Änderungen auf dem aktuellen Dokumentenmodell durchzuführen.

Der Scheduler bildet den zweiten Bereich des Backends. Dieser übernimmt die Aufgabe des Managements und der Ausführung der Nodes. Management umfasst dabei das Hinzufügen und Löschen von Nodes. Zudem ermöglicht der Scheduler, an weitere Informationen über die Nodes zu gelangen. Er dient auch als Schnittstelle für die Kommunikation eines einzelnen Nodes mit anderen in der gleichen Simulation. Die Ausführung von Nodes umfasst zum einen das Weiterleiten von externen Kommandos, welche über die API beim Scheduler angekommen, und zum anderen die semiautomatische Ausführung von Nodes beziehungsweise des gesamten Simulationsprozesses. Verdeutlichen soll dies ein kurzes Beispiel. Um die nächste Operation eines Nodes auszuführen, wird dieser beispielsweise nicht direkt sondern über eine separate Funktion angesprochen. Dies soll die Autarkie in der konkreten Implementation sicherzustellen, die für eine Verwendung in einer universellen Simulationsumgebung notwendig ist. Bei der semiautomatischen Ausführung wird der Scheduler entsprechend so implementiert, wie dies aus Ausführungsmodell der Simulation vorsieht. Details folgen im Kapitel 5.3 Semiautomatische Ausführung.

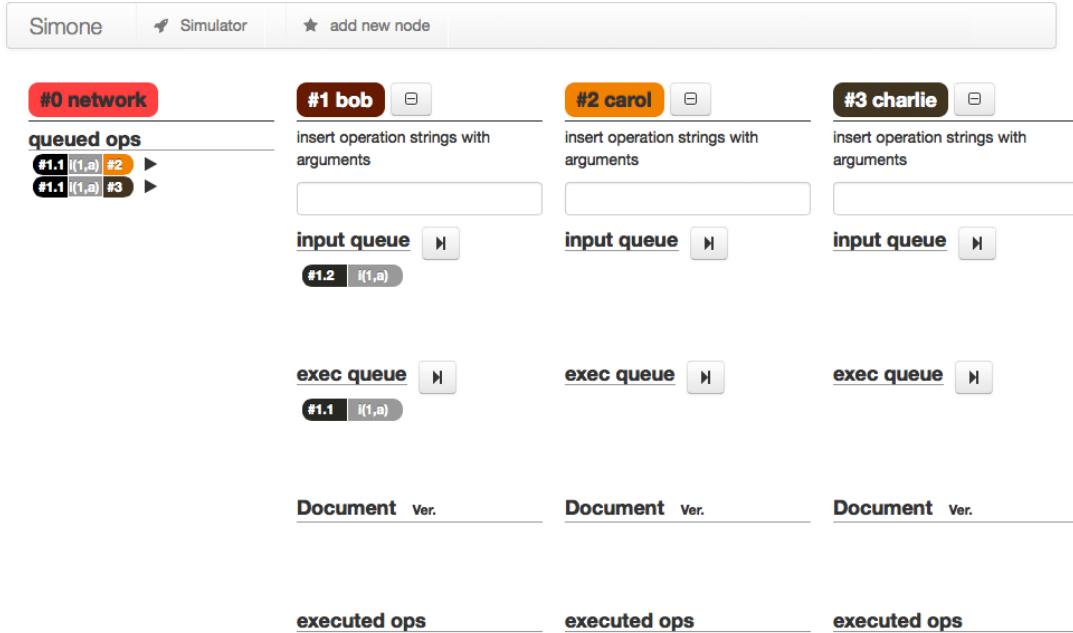
Die API ist der dritte Teilbereich des Backends. Sie ist der Anknüpfungspunkt für das Frontend und ermöglicht die Kommunikation mit diesem. Sie abstrahiert durch Kapselung komplexere Zusammenhänge in Form von Funktionsaufrufen zu einfachen und parametrisierten Aufrufen. Mittels des Tracers erfolgt auf Ebene der API das Erstellen von Traces. Ein Trace enthält alle relevanten Informationen, die für das spätere Nachvollziehen oder ein erneutes Ausführen zu einem späteren Zeitpunkt notwendig sind. Eine detaillierte Ausführung hierzu erfolgt im Abschnitt Traces.

4.4 Frontend

Das Frontend ist ein essentieller Teil der Simulationsumgebung und stellt alle Grundfunktionen bereit, um einen OT-Algorithmus gezielt ausführen und erproben zu können. In der konkreten Umsetzung ist es eine Implementation eines UIs mittels HTML, CSS und JavaScript. Das Zusammenspiel dieser drei Komponenten ermöglicht das Erstellen eines flexiblen und interaktiven UIs. Die Flexibilität bezieht sich dabei sowohl auf die Darstellungsform als auch auf die Erweiter- und Anpassbarkeit zu einem späteren Zeitpunkt. In der Darstellung werden beispielsweise unterschiedliche Bildschirmauflösungen und Formate unterstützt. So können auch große Simulationsanordnungen mit einer mehrstelligen Anzahl von Nodes je nach Bildschirmauflösung und Format flexibel dargestellt werden. Die Erweiterbarkeit ist ein wichtiger Punkt, da nicht alle im nächsten Kapitel 5 benannten Konzepte zur Veröffentlichung dieser Arbeit umgesetzt und implementiert sind. Durch die Verwendung einer einfachen und aussagekräftigen Bildsprache bei den Steuerelementen in Kombination mit der optionalen Einblendung von Hilfetexten sind die Grundlagen für eine einfache Bedienbarkeit gelegt. Sie ermöglichen auf der Basis dieser Elemente, dass das interaktive Interface auch mit neuen Funktionen versehen und erweitert werden kann. Durch die Strukturierung der Templates ist eine Erweiterung um weitere Interfaceelemente unter Beibehaltung einer einheitlichen Gestaltung jederzeit möglich.

Das Interface wird durch die Nodes, die sich in der Simulation befinden, unterteilt. Die Unterteilung der einzelnen Nodes ist typabhängig, innerhalb dieser jedoch einheitlich. Ein Node des Typs network erhält lediglich eine Queue. In dieser werden die aktuell im Netzwerk befindlichen Transport Capsules angezeigt werden. Jede Operation, die auf einem Client-Node ausgeführt wird, erzeugt dabei im Broadcast-Verfahren für jeden anderen Client-Node eine eigene Nachricht, welche sich im Netzwerk befindet. Der Nutzer des Simulators kann selbst entscheiden, welche Nachricht welchen Client-Node wann erreicht. Durch das erweiterte Konzept der Protokolltreue kann die Auswahl der Reihenfolge

auch an die Einhaltung von Regeln geknüpft werden, welche von Netzwerkprotokollen wie UDP oder TCP vorgegeben werden.



trace

```

6. gen_next_op:{'op_id': '1.1', 'node_id': 1}
5. create_op:{'node_id': 1, 'op_str': u'i(1,a)'}
4. create_op:{'node_id': 1, 'op_str': u'i(1,a)'}
3. add_node:{'node_id': 3, 'name': 'charlie'}
2. add_node:{'node_id': 2, 'name': 'carol'}
1. add_node:{'node_id': 1, 'name': 'bob'}

```

Abbildung 6 Screenshot des Frontends eines Simulators mit vier Nodes, bestehend aus drei Clients und einem Network-Node

Alle anderen abgebildeten Nodes sind vom Typ Client. Im Frontend wird jeweils die Input-Queue, die Exec-Queue sowie eine Sicht auf das aktuelle Dokumentenmodell dargestellt. Die Input-Queue enthält alle Operationen, welche auf dem Dokumentenmodell ausgeführt werden sollen. Die Eingabe erfolgt dabei implizit, in Form einer Operationsanweisung, und nicht explizit, durch das direkte ausführen dieser Mittels Tastatur auf dem Dokumentenmodell. Dieser abstraktere Weg ermöglicht die präzisere Definition auch komplexerer Simulationsausführungen. Am Ende eines jeden Nodes werden die bereits ausgeführten Operationen gelistet, welche sich in einer separaten Queue befinden. Die Queue enthält die tatsächlich auf dem Dokumentenmodell ausgeführten Operationen mit ihren gegebenenfalls transformierten Parametern. Das Betätigen des Buttons,

welcher neben der jeweiligen Queue positioniert ist, führt zu der Verarbeitung der vordersten Operation.

Am unteren Ende des Simulators ist der Trace der Ausführung hinterlegt.

Er wird bei jedem Ausführungsschritt automatisch erzeugt.

Über die Menüleiste sind wichtige Hauptfunktionen des Simulators erreichbar. In der Menüleiste hinterlegte Funktionen dienen dazu, den Simulator als Ganze zu kontrollieren und zu steuern. Hierzu zählt beispielsweise das Erstellen eines neuen oder das Klonen eines bestehenden Simulators.

5. Userinterface-fokussierte Konzepte

Nachdem im letzten Kapitel das Grundgerüst der Simulationsumgebung dargelegt und erläutert wurde setzen sich die folgenden zwei Kapitel mit einer Erweiterung dieses Grundgerüstes auseinander. Im aktuellen Kapitel erfolgt die Vorstellung und die Auseinandersetzung mit Erweiterungen, welche den Fokus auf der UI legen. Die Einordnung ist dabei eher subjektiv, als dass sie an konkreten Kriterien festgemacht werden kann. Konzepte mit ihrem Fokus auf das Frontend knüpfen auf einer übergeordneten Ebene an Funktionen von modernen IDEs und deren Debug-Interfaces an, ohne diese Referenz immer explizit aufzuzeigen.

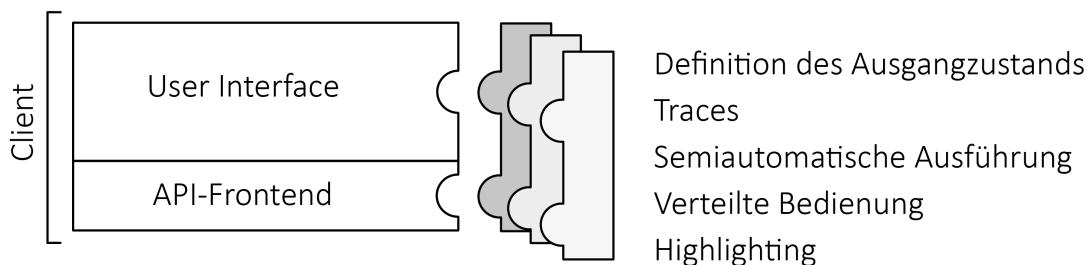


Abbildung 7 Übersicht der im Folgenden behandelten Konzepte mit dem Fokus auf dem UI

Zu einer Umsetzung der in Abbildung 7 dargestellten und im folgenden behandelten Konzepte kommt es in der Arbeit nicht. Stattdessen erfolgt eine Vorstellung eben jener, eine Diskussion, für welchen Anwendungsrahmen diese Konzepte gedacht sind, welche Umsetzungsmöglichkeiten es gibt und welche Fragestellungen mit diesen verbunden sind.

5.1 Definition eines Ausgangszustands

Aus unterschiedlichsten Gründen kann es erforderlich sein, bereits zum Zeitpunkt der Initialisierung eines Nodes sowohl die Queues als auch das Dokumentenmodell mit Daten initialisiert und für die eigentliche Simulation vorbereitet zu haben.

Es gilt, unterschiedliche Aspekte für diese Aufgabenstellung zu bedenken. Im Folgenden wird auf zwei denkbare Hauptansätze eingegangen. Dabei werden Möglichkeiten und Problemstellungen aufgezeigt.

Konzeptioneller Ansatz

Zum Initialzeitpunkt können unterschiedliche Queues bereits mit Daten vorbereitet werden. Welche dies sind, hängt hauptsächlich vom verwendeten OT-Algorithmus und der von ihm verwendeten Speicherstruktur ab. Im aktuell nur teilweise implementierten OT-Algorithmus, welcher in der aktuellen Simulationsumgebung enthalten ist, sind dies eine Vielzahl von Queues. Die Anzahl und ihre Verwendung variiert je nach OT-Algorithmus. Auf diese individuelle Struktur muss sowohl im Frontend als auch im Backend eingegangen werden, soll eine vollständige Definition des Nodes möglich sein. Zu beachten ist auch, dass die in den Queue definierten Daten immer im Kontext der gesamten aktuellen Simulationsinstanz und der Zustände des Dokumentenmodells stehen. Jede in einem Node ausgeführte Operation führt dazu, dass Nachrichten an alle anderen Nodes geschickt werden. Diese führen ihrerseits zu Operationen an den jeweiligen Nodes, welche durch ihre Ausführung das Dokumentenmodell des jeweiligen Nodes ändert. Dieser Umstand muss bedacht werden, wenn während einer laufenden Simulation neue Nodes mit vordefinierten Daten hinzugefügt werden. Die tatsächliche Schwierigkeit ist hierbei ebenso maßgeblich vom konkreten OT-Algorithmus sowie seinen Operationen abhängig und kann allgemein nicht genau bewertet werden. Bei der Erstellung von initialen Werten kann an dieser Stelle ein Editor unterstützend zum Einsatz kommen. Dieser kann den Nutzer bei der Erstellung konkreter Initialzustände helfen.

Eine Alternative zur vollständigen Definition des Initialzustands eines Nodes ist die Möglichkeit, nur einen Teil zu definieren. Besonders interessant für diesen Ansatz ist das Dokumentenmodell und die Queue mit Operationsanweisungen, welche ausgeführt werden sollen. Die Queue steht beispielhaft für jene Queues, welche in jedem OT-Algorithmus von Interesse sein können. Eine allgemeine oder schematische Charakterisie-

rung lässt sich hierbei nicht treffen, da es vom OT-Algorithmus abhängig ist, welche Queue bei der Initialisierung von Interesse ist.

Im Folgenden müssen zwei Fälle genauer erörtert werden. Zum einen der Fall der Initialisierung eines oder mehrerer Nodes, ohne dass andere Nodes bereits ausgeführt wurden. In diesem ist es möglich, einem oder mehreren Nodes einen initialen Zustand für das Dokumentenmodell zu übergeben. Dabei müssen nicht alle Nodes den gleichen initialen Zustand für das Dokumentenmodell erhalten. Ob eine Notwendigkeit für gleiche Ausgangszustände bei allen Nodes im Dokumentenmodell besteht, hängt vom konkreten OT-Algorithmus ab. Ebenso erlaubt ist die Definition von auszuführenden Operationen. Diese können sich von Node zu Node unterscheiden. Relevant sind auch hier wieder die Erfordernisse des OT-Algorithmus.

Der andere zu betrachtende Fall ist die Definition des Initialzustands eines Nodes zu einem Zeitpunkt, zu dem bereits andere Nodes im Simulator ausgeführt wurden. In diesem Fall können mehrere Optionen möglich und sinnvoll sein. Zum einen die Initialisierung des Dokumentenmodells mit einem Status, der dem eines bestimmten Nodes gleicht. So könnte der neue Node an der Stelle der Ausführung fortsetzen, an der die gesamte Simulation bereits angekommen ist. Sinnvoll ist dies dann, wenn sich keine Nachricht mehr im Netzwerk befindet und alle Operationen abgearbeitet wurden. Eine andere Option wäre die Definition eines freien und initialen Status des Dokumentenmodells. Nutzen und Sinnhaftigkeit ergibt sich hierbei auch erst im Kontext des konkreten OT-Algorithmus.

Eine für alle Ansätze sinnvolle Option ist es, die Speicherung von Presets zu ermöglichen. Preset versteht sich dabei als definierte Zusammenstellung von dem Initialzustand des Dokumentenmodells und der Queue mit den auszuführenden Operationen. Dies erlaubt die einfache Wiederverwendung für andere Simulationsläufe. In einem Klon des Simulators stehen die Presets ebenso zur Verfügung und können für die Arbeit genutzt werden.

Bei der Betrachtung der beiden benannten Hauptansätze, in Verbindung mit den aufgezeigten Möglichkeiten und den zu erwartenden Schwierigkeiten, fällt die Entscheidung auf den zuletzt genannten Ansatz. Im Vergleich ist er von seinen anfänglichen Möglichkeiten reduzierter, lässt den Raum für Erweiterungen aber offen. So besteht die Möglichkeit weitere Felder, welche in einem Preset gespeichert und nutzbar sein sollen, zu benennen und zu erweitern.

Aspekte der Umsetzung

Für die Realisierung dieses ausgewählten Ansatzes bedarf es der Anpassung des Frontends sowie des Backends der Simulationsumgebung. Im Frontend ist die Schaffung eines Workflows notwendig, der den Nutzer bei der Erzeugung eines neuen Nodes unterstützen kann, wenn er dies wünscht. Die Unterstützung erfolgt dabei in der Form, dass alternativ zum Standardverfahren für das Erstellen eines neuen Nodes ein Formular angeboten wird. In diesem existiert ein Feld für die Definition des Initialzustands des ersten Status des Dokumentenmodells. Als zweites gibt es ein Feld, in dem Operationen in der Reihenfolge eingetragen werden können, in der sie später ausgeführt werden, beginnend mit der ersten auszuführenden Operation. Diese beiden Felder können als Preset gespeichert werden, indem ihnen eine Bezeichnung gegeben wird. In einer Auswahlbox kann der Nutzer entscheiden, wie viele Nodes mit den vorgegebenen Werten erzeugt werden sollen. Als Alternative zu diesem Vorgehen existiert direkt neben dem Formular die Möglichkeit, die aktuellen Werte eines bereits im Simulator befindlichen Nodes zu übernehmen. Die Abbildung 8 zeigt in Form eines Mockups eine konzeptionelle Gestaltungsmöglichkeit.

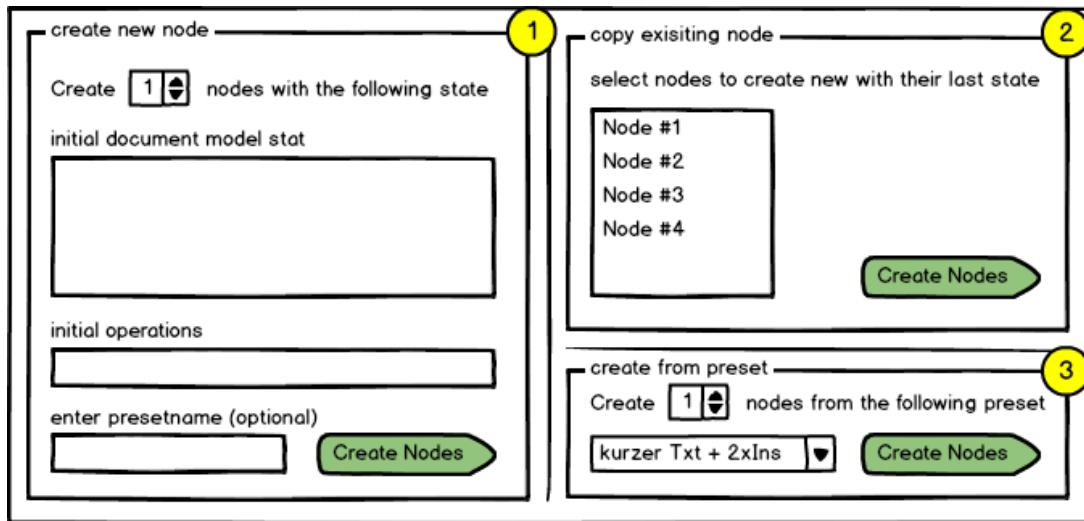


Abbildung 8 (1) dient zum Erzeugen neuer Nodes mittels manuell eingegebener Presets, (2) dient zum Erzeugen neuer Nodes auf Basis des aktuellen Zustands vorhandener Nodes, (3) zum Erzeugen von Nodes anhand eines vorhandenen Presets.

Das Backend muss dahingehend erweitert werden, dass es Daten dritter Erweiterungen speichern und verwalten kann. Im Konkreten sind dies hier die zuvor beschriebenen Presets. Darüber hinaus kann die Schnittstelle jedoch so offen angelegt werden, dass sie auch für andere Nutzungsmöglichkeiten verwendbar ist. Eine allgemeine Schnittstelle, in der in einem Key-Value-Store die notwendigen Daten im JSON-Format abgelegt werden, reicht dabei für die Zwecke der einfachen und bis zum Ende der Laufzeit der Simulationsumgebung anhaltenden Persistierung der Daten aus. Über die API ist der lesende und schreibende Zugriff durch das Hauptinterface möglich. Die Beschränkung des Zugriffs auf die Presets einer konkreten Instanz ist nicht notwendig. Dies ermöglicht die Verwendung selbiger auch über einzelne Instanzen hinaus. Als Schnittstelle kann hierbei die API, über die das Frontend bereits kommuniziert, erweitert und verwendet werden.

5.2 Traces

Nachvollziehbarkeit und Wiederholbarkeit sind zwei fundamentale Grundsätze, ohne diese eine Simulation sehr viel ihres Nutzens verliert. Die Erprobung, auf welcher das Hauptaugenmerkt bei *Simone* liegt, von Algorithmen hat eine weniger strikte Anforderung an eine Aufzeichnung. Dennoch ist es eine wünschenswerte Funktionalität, die Ausführung in Form eines Logs aufzuzeichnen. Steht ein Log einmal zur Verfügung kann

darüber nachgedacht werden, ob die dort gespeicherten Informationen für eine erneute Ausführung genutzt werden können. Hier setzt das Konzept von Traces im Folgenden an.

Konzeptioneller Ansatz

Mit der Erkenntnis, dass die Erstellung eines Logs über die Ausführung – im folgenden Trace genannt – notwendig ist, ergibt sich daraus die Aufgabe der Diskussion mehrerer Dinge im Kontext der Simulationsumgebung.

- 1) Welches **Datenformat** ist für den Trace sinnvoll, um die Grundlage für Nachvollziehbarkeit und Wiederholbarkeit mit nur einem Trace zu schaffen?
- 2) An welchen Stellen ist es, unter der Berücksichtigung der gewünschten und notwendigen **Flexibilität** des Simulators, sinnvoll, die Funktionalität zum Erstellen eines Traces zu implementieren?
- 3) Welche Möglichkeiten der **Erweiterung** sind sinnvoll, um auch der Flexibilität des Simulators gerecht zu werden.

In den folgenden Abschnitten werden diese drei Fragestellungen diskutiert.

Datenformat

Um die erste Fragestellung beantworten zu können, müssen zuerst die notwendigen Anforderungen festgehalten werden, die an einen Trace gestellt werden. Aus der allgemein formulierten Anforderung, die Ausführung des Simulators im Nachhinein nachvollziehen und erneut ausführen zu können, lassen sich konkrete Punkte ableiten.

- a) Speicherung der Information, welcher Schritt zu welchem Zeitpunkt mit welchen Parametern ausgeführt wurde
- b) Speicherung der Information, welches Ergebnis die Ausführung liefert
- c) Darstellung des Protokolls in chronologischer Reihenfolge

d) Schaffung der Möglichkeit der automatischen Verarbeitung eines Traces

Für die Speicherung der Werte muss ein Format verwendet werden, welches neben der automatischen Verarbeitung auch die Möglichkeit bietet, für den Nutzer ohne besondere Aufbereitung einfach interpretierbar zu sein.

Aus den Auswahlmöglichkeiten von CSV, XML und JSON fällt die Wahl auf letzteres Format. Es ermöglicht eine strukturierte Speicherung, ist durch seine enge Verzahnung mit Python einfach zu verarbeiten und kann durch die Nutzung des Key-Value-Verfahrens zur Speicherung von Werten und der Verwendung von aussagekräftigen Keys auch ohne eine separate Aufbereitung einfach verstanden werden. Hinzu kommt, dass es deutlich weniger Overhead beim Verarbeiten der Daten erzeugt als beispielsweise das Parsen von XML.

Das oberste Element, das jedes weitere Element aufnimmt, ist ein Array. Die Reihenfolge innerhalb des Arrays spiegelt die Chronologie der Ausführung wieder. Nur gültige JSON-Objekte sind gültige Elemente im Sinne des Traces. Jedes Objekt repräsentiert, je nach dem welchen Wert das Attribut „type“ aufweist, die Ausführung (exec), das Ergebnis dieser (result) oder sonstige Informationen (info) einer Simulation. Im Objekt vom Typ „exec“ werden im Attribut „parameter“ die Werte in einem Array gespeichert, die zusammen mit dem im Attribut „call“ gespeicherten Bezeichnern notwendig sind, um die Aktion auszuführen. „parameter“ kann dabei null sein, wenn keine Werte übergeben werden. Das Objekt vom Typ „info“ enthält das Attribut „context“ und „value“. In „context“ wird gespeichert, an welcher Stelle in der Simulation der Eintrag erstellt wurde. Dies kann für eine spätere Auswertung der Ausführung verwendet werden. Ein Kontext ist dabei immer ein Funktionsname auf Ebene der direkten Programmierung. In „value“ kann eine beliebige Zeichenkette mit gewünschten Informationen und Werten gespeichert werden. Es handelt sich um ein Freitextfeld. Jedes Objekt kann zusätzlich noch das Attribut „verbose“ enthalten. In diesem ist eine

ausführliche Beschreibung des aktuellen Objekts möglich. Beispielhaft kann ein Trace-Eintrag wie folgt aussehen:

```
[{"type": "exec", "call": "create_node", "parameter": null, "verbose": "create node"}, {"type": "result", "parameter": {"node_id": 2, "name": "bob"}}, {"type": "exec", "call": "create_node", "parameter": null, "verbose": "create node"}, {"type": "result", "parameter": {"node_id": 3, "name": "carol"}}, {"type": "exec", "call": "create_op", "parameter": "i(1,a);i(2,b)", "verbose": "add operations"}, {"type": "result", "parameter": {"ops": ["i(1,a)", "i(2,b)"], "matched_until_pos": 2, "node_id": 3}}]
```

Mit dem informal beschriebenen Aufbau des Schemas, nachdem das JSON aufgebaut ist, werden die beschriebenen Anforderungen a) bis c) erfüllt. Für die in a) geforderte Speicherung der Aufrufe und die in b) geforderte Speicherung der Ergebnisse existieren jeweils Attribute. Die chronologische Reihenfolge, welche in c) gefordert wird, ist durch die Nutzung der Array-Struktur umgesetzt. Folgende Ausführungen schaffen den inhaltlichen und argumentativen Rahmen für die Erfüllung der Anforderung d).

Die Grundlage für die automatische Verarbeitung bildet das JSON-Format. Es ermöglicht eine strukturierte Speicherung der einzelnen Schritte als Objekt und ihrer einzelnen Attribute in einem Key-Value-Format. In python steht eine elaborierte Verarbeitungsumgebung zur Verfügung, um mit Daten im JSON-Format zu verarbeiten. Die einheitliche Verwendung der Attribute innerhalb des JSON-Dokuments, verbunden mit der Festlegung einer Ausführungssemantik in Bezug auf die Attribute und ihrer Werte bieten einem Modul, dass einen Trace verarbeitet, den notwendigen Rahmen, um eine automatische Verarbeitung zu ermöglichen.

Beispielhaft wäre folgende Ausführungssemantik denkbar. Die Werte im Attribut „call“ dienen als Funktionsnamen. Der Aufruf einer Funktion mit eben diesem Namen in Verbindung mit der Übergabe der Werte als Funktionsparameter, die im Attribut „parameter“ hinterlegt sind, führt dazu, dass in einem vom ursprünglichen verschiedenen Simulator genau jene Ausführung vorgenommen wird, die ursprünglich zu eben jenen

Trace-Eintrag führte. Eine konkrete Semantik muss die Import-Komponente vorgeben.

Flexibilität

Insbesondere an zwei Stellen ist bei den Traces Flexibilität von großer Bedeutung. Zum einen muss es vom konkreten Algorithmus unabhängig möglich sein, einen Trace erstellen und diesen später für die erneute Ausführung nutzen zu können. Dies verlangt von der Simulationsumgebung sowohl die strukturelle als auch die implementatorische Bereitstellung dieser Möglichkeit. Zum anderen muss das Protokoll zum Speichern der Einträge bei der Attributvergabe flexibel genug sein, um auch nutzerspezifische Werte aufnehmen zu können. Im Folgenden werden diese Erfordernisse besprochen.

Um die Aufzeichnung eines Traces in jeglicher Situation zu ermöglichen sind verschiedene Strategien denkbar, welche auf unterschiedlichen Ebenen ansetzen. Die erste Strategie trifft dabei die Auswahl der entsprechenden Ebene, auf welcher die aufzuzeichnenden Daten abgegriffen werden, anhand des implementatorischen Ansatzes des Algorithmus. Dies bedeutet, dass die Erstellung von Traces bei selbst und in python implementierten Algorithmen so eng wie nötig mit dem Algorithmus verzahnt wird. Gleichzeitig heißt dies auch, dass ein Ansatz für das Erstellen von Traces diese externe Nutzung durch die Bereitstellung einer API ermöglichen müsste. Der zweite Teil der Strategie, der bestehende Implementationen und deren Nutzung über eine Proxy-Schnittstelle berücksichtigt, setzt deutlich oberhalb der Implementation an. Die Aufzeichnung des Traces geschieht auf der Ebene der Proxy-Schnittstelle. Der direkte Eingriff in die Implementation ist aus unterschiedlichsten Gründen nicht möglich. Diese sind auch die Ursache, weshalb eine Proxy-Schnittstelle geschaffen werden muss. Details hierzu finden sich im Abschnitt 6.6 Externe Schnittstellen.

Die Möglichkeit, einen für den speziellen Algorithmus sehr angepassten Trace durch die Wahl der geeigneten Stellen zu erzeugen, ist ein Vorteil dieses Ansatzes. Der Mehraufwand bei der Entwicklung durch eine individuelle und auf den Algorithmus angepasste Strategie der Trace-Erzeugung spricht allerdings gegen die Umsetzung dieser Strategie. Der

Fokus bei der Simulationsumgebung sollte auf deren Nutzung und nicht auf seiner Anpassung liegen. Ein weiterer kritischer Punkt bei der Umsetzung selbst gewählter Punkte, an denen das Tracing Daten aufzeichnetet, ist die Notwendigkeit Execute-Funktionen zu implementieren. Diese Funktionen sind Grundlage für das erneute Ausführen einer Simulation, dessen Trace aufgezeichnet wurde.

Ein alternativer und universeller Ansatz, der sowohl für die selbst implementierten als auch für Algorithmen verwendbar ist, die nur über die Proxy-Umgebung genutzt werden können, ist die Umsetzung des Tracings auf Ebene der API. Diese Ebene bildet einen guten Kompromiss zwischen universeller Nutzbarkeit und Granularität der nutzbaren Information. Sie verknüpft Funktionen des Backends mit dem im Frontend befindlichen Eingabe- und Steuerelementen. Diese sind für beide Gruppen von Simulationsobjekten, sowohl selbst implementierte als auch durch Dritte umgesetzte Algorithmen, einheitlich. Damit bieten sie eine Grundmenge an Funktionalität, welche bei ihrer Nutzung aufgezeichnet und zu einem späteren Zeitpunkt für die Wiederausführung genutzt werden kann. Eine darüber hinausgehende Aufzeichnung von Informationen zum Zwecke der besseren Nachvollziehbarkeit schließt dieser Ansatz nicht aus. Vielmehr legt er ein Grundschema vor, das in jedem Fall und ohne weitere Implementierung unabhängig vom verwendeten Algorithmus funktioniert.

Erweiterung

Der folgende Teilabschnitt beschäftigt sich damit, welche Möglichkeiten geschaffen werden können oder müssen, um eine detailliertere Aufzeichnung der Abläufe in der Simulation zu erweitern. Gerade bei komplexeren OT-Algorithmen kann der Wunsch entstehen, bestimmte Teile der Ausführung genauer betrachten und festhalten zu können.

Eine mögliche Annäherung bietet das Frontend. In diesem sind an all jenen Stellen der Simulation Interaktions- und Steuermöglichkeiten für den Nutzer eingebaut, an denen ein Eingriff für einen einfachen OT-Algorithmus sinnvoll erscheint. Von besonderem Vorteil ist diese Ebene und sind diese Stellen auch deshalb, weil sie einen künstlichen Unterbrechungspunkt in der Ausführung der Abarbeitung des OT-Algorithmus

schaffen. Diese Punkte bieten für eine erneute Ausführung mittels der aufgezeichneten Daten einen idealen Ansatzpunkt für die Exec-Funktionen.

Über den OT-Algorithmus von Ellis und Gibbs [1], welcher als Vorlage für die Simulationsumgebung verwendet wurde, hinausblickend, ist beispielsweise am OT-Algorithmus von Sund und Ellis [4] bereits erkennbar, dass komplexere OT-Algorithmen auch zu einer Erweiterung um neue Funktionalitäten führen können. Wenn also über das Frontend weitere Steuerungsmöglichkeiten implementiert würden, so besteht die Möglichkeit im gleichen Zuge die API zu erweitern. An eben dieser Stelle böte sich dann die Gelegenheit, für die Erzeugung eines Trace-Objekts anzusetzen. Durch die Verankerung auf Ebene der API ist die Hürde zur Umsetzung von Exec-Funktionen geringer als in anderen Teilen des Codes.

Nachteilig an diesem Ansatz ist die Komplexität in der Umsetzung. Nicht nur im python-Teil der API sondern auch im HTML- und JavaScript-Teil des Frontends müssten Änderungen vorgenommen werden. Unterstützen kann hier die ausführliche Dokumentation, mit umfangreichen Beispielen unterfüttert, sowie die stringente Kapselung des Codes.

Ist die Widerausführbarkeit keine priorisierte Anforderung für den gewählten OT-Algorithmus, sondern steht die Nachvollziehbarkeit im Kernfokus, so ist die Verankerung der Tracing-Funktionen direkt im OT-Algorithmus eine denkbare Alternative. In diesem Fall kann auf für diesen Anwendungsfall geschaffene Funktionalität des Tracing-Modules zurückgegriffen werden. Da der OT-Algorithmus selbst in python implementiert werden muss, um ihn direkt mit der Simulationsumgebung zu verwenden, ist der Komplexitätsgrad dieser Erweiterung weniger gering als im zuvor beschriebenen Ansatz. Geschlossene Ansätze, die über eine Proxy-Schnittstelle angesprochen werden müssen, bleiben hier Prinzip bedingt außen vor.

Für beide Ansätze ist es notwendig, eine allgemeine Schnittstelle vorzubereiten und diese nach außen für den Entwickler bereitzustellen. Zudem bleibt anzumerken, dass sie sich nicht ausschließen sondern ergänzen und gleichzeitig verwendet werden können.

Neben den Stärken des jeweiligen Ansatzes gilt es auch die Herausforderungen zu betrachten. Dem ersten Ansatz wohnt, neben der Komplexität seiner Umsetzung im Konkreten, die Schwierigkeit inne, eine allgemeingültige und einfach zu nutzende Lösung zu schaffen, die eine Verwendung durch Dritte an beliebiger Stelle erlaubt. Die Implementierung von Steuerelementen im Frontend, der Verbindung dieser mit der API sowie das Aufzeichnen der notwendigen Daten und deren Verwendung für eine erneute Ausführung ist bereits für die Umsetzung einer einzigen Funktionalität hinreichend aufwendig.

Die Umsetzung des zweiten Ansatzes trifft dadurch, dass der Algorithmus in python-Code vorliegt, eine entscheidende Annahme. Diese impliziert eine Steigerung der Komplexität bei der Umsetzung. Sowohl die Tracing- als auch die Exec-Funktionalität müssen separat durch den Nutzer der Simulationsumgebung umgesetzt werden. Anders als im ersten Ansatz wird das Tracing direkt im Backend verankert. Das Aufzeichnen von bestimmten Zuständen wird dadurch vereinfacht und auf einen Bereich begrenzt.

5.3 Semiautomatische Ausführung

Gewisse Problemfälle sind schwierig zu konstruieren. Sie treten erst bei größeren Simulationsanordnungen auf und benötigen viele Schritte, um einen Zustand zu erreichen. Es wäre hilfreich, wenn der Nutzer diese Schritte nicht alle manuell ausführen müsste, sondern die Simulationsumgebung dies zu einem Teil und bis zu einem beschriebenen Punkt selbsttätig gehen könnte. Ein Konzept hierfür ist die semiautomatische Ausführung.

Konzeptioneller Ansatz

Der Modus der semiautomatischen Ausführung hat die Aufgabe, dem Nutzer bei der Ausführung eines konkreten Simulators zu unterstützen. Er soll die Simulation solange ausführen, bis er vom Nutzer oder einem anderen Ereignis unterbrochen wird. Die Ausführungsgeschwindigkeit unterliegt dabei einem Takt, der vom Nutzer eingestellt wird. Das Tempo, dass der Takt vorgibt, richtet sich dabei danach, wie schnell und gut der Nutzer die passierenden Veränderungen wahrnehmen kann oder möchte.

Von gar keinem Verzug bis zu mehreren Sekunden kann alles praktisch zur Anwendung kommen und sinnvoll sein.

Für die Ausführung der semiautomatischen Simulation sind unterschiedliche Modelle denkbar. Dabei geht es konkret darum, in welcher Reihenfolge die auszuführenden Nodes der Simulation ausgewählt werden. Beispielsweise kann das Bevorzugen eines bestimmten Nodes gewünscht sein. Oder aber das Einhalten eines bestimmten Verhältnisses bei der Ausführung eines bestimmten Nodes. Durch den hohen Grad an Kontextualität ist es hier jedoch wenig zielführend, auf unklaren Annahmen eine Konzeption vorzunehmen. Eine modulare Gestaltung ermöglicht es dem Nutzer einen für seine Versuchszwecke passenden Auswahlalgorithmus selbst zu formulieren, sollte dies notwendig sein. Um dies zu ermöglichen, ist die Vorgabe eines gewissen Entwicklungsrahmens notwendig. Dieser gibt dem Nutzer den Rahmen vor, in dem er ein bestimmtes Modul implementieren kann. Er ermöglicht es dann, beliebige Auswahlalgorithmen, die über eine einfache randomisierte Auswahl hinausgehen, umzusetzen.

Die abstrakte Klasse, welche den Entwicklungsrahmen bildet, gibt dabei Standardschnittstellen vor, über welche die semiautomatische Ausführung mit dem Modul kommunizieren und die Auswahl des nächsten Nodes treffen kann. Zum anderen implementiert sie Funktionen, mit denen die Simulationsumgebung Informationen über das Modul auslesen kann, um diese, unter anderem, für die Darstellung im Frontend zu nutzen.

Als universelle Möglichkeit, also vom konkreten OT-Algorithmus losgelöst, einen Node aus der Menge von verfügbaren Nodes automatisch auszuwählen, ist die zufällige Auswahl jene, welche in der Simulationsumgebung standardmäßig vorgegeben ist. Sie stellt sicher, dass in einer festen aber zufälligen Reihenfolge alle verfügbaren Nodes solange ausgeführt werden, bis keine Operationen mehr zur Ausführung zur Verfügung stehen.

Die Möglichkeit zur Wiederholung einer Ausführung ist in der Simulationsumgebung sehr wichtig. Sie ermöglicht dadurch die Überprüfung, ob sich ein gewisses Verhalten reproduzieren lässt. Mittels eines Seeds, der

an den Zufallszahlenalgorithmus übergeben wird, kann sichergestellt werden, dass die erzeugten Zahlen immer wieder in der gleichen Reihenfolge ausgegeben werden. Dies ist für das spätere und erneute Ausführen von Bedeutung. Wird die Ausführung durch ein Ereignis unterbrochen, so kann diese durch den Eingriff des Nutzers wieder aufgenommen werden. Neben Break Points können dies auch systemeigene Ereignisse sein. Beispielsweise die schlichte Tatsache, dass alle Operationen in allen Nodes abgearbeitet wurden.

Aspekte der Umsetzung

Für die Umsetzung sind zwei Aspekte zu betrachten: Zum einen die Integration in das Frontend und zum anderen muss das Backend und die dortige Integration beleuchtet werden.

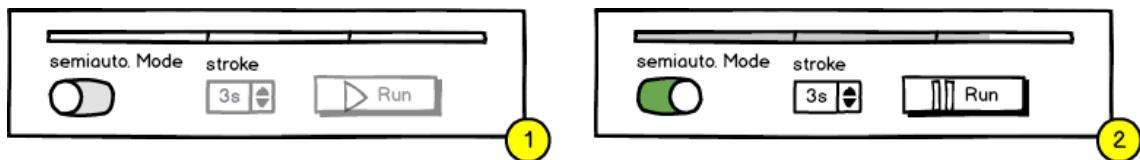


Abbildung 9 Mockup der Bedienelemente. 1 zeigt dabei den Zustand, in dem die semiautomatische Ausführung deaktiviert ist, 2 den Zustand, in dem die semiautomatische Ausführung mit einem Takt von 3 Sekunden Dauer ausgeführt wird

Für die Steuerung der semiautomatischen Ausführung braucht es im Frontend entsprechende Steuerelemente. Zum einen, um zwischen der manuellen und der semiautomatischen Ausführung zu wechseln. Zum anderen um die semiautomatische Ausführung selbst zu starten und anzuhalten. Ebenso braucht es eine Möglichkeit für den Nutzer, den Ausführungstakt zu beeinflussen und festzulegen. Zudem sollte visuell kenntlich gemacht werden, wann der nächste Ausführungstakt erfolgt. Dies gibt dem Nutzer bei der Beobachtung der Ausführung ein Gefühl dafür, wann die nächste Aktion zu erwarten ist. All diese Steuerelemente sind nur im Hauptinterface sichtbar. Werden weitere Instanzen des Simulators für die verteilte Bedienung gestartet, so können diese während der semiautomatischen Ausführung lediglich für die Beobachtung verwendet werden. Alle direkten Steuerelemente sind deaktiviert oder nicht eingeblendet. Dies verhindert die ungewollte Einflussnahme in die Ausführung, welche während der semiautomatischen Ausführung nicht gewollt ist. Unbeeinflusst davon ist die Möglichkeit, über die Timeline auf

vergangene Ereignisse zu navigieren und sich Veränderungen anzuschauen.

Das Backend benötigt auf der Ebene der API eine Anpassung. An dieser Stelle muss die Steuerung der Ausführung über eben jene ermöglicht werden. Zudem braucht es die Umsetzung eines Mechanismus, der die durch den Nutzer implementierten Auswahlalgorithmen einliest und im Frontend zur Auswahl anbietet. Darüber hinausgehend werden alle weiteren Schnittstellen in der abstrakten Klasse für die semiautomatische Ausführung definiert und mit dem Scheduler der Simulationsumgebung direkt verbunden.

5.4 Verteilte Bedienung

Die Simulationsumgebung sieht vor, dass die primäre Bedienung durch einen Nutzer erfolgt. Dieser steuert die Simulationsumgebung während ihrer Ausführung. In gewissen Szenarien kann es gewünscht sein, dass die Steuerung von Teilen der Simulationsumgebung durch mehrere Nutzer gleichzeitig erfolgen soll. Um dies zu ermöglichen bedarf es einer Erweiterung der Simulationsumgebung.

Konzeptioneller Ansatz

Eine verteilte Bedienung zu ermöglichen hat Primärziele: losgelöste Darstellung des einzelnen Nodes vom Hauptinterface, verteilte Bedienung einzelner Nodes und das Anzeigen aller relevanten Informationen bezüglich der Simulationsinstanz außerhalb des Hauptinterfaces. Die Möglichkeit der losgelösten Darstellung einzelner oder mehrerer Nodes vom Hauptinterface ist die Grundlage für die Umsetzung des Eingangs erwähnten Wunsches der Bedienung durch mehrere Nutzer. Neben der Darstellung der Nodes selbst ist es notwendig, den Nutzer auch über das Kontextgeschehen zu informieren. Um dies zu ermöglichen, wird einerseits der Netzwerk- beziehungsweise Server-Node in jeder externen Ansicht integriert und eingeblendet. Ebenso wird der Trace der gesamten Simulationsumgebung angezeigt. Dies dient dazu, dem Nutzer einen tatsächlichen Einblick in die Ausführung zu gewähren und die eigenen Aktionen in den Kontext der gesamten Ausführung einordnen zu können. Damit wird zudem das dritte Ziel erfüllt.

Durch die Möglichkeit der Bedienung mehrerer Nutzer gleichzeitig entstehen unterschiedliche Problemstellungen, welche bei der Bedienung durch einen Nutzer nicht auftreten können. Sollten diese die Ausführung dahingehend beeinflussen können, dass es zu einer Verfälschung des Verhaltens kommt, muss das Konzept auf eben jene Problemstellungen eingehen. Im Folgenden geschieht diese Auseinandersetzung anhand zweier Punkte.

Zum einen besteht die Möglichkeit, dass unterschiedliche Nutzer ein und den selben Node in unterschiedlichen externen Ansicht zur gleichen Zeit darstellen und bedienen. Dabei muss verhindert werden, dass sie sich gegenseitig bei der Bedienung behindern oder gar die Ausführung störend beeinflussen. Eine einfache und für den Kontext ausreichende Lösung ist die Umsetzung des wechselseitigen Ausschlusses, sollten zwei Nutzer gleichzeitig denselben Node ausgewählt haben. Er verhindert das gegenseitige Stören bei der Bedienung und lässt es dennoch zu, dass unterschiedliche Nutzer auf den gleichen Node steuernd zugreifen können. Ein allein lesender Zugriff ist jederzeit durch alle Nutzer möglich. Zum anderen ist es die Hierarchie der Rechte. Der Nutzer, der über das Hauptinterface agiert, verfügt über mehr Steuermöglichkeiten als der Nutzer, der in einer externen Ansicht nur einzelnen Nodes darstellen lässt. Diese Hierarchisierung hat zur Folge, dass Nutzer der externen Ansichten über Aktionen, die im Hauptinterface ausgeführt werden und ihre eigenen Instanz beeinflussen, mindestens informiert werden müssen. Hierzu zählen das Löschen von Nodes, die Nutzung der Snapshot-Funktion sowie das Beenden einer Simulation. Insbesondere beim Löschen eines Nodes und beim Wechsel in einen anderen Snapshot sollten andere Nutzer darüber informiert werden, um Irritationen der betroffenen Nutzer zu vermeiden.

Nach der konzeptionellen Beschreibung des Ansatzes wird nachfolgend auf Aspekte der Umsetzung eingegangen.

Aspekte der Umsetzung

Der Aufruf der externen Ansicht kann über ein Menüelement in der Node-Darstellung des Hauptinterfaces erfolgen. Ein Klick auf dieses Element erzeugt ein neues Fenster, in dem der neue Node dargestellt

wird. Ein direkter Aufruf dieser Ansicht ist ebenso möglich und kann über die Erweiterung der URL für die Hauptansicht erfolgen. Dabei werden der URL weitere Parameter angefügt. Über den Parameter `ids` können Kommata-getrennt Node-IDs übergeben werden, besteht der Wunsch mehr als einen Node in einem eigenen Fenster anzuzeigen. Pflichtangabe ist eine ID.

Alle Interface-Funktionen, über die ein Node im Hauptinterface verfügt, sind auch in der separaten Darstellung verfügbar. Ausgenommen davon ist die Funktion zum Löschen und zum Erzeugen von Nodes, sowie das Springen zu einem Snapshot.

Neben den Steuerelementen werden auch Kontextinformationen und Kontextelemente dargestellt. Zu den Kontextinformationen gehören mindestens die Daten, um welche Session es sich handelt und, wie viele Nodes in der Session existieren. Kontextelemente sind die Tracedarstellung und der Netzwerk-Node, so dieser auch im Hauptinterface vorhanden ist. Er enthält ebenso Steuerelemente wie im Hauptinterface. Sie erlauben jedoch nur die Steuerung der eigenen Operationen.

5.5 Highlighting

Die Ursache für Änderungen des Zustands eines einzelnen Nodes, verbunden mit der Anzeige dieser Änderung im Frontend der Simulationsumgebung, kann unterschiedliche Ursachen haben. Bei der semiautomatischen Ausführung geschehen diese nach dem Start ohne direkten Eingriff des Nutzers. Auch bei der verteilten Ausführung der Simulationsumgebung ist eine Änderung nicht notwendiger Weise mit einer Interaktion durch den Hauptnutzer des Simulationsumgebung verbunden. Selbst in Fällen, in denen der Nutzer die Simulationsumgebung direkt bedient, kann es schwierig sein, auf einen Blick zu erkennen, zu welchen Veränderungen das Ausführen einer bestimmten Aktion geführt hat. Insbesondere wenn mehrere Nodes an der Simulation beteiligt sind, kann die Übersicht darüber, welche Aktion welche Veränderung an welcher Stelle bewirkt hat, verloren gehen. Der nachfolgende Ansatz soll aufzeigen, welche Besonderheiten bei Verbesserungen

der Wahrnehmbarkeit von Veränderungen beachtet werden müssen. Im Anschluss wird ein Umsetzungsvorschlag vorgestellt.

Konzeptionelle Diskussion

Bei der Überlegung, wie ein Lösungsvorschlag aussehen kann, kommen unterschiedliche Gedanken und Ansätze auf. Diese sollen mit einer Diskussion im Folgenden gewürdigt werden.

Neben den mit den im Hauptinterface gekoppelten Aktionen gibt es eben auch solche, welche losgelöst von diesem ausgeführt werden. Diese gilt es im Folgenden gesondert zu betrachten. Die semiautomatische und manuelle Ausführung der Simulationsumgebung über das Hauptinterface werden höchstens durch einen Nutzer gesteuert. Diese Steuerung geschieht immer direkt und kann im Kontext zur Ausführung erfolgen. Das bedeutet, dass der Nutzer während der manuellen Ausführung die Möglichkeit hat, die nächste Aktion erst dann auszuführen, wenn er dies für richtig hält. Im Kontext der Hervorhebung von Veränderungen bedeutet dies, dass der Nutzer zwischen zwei Aktionen eine beliebig lange Zeit vergehen lassen kann. Diese kann für die Erfassung der Veränderungen, sowohl optisch als auch logisch, verwendet werden. Auch im semiautomatischen Modus kann der Nutzer entweder in eine Situation eingreifen und diese anhalten. Oder aber er hat die Möglichkeit, das Zeitintervall zwischen zwei Schritten derart zu vergrößern, dass dieses genügend Zeit für die Erfassung der Veränderungen bietet.

Anders verhält es sich bei der verteilten Ausführung. Unter der Annahme, dass zwei oder mehrere Nutzer nicht gegenseitig blockiert werden sollen, ist es nicht einfach möglich, die Ausführung der gesamten Simulation für einen Nutzer anzuhalten. Der zweite, genauso wie auch alle weiteren Nutzer, arbeitet, wenn auch nur mit einem Teil aller Nodes, unabhängig vom ersten Nutzer und damit auch losgelöst von dessen Interface. Um ihnen dennoch die Möglichkeit zu geben, die Veränderung in selbstgewählter Geschwindigkeit nachvollziehen zu können, bedarf es der Entkopplung von der Rückgabe des Ergebnisses eines Schritts und deren Darstellung. Eine solche Entkopplung ermöglicht es dem einzelnen Nutzer, unabhängig von der tatsächlichen Ausführungsgeschwindigkeit, die Darstellung an das selbst gewählte Tempo anzupassen.

Das im ersten Teil beschriebene Verhalten ist ein Sonderfall des im zweiten Teil beschriebenen. Die Ausnahme zeichnet sich dadurch aus, dass es nur eine Geschwindigkeit in der Darstellung gibt, welche durch den Nutzer des einzigen Interfaces bestimmt wird. Durch diesen günstigen Fall bedarf es keines separaten Ansatzes.

Neben der Geschwindigkeit ist die Art und Weise der Darstellung der Veränderungen entscheidend für eine gute Wahrnehmbarkeit selbiger. Die fokussierte Beobachtung schränkt das Sichtfeld des Beobachtenden nachhaltig ein. Liegt der Betrachtungsfokus auf einem bestimmten Teil des Interfaces, kann ein anderer deutlich schlechter wahrgenommen werden. Hinzu kann im semiautomatischen Modus durch die randomisierte Auswahl passieren, dass der Ort der Veränderung durch den Nutzer nicht jederzeit vorhersehbar ist. Auch Veränderungen innerhalb einer größeren Menge an Zeichen exakt wahrzunehmen und diese auch korrekt zuzuordnen, kann hinreichend schwierig sein, wenn diese nicht gesondert hervorgehoben und sichtbar gemacht werden. Diese Schwierigkeit wurde bereits in aktuellen kollaborativen Anwendungen wie GoogleDocs [12] erkannt und durch farbige Hervorhebungen der aktuellen Bearbeitungsstellen gelöst. Andere Anwendungen wie Etherpad Lite [13] stellen nur den bereits editierten Text heraus, was bei der Wahrnehmung des aktuellen Orts der Bearbeitung zu Schwierigkeiten führt. Eine denkbare Lösung wäre das Hervorheben von Veränderungen. Die Verwendung unterschiedlicher farbiger und semantisch konnotierter Markierung in Verbindung mit Animationen könnte eine solche Möglichkeit sein. Die semantische Annotation von Farben an gewisse Operationen oder Aktionen kann hierbei helfen.

Für die Darstellung aller Ereignisse in der Reihenfolge ihres Auftretens könnten zwei Ansätze alternativ zueinander verwendet werden. Zum einen die Darstellung der Einträge in einer Liste. Diese, eher technische Darstellung, bietet die Möglichkeit zusätzliche Informationen in Textform innerhalb der Liste anzuzeigen. Die zweite Möglichkeit ist die Verwendung der Timeline-Metapher, ähnlich wie bei einem Software-Videoplayer. Zum einen trägt sie die Assoziation der Wiedergabe anstelle der direkten Ausführung von Ereignissen, was der gedachten Umsetzung

entspricht. Zum anderen wird durch den Slider dem Nutzer die direkte Option gegeben, sich auf einer Zeitliste zwischen Ereignissen und ihren Veränderungen hin und her zu bewegen.

Bei der Verwendung der Timeline werden die beiden zuvor genannten Vorschläge teilweise aufgegriffen. Auf der Timeline werden alle vergangenen Veränderungen abgetragen. Dem Nutzer wird erlaubt einen Slider auf dieser Timeline auf die Position zu bewegen, an welcher er die aktuelle Veränderung nachvollziehen möchte. Wird die Wiedergabe der Veränderung angehalten und findet im Hintergrund eine fortschreitende Veränderungen statt, so bildet der Teil rechts vom aktuell betrachtete und hervorgehobenen Ereignis die noch nicht betrachteten Veränderungen ab, die der Nutzer noch nicht gesehen hat. Durch die semantische Zuordnung von Farben zu Operationen und Aktionen, im Folgenden als Ereignisse benannt, wird diese auch für die Hervorhebung von Einträgen in der Timeline verwendet werden. Eine detaillierte Auseinandersetzung mit dem Slider erfolgt im nächsten Abschnitt.

Aus technischer Sicht auf das Backend gilt es die Frage zu beantworten, wie die Ereignisse außerhalb des einzelnen Nodes gespeichert werden, so dass sie im Frontend für die beschriebenen Zwecke verwendet werden können. Eine globale Speicherung, auf die alle Frontend-Instanzen eines Simulators zugreifen können, wäre eine sehr naheliegende Möglichkeit. Alle Instanzen können über die API des Backends auf diese zentrale Queue zugreifen und die für sich relevanten Ereignisse herausfiltern. Die Filterung erfolgt dabei über die IDs der Nodes, welche in der jeweiligen Instanz angezeigt werden. Neue Ereignisse werden bei ihrer Aufnahme direkt an das Frontend übertragen.

Aspekte der Umsetzung

Die drei benannten Teilespekte, die einer verbesserten Darstellung der Ereignisse und Veränderungen innerhalb des Simulators dienen können, bedürfen der konkreten Umsetzung, um den umrissenen Rahmen und die beschriebenen Attribute im Detail und im Zusammenspiel sehen zu können.



Abbildung 10 konzeptionelle Darstellung der Interfacekomponenten, welche die Wiedergabe der Ereignisse auf der Timeline ermöglichen. (1) Timeline mit schematisch dargestellten Ereignissen, einem Slider sowie einem Navigationselement zum Navigieren auf der Ansicht. (2) Steuerelement für die Wiedergabe und die Anzeigedauer.

Da sich alle drei beschriebenen Konzepte in der Timeline und den Steuerelementen kristallisieren, wird im Folgenden auf explizit diese zuerst eingegangen. Der Slider ist das Auswahllement einer konkreten Aktion auf der Timeline. Gleichzeitig dient er als Anzeige, welcher Punkt auf der Timeline momentan ausgewählt ist und kann auf der Timeline durch den Nutzer verschoben werden. Die Auswahl erfolgt durch das Absetzen des Sliders auf dem operations- oder aktionsrepräsentierenden Element. Durch die Auswahl wird die im Frontend ausgeführte Operation oder Aktion entstehende Veränderung hervorgehoben. Zusätzlich zum Slider gibt es auch noch Steuerelemente, die es ermöglichen sich Ereignis für Ereignis vorwärts und rückwärts auf der Timeline zu bewegen. Weiterhin gibt es einen Wiedergabe- und Pause-Button sowie die Möglichkeit auszuwählen, wie lange auf einem Ereignis höchstens verweilt werden soll, bevor im Wiedergabemodus auf das nächste Ereignis gesprungen wird. Ist der Slider am aktuellen Ereignis angelegt, so ist das nachfolgende Verhalten davon abhängig, in welchem Modus der Nutzer sich befindet. Im semiautomatischen Modus wird die nächste Aktion ausgeführt, die durch den Scheduler vorgegeben wird. Im manuellen Modus erfolgt eine Steuerung und Auslösung neuer Aktionen ausschließlich über die direkten Steuerelemente neben den Nodes. Der Nutzer muss einen Node beziehungsweise eine Queue auswählen und dort die nächste Aktion ausführen. Der Slider bewegt sich dann immer an der Spitze mit.

Jede Operation oder Aktion, die auf der Timeline dargestellt wird, ist auch mit einer Veränderung im Frontend verbunden. Damit diese für den Nutzer gut ersichtlich ist wird sie farblich und mit einer dezenten Animation hinterlegt auf dem Frontend angezeigt. Diese Hervorhebung hält mindestens bis zum Wechsel zur nächsten Operation oder Aktion an und wird dann ausgeblendet.

6. Backend-fokussierte Konzepte

Im vorherigen Kapitel wurden Konzepte dargelegt und erläutert, die ihren Fokus auf das UI legen. Das nun folgende handelt von Konzepten, die ihren Fokus eher auf das Backend legen. Die Einordnung ist dabei auch wie bereits im vorherigen Kapitel eher subjektiv, als dass sie an konkreten Kriterien festgemacht werden kann. Konzepte mit ihrem Fokus auf das Backend wirken sich eher weniger auf das UI aus. Vielmehr konzentrieren sie sich auf die gesamten Möglichkeiten, welche die Simulationsumgebung bietet.

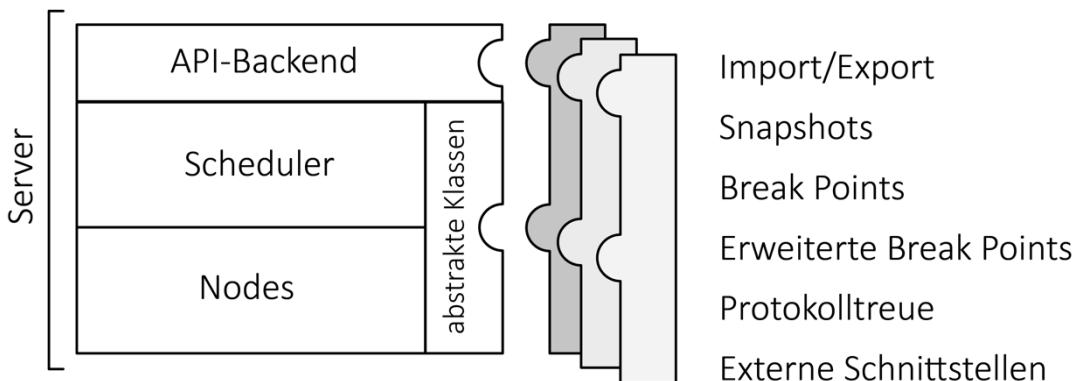


Abbildung 11 Übersicht der im Folgenden behandelten Konzepte mit dem Fokus auf dem Backend

Zu einer Umsetzung der in Abbildung 11 dargestellten und im Folgenden behandelten Konzepte kommt es in der Arbeit nicht. Stattdessen erfolgt eine Vorstellung eben jener, eine Diskussion, für welchen Anwendungsrahmen diese Konzepte gedacht sind, welche Umsetzungsmöglichkeiten es gibt und welche Fragestellungen mit diesen verbunden sind.

6.1 Import und Export von Traces

Traces werden auch über die Laufzeit der Simulationsumgebung hinaus benötigt. Im Besonderen nach der Ausführung können sie dabei helfen nachzuvollziehen, welche konkreten Schritte in der Ausführung zu welchen Ergebnissen führen. Um einen Trace auch über Laufzeit der Simulationsumgebung hinweg aufzubewahren zu können, ist es notwendig diesen zu persistieren. Dies kann von Seiten der Simulationsumgebung

erfolgen. So wären nach der Beendigung und der erneuten Ausführung selbiger die Traces der vorherigen Ausführungen noch vorhanden und könnten im gewohnten Umfeld betrachtet werden.

Export

Eine zweite und mindestens ergänzende Möglichkeit ist der Export eines Traces in eine eigenständige Datei und möglicherweise in unterschiedliche, dem Verwendungszweck angepasste, Formate. Dies würde zweierlei ermöglichen. Zum einen die Weiterverarbeitung der Traces in einer anderen Anwendung. Denkbar wäre eine Aufbereitung der Traces, um bestimmte Muster zu erkennen und Problemfälle zu entdecken – insbesondere bei semiautomatisch ausgeführten Simulationen.

Zum anderen ermöglicht der Export einen Austausch mit Dritten, außerhalb der Simulationsumgebung. Insbesondere bei der Veröffentlichung von Ergebnissen ist es wenig praktikabel und nicht sinnvoll direkt auf den Simulator und den dort gespeicherten Trace zu verweisen. Der Verweis auf den Trace als dediziertes Dokument ist hingegen deutlich zielführender.

Import

Ist ein Export der Traces umgesetzt, so ist die Möglichkeit zum Import des selbigen eine logische Schlussfolgerung. Insbesondere deshalb, weil durch das Trace-Objekt vom Typ „exec“ die Möglichkeit existiert, einmal ausgeführte Simulationen erneut und auf dem exakt gleichen Wege auszuführen wie zuvor. Detailliert wurde hierauf im vorherigen Unterkapitel eingegangen. Der Import kann im Prinzip in zwei Phasen unterteilt werden. Die erste Phase ist das Einlesen der Datei, die zweite Phase das Verarbeiten der Daten. Beim Einlesen der Datei werden die Daten überprüft, ohne dass es bereits zu einer Ausführung dieser durch einen Simulator kommt. Bei der ersten und indirekten Überprüfung, die systembedingt automatisch passiert, erfolgt die Wandlung der Textdatei, welche die Daten im JSON-Format enthält, in ein python-Objekt. Dabei wird der grammatischen Test durch die python-eigene Methode, welche für die Transformation verantwortlich ist, durchgeführt. Im Fall eines Fehlers im Datenformat geschieht an dieser Stelle bereits eine

Unterbrechung und die Hinweisgabe an den Nutzer. Eine zweite Überprüfung erfolgt auf der semantischen Ebene. Hierbei wird geprüft, ob für jedes Trace-Objekt vom Typ „exec“ im Attribut „call“, verbunden mit der Anzahl der Werte im Attribut „parameter“, eine entsprechende Funktion in der Simulationsumgebung hinterlegt ist, welche das erneute Ausführen überhaupt erst ermöglicht. Auch hierzu finden sich Details im vorherigen Unterkapitel. Im Fehlerfall wird ein Hinweis an den Nutzer gegeben und die Abarbeitung gestoppt. Die vorherige Überprüfung vermindert die Wahrscheinlichkeit für das Auftreten von Problemen, welche zu einem Abbruch führen würden, während der Ausführung.

Verarbeitung

Die zweite Phase des Imports stellt die Verarbeitung der Daten dar. Ein Trace besteht aus Objekten, welche unterschiedlichen Typs sein können. Aktiv verarbeitet werden beim Import Objekte vom Typ „exec“ und „result“, letztere jedoch optional. Eine Verarbeitung kann dabei auf drei Wegen erfolgen: automatisch, manuell oder semiautomatisch. Bei der automatischen Verarbeitung erfolgt die Ausführung der Funktionen, die mittels der Werten im „call“-Attribut und „parameter“-Attribut aufgerufen werden können. Die Abarbeitung wird bis zum letzten Eintrag ausgeführt. Das Ergebnis ist in einem neuen Trace und im ausgeführten Simulator ersichtlich. Die manuelle Ausführung erlaubt eine Eintrag für Eintrag basierte Abarbeitung des importierten Traces. Dabei wird immer ein ganzer, durch ein Trace-Objekt repräsentierter, Schritt abgearbeitet. Eine semiautomatische Abarbeitung wird dadurch ermöglicht, dass ein beliebiges und noch nicht verarbeitetes Trace-Objekt ausgewählt werden kann. Alle bis dort noch nicht verarbeiteten Trace-Objekte werden jetzt ausgeführt, bis das ausgewählte Trace-Objekt selbst ausgeführt wurde. Zusätzlich besteht jederzeit die Möglichkeit den Trace zu verlassen und einen vom ursprünglichen nächsten Trace-Objekt verschiedenen Schritt in der Ausführung über das Frontend zu wählen. Eine Ausführung des importierten Traces ist danach nicht mehr möglich. Als Ursache wirkt hier, dass im Allgemeinen mit der Ausführung eines beliebigen anderen Schritts der ursprüngliche Ausführungspfad verlassen wird.

6.2 Snapshots des Simulators

Bei einer Erprobung treten immer wieder Situationen auf, in denen die Ausführung als nächstes mehrere unterschiedliche Schritte erlaubt. Dabei sind die Situationen, die unabhängig von der Entscheidung ohnehin zum gleichen Endergebnis führen, weniger interessant. Für eine spätere und erneute Ausführung sind jene von Bedeutung, in denen unterschiedliche Entscheidungen zu unterschiedlichen Endergebnissen der Simulation führen – also unterschiedliche Ausführungspfade beschritten werden. Dem Nutzer soll an eben diesen Punkten die Möglichkeit geboten werden, zu einem späteren Zeitpunkt an eine bestimmte Stelle zurückzukehren, um eine alternative Entscheidung treffen zu können.

Konzeptionelle Aspekte

Die Umsetzung bietet mindestens zwei mögliche Wege, welche separat voneinander verfolgt werden können. Der erste basiert auf der Implementierung des Trace-Konzepts und einer Erweiterung dessen (Abbildung 12).

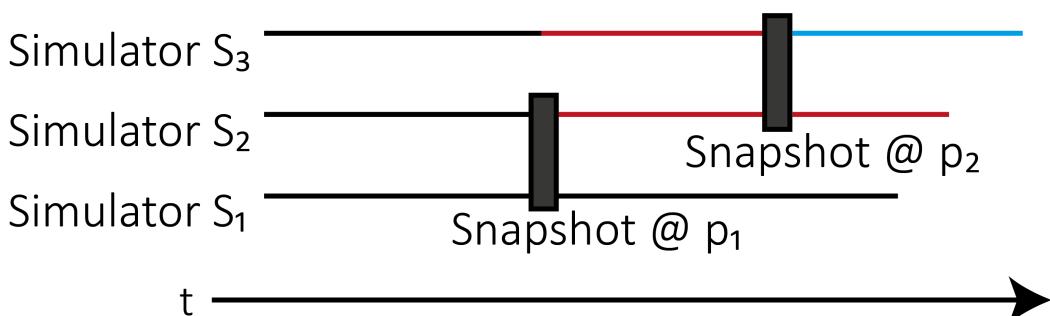


Abbildung 12 Schematische Darstellung des ersten möglichen Wegs.

Letztere ermöglicht dem Nutzer die Wahl eines Punktes p_1 auf dem Trace, der sich in der Vergangenheit befindet. Mit der Auswahl wird der bisherige Trace in einem neuen und eigenständigen Simulator bis zu p_1 ausgeführt. Das Ergebnis wird als Snapshot bezeichnet. Es ermöglicht fortan eine vom Simulator₁ unabhängige Ausführung ab der Stelle, ab welcher der Snapshot erstellt wurde. Fortan existieren zwei Simulatoren, mit einem gemeinsamen Trace bis zum Punkt p_1 , parallel zueinander. Im Interface und insbesondere im Bereich der Darstellung des Traces wird der Punkt des Pfades hervorgehoben, an der ein Snapshot des aktuellen

Simulators zu einem direkten Klon geführt hat. An diesem Punkt ist auch ein Wechsel zu dem jeweils anderen Simulator möglich.

Der zweite mögliche Weg (Abbildung 13) besteht darin, zur Zeit der Ausführung an einem durch den Nutzer gewünschten Punkt p der Ausführung den bestehenden Simulator S_n zu klonen. Klonen bedeutet dabei, dass der Speicher, den der Simulator S_n für die Ausführung belegt, vollständig kopiert wird und zwei zusätzliche Simulatoren S_{nc} und S_{nc1} erstellt werden.

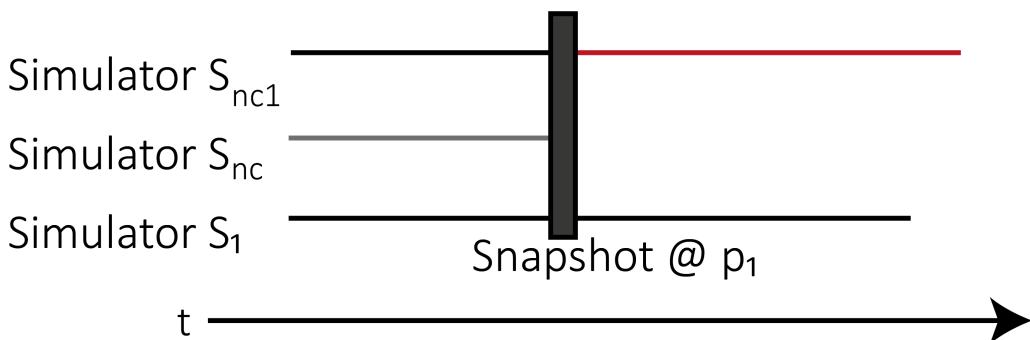


Abbildung 13 Schematische Darstellung des zweiten möglichen Wegs

Explizit ausgeschlossen von diesem Vorgang sind bereits existierende und mit dem Simulator verknüpfte Simulatoren beziehungsweise deren Kopien. Dies sind beispielsweise in der Vergangenheit erzeugte Snapshots des Simulators. Der ursprüngliche Trace hingegen wird kopiert. Der Simulator S_{nc} verbleibt im Hintergrund und dient als Ausgangsbasis für spätere Klone des Simulators S_n . Der zweite Simulator S_{nc1} steht ebenso wie die Ausgangsversion S_n zur Ausführung bereit. Änderungen an den unterschiedlichen Simulatoren beeinflussen die jeweils anderen nicht mehr. Zwischen allen Simulatoren bleibt, wie beim zuerst beschriebenen Ansatz, die Verbindung bestehen. Die Basis hierfür ist der Punkt p , an dem die initiale Kopie erzeugt wurde.

Eine mögliche Abwandlung und Erweiterung dieses Ansatzes besteht darin, nach jedem Ausführungsschritt einen versteckten Klon S_{nc} des Simulators im aktuellen Zustand zu erstellen. Dies entlastet den Nutzer des Simulators in der Auswahl des Moments, in der ein Klon erstellt werden muss.

Eine weitere mögliche Abwandlung kann darin bestehen, für die letzten k Schritte der Ausführung automatisch einen Klon S_{nc} zu erstellen. Dieser wird dann nach $n - k$ Schritten in der Ausführung verworfen, so der Nutzer einen konkreten Klon nicht explizit behalten möchte und eine erste Simulatorkopie erstellt hat.

Für die Visualisierung der Klone erfolgt eine Annotation im Trace. Sie zeigt dem Nutzer zum einen direkt an, aus welchem Trace-Punkt Simulatoren hervorgegangen sind und welche Simulatoren existieren. Zudem ermöglichen sie dem Nutzer den direkten Sprung zu den jeweils anderen Simulatoren. Gelöschte Simulatoren werden hingegen nicht mehr angezeigt.

Diskussion

Beide Hauptwege haben sowohl Vor- als auch Nachteile, welche es für die Umsetzung zu beachten gilt. Beim zuerst beschriebenen Weg, der Erweiterung des Trace-Konzepts, ist der geringe Speicherverbrauch der augenfälligste Vorteil. Egal ab welchem Zeitpunkt der Ausführung ein alternativer Ausführungspfad beschritten werden soll: es kommt erst im Moment der Auswahl zu einer Belegung des Speichers. Dies geschieht durch die erneute Ausführung der Simulation mit den gespeicherten Werten des Traces bis an eben jenen ausgewählten Punkt.

Beim zweiten Weg, der einfachen Erstellung eines Klons, wird hingegen zum Zeitpunkt des Klonens der Speicher belegt. Allgemein wird für jeden Klon Speicher verbraucht, welcher aktiv im System gehalten werden muss.

Der, im Vergleich zum direkten Erstellen eines Klons, geringe Speicherverbrauch bei der Nutzung des Traces-Konzept wird mit einer Zunahme an Zeit, welche für eine erneute Ausführung der Simulation nötig ist, erkauft. Bei jeder erneuten Ausführung muss vom Anfang bis zum ausgewählten Zeitpunkt der Simulation eine Neuberechnung durchgeführt werden. Insbesondere bei Simulationen mit einer großen Anzahl von Nodes, rechenaufwendigen OT-Funktionen oder vielen bereits ausgeführten Simulationsschritten kann dies spürbar sein. Ohne auf empirisches Wissen zurückgreifen zu können erlaubt die nähere Betrachtung das

Aufstellen der Hypothese, dass es insbesondere in der semiautomatischen Ausführung der Simulation, verbunden mit dem Nutzung von erweiterten Break Points, zu einer potentiell höheren Anzahl an Ausführungsschritten und einer damit verbunden verlängerten Ausführungszeit von Klonen kommen kann. Diese Hypothese beruht auf der Annahme, dass es durch die semiautomatische Ausführung, im Gegensatz zur manuellen, sehr einfach ist, eine große Anzahl von Schritten der Simulation auszuführen. Soll der Simulator an einem ausgewählten Punkt p auf dem Pfad der Ausführung geklont werden, so ist erneut die gleiche Ausführungszeit notwendig, wie sie bei der ersten Ausführung der Simulation notwendig war, um zum Punkt p zu gelangen. Relevant ist dabei die reine CPU-Zeit und nicht jene, welche sich durch andere Unterbrechungen ergibt.

Die Erstellung einer Speicherkopie des aktuellen Simulators als Ausgangsbasis für weitere Klone ist unabhängig von der Anzahl der Ausführungsschritte, welche die Simulation schon ausgeführt hat, eine sehr schnelle Möglichkeit, eine Kopie des Simulators zu erstellen. Alle relevanten Aktionen finden dabei im Arbeitsspeicher statt und eine erneute Abarbeitung der Schritte ist nicht mehr notwendig.

Neben der Erstellung von Kopien muss auch deren Löschung bedacht werden. Eine Löschung es Ausgangssimulators ist mit der Löschen aller anderen Simulatoren, die mit ihm durch das Klonen in Verbindung stehen, verbunden.

Bei der Diskussion nicht berücksichtigt wurden Optimierungsmöglichkeiten und Strategien in Bezug auf den Speicherverbrauch bei den unterschiedlichen Ansätzen. Dies müsste separat erfolgen

6.3 Break Points

Um bei der Ausführung nicht alle Nodes, ihre unterschiedlichen Variablen und die daraus erwachsenden Zustände beobachten und daraufhin überprüfen zu müssen, ob ein bestimmter Zustand eingetreten ist, wäre die Möglichkeit dies zu automatisieren eine sehr wünschenswerte. Die Idee für einen Ansatz geben moderne Integrated Development Environments (IDEs) mit der Bereitstellung der Funktionalität zum Erstellen von

Break Points. Diese erlauben den Nutzer das Markieren einer Quellcodezeile und das unterbrechen der Ausführung, sobald diese erreicht wird. Eine Erweiterung dieses Ansatzes stellen bedingte Break Points da. Dabei muss neben der eigentlichen Codezeile auch eine bestimmte durch den Nutzer definierte Bedingung erfüllt sein, damit die Ausführung unterbrochen wird.

Im Folgenden werden unterschiede Aspekte zu dem Break Point-Ansatz und seiner Umsetzung erörtert.

Konzeptioneller Ansatz

Der Grundgedanke ist einfach, die Umsetzung im Rahmen der Simulationsumgebung bedarf dennoch einer Einordnung und Anpassung an die Gegebenheiten. Erster und wichtigster Punkt dabei ist die Tatsache, dass anders als in einer IDE die Break Points nicht frei mit jeder beliebigen Zeile Quellcode verknüpft werden können. Dies ist zum einen nicht gewollt, da bei der Arbeit auf der Ebene des Quellcodes die Abstraktion zu einem gewissen Grad verloren geht. Zum anderen würde an dieser Stelle eine Funktionalität implementiert werden, welche durch eine IDE viel besser geleistet werden kann. In jener Umgebung könnte mit dem Quelltext auf einer ganzheitlichen Ebene gearbeitet werden. Dies würde neben dem Quellcode des Algorithmus und des Simulators auch zusätzliche den Code einbeziehen, der durch dritte Bibliotheken bereitgestellt wird, was eine detaillierte Auseinandersetzung erlaubt.

Anstelle des freien Setzens von Break Points können bestimmte und im Kontext der Nutzung des Simulators besonders interessante Variablen mit einem Break Point versehen werden. Variablen können in diesem Zusammenhang nicht nur in python übliche Typen sein. Variablen können komplexe Objekte, Queues oder andere Datenstrukturen, die in der Simulationsumgebung genutzt werden, sein. Interessante Variablen sind beispielsweise jene Queues, die abzuarbeitende oder abgearbeitete Operationen aufnehmen. Das Dokumentenmodell ist eine ebenso interessante Variable, welche durch gesetzte Break Points beobachtet werden kann. Interessant werden diese, stellvertretend aufgezählten Variablen durch ihren Charakter innerhalb der Simulationsumgebung. Variablen, die nicht nur Zwischenergebnisse der Ausführung speichern,

sondern die Endergebnisse der Berechnung aufnehmen, sind eben solche von besonderem Interesse.

Diese Variablen sind üblicherweise nicht Teil des OT-Algorithmus selbst, sondern gehören zu den Nodes, welcher den Algorithmus ausführt, oder zur Simulationsumgebung selbst. Auf Grund dieses Umstands bedarf es in der ersten Umsetzung nicht der Möglichkeit, dass der Nutzer im zu simulierenden Algorithmus mögliche Break Points definieren kann. Eine Alternative kann eine Vereinfachung dieser Möglichkeit sein, in dem feste Punkte für die Überprüfung vorgegeben werden.

Für eine Umsetzung gibt der bis jetzt beschriebene Ansatz bereits eine Richtung vor. Im Detail bedarf es jedoch einer detaillierteren Auseinandersetzung, da unterschiedliche Vorgehensweisen denkbar sind. Nachfolgende Detailfragen sind noch zu erörtern:

- a) **Was** sollen die Break Points **prüfen** können?
- b) **Welche Reaktion** soll erfolgen, wenn ein Break Point positiv überprüft wurde?
- c) Soll der Nutzer selbst **Bedingungen formulieren** oder nur vorformulierte auswählen und parametrisieren können?

Was prüfen?

Es sind ganz unterschiedliche Bedingungen vorstellbar, welche in Form eines Break Points überprüft werden können. Im Folgenden werden Abfragen aufgelistet, welche es in der Zeit der Arbeit mit unterschiedlichen OT-Algorithmen immer wieder zu überprüfen galt. Diese nicht abschließende Auflistung beruht also auf der Grundlage von Erfahrungswerten und keinem Formalismus. Nach der Überprüfung der Abfrage kann entweder True oder False als Wert zurückgeliefert werden. Wird True zurückgegeben, so ist der Vergleich mit einem Parameter positiv.

- Befindet sich der Simulator in der k -ten Iteration?
- Befindet sich der Node c in der k -ten Iteration?
- Wurde durch den Node c die Operation o ausgeführt?

- Wurde durch den Node c die Operation o mit den Werten x ausgeführt?
- Enthält das aktuelle Dokument von Node c das Wort s ?
- Enthält das aktuelle Dokument aller Nodes das Wort s ?

Diese Regeln sinnvoll eingesetzt und mit den Operatoren der booleschen Algebra verknüpft ermöglichen es dem Nutzer die Überprüfung komplexer Sachverhalte.

Wie reagieren?

Die Bezeichnung Break Point intendiert zuweilen, dass eine Art Unterbrechung als Reaktion erfolgen muss. Wie Eingangs erwähnt, unterscheidet sich die Umsetzung von Break Points in der Simulationsumgebung von jener in einer IDE. Eine Unterbrechung der Ausführung ist dabei dennoch die vom Nutzer erwartbarste aller möglichen Aktionen, welche ausgelöst werden können. Insbesondere bei einer semiautomatischen Ausführung ist dies von hohem Nutzen. Aber auch bei der manuellen Ausführung wird durch die Unterbrechung der Ausführungsmöglichkeit ein unbeabsichtigtes Fortführen verhindert.

Über eine einfache Unterbrechung der Ausführung hinausgehende Aktionen können, insbesondere im Zusammenspiel mit anderen in diesem Kapitel vorgestellten Konzepten, einen hohen Nutzen entfalten. Aufzuzählen ist dabei beispielsweise das Erstellen eines Snapshots, wenn ein bestimmter Zustand erreicht wurde. Den Rahmen für das Mögliche stellen an dieser Stelle nur das Kriterium der Nützlichkeit im Kontext der Simulationsumgebung und die Nutzungshürden in der Hinsicht auf, dass diese so gering wie möglich sein soll.

Wie formulieren?

Um die Bedingungen auszudrücken, welche es zu überprüfen gilt, sind unterschiedliche Ansätze denkbar. Da die Formulierung über das Frontend möglich sein soll, werden einige Optionen von vornherein ausgeschlossen. So ist die Annotation direkt im Quellcode nicht möglich. Auch die Möglichkeit, Regeln in einer formalen Sprache über das Frontend zu formulieren, ist für den geplanten Nutzungsrahmen und vom

geschätzten Aufwand der Umsetzung zu hoch angesetzt. Insbesondere der Freiheitsgrad bei der Eingabemöglichkeit, welche durch eine gesonderte Sprache und ihre Interpretation entsteht, schafft einen großen und im Kontext der Aufgabenstellung nicht zielführenden Arbeitsrahmen. Die Opportunität einer Kombination von Bedingungen ist jedoch eine Funktion, welche nicht außen vor gelassen werden darf, da diese die Verwendbarkeit durch die Kombination unterschiedlicher Regeln erst ermöglicht.

Ein Interface für die Formulierung der Regeln muss es also ermöglichen, die existenten Regeln zu parametrisieren und diese mit booleschen Algebra zu verknüpfen. Hierfür wäre die Umsetzung durch ein Drop-Down-Menü denkbar. Dieses würde die Auswahl der Regeln in Kombination mit einem Freitextfeld für die Eingabe der Parameter sowie der booleschen Operationen erlauben. Das Verschieben und Vertauschen erlaubt eine einfache Modifikation des Ausdrucks.

6.4 Erweiterte Break Points

Die zuvor eingeführten Break Points sind, durch ihre Möglichkeiten der Parametrisierung und in Kombination mit der booleschen Algebra, bereits sehr vielseitig verwendbar. Wie beschrieben, kann es den Nutzer bei der Arbeit mit der Simulationsumgebung unterstützen und diese erleichtern. Darüber hinaus sind Erweiterungen denkbar, welche bei besonderen Aufgabenstellungen zum Tragen kommen können. Nachfolgend werden Möglichkeiten eines erweiterten Konzepts aufgezeigt und erörtert.

Konzeptioneller Ansatz

Das Unterbrechen der Ausführung ist die einfachste aller denkbaren Aktionen, welche durch die Erfüllung einer Bedingung ausgelöst werden können. Darüber hinaus ist das Ausführen von bestimmten Aktionen, welche hier bereits an anderer Stelle erörtert und konzeptionell umrissen wurden, ein denkbarer Mehrwert. So können explizit Situationen überprüft und bei ihrem Eintritt eine Aktion ausgelöst werden. Das Erstellen von Snapshots, sobald ein bestimmter Zustand erreicht wird, ist eine solche Aktion. Aufbauend darauf wäre eine weitere Möglichkeit das Anhalten der Ausführung und das Fortsetzen an einem anderen Snapshot,

wenn beispielsweise eine Situation erkannt wurde, die zu keinem gewünschten Ergebnis mehr führen kann. Auch das Ausführen aller Snapshots, nachdem der erste Durchlauf durch den ursprünglichen Simulator, wäre eine Aktion, welche im Zusammenspiel mit dem semiautomatischen Ausführen des Simulators denkbar wäre.

Die zuvor genannten Aktionen sind nur Beispiele und sollen zur Verdeutlichung dienen. Um diese und darüber hinausgehende Erweiterungen zu ermöglichen, die primäre durch den Nutzer der Simulationsumgebung erstellt werden sollen, bedarf es der Grundlage.

Um Aktionen Dritter auslösen zu können, muss es eben diese die Möglichkeit geben, solche Aktionen bereitzustellen. Hierfür kann eine abstrakte Klasse, welche alle externen Schnittstellen definiert, über die die Simulationsumgebung kommuniziert, mit der Umsetzung der erweiterten Break Points bereitgestellt werden. Eine Konfiguration für die Nutzung der Funktionalität wird beim ersten Initialisieren der Klasse an die Simulationsumgebung übergeben. Darin werden Informationen für die Darstellung im Hauptinterface, also Hinweistexte und Annotationen, übergeben. Der Aufruf der eigentlichen Funktionalität erfolgt über eine in der abstrakten Klasse definierte Funktion, welche immer gleich ist. Ihr Verhalten wird durch den Nutzer implementiert und ist dann nutzbar.

6.5 Protokolltreue

Die Reihenfolge von Nachrichten spielt insbesondere in der Netzwerk-kommunikation eine besondere Rolle. Durch die Vermittlung von Nachrichten im Gegensatz zu Vermittlung von Leitungen können die Nachrichten unterschiedliche Routen auf dem Weg zum Empfänger nehmen, wodurch auch die Dauer der Übertragung variieren kann. Gewisse Protokolle sichern dabei zu, dass sich die Reihenfolge der Nachrichten durch die Kommunikation über das Netzwerk nicht verändert. Sie tragen dafür Sorge, dass die Nachrichten in der gleichen Reihenfolge bei *B* angekommen, wie sie *A* verschickt hat. Andere Protokolle bieten diese Möglichkeit explizit nicht und übertragen diese Aufgabe an die Anwendung.

Im Folgenden wird beschrieben, wie das Verhalten für die Simulationsumgebung umgesetzt werden und konsistent nutzbar gemacht werden kann.

Konzeptioneller Ansatz

Die Übertragung der Nachrichten der Nodes ist ein Teil der Simulation. Die Möglichkeit, den Nachrichtenversand unter der Einhaltung eines gewissen Protokolls sicher zu stellen, kann die Güte der Simulation nachhaltig beeinflussen. Die Simulationsumgebung des Netzwerkes legt das Augenmerk auf die OSI-Schicht 4, die Transportschicht. Als Netzwerkprotokolle auf der OSI-Ebene 4 sind TCP und UDP für das tatsächliche Erreichen der Nachrichten sowie die Einhaltung der Reihenfolge verantwortlich. Die hier besprochene Komponente befasst sich nur mit dem Aspekt der Einhaltung der Nachrichtenreihenfolge. Insbesondere wird nicht der Aspekt des Paketverlustes bei UDP sowie die sich daraus für die Anwendung ergebenden Aufgaben betrachtet. Zudem sind nur ausgewählte Eigenschaften Gegenstand der Simulation. So wird die Übertragungsdauer der Nachrichten innerhalb des Netzes nicht angegeben. Die Laufzeit der Nachrichten im simulierten Netzwerk ergibt sich allein durch die relative Ordnung der Nachrichten zueinander beim Empfänger-Node. Kommt Nachricht m_2 vor m_1 bei B an, so war die Laufzeit von m_2 kürzer als die von m_1 . Eine konkret benennbare Laufzeit beziehungsweise Unterschiede dieser gibt es hingegen nicht.

Bei der verwendeten Speicherstruktur für den Netzwerk-Node handelt es sich um eine Queue. Die Verwendung einer Queue sichert ohne explizite Vorgabe und ohne verändernde Eingriffe des Nutzers zu, dass neue Nachrichten am Ende der Queue eingefügt und alte Nachrichten am Anfang aus der Queue herausgenommen werden. Eine Änderung der Reihenfolge geschieht dabei nicht.

Dieses Verhalten ist für die Implementierung von TCP besonders günstig. Die First-In-First-Out-Eigenschaft (FIFO) stellt sicher, dass die Nachrichten immer in der Reihenfolge beim Empfänger-Node ankommen, wie sie den Sender-Node verlassen haben. Komplexer wird es in diesem Zusammenhang an folgender Stelle. Zum besseren Verständnis wird zuvor ein neuer Begriff eingeführt. Ein logischer Nachrichtenblock

bezeichnet eine Menge von Nachrichten, die ein Node A auf Grund der Ausführung einer Operation an alle anderen Nodes B_1 bis B_n verschickt.

Um den unterschiedlichen Laufzeiten in der Simulation gerecht zu werden bedarf es der Möglichkeit diese auch zu simulieren. Beschrieben werden kann das Verhalten wie folgt. Es befinden sich drei logische Nachrichtenblöcke in der Netzwerkqueue. Zwischen dem Sender-Node A und dem Empfänger-Node B_1 besteht eine Verbindung mit einer signifikant kürzeren Nachrichtenlaufzeit als zwischen A und B_2 oder B_3 . Daraus ergibt sich, dass die drei Nachrichten an B_1 aus den jeweiligen drei logischen Nachrichtenblöcken vor allen anderen Nachrichten, die aus den selben logischen Nachrichtenblöcken an B_2 oder B_3 gesendet werden, bei B_1 eintreffen. Dabei bleibt im TCP-Modus die beim Senden entstandene Reihenfolge beim Empfänger erhalten.

Dieses Verhalten spiegelt die Tatsache wieder, dass die Laufzeit einer Nachricht zwischen dem Node A und B_n sowie A und B_m unterschiedlich lang sein können. Daraus ergibt sich auch ein unterschiedlicher Ankunftszeitpunkt der Nachricht beim Empfänger-Node. Die Länge der Nachrichtenlaufzeit bestimmt sich in der Simulationsumgebung dabei aus der Differenz zwischen Ankunft in der Netzwerkqueue des Netzwerk-Nodes und ihrem Entfernen aus selbiger, was mit der gleichzeitigen Ankunft beim Empfänger-Node verbunden ist.

Für den semiautomatischen Modus bedarf es der Überlegung, wie das beschriebene Verhalten in der Simulationsumgebung zugesichert werden kann. Wie also die Festlegung erfolgt, zwischen welchen beiden Nodes eine priorisierte Verbindung besteht. Die Auswahl könnte einerseits zu Beginn der Simulation zufällig vorgenommen und über die ganze Laufzeit beibehalten werden. Die Verwendung eines Seeds bei der Auswahlfunktion könnte an dieser Stelle die Wiederholbarkeit ermöglichen. Andererseits ist die Festlegung von Netzwerkprioritäten durch den Nutzer vorstellbar. Das Verhalten im UDP-Modus ändert sich im Vergleich zum TCP-Modus an bestimmten Stellen. Dem Nutzer wird die freie Wahl darüber gegeben, in welcher Reihenfolge er die Pakete, welche sich in der Netzwerkqueue befinden, weiterleiten möchte.

Bei der semiautomatischen Simulationsausführung bedarf es einer Lösung, die eine Reihenfolge der Nachrichten außerhalb der gegebenen ermöglicht. Hierbei ist die Nutzung eines Mechanismus interessant, der sicherstellt, dass ein gewisses Verhältnis von versendeten Nachrichten, die in FIFO- und in non-FIFO-Reihenfolge, verschickt werden, gewahrt bleibt. Eine komplette nichtzusammenhängende Reihenfolge ist dabei auch denkbar. Eine Simulation eines solchen Verhaltens sollte also möglich sein, stellt jedoch einen Extremfall da, der in der Regel nicht auftreten sollte. Denkbar wäre eine Funktion, welche auf Basis aller vorangegangenen Entscheidungen darüber befindet, ob als nächstes eine zufällige oder die nächst mögliche Operation in der Queue versendet wird. Als Parameter könnte dieser Funktion ein Wahrscheinlichkeitswert übergeben werden, der die Wahrscheinlichkeit für eine Operation außerhalb der FIFO-Reihenfolge ausdrückt.

Der Aspekt der Wiederholbarkeit der Simulation spielt im UDP-Modus noch einmal eine andere Rolle als im TCP-Modus. Insbesondere bei der zufälligen Reihenfolge der Netzwerkpakete wird dies sichtbar. Es muss sichergestellt werden, dass die Netzwerkpakete bei der Wiederholung in der gleichen Reihenfolge versendet werden, wie sie bei der ersten Ausführung versendet wurden. Auch hier währe die Einführung eines Seeds für die Zufallsfunktion eine denkbare Möglichkeit. Durch die zusätzliche Angabe dieses Seeds kann auch außerhalb der erneuten Ausführung des Traces für spätere Simulationen die gleiche Zufallsreihenfolge verwendet werden.

Aspekte der Umsetzung

Für das Frontend wird beschrieben, wie eine Umsetzung der zuvor beschriebenen Funktionalität auf Ebene des Nutzers erfolgt. Für das Backend wird darauf eingegangen, wie die besprochenen Eigenschaften umgesetzt und gleichzeitig Raum für eine spätere Erweiterung geschaffen werden kann.

Der Wechsel zwischen den einzelnen Netzwerk-Modi über das Frontend ist immer dann möglich, wenn sich keine Nachrichten mehr im Netzwerk befinden. Dies kann über einen einfachen Schalter im Frontend erfolgen.

Dieser dient auch als Anzeige, welcher Modus momentan aktiviert ist. Ein Wechsel während einer aktiven Netzwerkkommunikation würde dem realen Verhalten des Netzwerks widersprechen, weshalb dies nicht möglich ist. Auf die Simulation mehrerer Verbindungen über unterschiedliche Protokolle wurde ebenso verzichtet, da diese zu einer Erhöhung der Komplexität der gesamten Simulationsumgebung und des einzelnen Simulationslaufes führen würde. Zudem ist diese in der Simulation, anders als die tatsächliche Ankunftsreihenfolge der Nachrichten beim Empfänger, nicht relevant.

Durch die Aktivierung des TCP-Modus es ist nicht mehr möglich, die Reihenfolge der Nachrichten eines Nodes in ihrer absoluten Reihenfolge zu verändern. Neue Nachrichten werden an das Ende der Queue angefügt. Elemente die versendet werden, verlassen die Queue am Anfang. Die Möglichkeit der Reihenfolgenänderung von Nachrichten innerhalb eines logischen Nachrichtenblocks bleibt bestehen und wird durch ein Verschieben der Nachrichten im Frontend ermöglicht. Alternativ dazu können diese Nachrichten auch direkt verschickt werden, indem dies explizit durch den Klick auf ein Steuerelement ausgelöst wird. Der Wechsel in den UDP-Modus ändert das zuvor beschriebene Verhalten in Bezug auf die Implementation im Frontend in einem Punkt. Durch die Tatsache, dass die Nachrichtenreihenfolge bei UDP beliebig sein kann, bedeutet dies, dass auch die Nachrichten auch über logischen Nachrichtenblöcke hinaus eine andere Reihenfolge einnehmen können. Konkret kann also die Nachricht zwischen A und B_1 aus dem zweiten Block vor der Nachricht aus dem ersten Block bei B_1 ankommen.

Für beide Modi gibt es in jedem Fall ein Element, mit dem die nächste Nachricht im Netzwerk verschickt werden kann. Ein zweites Element ermöglicht es, nicht die in der Ordnung als nächstes kommende Nachricht sondern eine der Nachrichten aus der Menge der nächst möglichen zu verschicken. Je nach Modus wird, über die API, ein unterschiedlicher Algorithmus im Backend für die Nachrichtenauswahl angesprochen.

Auf der Ebene des Backends muss der Scheduler genauer betrachtet werden. Der Scheduler ist die Einheit, welche die Nodes verwaltet. Er hat

somit Zugriff auf alle Nodes und die entsprechenden Speicherstrukturen. An dieser Stelle besteht die Möglichkeit der Implementierung der entsprechenden Funktion, die eine Auswahl anhand der gegebenen Kriterien wählt. Damit eine gewisse Vereinheitlichung erfolgt, wird eine allgemeine Funktion bereits im abstrakten Scheduler umgesetzt. Diese Funktion ruft mit den übergebenen Parametern die eigentliche Auswahlfunktion auf. Als Parameter muss der eindeutige Bezeichner des ausgewählten Modus übergeben werden. Die Übergabe zusätzlicher Parameter ist ebenso möglich und abhängig vom konkreten Auswahlalgorithmus. Sie werden ebenso wie der Bezeichner des Modus über die API übergeben. Die Behandlung der Rückgabewerte erfolgt dann wie bei jeder anderen Ausführung eines Nodes.

Durch die erfolgreiche Auswahl wird die Übertragung aller Parameter an den Tracer ausgelöst. Dieser speichert diese und ermöglicht später somit ihre Wiederverwendung.

6.6 Externe Schnittstellen

Neben der Möglichkeit der Erprobung von OT-Algorithmen, welche durch den Nutzer der Simulationsumgebung selbst umgesetzt wurden, ist es das zweite Teilziel, auch bestehende und durch Dritte implementierte OT-Algorithmen im Rahmen der Simulationsumgebung erproben und ausprobieren zu können. Da diese in aller Regel nicht direkt den Anforderungen der Simulationsumgebung genügen, um mit diesen zu arbeiten, ist es notwendig, eine Schnittstelle für die Kommunikation zwischen beiden Stellen zu konzipieren. Nachfolgend wird ein Vorgehen diskutiert und ein darauf aufbauendes Konzept erörtert.

Untersuchung

Bevor ein Lösungsansatz auf konzeptioneller Ebene erörtert und diskutiert werden kann, bedarf es einer Untersuchung, mit welcher Art von Anwendungen und welchen Besonderheiten zu rechnen ist. Abschließend kann dies in dem Rahmen dieser Arbeit sowohl theoretisch als auch praktisch nicht passieren. Aus diesem Grund werden stellvertretend zwei existierende OT-Implementationen genauer betrachtet. Dies dient einerseits zur Schaffung einer Diskussionsgrundlage, mit welchen

Herausforderungen bei der Schaffung einer externen Schnittstelle zu rechnen ist. Andererseits können darauf aufbauend Anforderungen an Lösungsansätze diskutiert und erörtert werden.

Für eine Betrachtung wurden die bestehenden Implementationen share.js [14] und Etherpad Lite [13], beides OpenSource Software, näher untersucht. Bei share.js handelt es sich um eine OT-Bibliothek, welche in JavaScript und unter Verwendung der Plattform node.js [15] entwickelt wurde. Es kapselt die OT-Funktionalität und ermöglicht so, diese in einer beliebigen – vornehmlich webbasierten – Anwendungen umzusetzen. Etherpad Lite ist „a really-real time collaborative word processor for the web“ [13]. Es wird die OT-Funktionalität soweit abstrahiert, dass dem Nutzer lediglich ein Frontend zur Verfügung steht. Dieses unterteilt sich in eines zum kollaborativen Arbeiten und eines zum Administrieren. Auf die Untersuchung einer CloseSource Software wurde verzichtet. Stattdessen werden die Erkenntnisse, die aus der Untersuchung der OpenSource Software gewonnen wurden, genutzt, um eine Einschätzung bezüglich CloseSource Software abzugeben.

Die Auseinandersetzung mit share.js ist auf der Ebene des Quellcodes gradlinig. Der gesamte Quellcode ist OpenSource und kann somit vollständig angeschaut und untersucht werden. Dieser ist überwiegend mit Kommentaren versehen, welche gebündelt auch als eine externe Dokumentation verfügbar sind. Die Qualität der Kommentare schwankt dabei und lässt viele Themen offen. Es wird beispielsweise nur sehr kontextuell darauf eingegangen, welche Funktionen bestimmte Teile im Quellcode erfüllen. Hierfür ist jedoch Vorwissen notwendig, welches nicht gebündelt zur Verfügung gestellt wird.

share.js verarbeitet die Änderungsoperationen an einem Dokument clientseitig und leitet sie an die Server-Instanz weiter. Dieser übernimmt seinerseits die Speicherung und Weiterleitung der Anfragen und Daten.

Nach der genaueren Untersuchung des Quellcodes ergibt sich das Bild, dass die Funktionalität zum Verarbeiten von Änderungen, die von unterschiedlichen Nutzern an einem Dokument durchgeführt werden, sehr dicht mit dem Dokumentenmodell verzahnt ist. Konkret enthält das Dokumentenmodell selbst die vollständige Funktionalität, um Änderun-

gen von außen entgegen zu nehmen oder selbst Änderungen von anderen Teilnehmern mitzuteilen. Diese Änderungen werden immer direkt auf das jeweilige Dokumentenmodell angewandt.

Diese Tatsache führt zu der ersten Einschätzung, dass eine externe Anbindung gewissen Schwierigkeiten unterliegen wird. Unter anderem ist zu vermuten, dass noch nicht konkret abzuschätzende Seiteneffekte dadurch eintreten, dass zwei Dokumentenmodelle verwaltet werden müssen. Auch weil durch die erste Untersuchung nicht abschließend geklärt werden kann, welche Funktionalität noch in anderen Teilen des Quellcodes gekapselt ist, fällt das Treffen einer hinreichend zuverlässige Aussage darüber, ob und wie die Verwendung des Codes in einem von share.js losgelösten Rahmen, schwer.

Im Gegensatz zu share.js ist Etherpad Lite ein deutlich größeres und erwachseneres Projekt. Die Größe ist durch die Lines of Code direkt ersichtlich. Dass Etherpad Lite ein reiferes Projekt ist zeichnet sich durch unterschiedliche Dinge aus: die umfangreiche Dokumentation, die von der Einrichtung bis zur Verwendung der API reicht, die gute Struktur des Quellcodes, aber auch die umfangreichere Kommentierung des selbigen. Darüber hinaus gibt es ein eher theoretisches Paper [16] über die verwendeten OT-Algorithmen.

Die beiden Punkte „Struktur des Quellcodes“ und „Kommentierung“ sind besonders wichtig und können entscheidend für den Erfolg des Vorhabens sein, das Projekt und insbesondere der OT-Algorithmus per Schnittstelle mit der Simulationsumgebung zu koppeln.

Genauso wie share.js führt auch Etherpad Lite die OT-Operationen clientseitig aus. Im Unterschied zu share.js ist die Funktionalität um Änderungen an einem Dokument zu verarbeiten, vom eigentlichen Dokument losgelöst. In wenigen JavaScript-Dateien ist die relevante Funktionalität so zusammengefasst, dass sie durch einen kundigen Entwickler auch außerhalb von Etherpad Lite verwendet werden kann. Mit einem Script, mit welchem unterschiedlichste Testfälle ausführt werden können, lässt sich diese Aussage untermauern. In diesem Script werden alle Testfälle außerhalb der direkten Umgebung von Etherpad

Lite ausgeführt. Direkt benötigt und eingebunden werden lediglich zwei Bibliotheken.

Durch die Untersuchung der beiden Beispiele können folgende Schlussfolgerungen festgehalten werden. Je nach konzeptionellem Aufbau und Struktur ist es ohne detailliertes Hintergrundwissen über den Quellcode schwierig abzuschätzen, ob der Code zum einen überhaupt und zum anderen mit einer Schnittstelle an die Simulationsumgebung zu verknüpfen und anzusprechen ist. Es ist sehr stark davon abhängig, ob und wie die Codeteile, die für die OT-Funktionalität entscheidend sind, losgelöst vom restlichen Projektcode umgesetzt sind. Bei der Umsetzung von Etherpad Lite ist die Kommunikation mit dem Code durchaus vorstellbar. Bei share.js bleiben diesbezüglich Fragen offen, die allein mit einem besseren Verständnis des Quellcodes und einer Testumsetzung beantwortet werden können. Bei CSS ist eine solche Untersuchung gar nicht möglich. Die für die Simulation relevanten Teile des Codes könnten nur dann genutzt werden, wenn eine Schnittstelle und eine entsprechende Dokumentation ihrerseits bereitgestellt wird. Hiervon kann jedoch nicht ausgegangen werden.

Nach der Untersuchung dritter Software folgt als nächstes ein Schritt zurück und der Fokus verlagert sich auf die Simulationsumgebung selbst. Unter Beachtung der vorherigen Erkenntnisse muss geklärt werden, ob überhaupt und wenn ja, für welche Teile der Simulationsumgebung Schnittstellen nach außen zum einen gebraucht werden und zum anderen notwendig sind.

Die Schwierigkeiten, die die heterogene Struktur fremder Projekte bietet, erlaubt es nicht, eine zuverlässige Schnittstelle anzubieten, über die selbst mit leicht angepasstem Code direkt kommuniziert werden kann. Selbst bei der Entwicklung einer möglichst offenen und viele Eventualitäten abdeckenden Schnittstelle kann konkreten Sonderfällen damit in keinster Weise Rechnung getragen werden. Alternativ zu einer Schnittstelle, die wie ein Universalstecker in jede Steckdose passt, kann jedoch ein anderer Weg versucht werden: die Schaffung einer Steckdose, für die in der entsprechenden Programmiersprache und mit den entsprechenden

Mitteln als ein passender Stecker gebaut werden kann. Der Unterschied zwischen den beiden Ansätzen ist, dass der zuletzt genannte einen Rahmen von Seiten der Simulationsumgebung vorgibt, der durch die zu simulierende Anwendung erfüllt wird. Dies erfordert Entwicklungsaufwand auf Seiten der zu simulierenden Anwendung, ermöglicht aber so eine maximale Kompatibilität zwischen Simulationsumgebung und zu simulierender Umsetzung.

Schnittstelle

Die Schnittstelle nach außen setzt jeweils an den Stellen an, an der die Funktionalität extern gekapselt ist. Gekapselt werden dabei lediglich die Algorithmen, die für eine Transformation der Operationen notwendig sind, sowie die Operationen selbst. Sollte Verwaltungslogik bezüglich eigener Datenstrukturen erforderlich sein, so müssen diese Daten selbst durch den extern gekapselten Code verwaltet werden. Der Simulationsumgebung werden nur diese Daten übergeben, welche auch durch diese explizit angefordert werden. Alle relevanten Daten werden dabei durch einen Commandline-Aufruf nach außen im JSON-Format über stdout übergeben und auch in diesem Format an stdin zurückerwartet.

Alternativ zu diesem Vorgehen wäre noch die Kommunikation über eine REST-Schnittstelle denk- und umsetzbar. Dies hätte, so eine solche Schnittstelle in der den Algorithmus bereitstellenden Anwendung implementier- oder nutzbar ist, den Vorteil, dass der entsprechende Code nicht separat extrahiert und gekapselt werden müsste. Stattdessen könnte diese Schnittstelle so erweitert werden, dass sie die von der Simulationsumgebung geschickten Daten annimmt, verarbeitet und die Ergebnisse zurückschickt.

7. Fallstudie

Nachdem in den vorherigen Kapiteln die theoretischen Grundlagen eingeführt, der Ansatz vorgestellt und die Konzepte, welche auf das Grundgerüst der Simulationsumgebung aufbauen und arbeiten, erörtert wurden, wird der folgende Abschnitt zeigen, wie die Verwendung von *Simone* aussehen und erfolgen kann. Dies geschieht in der Form einer Fallstudie. Dieses soll anhand des ausgewählten Beispiels zeigen, wie *Simone* die Arbeit an OT-Algorithmen erleichtern kann.

Vorgehen

Gegenstand der Untersuchung dieser Fallstudie ist die Simulationsumgebung in der letzten, zum Zeitpunkt der Entstehung der Arbeit aktuellen, Version und den dort implementierten Grundkonzepten. Bei dem für das Fallbeispiel verwendeten OT-Algorithmus handelt es sich um den ursprünglichen von Ellis und Gibbs [1] konzipierten und im Kapitel 2.1 bereits besprochenen.

Die Fallstudie hat das Ziel zu zeigen, dass die Erprobung von OT-Algorithmen unter Verwendung von *Simone* möglich und zugleich gezielter als mit einem alternativen Vorgehen ist.

7.1 Simulationsumgebung

Im Folgenden werden zwei unterschiedliche Arten, wie ein OT-Algorithmus zum Zwecke der Erprobung implementiert werden kann, vergleichend beschrieben. Zum einen wie bei der Untersuchung des benannten Algorithmus von Ellis und Gibbs [1] verfahren wurde, wenn *Simone* zur Anwendung kommen kann und zum anderen, wie vorgegangen wird, ohne dass die Simulationsumgebung verwendet werden konnte.

Im ersten Schritt erfolgt, und dieses Vorgehen ändert sich nicht durch die Verwendung der Simulationsumgebung, eine Analyse des OT-Algorithmus in der Form durchgeführt, dass das veröffentlichte Papier gelesen und studiert wird. Unterstützendes Werkzeug ist dabei insbeson-

dere Papier und Stift, um das beschriebene Verhalten doppelt nachzuvollziehen: zu eigenen Gedanken und zum anderen auf dem Papier. Dabei werden einfachsten Beispiele skizziert und mit der Abarbeitungsvorschrift, die der Algorithmus darstellt, ausgeführt.

Im Papier von Ellis und Gibbs [1] ist dies insbesondere wichtig um die genaue Funktionsweise zu verstehen, da es eine gewisse Herausforderung darstellt, den Algorithmus durch seine zum Teil sehr informale Notation bis ins Detail korrekt zu verstehen. Die Abarbeitung aller Abschnitte hilft auch dabei die unterschiedlichen Bestandteile des Algorithmus im Zusammenspiel richtig einzuordnen und ihre Funktionsweise zuzuordnen.

Ist ein Grundverständnis geschaffenen, so kann zum nächsten Schritt übergegangen werden. Dieser ändert das Vorgehen im Vergleich zu jedem, bei dem die Simulationsumgebung nicht verwendet wird, maßgeblich. Durch die Bereitstellung vieler Funktionen zur Ausführung eines beliebigen OT-Algorithmus braucht im Folgenden deutlich weniger Arbeit bei der Implementation geleistet werden. Bereits umgesetzt sind Funktionalitäten für das Ausführen der Nodes, für die Speicherung der Daten und die Bedienung der gesamten Simulationsumgebung über ein UI. Es bedarf nun der Implementation der Operationen, welche das Papier vorgibt, in python-Code. Hierfür wird von bestehenden python-Klassen geerbt und die jeweils noch nicht implementierte Funktionalität umgesetzt. Als nächstes erfolgt die Implementation der Nodes. Diese sind notwendig zur Ausführung und der individuellen Datenhaltung. Jeder Client-Node repräsentiert im Falle dieses Algorithmus eine Instanz eines Nutzers, der Netzwerk-Node bildet das globale Netzwerk ab. Beide Node-Typen werden vor vorgegebenen Klassen abgeleitet und erben auf diesem Weg Grundfunktionalitäten. Bei Client-Nodes müssen viel Teile explizit implementiert werden, die Teil des Algorithmus sind.

Die Implementation der Transformationsfunktionen erfolgt im weitesten so, wie sie im Algorithmus genannt werden. Hier ist dem Nutzer freie Hand gegeben, ob er sie genau so umsetzt, wie sie im Algorithmus strukturiert sind oder er eine Zusammenfassung vornimmt.

Das Ausführen von Operationen muss so implementiert werden, dass immer eine einzelne Operation genommen und diese verarbeitet wird. Dies ist anders als im Algorithmus beschrieben. Dort werden die Funktionen aus einer bestimmten Queue nacheinander ausgeführt, bis alle abgearbeitet sind. Das Empfangen und Versenden muss ebenso implementiert werden, bedarf aber keiner explizierten Anpassung zum im Algorithmus beschriebenen Vorgehen. Der Netzwerk-Node bedarf keiner weiteren Konfektionierung. Die Standardfunktionalität, die in der abstrakten Klasse bereits umgesetzt ist, erfüllt die Anforderungen des Algorithmus. Hierzu zählen insbesondere das Empfangen, das Weiterleiten und die Steuerungsmöglichkeiten über das Frontend.

Neben diesen konkret benennbaren Funktionalitäten und Objekten gibt es einige Hilfsfunktionen, welche implementiert werden müssen, damit ein Zusammenspiel der Simulationsumgebung mit dem Algorithmus möglich ist. Der Umfang ist dabei vom Programmierstil des Nutzers abhängig.

Während der Implementation bedarf es regelmäßiger Testläufe, ob der geschriebene Code lauffähig ist und sich nach augenscheinlicher Betrachtung richtig verhält. Hierfür kann die Simulationsumgebung in einem Debugging-Umfeld gestartet werden. Auf diese Weise kann die Ausführung im Detail beobachtet und mitverfolgt werden. Anders als bei der detaillierten Auseinandersetzung mit dem Algorithmus ist deren Implementation und Integration vorwiegend auf der Ebene des Quellcodes statt. Durch die Bereitstellung des vollständigen Quellcodes besteht auch die Möglichkeit Zwischenschritte, welche durch die Simulationsumgebung selbstständig ausgeführt werden, nachzuvollziehen und so einen Gesamtüberblick über die Ausführung zu erhalten.

Zudem kann die Simulationsumgebung losgelöst von einer Debugging-Umgebung ausgeführt werden. Hierzu erfolgt der Start über ein Initialisierungsscript. Danach kann, genauso wie bei der Ausführung innerhalb der Debugging-Umgebung, über den Webbrowser und die Adresse localhost:5000 auf das Interface zur Bedienung zugegriffen werden. Das Interface erlaubt das Erstellen eines neuen Simulators, das hinzufügen von Nodes und das festlegen von Operationen, welche ausgeführt werden sollen. Ist dies geschehen kann ein entsprechender

Node und eine entsprechende Operation ausgewählt und die Ausführung gestartet werden. Die Ausgabe der Ergebnisse des Ausführungsschritts erfolgt im Trace am Ende des Interfaces. Dabei wird eine Vorstufe des Formats eingehalten, dass im Kapitel 5.2 detailliert erklärt und beschrieben wird.

Um zu sehen, wie die Ausführung mit der Simulationsumgebung erfolgen kann, wird im Folgenden ein Beispiel ausgeführt. Dieses Beispiel stammt aus dem Papier von Sun und Ellis [4] und wurde bereits in Kapitel 2.1 als Gegenbeispiel für den Algorithmus von Ellis und Gibbs [1] verwendet.

Walkthrough

Im ersten Schritt wird nach dem Starten der Simulationsumgebung ein neuer Simulator erstellt. Das Interface zur Steuerung ist nach dem Initialisieren über eine eindeutige Adresse erreichbar. Der Simulator verfügt nach der Erstellung automatisch über einen Netzwerk-Node mit der ID 0. Über diesen wird die gesamte Kommunikation aller Nodes geleitet. Als nächstes wird die Anzahl von Nodes erstellt, die für die Erprobung gebraucht werden. Im konkreten Beispiel sind dies drei Client-Nodes.

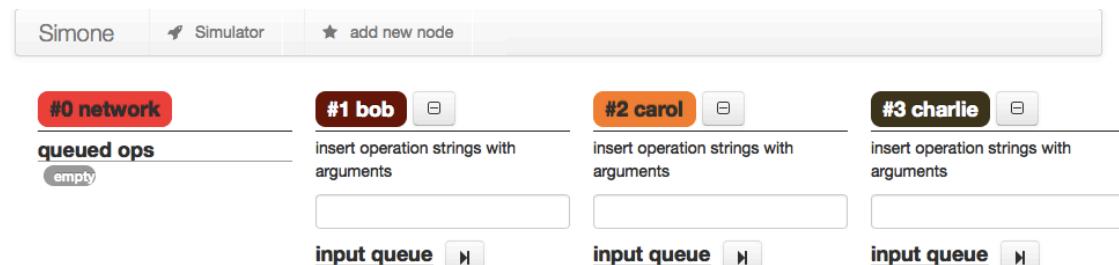


Abbildung 14 Ansicht des Frontends nach dem erstellen der Nodes

Jeder Node hat eine ID, einen Namen sowie eine eindeutige Farbe. Dies dient der einfachen Erkennung und Unterscheidung, insbesondere dann, wenn viele Nodes im Simulator gleichzeitig ausgeführt werden. Beim darauffolgenden Schritt werden den jeweiligen Nodes die Operationen übergeben, welche ausgeführt werden sollen. Die Reihenfolge kann beliebig gewählt und im Nachhinein noch verändert werden. Es gibt die

Operation „insert“ und „delete“. Das Einfügen des Zeichens „b“ an der

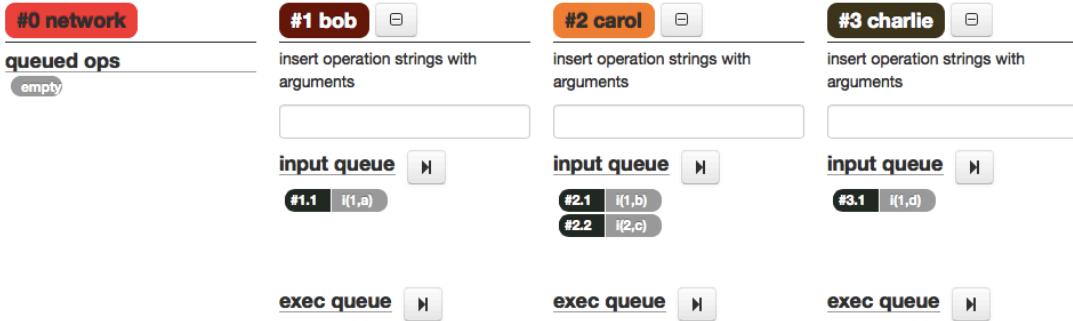


Abbildung 15 Frontend nach der Eingabe der ersten Operationsanweisungen

Position 1 ist die Anweisung $i(1,b)$ – auch Eingaben mit mehreren Zeichen sind möglich. Das Löschen von zwei Zeichen ab der Position 3 im Text ist über die Anweisung $d(3,2)$ möglich. Mehrere Anweisungen können, über die Nutzung des Semikolons als Trennzeichen, gleichzeitig eingegeben werden. Das Ausführen einer Operation ist ein zweiteiliger Schritt.

Im ersten verlässt die Operationsanweisung zum einen die input-queue und wird in eine Operation überführt, welche in der exec-Queue gespeichert wird. Die input-Queue dient allein der Aufbewahrung der vorbereiteten Operations-Anweisungen wohingegen die exec-Queue die Operationen aufnimmt. Gleichzeitig werden Nachrichten an den Netzwerk-Node übergeben, die dort auf die Weiterleitung an die anderen Nodes warten. Im zweiten Schritt erfolgt die Ausführung der Operation in der exec-Queue.

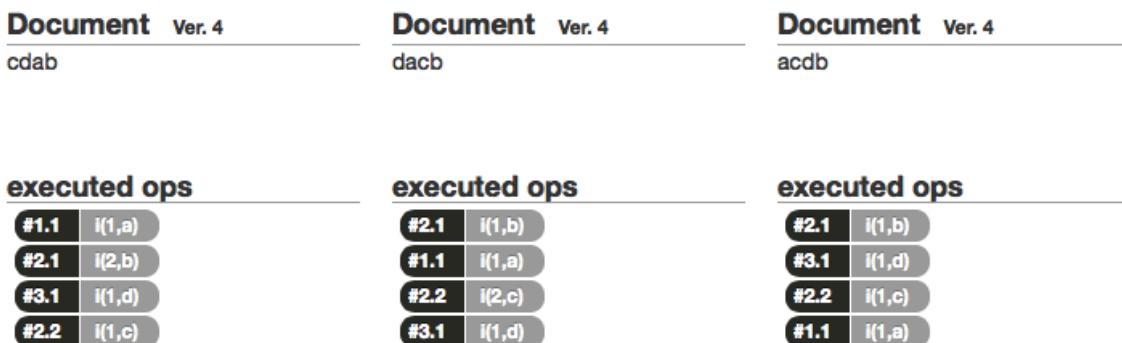


Abbildung 16 Zusammengesetzte Darstellung der einzelnen Client-Nodes, des jeweiligen Dokumentenmodells und der Liste der ausgeführten Operationen

Nachfolgenden hat der Nutzer die Freiheit zu entscheiden, wie er fortfahren möchte. Dies ist vor allem von der Ausführungsanordnung

abhängig und wird durch sie bestimmt. In der Abbildung 1, welche auf Seite 7 zu sehen ist, wird dabei der zeitliche Verlauf und die daraus resultierende Reihenfolge der Operationen für das konkreten Beispiel abgebildet. Wie im Interface zu sehen, ist kann auf jeden Schritt einzeln Einfluss genommen werden. Während der Ausführung der Operationen wird Schrittweise die Darstellung des Dokumentenmodells und die Anzeige der bereits ausgeführten Operationen aktualisiert. Die Ansicht des Dokumentenmodells offenbart den aktuellen Zustand des selbigen. Nach der Ausführung der Operation wird die Änderung kurzzeitig hervorgehoben. Darüber hinaus werden die Operationen in der letztendlich ausgeführten, das heißt den Transformationsprozess durchlaufenen, Version angezeigt.

Nach der Ausführung aller Operationen kann das Ergebnis an drei Stellen abgelesen werden. Zum einen am aktuellen Status des Dokumentenmodells. Für jeden Node ist dort ablesbar, welcher der letzte Status ist, ob sich der aller Nodes gleicht oder wo Unterschiede bestehen. Als zweites können die ausgeführten Operationen und ihre Werte betrachtet werden. Diese werden in der Reihenfolge ihrer Ausführung aufgelistet. Als letztes wird der Trace ausgegeben, in dem alle Aktionen, die im Simulator ausgeführt wurden, aufgelistet sind.

```

24. exec_op:{'op_id': '3.1', 'node_id': 2}
23. exec_op:{'op_id': '2.2', 'node_id': 1}
22. exec_op:{'op_id': '2.2', 'node_id': 1}
21. exec_op:{'op_id': '2.2', 'node_id': 1}
20. exec_op:{'op_id': '2.2', 'node_id': 1}
19. forward_op:{'receiver_id': 2, 'op_str': u'i(1,d)', 'sender_id': 3, 'op_id': '3.1'}
18. forward_op:{'receiver_id': 3, 'op_str': u'i(1,a)', 'sender_id': 1, 'op_id': '1.1'}
17 forward op:{'receiver id': 3 'op str': u'i(1 a)' 'sender id': 1 'op id': '1.1'}

```

Abbildung 17 Ausschnitt des Traces, wie er im Frontend ausgegeben wird

Im Ergebnis des konkreten Beispiels wird beim Vergleich des letzten Status des jeweiligen Dokumentenmodells ersichtlich, dass sich alle endgültigen Status in der Version 4 zueinander unterscheiden. Dies führt zu dem Schluss, dass die Transformation nicht korrekt gearbeitet hat. Mittels einer erneuten Ausführung und einer detaillierten schrittweisen Beobachtung könnte mit Hilfe der Simulationsumgebung nun nachvollzogen werden, an welcher Stelle der Ausführung das Fehlverhalten erzeugt wird.

Beispiel

Im Folgenden soll anhand eines Beispiels demonstriert werden, dass bereits die Verwendung des Grundgerüsts von *Simone* wesentliche Vorteile gegenüber einem alternativen und in Kapitel 7.2 beispielhaft beschriebenen Vorgehen bietet.

Als Fallbeispiel wird ein Beispiel aus dem mehrfach zitierten Papier von Ellis und Gibbs ([1], S. 405) verwendet. „Suppose that initially the site object is the null string at all sites and that site 1 initiates operation insert[x;1]; site 2 initiates insert[y;1]; and site 3, insert[z;1]. [...] The final string is xyz.“

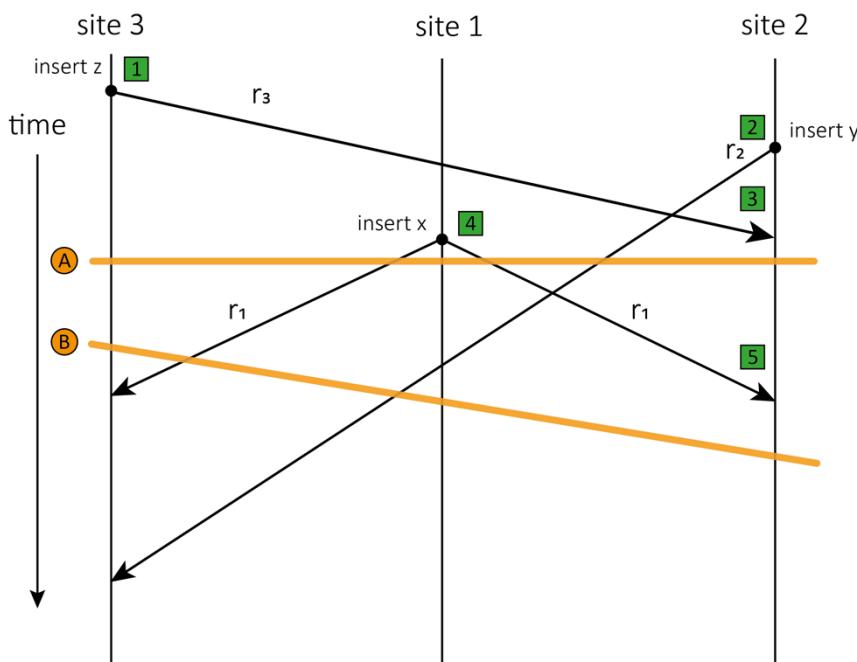


Abbildung 18 Beispiel aus dem Papier von Ellis und Gibbs; zudem eingezeichnet die Ausführungsreihenfolge der Nodes (grüne Quadrate) und die beiden Zuständen A und B

Um dieses Beispiel nachzuvollziehen zu können muss zunächst eine Simulationsinstanz aus 3 Nodes erstellt und müssen die drei Operationen angelegt werden. Danach können die einzelnen Schritte, wie im Messages Sequenzdiagramm (Abbildung 18) zu sehen, ausgeführt werden. Die Reihenfolge ist dabei in den grünen Quadranten hervorgehoben. Werden Schritt 1 bis 4 ausgeführt, so ergibt sich der Zustand des Simulators. An der Position A ist er in der Abbildung 18 durch eine horizontale Line markiert. Auf einen Blick können so die unterschiedlichen Zustände der Simulationsinstanz gesehen und durch den Nutzer bewertet werden. So

ist auf einen Blick der Inhalt der Queues und des Dokumentenmodells aller Nodes zu sehen. Auch welche Nachrichten sich noch im Netzwerk befinden und welche Operationen noch gar nicht ausgeführt wurden kann einfach gesehen werden, wie Abbildung 19 zeigt.

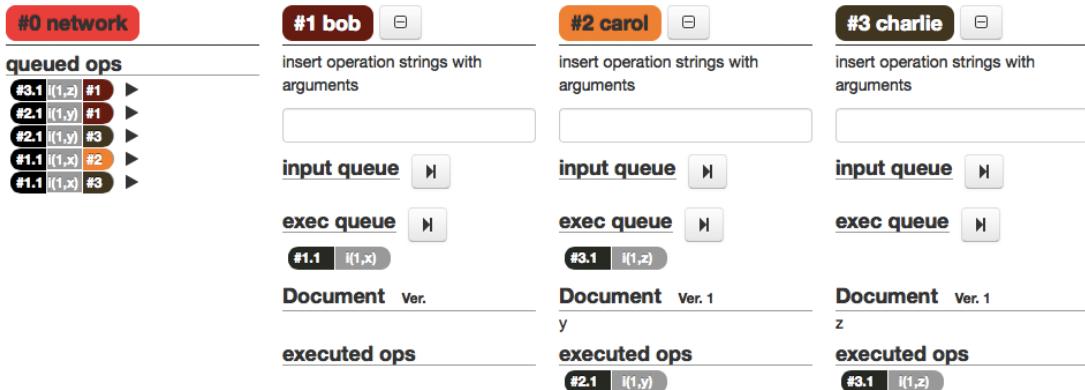


Abbildung 19 Der über das UI einsehbarer Zustand der Simulationsumgebung an Punkt A

Die orangene Linie an Punkt B soll andeuten, dass die Simulation es ermöglicht in jedem Zwischenschritt der Ausführung einen Einblick in den genauen Zustand der Simulation nehmen zu können.

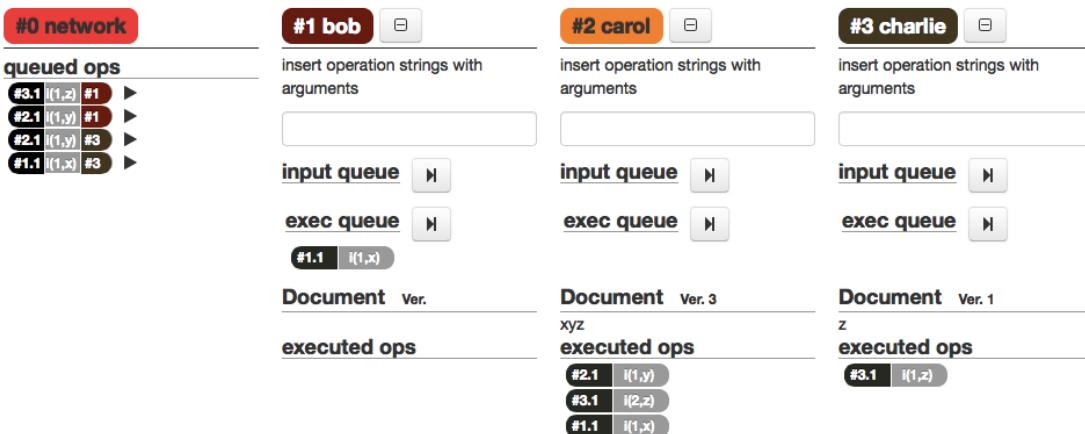


Abbildung 20 Der über das UI einsehbarer Zustand der Simulationsumgebung an Punkt B

Anders als in einem Debugging-Interface bietet das UI der Simulationsumgebung die Möglichkeit, die für den konkreten Algorithmus relevanten Queues von allen Nodes auf einem Blick zu betrachten. Beispielhaft ist dies in Abbildung 21 zu sehen. So kann nachvollzogen werden, welche Operationen in welcher Reihenfolge und mit welchen Werten ausgeführt wurden (executed ops). Auch ist es über den Netzwerk-Node und dessen Queue möglich jederzeit alle Operationen zu sehen, welche sich im Netzwerk befinden.

7.2 Eigenständiger Ansatz

Anders muss vorgegangen werden, soll der Algorithmus ohne die Simulationsumgebung in ein Stadium überführt werden, in dem der Nutzer über die Möglichkeit verfügt, den Algorithmus auszuführen und zu erproben. Bezugnehmend auf die erste Implementierung des Algorithmus im Anfangsstadium der Arbeit, wird im Folgenden von den dort gemachten Erfahrungen zusammenfassend berichtet. Dieses Vorgehen wird als Grundlage für eine im Anschluss erfolgende Auswertung genutzt.

Um den Algorithmus ausführen zu können wurden alle im Papier benannten Teilespekte des Algorithmus in mehrere python-Scripte überführt und Teilweise in Klassen strukturiert. Dabei gab es im Ergebnis eine, zum vorher beschriebenen Aufbau, ähnliche Struktur. Genauer gesagt, ist die hier verwendete Struktur aus der vorherigen hervorgegangen. Die Client-Nodes kommunizierten über einen Server-Node, welche für die Simulation des Netzwerkverhaltens zuständig war. Seine Aufgaben bestanden in der Zwischenspeicherung der gesendeten Operationen und der Weiterleitung der entsprechenden Anweisung vom Scheduler an die jeweiligen Empfänger-Nodes. Die Ausführung der Nodes erfolgte durch einen Scheduler, welcher im Rahmen der Entwicklung geschaffen wurde. Ein Parser, welcher die Eingaben verarbeitete und ihnen entsprechenden Operationen zuordnet, ist ebenfalls direkt in den Client-Nodes angesiedelt.

Die Bedienung des Algorithmus erfolgte über ein weiteres python-Script. Dabei wurden mehrere Nodes erstellt und mit einer Menge an Operationen initialisiert. Diese Nodes wurden an den Scheduler übergeben und mit dem entsprechenden Modus ausgeführt. Durch die Entwicklung bedingt waren nur zwei Ausführungsvarianten möglich: die randomisierte Ausführung und die auf Basis einer vom Nutzer vorgegebenen Ausführungsreihenfolge. Letzterer Modus war explizit geschaffen worden, um eine bestimmte und vom Nutzer vorgegebene Ausführungsreihenfolge abzuarbeiten, um beispielsweise einen bestimmten Fall zu überprüfen. Die Steuerung der Ausführung über die Konsole war durch den großen Umfang zu aufwendig in der Realisierung. Die Ergebnisse der

Ausführung wurden als Ausgabe des Scripts direkt in die Konsole geschrieben und konnten dort betrachtet werden. Dabei wurde ein Dictionary in JSON umgewandelt und auf stdout ausgegeben. Es enthielt jeweils die wesentlichen Variablen des zuletzt ausgeführten Nodes. Mittels der Weiterleitung der Ausgaben in der Konsole konnten diese auch in eine Datei persistiert und später aufgerufen werden. Ebenso war die Ausgabe der Reihenfolge, in welche die Nodes aufgerufen und ihre jeweilig nächste Funktion ausgeführt wurde, möglich.

Im Folgenden wird das gleiche Beispiel, das bereits in der Simulationsumgebung durchgespielt wurde, erneut genutzt und für die Ausführung in Individuallösung verwendet.

Walkthrough

Das Script, das zum Ausführen aufgerufen wird, enthält neben grundlegendem python-Code für den Import weiterer Funktionalitäten die Initialisierung der Nodes.

```
s0 = EllisionGibbsServer(0,"simson")
c1 = EllisonGibbsClient (1, "alice", ['IM(1,a)'], text=" ")
c2 = EllisonGibbsClient (2, "bob", ['IM(1,b)', ' IM(1,c)'],text=" ")
c3 = EllisonGibbsClient (3, "carol", ['IM(1,d)'],text=" ")
```

Diese initialisierten Objekte werden an den Scheduler übergeben, der die Ausführung übernimmt. Nach der Initialisierung des Schedulers erfolgt die Festlegung der Reihenfolge, in welcher die dieser die Nodes ausführt werden soll.

```
samson.exec_list_simulation ( [2,1,0,3,0,0,2,0,0,0,0,0],
continue_randomly=True)
```

Dabei wird nur die Reihenfolge der Node-Aufrufe festgelegt, bis zu welcher die Ausführung von Operationen die Ergebnisse anderer Nodes beeinflussen kann. Konkret bedeutet dies, dass die Reihenfolge, in welcher die Nodes lokale Operationen abarbeiten, spätestens ab dann beliebig sein kann, wenn alle Nodes von allen anderen Nodes alle ausstehenden Nachrichten erhalten haben. Die Abarbeitung innerhalb eines Nodes erfolgt der Queue entsprechend im FIFO-Prinzip. Um also eine Ausführung zu ermöglichen, wurde der Scheduler so initialisiert, dass die Ausführung nach der Abarbeitung der vorgegebenen Reihenfolge

randomisiert fortgesetzt wird. Um dabei einen möglichst kurzen Ausführungspfad zu erzeugen, werden nur Nodes ausgeführt, welche auch noch über auszuführende Operationen in der exec-Queue verfügen. Nachdem die Ausführung vorbereitet ist, kann im nächsten Schritt die selbige gestartet werden.

Die Abarbeitung erfolgt dann wie zuvor konfiguriert. An dieser Stelle kann nicht mehr in der Form von Werteveränderung eingegriffen werden. Für die Verfolgung der Ausführung bieten sich zwei Möglichkeiten. Zum einen die Ausgabe der relevanten Werte direkt über die Konsole. Der Übersicht wegen beschränken sich diese Möglichkeiten jedoch auf einen sehr ausgewählten Bereich. 80 Zeichen in der Breite und ein sich kontinuierlich bewegender Bildschirm bieten wenig Raum, für die Darstellung von Informationen und schlechte Voraussetzung für deren Aufnahme. Der zweite Aspekt, der gegen diese Umsetzung spricht, ist die fehlende Möglichkeit der Unterbrechung des Ausführungsflusses. Eine solche Funktionalität bedürfte der expliziten Implementierung sowohl auf der technischen Ebene als auch auf der Nutzerinteraktion. Dies beides hätte die Komplexität und den Aufwand bei der Umsetzung sehr gesteigert.

Als zweite und in mehrerlei Hinsicht bessere Möglichkeit, einerseits Daten auszugeben und andererseits in die Ausführung einzugreifen, bietet die Debug-Umgebung einer elaborierten IDE. Sie bietet zum einen die Möglichkeit, die interessanten Variablen zu beobachten und die Ausführung Schrittweise fortzuführen, ohne einen weiteren Implementierungsaufwand zu haben. Die Ausgaben, die ohnehin anfallen, können dennoch in der Ausgabe der Konsole nachverfolgt und auf Wunsch exportiert werden.

Die obige Ansicht veranschaulicht, wie eine solche Darstellung aussehen kann. Die Anwendung teilt sich dabei in zwei Bereiche auf. Im oberen Bereich ist der Quellcode mit der hervorgehobenen Zeile zu sehen, an der die Ausführung unterbrochen wurde. Im unteren Bereich werden die Variablen angezeigt, die an genau dem Punkt, an dem die Ausführung angehalten wurde, zur Verfügung stehen. Die Steuerelemente im

Bildmittebereich erlauben die Schrittweise Fortführung der Ausführung.

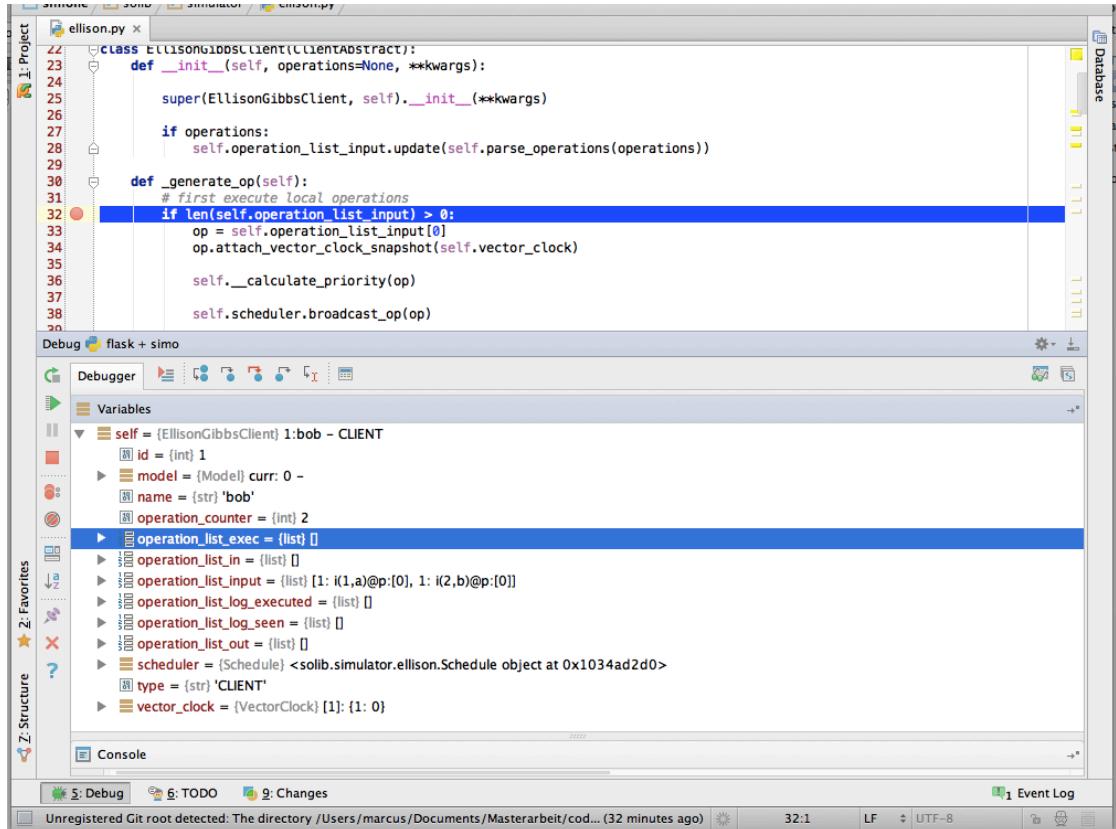


Abbildung 21 Screenshot der IDE mit aktiviertem Debugger

Alternativ wird diese beim nächsten Break Point angehalten. Nach dem Ende der Ausführung leert sich die Detailansicht im Debugger und es werden nur noch der Quellcode und die Ausgaben auf der Konsole angezeigt.

Die Ausführung kommt zu gleichen Ergebnis wie die Ausführung bei der Nutzung der Simulationsumgebung. Dies war nicht anders zu erwarten, handelt es sich ja um den selben Algorithmus. Die entscheidenden Unterschied sind bei der Ausführung auftreten, worauf im Folgenden eingegangen wird.

7.3 Auswertung

Betrachtet man die beiden zuvor beschriebenen Verfahren, so werden die qualitativen Unterschiede schnell offensichtlich. Bietet die Simulationsumgebung dem Nutzer ein speziell auf die Erfordernisse von OT-Algorithmen angepasstes Interface, das er sowohl aktiv in die Ausführung eingreifen als auch nur die selbige passiv verfolgen kann, so bietet dies die

im ersten Schritt geschaffene und allein für den konkreten Algorithmus entwickelte Ausführungsumgebung nicht. Der geschätzte Aufwand in die Umsetzung erweiterter Funktionen für die Steuerung und die Betrachtung stehen nicht im Verhältnis zum Ziel, den Algorithmus in python funktionsfähig umzusetzen. Der im Anschluss entstandene Wunsch, den Algorithmus mit echten Daten zu füllen und bei der Ausführung zu beobachten, führte in der Folge dazu, eine alternative Strategie zu verwenden. Dabei wurden viele Nachteile dieser deutlich erfahrbar, welche in die Konzeption der Simulationsumgebung einflossen und dadurch zu kompensieren versucht wurden.

Zum einen fehlt die Möglichkeit die Ausführungsumgebung starten zu können, ohne bereits zum Start auf die Menge der ausführenden Operationen festgelegt zu sein und diese auch während der Nutzung nicht weiter ergänzen zu können. Dies ist verbunden mit der fehlenden Möglichkeit, in die aktive Ausführungsumgebung eingreifen und den Ablauf flexibel verändern zu können. Diese zwei Punkte lassen sich mit einem dritten immer in Verbindung bringen: es fehlt an der Übersicht über den gesamten Ausführungsvorgang des Algorithmus und seine durch die Ausführung hervorgerufene Veränderungen.

8. Ausblick

In diesem letzten Kapitel soll zunächst ein Gesamtüberblick über die durch die Arbeit behandelten Themenstellungen gegeben werden. Anschließend wird aufbauend auf den abschließenden Ergebnisstand der Arbeit darauf eingegangen, welche Fortsetzungsmöglichkeiten sich daraus ergeben.

Die Arbeit und ihre Ergebnisse betrachtend bieten sich zwei Richtungen, in die weiter gearbeitet werden könnten. Zum einen in die praktisch-implementatorische, zum anderen in die konzeptionelle-experimentelle Richtung.

Im Kapitel 5 und 6 wurden insgesamt elf Konzepte benannt und ausführlich dargelegt. Jedes einzelne für sich verspricht der Simulationsumgebung einen spürbaren Mehrwert im Verwendungsalltag zu geben. Von besonderem Interesse kann das Erstellen von Snapshots sein. Es ermöglicht ohne den Aufbau auf anderer Konzepte eine viel agilere Bedienung der Simulationsumgebung. Die Möglichkeit, schneller und einfacher unterschiedliche Szenarien auszuprobieren zu können ist sehr vielversprechend.

Zwei ebenso interessante, mit wenig auf und großem Mehrwert umzusetzende Konzepte sind die der Traces und des Highlightings. Sie würden die Nutzbarkeit des Frontends noch einmal deutlich verbessern und den Nutzer eine agilere Arbeit ermöglichen.

In der zweiten angesprochenen Richtung ist die Untersuchung, welche Anpassungen der Simulationsumgebung notwendig sind, um den universellen Charakter nachhaltig umzusetzen. Die momentane Umsetzung der Simulationsumgebung ist so gestaltet, dass sie versucht unterschiedlichsten Ansprüchen durch Algorithmen gerecht zu werden. Überprüft und auf ein ausgearbeitetes Konzept gestellt ist dies jedoch noch nicht.

I. Appendix

A. Verwendete Techniken

Die nachfolgende Übersicht fasst kurz zusammen, welche Techniken zur Anwendung kamen und warum dies geschah. Detaillierte Informationen finden sich auf den dazu angegebenen Websites der jeweiligen Projekte.

Python und flask

Für die Umsetzung des Backends wurde die Programmiersprache python in der Version 2.7 verwendet. Als Programmiersprache eignet sich python für eine schnelle Entwicklung durch seine einfache Handhabung bei der Entwicklung und den großen Umfang an bereits vorhandenen Bibliotheken für unterschiedlichste Zwecke. Das Ausführen von externen Shell-Skripten ist ebenso einfach möglich wie das Einbinden von externe python-Bibliotheken dritter Entwickler. Auch in der Art der Programmierung erlaubt python einen weniger strengen Programmierstil als beispielsweise die Programmiersprache C. python bietet viele Voraussetzungen für ein agileres Programmieren und kurze Entwicklungszyklen.

Mit flask [17] wird auf ein Framework zurückgegriffen, welches bei der Entwicklung von Webanwendungen sehr viele Funktionalitäten bereitstellt, die immer wieder benötigt werden. So sind ein URL-Handling für den Umgang mit parametrisierten URLs und ein Webserver, um das Frontend an den Webbrower auszuliefern, bereits im Framework enthalten. Weiterhin bietet flask durch eine Erweiterung die Template-Engine jinja2 [18] an. Sie ermöglicht den einfachen und effizienten Umfang mit HTML-Templates, die für die Erstellung des Frontends benötigt werden.

WebSocket und gunicorn

Die Kommunikation zwischen Frontend und Backend erfolgt auf der Basis von WebSockets [19]. Der spezifizierte Standard ermöglicht, anders als normale HTTP-Verbindungen, eine duplex-Verbindung zwischen Frontend und Backend. So kann Letzteres selbst initiiert Daten an das Frontend schicken, wenn neue Daten durch die Verarbeitung vorliegen,

ohne dass der Nutzer diese direkte anfordern muss. Bei HTTP-Verbindungen, welche üblicherweise für Webprojekte verwendet werden, erfolgt der Datenversand nur, wenn der Client explizit diese anfragt. Alternativ zu Websockets bestünde auch die Möglichkeit über eine AJAX-Verbindung in Kombination mit dem Long-Polling-Verfahren eine duplex-Verbindung zu emulieren. Dabei wird eine HTTP-Verbindung solange aktiv offen gehalten, bis das Backend Daten an das Frontend senden will. Dies ist jedoch vielmehr ein Hack der Technologie als ein beabsichtigtes Feature, weshalb auf Websockets zurückgegriffen wurde. Diese Technik ist explizit für das beschriebene Nutzungsszenario entwickelt worden.

Für die Umsetzung der WebSocket-Komponente wurde auf die flask-Erweiterung flask-sockets [20] zurückgegriffen. In Kombination mit dem Webserver gunicorn [21] erweitert sie flask mit der Möglichkeit auf Basis von WebSocket zu kommunizieren. gunicorn kommt als leistungsfähige Alternative zum von flask selbst angebotenen Webserver zum Einsatz. Dies ist der Tatsache geschuldet, dass die Erweiterung flask-sockets gunicorn als Basis voraussetzt.

HTML und CSS

Das HTML-Grundgerüst wurde mit der Template-Sprache Jinja2 erstellt. Diese ermöglicht durch Verschachtelung und Vererbung, dass unterschiedliche Teile des Templates unabhängig voneinander erstellt und erst bei ihrer Verwendung zusammengeführt werden. Durch das Verwenden von Platzhaltern, Schleifendurchläufen und bedingter Ausführung ermöglicht es auch komplexere Zusammenhänge auf Template-Ebene zu behandeln und diese aus dem python-Code herauszuhalten. Dies ist eine Voraussetzung für eine saubere Trennung von Quellcode und Design.

Die Grundlage für die interaktive Gestaltung des Frontends basiert auf der technischen Ebene auf Zweierlei. Zum einen die Nutzung von Bootstrap [22], was ein sehr verbreitetes und weitentwickeltes Template-Framework ist. Es setzt sich aus drei Teilen zusammen: dem Cascading Style Sheets für die Strukturierung des Seitenaufbaus, dem sie für die gestalterische Umsetzung der sonstigen Elemente wie Buttons, Eingabefelder oder anderer Elemente, und dem für die interaktiver Funktionen

notwendigen JavaScript. Das Zusammenspiel dieser drei Komponenten erlaubt mit Hilfe von wenig HTML-Deklarationen ein UI zu erstellen, das komplexe Aufgaben erfüllt.

Bootstrap setzt mit Hilfe der drei Komponenten grundlegende UI-Designpattern um. Durch die Kapselung und gute Strukturierung in wenige CSS-Klassen wird die Verwendung sehr einfach gestaltet. Der Rückgriff auf bestehende UI-Designpattern hilft dabei, den Einarbeitungsaufwand in eine neue Anwendung für den Nutzer zu senken. Gleichzeitig wird durch den Rückgriff auf solche bestehenden Musterlösungen die Entwicklung erleichtert und der Aufwand bei der Implementierung massiv verringert.

JavaScript

JavaScript kommt nur im Frontend zum Einsatz. Dort wird es verwendet, um ein interaktives UI zu ermöglichen. Aufgebaut wird dabei auf jQuery [23] und insbesondere seine Erweiterung jQuery-UI [24]. Beide bieten eine Vielzahl an interaktiven UI-Designpattern an, welche eine schnelle Entwicklung sehr vereinfachen und die Bedienung unterstützen. Die beiden JavaScript-Bibliotheken bieten einen großen und elaborieren Funktionsumfang, der bei der Entwicklung und auch bei einer späteren Erweiterung des interaktiven Interfaces sehr nützlich ist. Während der Entwicklung kommt eine nicht komprimierte Version der jeweiligen Bibliothek zum Einsatz. Dies ermöglicht ein Debugging in dieser besonderen Phase der Software.

II. Literaturverzeichnis

- [1] Clarence Ellis and Simon J Gibbs, "Concurrency Control in Groupware Systems," in *SIGMOD*, Austin, Texas, 1989, pp. 399-407.
- [2] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping, "High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System," in *Proceedings of the 8th annual ACM symposium on User interface and software technology*, New York, NY, USA, 1995, pp. 111-120.
- [3] Ressel Matthias, Doris Nitsche-Ruhland, and Rul Gunzenhäuser, "An integrating, transformation-oriented approach to concurrency control and undo in group editors," in *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, 1996, pp. 288-297.
- [4] Sun Chengzheng and Ellis Skip Clarence, "Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements," in *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, New York, NY, USA, 1998, pp. 59-68.
- [5] The Etherpad Foundation. (2008, Januar) Etherpad. [Online].
<http://etherpad.org>
- [6] Google Inc. (2009, Januar) Apache Wave. [Online].
<http://incubator.apache.org/wave/>
- [7] Aurel Randolph, Hanifa Boucheneb, Abdessamad Imine, and Alejandro Quintero, "On Consistency of Operational Transformation Approach," in *Infinity'12*, 2013, pp. 45-59.
- [8] Philip A. Bernstein and Nathan Goodman, "Multiversion Concurrency Control-Theory and Algorithms ,," in *ACM Transactions on Database Systems*, vol. Vol. 8 No. 4, 1983, pp. 465-483.
- [9] Stewart Robinson, *Simulation – The practice of model development and use*, 1st ed. Chichester, West Sussex, England: John Wiley & Sons Inc., 2004, vol. 1.
- [10] Wolfram Research. (2014, März) Wolfram Mathematica: Technical Computing Software. [Online].
<http://www.wolfram.com/mathematica/>
- [11] Alfred Hartmann and Herb Schwetman, "Discrete-Event Simulation of Computer and Communication Systems," in *Handbook of Simulations*. Atlanta, Georgia, USA: John Wiley & Sons, Inc., 1998, pp. 659-676.

- [12] Google Inc. (2014, März) Google Docs. [Online].
<https://docs.google.com>
- [13] The Etherpad Foundation. (2014, März) etherpad.org. [Online].
<http://etherpad.org>
- [14] Joseph Gentle. (2014, März) sharejs.org. [Online]. <http://sharejs.org>
- [15] Joyent Inc. (2014, März) node.js. [Online]. <http://nodejs.org>
- [16] AppJet, Inc.; Etherpad Foundation. (2011, März) Easysync Protocol. [Online]. <https://github.com/ether/etherpad-lite/blob/e2ce9dcc02e2790a34d76ca148f965494bb5055e/doc/easysync/easysync-notes.pdf>
- [17] Armin Ronacher. (2013, März) Flask (A Python Microframework). [Online]. <http://flask.pocoo.org>
- [18] Armin Ronacher. (2013, März) Jinja2 (A Python Template Engine). [Online]. <http://jinja.pocoo.org/>
- [19] I. Fette and A. Melnikov, "The WebSocket Protocol," Internet Engineering Task Force (IETF) , PROPOSED STANDARD RFC 6455, 2011.
- [20] kennethreitz, mbildner, and aaugustin. (2013, März) flask-socket on github. [Online]. <https://github.com/kennethreitz/flask-sockets>
- [21] (2013, März) Gunicorn - Python WSGI HTTP Server for UNIX. [Online]. <http://gunicorn.org>
- [22] Mark Otto, Jacob Thornton, Chris, Thilo, Julia Rebert, and XhmikosR. (2014, Februar) Bootstrap. Bibliothek.
- [23] The jQuery Foundation. (2014, Februar) jQuery - write less, do more. Bibliothek.
- [24] The jQuery Foundation. (2014, Februar) jQuery user interface. Bibliothek.

III. Abbildungsverzeichnis

Abbildung 1 Beispielszenario, in dem es dem Algorithmus nicht möglich ist zu einem Ergebnis zu gelangen, dass alle geforderten Eigenschaften erfüllt. [4]	7
Abbildung 2 Strukturelle Gliederung des Ansatzes	12
Abbildung 3 Vergleichstabelle von Ansätzen für eine Simulation – Robinson [9] S. 42	16
Abbildung 4 Auf der linken Seiten ist die Gliederung des Grundgerüsts der Simulationsumgebung hervorgehoben	18
Abbildung 5 symbolische Darstellung eines ausgeführten Servers mit mehreren Instanzen	23
Abbildung 6 Screenshot des Frontends eines Simulators mit vier Nodes, bestehend aus drei Clients und einem Network-Node	28
Abbildung 7 Übersicht der im Folgenden behandelten Konzepte mit dem Fokus auf dem UI	30
Abbildung 8 (1) dient zum Erzeugen neuer Nodes manuell eingegebener Presets, (2) dient zum Erzeugen neuer Nodes auf Basis des aktuellen Zustands vorhandener Nodes, (3) zum Erzeugen von Nodes anhand eines vorhandenen Presets.	34
Abbildung 9 Mockup der Bedienelemente. 1 zeigt dabei den Zustand, in dem die semiautomatische Ausführung deaktiviert ist, 2 den Zustand, in dem die semiautomatische Ausführung mit einem Takt von 3 Sekunden Dauer ausgeführt wird	43
Abbildung 10 konzeptionelle Darstellung der Interfacekomponenten, welche die Wiedergabe der Ereignisse auf der Timeline ermöglichen. (1) Timeline mit schematisch dargestellten Ereignissen, einem Slider sowie einem Navigationselement zum Navigieren auf der Ansicht. (2) Steuerelement für die Wiedergabe und die Anzeigedauer.	50
Abbildung 11 Übersicht der im Folgenden behandelten Konzepte mit dem Fokus auf dem Backend	52
Abbildung 12 Schematische Darstellung des ersten möglichen Wegs.	55
Abbildung 13 Schematische Darstellung des zweiten möglichen Wegs	56
Abbildung 14 Ansicht des Frontends nach dem erstellen der Nodes	76
Abbildung 15 Frontend nach der Eingabe der ersten Operationsanweisungen	77
Abbildung 16 Zusammengesetzte Darstellung der einzelnen Client-Nodes, des jeweiligen Dokumentenmodells und der Liste der ausgeführten Operationen	77
Abbildung 17 Ausschnitt des Traces, wie er im Frontend ausgegeben wird	78
Abbildung 18 Beispiel aus dem Papier von Ellis und Gibbs; zudem eingezzeichnet die Ausführungsreihenfolge der Nodes (grüne Quadrate) und die beiden Zuständen A und B	79
Abbildung 19 Der über das UI einsehbarer Zustand der Simulationsumgebung an Punkt A	80
Abbildung 20 Der über das UI einsehbarer Zustand der Simulationsumgebung an Punkt B	80
Abbildung 21 Screenshot der IDE mit aktiviertem Debugger	84

Selbstständigkeitserklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

Marcus Sümnick

Rostock, den 21. April 2014