Integration of a Java Physics Framework in a Virtual World

Techniques Applied for Redesigning an Educational 3D Software to Run in Distributed Environments

Master's Thesis
at
Graz University of Technology

submitted by

Christian Schratter

Supervisor: Dipl.-Ing. Dr. techn. Christian Gütl

Institute for Information Systems and Computer Media (IICM)
Graz University of Technology
A-8010 Graz, Austria

Co-Supervisor: Professor John W. Belcher

Department of Physics and Center for Educational Computing Initiatives (CECI)
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

TU Graz
Graz University of Technology

Massachusetts Institute of Technology

November 4th, 2013

Integration eines Java Physik-Baukastens in einer Virtuellen Welt

Angewandte Techniken um eine 3D Lernsoftware für Verteilte Umgebungen zu Adaptieren

Masterarbeit
an der
Technischen Universität Graz

vorgelegt von

Christian Schratter

Betreuer: Univ.-Doz. Dipl.-Ing. Dr. techn. Christian Gütl

Institut für Informationssysteme und Computer Medien (IICM)
Technische Universität Graz
A-8010 Graz, Österreich

Co-Betreuer: Professor John W. Belcher

Department of Physics und Center for Educational Computing Initiatives (CECI)
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

04. November 2013

Abstract

E-learning is a technique on the rise, especially in tertiary education. Nowadays, most universities offer their students electronic tools to enable or support their learning tasks. The Massachusetts Institute of Technology (MIT), which is well known for its continuing efforts in this area within their OpenCourseWare project, is constantly increasing its e-learning portfolio, with its latest joint project, edX, being one more step towards the goal of making the university's complete course catalog available in virtual form. This development provided the initial impetus for the present master's thesis, while a further concept developed at MIT, Technology Enabled Active Learning (TEAL), provided additional motivation. In short, physics professors at MIT use TEAL to change lectures from a recitation to an interactive format, in which students collaborate in smaller groups. Initially, a third incentive existed as well: employing virtual worlds as host environment for educational setups. However, as outlined in the course of this thesis, a substantial amount of additional work would be required to properly embed the TEAL learning tools in the virtual world, for which reason this aspect was eventually de-emphasized in the present work.

Starting from these three motivational factors, the question emerged of how existing e-learning tools could be further developed to support educational efforts. To this end, a concept was developed which enables one of the TEAL learning aids – the TEAL Simulation Framework (TEALsim) – to run in distributed environments. The guiding principles for the design were scalability, flexibility and consistency. To achieve these design goals, TEALsim was converted to a client-server architecture. In this process, many different areas of the framework had to be adapted. The most important of these changes, such as the new characteristics of the simulation engine, are outlined in this document to support the inline documentation of the source code. These explanations usually include a preceding analyses of the issues encountered that led to the changes, along with subsequent summaries of the designs implemented.

Ultimately, most of the chapters herein revolve around the topic of how to ensure the synchronization of the simulation's calculations, as well as its visual representation. This requires a comprehensive network package, the essentials of which are discussed in multiple chapters here, including definitions of the expected system behavior, an analysis of how the network layer is integrated in TEALsim, and the documentation of the exchangeable connection system. In this context, the mechanics of the TEALsim OpenWonderland module are also outlined to provide a reference point for other external projects seeking to utilize the TEALsim framework.

The final section of this thesis offers a brief summary of the personal experiences gained in the course of this thesis and also provides a list of suggested future projects that would need to be completed in order to develop the prototype implemented here into a releasable software product.

Kurzfassung

E-Learning ist eine Methode die speziell im tertiären Bildungssektor zunehmend an Bedeutung gewinnt. Heutzutage bieten die meisten Universitäten ihren Studenten bereits elektronische Hilfsmittel, um die Lehre zu unterstützen oder überhaupt erst zu ermöglichen. Das Massachusetts Institute of Technology (MIT) ist in dieser Hinsicht schon länger bekannt durch sein OpenCourseWare Projekt und ständig am Erweitern seines E-Learning Angebots. Eines seiner letzten Projekte, edX, ist ein weiterer Schritt das gesamte Studienangebot in Form von Virtual Education verfügbar zu machen. Diese Entwicklung stellt eine der Hauptmotivationen zur Durchführung dieser Masterarbeit dar. Ein weiterer Anreiz ist das ebenfalls am MIT entwickelte „Technology Enabled Active Learning“ (TEAL) Konzept. Kurz zusammengefasst handelt es sich dabei um die Idee, Vorlesungen mit Frontalvortrag durch eine interaktive Variante zu ersetzen, bei der Studenten in Kleingruppen zusammenarbeiten. Der ursprünglich dritte Anreiz für diese Arbeit bestand im Bestreben virtuelle Welten als Lernumgebung einzusetzen. Dieser Ansatz verlor, wie im Laufe dieser schriftlichen Arbeit genauer dargelegt, jedoch an Bedeutung, da es eines beträchtlichen Aufwandes bedurft hätte die elektronischen Hilfsmittel von TEAL adäquat in einer virtuellen Welt zu integrieren.

Ausgehend von diesen drei Motivationsfaktoren stellte sich die Frage wie verfügbare E-Learning-Werkzeuge noch besser für die Lehre eingesetzt werden können. Aus diesem Grund wurde eines der TEAL Hilfsmittel – das TEAL Simulation Famework (TEALsim) – angepasst um in verteilten Systemen zu laufen. Der Leitgedanke des neuen Designs basierte dabei auf den Aspekten Skalierbarkeit, Flexibilität als auch Konsistenz. Um diese Ziele zu erreichen wurde TEALsim mit einer Client-Server-Architektur erweitert, was umfangreiche Änderungen in vielen Bereichen des Frameworks erforderte. Die wichtigsten dieser Änderungen, wie zum Beispiel Details des neuen Algorithmus zur Simulationsberechnung, werden in diesem Dokument genauer erläutert. In der Regel umfassen diese Darlegungen eine einleitende Analyse der aufgetretenen Probleme welche die Anpassungen erforderlich machten, gefolgt von einer Zusammenfassung der implementierten Lösung.

Schlussendlich drehen sich die meisten Kapitel um die Frage wie die Synchronisierung der Simulationsberechnungen sowie der visuellen Darstellung in verteilten Systemen erreicht werden kann. Dafür wurde ein umfangreiches Netzwerkpaket entworfen, dessen Details in mehreren Kapiteln beschrieben werden. Diese Dokumentation reicht von der Definition des erwarteten Systemverhaltens über eine Beschreibung der integrierten Netzwerkschicht in TEALsim bis hin zum austauschbaren Connection System. In diesem Zusammenhang werden auch Aspekte des TEALsim OpenWonderland Moduls erläutert, um als Leitfaden zu dienen wie das TEALsim Framework in externen Projekten eingebunden werden kann.

Der abschließende Teil dieser Thesis fasst die persönlichen Erfahrungen zusammen die im Laufe dieser Masterarbeit gesammelt wurden und listet in weiterer Folge Vorschläge auf, welche zusätzlichen Entwicklungsschritte notwendig wären um den verfügbaren Prototypen zu einem in der Praxis einsetzbaren Produkt weiterzuentwickeln.

Affirmations

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

………………………………..   ………………………………..

(Place, Date)   (Signature)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

………………………………..   ………………………………..

(Ort, Datum)   (Unterschrift)

Acknowledgments

First, I would like to thank my supervisor at Graz University of Technology, Christian Gütl, for supporting me with his expertise, input and ideas for how to conduct this thesis so as to produce useful results. His continued belief in me paved the way for an enduring collaboration over the course of the last couple of years, which eventually led him to recommend me to his colleagues at the Massachusetts Institute of Technology (MIT), thereby enabling this university-spanning project.

In addition, I would like to express my gratitude to all the people who worked with me at the Center for Educational Computing Initiatives department at MIT: John Belcher for co-supervising me, Judson Harward and Phil Bailey for all their explanations, input and constructive talks about how to further develop TEALsim, and of course Meg, Kirky, Maria, Jim and Mark for everything they did to make my stay in the United States as enjoyable and productive as possible.

Furthermore, I would like to thank the following institutions for funding this thesis, in particular with regard to my extended stay in the USA: the Marshall Plan Foundation[1] for granting a 'Marshall Plan Scholarship', the Industriellenvereinigung Kärnten[2] for granting an 'Exzellenzstipendium', the Internet Foundation Austria for granting a scholarship in the context of the call for the 'Netidee 2012'[3], the Verband selbstständig Wirtschaftstreibender Kärntens for granting an 'Auslandsstipendium', Dr. Josef Martinz for granting a scholarship for studies in foreign countries, and the Faculty of Informatics[4] at Graz University of Technology for granting a 'Förderungsstipendium'.

Last but not least, I would like to take this opportunity to thank my family for supporting my studies at every point in time and in every way possible, which has allowed me to have a wide variety of experiences and a great deal of enjoyment during my time as a student.

Christian Schratter

Graz, Austria, October 2013

[1] Marshall Plan Foundation – http://www.marshallplan.at
[2] IV Kärnten – http://www.iv-kaernten.at
[3] Netidee 2012 by Internet Foundation Austria – http://www.netidee.at
[4] Faculty of Informatics – http://www.dinf.tugraz.at/

Contents

Chapter 1

Introduction

The starting point for this thesis was the results obtained by Scheucher [2010] and Berger [2012]. Their work was primarily based on the TEAL Simulation Framework[5] (TEALsim), as well as an open source framework for 3D virtual worlds called Open Wonderland[6] (OWL). The official definition of TEALsim provides the most succinct explanation of its purpose: "*The TEAL Simulation Framework is an environment for creating, presenting, and controlling simulations that represent physical and mathematical concepts*" [Center for Educational Computing Initiatives (CECI), 2012]. The vision of OWL is to enable programmers to "*Create dynamic learning environments, collaborative business applications, or interactive, multi-user simulations. Start with a blank slate, or modify an existing world.* […] *As a developer, you can extend any part of the system and add functionality by creating modules, the Wonderland version of plugins.*" [Open Wonderland Foundation, 2012]

1.1 Motivation

In the course of the development of human society within recent decades, a multitude of interesting developments have emerged, such as the ongoing globalization, the increase of women's rights or the gradual transformation into an information society. Cultural changes inevitably give rise to the need for new, innovative solutions, which in turn open up further possibilities to sustainably alter our society into a more just, livable and efficient one. The goal of this research project is to develop exactly such a solution to tackle some of the aforementioned, prevailing social challenges, namely to provide a tool(-box) for distributed computer environments that allows for interactive learning and collaboration between geographically distributed users.

[5] TEALsim project at MIT - http://web.mit.edu/viz/soft/visualizations/tealsim/index.html
[6] Open Wonderland - http://www.openwonderland.org/

Although computers have been used in education for quite some time, due to the lack of affordable and reliable network capabilities in the first era of the 'computer age', initial learning tools and their concepts were usually built upon the idea that individuals were essentially using the corresponding hardware and software in isolation. However, the increasing adoption of the internet in the last decade laid the foundation for the implementation of new systems with a strong emphasis on social interaction. In this context, the goal of this thesis is not to compare the pros and cons of these diverging learning paradigms, but rather to focus solely on the traits of multi-user environments and, in particular, how TEALsim can be used to exploit their potentials.

1.2 Definition of Objectives

After an initial, rudimentary analysis of the status quo of TEALsim and OWL, the following topics were identified as potentially interesting tasks for this thesis:

a) Integrate additional physical simulations of the TEALsim project into OWL. Ideally, the whole set of existing simulations should run within OWL as well.

b) Create new, game-like interactive applications in TEALsim which can be used in OWL (either of a collaborative or competitive nature). One example is the multiplayer version of the electrostatic videogame application in TEALsim, whereby 2 users are required to drag a point charge through a maze (for a single-player version, see 'EM Videogame[7]').

c) Further redesign the TEALsim project to allow for a streamlined interchangeability between the graphical rendering engines jMonkeyEngine[8] (JME) and Java3D.

d) Refine the user interface (UI) of TEALsim. Since the 2-dimensional, Swing-based user interface (the simulation control) is not considered ideally user friendly within the context of a virtual world, alternate approaches should be evaluated. Currently, three-dimensional elements for user inputs are considered as a potentially efficient replacement of the former Swing UI, not least because they are visually native components of a 3D world. This circumstance would allow the controls to meld together perfectly with the visualization of the physical experiments (see figure 1.1 for a screenshot of the prevailing UI at the start of this thesis, and figure 1.2 for a sketch of a revamped 3D UI).

e) In terms of the redesign of the user interface, there is the intention to evaluate and implement features of an intelligent guidance system for ease of use. This broad field of studies ranges from simple guidance mechanisms (e.g. tooltips and hints) to complex variants (e.g. computer controlled in-game avatars that offer advice). Research in this field is especially interesting because it is heavily influenced by user perception. With

[7] EM Videogame in TEALsim - http://web.mit.edu/viz/EM/simulations/TEALsim/TEALsim.jnlp
[8] jMonkeyEngine (JME) - http://www.jmonkeyengine.com/

an awareness of the imperative of undisputed comprehensibility and accessibility of technical system (i.e. to establish a significant user tolerance), this topic would serve to link this thesis with studies of human-computer interaction.

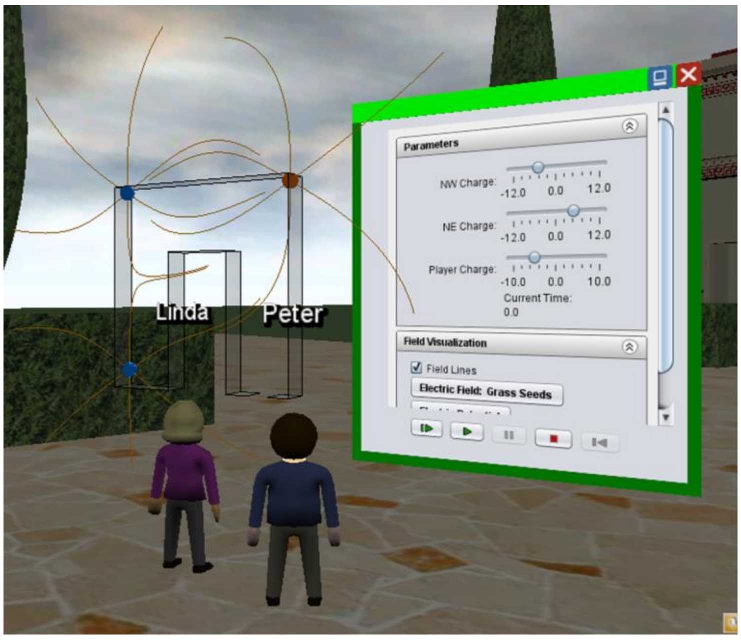


Figure 1.1: Screenshot of an instance of TEALsim running within OWL

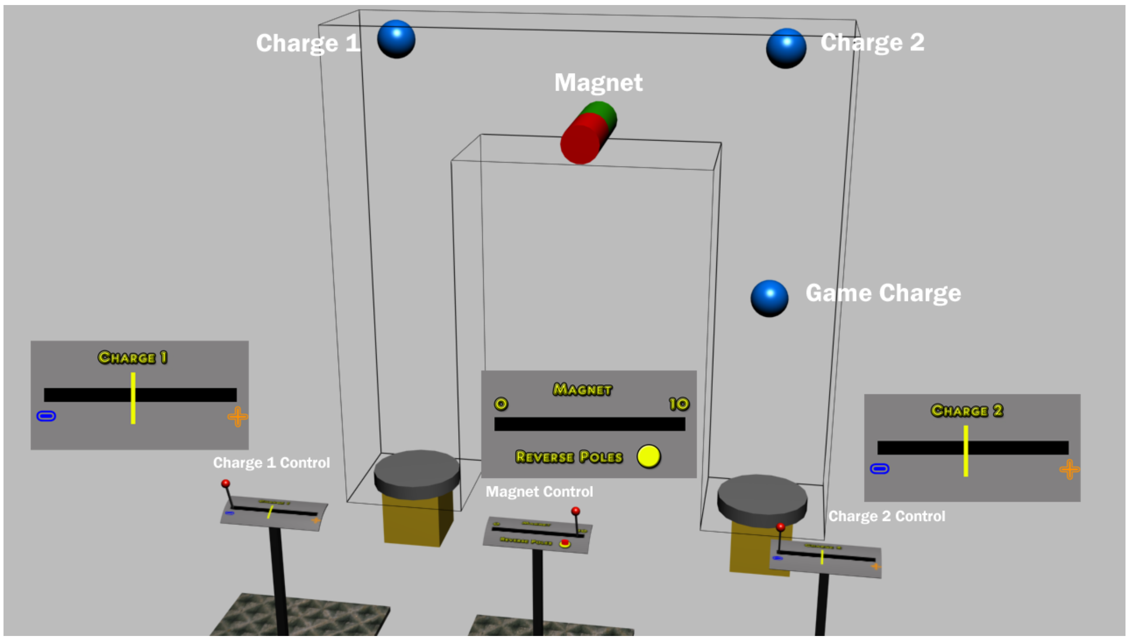


Figure 1.2: Sketch of a potential UI consisting of 3D elements created by Mark Bessette

Constraints of this thesis' objectives

The constraints already outlined by Berger [2012, p. 41] are also relevant for the present objectives, such as:

- The maximum possible number of existing simulations for the stand-alone TEALsim framework should be capable of running in a distributed environment in general, and OWL in particular. In addition, the maximum possible number of simulation functions should be usable in the target environment as well.
- In general, the creation of new simulations in TEALsim should not become more complicated. In particular, the number of effective changes to the application programming interface (API) exposed by TEALsim should be kept as small as possible, in order to reduce the amount of training required for established content creators.
- Alterations of the TEALsim framework should impair neither the existing level of the software design nor the framework's performance characteristics.
- TEALsim should remain executable as both a stand-alone application and an applet in a web browser.

1.3 Structure

This document is structured into four major parts, beginning on a very formal level and then becoming increasingly technical and specific, as the document moves into the practical work performed as a foundation for this thesis.

The first part covers various theoretical topics (e.g. chapter 2 introduces concepts associated with the programming performed for this thesis). Chapter 3 outlines some potential learning scenarios and compares the existing software basis with an alternative design suggestion. Chapters 4 and 5 go on to explain the concepts and requirements for achieving an implicit execution synchronization for software running in a distributed environment.

The second part of this work begins with a summary of organizational adaptations made for the TEALsim project. Subsequently, chapter 7 provides a detailed technical explanation of the integrated network layer and the tools available for user input synchronization. This part ends with chapter 8, which documents the process for running TEALsim as stand-alone application and explains the concept for embedding the framework in a 3rd-party application such as OWL.

The third part of this thesis provides a brief overview of the personal experiences gathered and offers ideas for further research and development.

Finally, the appendix contains all relevant references and listings. In addition, chapter 12 provides an explanation of the notations used in the diagrams, as well as some diagrams that did not fit into a particular chapter of this thesis, but nevertheless might be interesting for follow-up projects.

Part I
General Background

Chapter 2

Terms and Definitions

The main motivation to conduct this thesis was to come up with a suitable software to perform education in an interactive way by means of connected computers. The commonly used description for this kind of education is ‘e-learning’. This introduction already indicates the large range of aspects influencing the objectives of this thesis, reaching from profound engineering sciences all the way to more elusive fields of research like human sciences.

Although in the end a major fraction of the entire work on this thesis has been spent with software design and implementation – or in other words activities of rather technical, delicate nature – the following pages are intentionally kept non-technical and superficial to give a brief overview of some fundamental concepts which are in one or the other way of relevance.

2.1 STEM Education

‘STEM[9] education’ is the placeholder term for education in those fields of science which are considered to be supposedly of particular importance for a countries (economical) success that is, ‘Science, Technology, Engineering, and Mathematics’ [Morella, 2012]. Although the use of the acronym ‘STEM (education)’ to unambiguously refer to this mindset is a rather new phenomenon, the importance of STEM for a nation’s economical prosperity has been in discussion for a much longer period of time. For example in the 1950s economist Robert Solom already identified that “[…] *innovation drives more than 50 percent of future economic growth.*”, where innovation is considered to require background knowledge in STEM fields [Trevey, 2008, p. 34]. Apart from that it is always debatable how to define a nation’s ‘*real*’ success aside from monetary values, especially in the face of recent developments where countries like Bhutan introduced new national performance indicators benchmarking gross national happiness instead of exclusively

[9] STEM equals the term ‚MINT‘ used through the German speaking language area

focusing on financial figures. Nevertheless sciences belonging to these categories are doubtlessly some of the most lucrative [Anger, Geis, & Plünnecke, 2012, S. 15]. Additionally another undeniable circumstance is that nowadays – in our technological world – hardly anyone is able to pursue his main profession and tap his full potentials without profound knowledge in one or more of the STEM fields. Bybee describes the benefits of STEM in this context by outlining that "*A true STEM education should increase students' understanding of how things work and improve their use of technologies.*" [Bybee, 2010, p. 996]

In the context of this master's thesis STEM is of importance because it constitutes the imaginary foundation (or root) for all the other technical and conceptual models involved. One could also say that this thesis was conducted to support the idea to foster STEM education.

2.2 What is TEAL?

TEAL stands for Technology-Enabled Active Learning. This catch phrase was used to describe the corresponding project started at Massachusetts Institute of Technology (MIT). More precisely this project "[…] *is a studio format course designed to accommodate large enrollment in freshman physics at MIT. It is aimed at serving as a model for a new format of undergraduate science courses for large groups of students at MIT and possibly elsewhere.*" [Dori & Belcher, 2005, p. 252]

To briefly summarize the ideas and concepts of the TEAL project: With the aid of modern technologies (e.g. computers for simulations and visualizations) the way physics was taught at MIT should be transformed from its previous passive lecture respectively recitation format to a more collaborative and engaged style where students interact with each other and to some extend truly 'experience' the content of their lecture. Part of this idea was to arrange students in small workgroups for which reason a specific classroom design was envisioned to support this group-oriented approach of teaching [Belcher J. W., 2001]. See figure 2.1 for an early rendering of the proposed classroom design which was later realized at MIT. While the educational results of this new setup seem rather pleasing (see [Dori Y. J., et al., 2003]) one of the few major drawbacks is the relatively high up-front cost to adapt such a special teaching environment [Massachusetts Institute of Technology, 2005][10].

[10] also stated by Christian Gütl during his presentation at iED Europe Summit in Paris 2012 to be approximately 1.5 million US Dollars for the particular classroom (see minute ~8:20 at http://vimeo.com/55862928)

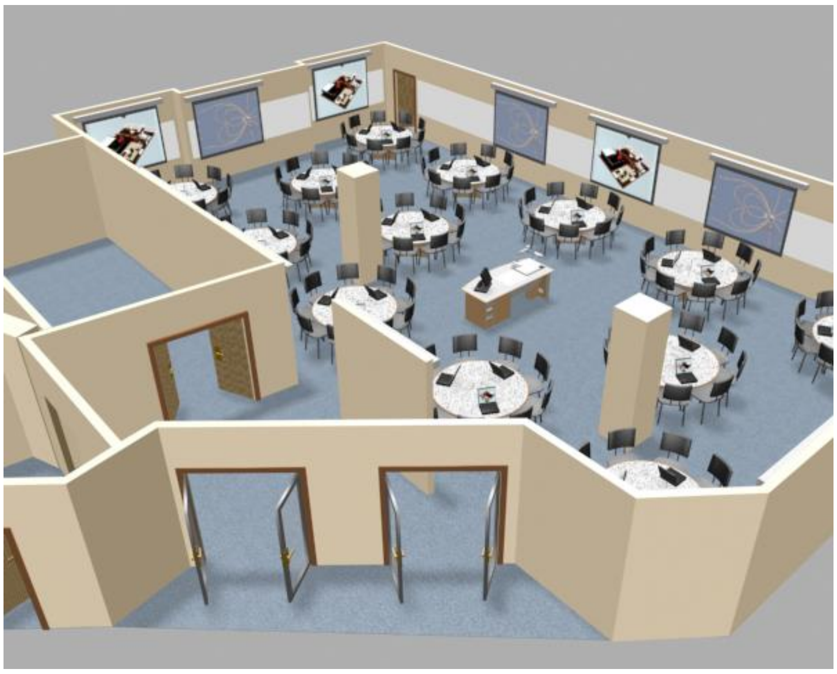


Figure 2.1: Artistic rendering of a classroom suitable for TEAL courses by Mark Bessette[11]

2.3 What is e-Learning

2.3.1 Formal Definitions

As mentioned one of the driving powers to conduct this thesis was the vision to foster e-learning. To define what e-learning actually is we start with its most striking term that is, ‘learning’. However, this is also the point where it already gets difficult since ‘learning’ – although nowadays being an omnipresent term in western civilization’s everyday life – is a very intangible concept for which no universally accepted definition exists [Domjan, 2009, p. 17]. Apart from this dilemma Domjan still tries to summarize the most important aspects of learning by defining it to be “[…] *an enduring change in the mechanisms of behavior involving specific stimuli and/or responses that results from prior experience with those or similar stimuli and responses.*” [Domjan, 2009, p. 17] Subsequently, the logic deduction leads to the self-definition that e-learning is a technical aid to provide exactly such a stimuli respectively response. A different definition for e-learning by Juneidi & Vouros backs up this deduction by asserting that “*E-learning refers to a wide range of applications and processes designed to deliver instruction through computational means.*” [Juneidi & Vouros, 2005], which is semantically equivalent and merely uses a different verbalization. The American Society for Training & Development (ASTD) defines e-learning in a similar but technically more explicit way by stating that “*E-learning (electronic learning): Term covering a*

[11] TEAL project animation specialist Mark Bessette - http://ceci.mit.edu/people/bessette.html

wide set of applications and processes, such as web-based learning, computer-based learning, virtual classrooms, and digital collaboration. It includes the delivery of content via Internet, intranet/extranet (LAN/WAN), audio- and videotape, satellite broadcast, interactive TV, CD-ROM, and more." [American Society for Training & Development, 2013]

Difference between distance learning and e-learning

One important thing to keep in mind when talking about e-learning is the fact that at its core this term refers to something different than 'distance learning'. Even though the term 'e-learning' is related to the use of technical aids like connected computers which are geographically distributed (internet/intranet), the deduction that e-learning and distance learning are synonyms is not the case. The formal definition of distance learning describes it as "[...] *a variety of educational programmes and activities. The major common features are that learner and teacher are physically separate but that deliberate efforts are made by educators to overcome this separation using a variety of media.*" [Unesco, 1987, p. 5] Thus e-learning is rather a specific application of distance learning, which is backed up by the definition of ASTD[12]: "*Distance education: Educational situation in which the instructor and students are separated by time, location, or both. Education or training courses are delivered to remote locations via synchronous or asynchronous means of instruction, including* [...]. *Distance education does not preclude the use of the traditional classroom. The definition of distance education is broader than and entails the definition of e-learning.*" [American Society for Training & Development, 2013]

A compact classification outlining the different characteristics of e-learning and distance learning is given by Mencke & Dumke:

instruction delivery technology	physical separation yes	physical separation no
computational	Distance education & e-Learning	e-Learning
other	Distance education	-

Table 2.1: Scope of distance learning respectively e-learning [Mencke & Dumke, 2007, p. 40]

2.3.2 Different Kinds of e-Learning

From a more abstract point of view differing types of computer-aided education systems can be distinguished by their style of interaction with human users. Soh et al. summarize three different scenarios [Soh, Miller, Blank, & Person, 2004, p. 2]:

a) *"Computer-Assisted Instruction, the system provides drill and practice exercises and tutorial instruction"*

[12] 'distance education' and 'distance learning' are used synonymously in this context

b) *"Computer-Managed Instruction, the system evaluates and stores student performance and guides students to appropriate instructional resources"*
c) *"Computer-Enriched Instruction, the system satisfies student requests such as solving a mathematical equation, generating data, and executing programs"*

From a more technical point of view the utilized technique to 'transmit' the information also allows to specify various different kinds of e-learning. Amongst the categories somewhat related to this thesis, in the document by Mencke & Dumke are examples such as [Mencke & Dumke, 2007, p. 41]:

Web-based teaching (WBT)

By definition web-based teaching is the "*Delivery of educational content via a Web browser over the public Internet, a private intranet, or an extranet. web-based training often provides links to other learning resources such as references, email, bulletin boards, and discussion groups. WBT also may include a facilitator who can provide course guidelines, manage discussion boards, deliver lectures, and so forth.*"[American Society for Training & Development, 2013]

This kind of teaching became very popular within the last years. One of its biggest advantages is its accessibility requiring nothing else but a web browser and internet. Requirements for special knowledge on the client side to engage in the technology can therefore be reduced to a minimum; most of the environment's complexity can be dealt with on the administration-level. In conjunction with the worldwide ever increasing penetration rate of internet connections per capita [Miranda & Lima, 2012] this approach allows to reach a maximum of potential users.

Virtual education

Compared to WBT, virtual education explicitly adds the aspect of interaction between trainer and trainee. For this purpose virtual classrooms ("*The online learning space where students and instructors interact.*" [American Society for Training & Development, 2013]) shall create an atmosphere humans are familiar with from real-world learning experiences. If implemented to run within a web browser, virtual education could be considered as a special type of WBT.

Mobile learning

This term has its emphasis on the mobile nature of the utilized devices to access the educational content. In conjunction with the WBT paradigm this field is of continuously growing interest due to the thriving success of smart phones and tablet computers [Ericsson, 2012]. Usually such mobile devices come with limited computational powers and narrow tolerances to run arbitrary applications, making them an ideal candidate to use for WBT since pretty much all devices come with a web browser nevertheless. However, a particular challenge within this context is the great range of differing application-environments (screen size, operating systems, CPU power, etc.), which is also expressed by Mencken & Dumke by stating that "*The appropriateness of technic, learning content and learning activities need to be taken into account for the application of this e-Learning type.*" [Mencke & Dumke, 2007, p. 44]

Blended learning

This term denotes the technique to combine respectively enrich traditional style lectures and courses with electronic tools. Consequently the benefit of this paradigm is that it allows teachers to gradually adapt their teaching styles and methods to take advantage of modern teaching aids without overburdening them. [Mencke & Dumke, 2007]

Educational games

Like the term 'game worlds' discussed in chapter 2.4 the term 'educational games' is also a somewhat difficult category to define precisely. The notion of gaming – and especially in conjunction with education – is discussed very controversially in literature. Mencke & Dumke for example define an educational game to be "[…] *a computer-based game that motivates and engages the player/learner to learn.*" [Mencke & Dumke, 2007, p. 45] In contrast, the definition of games by Salen & Zimmerman referenced in chapter 2.4 actually does not mention anything regarding the incitements of players to play the game at all.

The problem is that the term 'educational game' is not of pure technical nature – which would make it easier to find an all-purpose definition – but rather an ethical and philosophical concept, hence highly subjective. Under this preposition it becomes clear that for most assertions given in literature one may find contradictory counterexamples.

As an example O'Brien [2010] outlines that 4 specific traits of commercial games "[…] *are either not desirable in an educational game or must be minimized: chance, critical competition, inappropriate material, and advertising.*" [O'Brien, 2010, p. 2] Doubtlessly the reasons given to avoid inappropriate material and advertising seem coherent. However, the two other traits are rather debatable. The justification to avoid chance is based on the observation that when chance impacts a player's possibilities to succeed it would result in unequal (unfair) learning returns for various players. Nevertheless it gets acknowledged that yet chance is "[…] *for many, highly engaging and highly entertaining, we know them* [games involving a certain amount of chance] *to be highly addictive.*" [O'Brien, 2010, p. 2] Now the question arises why someone designing an educational game would not want it to be highly engaging, highly entertaining, and thus highly addictive? The rule of thumb says practice makes perfect. True, not everyone would take away exactly the same experience but one common opinion about education is that it has to ensure that all students meet a certain minimum level of knowledge, neglecting to some extend how many particular individuals truly excel (and if only due to chance). Without doubt there is (yet) no final call to this issue. The next trait – critical competition – is justified by gender equality. Based on the research of Kafai [1996] O'Brien concludes that competitive games are less appealing to girls than boys. Assuming the soundness of this assertion, it still remains situational whether or not competition should be part of an educational game. Potential questions to evaluate in this context are e.g.: Might the game aim for a certain gender intentionally or should it be liked equally by both genders? Does the assumption that boys prefer competition more than girls hold globally for any culture?

An interesting approach to deal with the apparent difficulties to properly classify and evaluate educational games (in this context synonymously referenced to as 'serious games') is proposed by

Jantke [2010]. According to him any game may be seen as a serious game, it only "[…] *depends in many ways on your theoretical point of view and on concepts you rely on.*" [Jantke, 2010, p. 859] He subsumes that research done in the field of educational games is less about finding compact definitions but rather about creating a comprehensive taxonomy which is "[…] *essential to every scientific communication.*" [Jantke, 2010, p. 859] On the basis of an appropriate taxonomy any game's benefit for specific groups of users could be evaluated individually (or in other words one could formulate proper questions to receive the desired answers).

2.4 Different Kinds of Multi-User Environments

Based on the key technology 'internet' a multitude of multi-user environments have been established over the course of the past years. To name a few Usenet, social networks, blogs and virtual worlds would promptly come to one's mind. Even though nowadays some environments may be already considered as (technologically) outdated, at least within very specific niches they still manage to maintain a minimum amount of users. A reason for this situation is that each environment's distinct set of system traits ultimately gives them a competitive edge for particular fields of application explaining their continued use in these areas. Certain advantages might be difficult to identify at first glance whereas other aspects setting apart one from another system are immediately evident such as:

- technical requirements
    - to run the service (e.g. does it require a single, low-end server or contrariwise a full-fledged server farm; is auxiliary, proprietary software needed or not)
    - which pose as an entrance barrier for new users to join and participate in the system (e.g. what kind of client hardware is needed; do clients have to install a special software on their computer; is a reliable, fast and/or permanent internet connection required)
- user permissions for interaction and participation (e.g. do clients have to register; do new registrations have to be approved by humans; who is allowed to add content to the environment and who may only consume it; who may see which information; is every user allowed and able to get in contact with any other user)
- visual representation (e.g. 2D or 3D; in a web browser or in a special client)
- type of personal rewards for users of a system that is, the incentives which motivate people to use respectively participate in the system (e.g. increased self-esteem derived from helping others; pure information gain; simply distraction from real-life)

Amongst the exemplarily mentioned multi-user environments the most important one for this thesis was the category of 'virtual worlds' since OWL, which should serve as host environment to run the physics experiments of the TEALsim framework (see chapter 2.3.2), constitutes exactly such a virtual world. Besides this particular focus – predefined at project start – over the course of the practical work for this thesis eventually other kinds of multi-user environments became of interest to serve as potential hosts for TEALsim.

Virtual worlds

Today the term ‘virtual world’ is usually associated with three dimensional game-like computer programs. According to Bell [2008] this does not precisely address the term ‘virtual world’ though. Based on his definitions, ‘virtual worlds’ are rather a super-category including the introductory outlined scenario. As summarized in Peachey et al. [2010] virtual worlds have to meet 4 specific traits [Peachey, Gillen, Livingstone, & Smith-Robbins, 2010, p. xviii]:

a) *“Virtual worlds are persistent. They exist regardless of whether any specific individual is logged in. Typically, there are processes in these worlds such as time and economy that continue to progress in some real time scale even when an individual user isn’t logged in.”*
b) *“Virtual worlds exist on wide area networks (WAN). To reach the scale of a “world” rather than an “environment” or “space” a virtual world must be accessible on a large scale and not contained behind a firewall or similar limitation.”*
c) *“Virtual worlds are massively multi-user. This is an important differentiation between virtual spaces built for a few users and worlds which can accommodate a global scale of users.”*
d) *“Virtual worlds employ avatars to represent users. Avatars are semi-autonomous agents represented in the digital space and capable of performing actions when commanded by a user. We differentiate avatar from icon or profile which represent a user but cannot perform actions.”*

In other words the concept of ‘virtual worlds’ has to be seen as an archetype of software based on the definitions above. The main justification for this thought is that none of the 4 traits give the world a true purpose that is, a motivation for people to use the software. For this reason a more precise classification of different virtual worlds is proposed in Peachey et al. [2010] with the exemplary introduction of ‘game worlds’ and ‘social worlds’.

Game worlds

Not least due to the commercial successes of Massive Multi Player Online Role Playing Games (MMORPG) like Ultima Online, Everquest or especially World of Warcraft this sub-category of virtual worlds became very prominent within the last few years and accounts for a multibillion dollar industry nowadays [Zhong, 2011, p. 2352]. Thanks to this recent popularity everybody would most likely assert to know what game worlds ought to be. On closer inspection from a scientific point of view it becomes quite difficult though to exactly define this term that is, the ‘game’ component promoting game worlds to be a more specialized form of a ‘virtual world’. The dilemma is that while there is less debate on what a virtual world is (see the 4 traits in the former sub-chapter) the notion of games has been discussed for a rather long period of time (and therefore a multitude of definitions exist). For example the thoughts of philosopher Ludwig Wittgenstein about the concept of games have already been published in 1953 [Wittgenstein, 1958]. More recent, and also more comprehensible for anyone not into philosophical essays, are the findings by Salen & Zimmerman [2003]. After analyzing and comparing 8 different formalizations of the term ‘game’ they conclude with their own definition, which essentially is the aggregation of all their examined ones: "*A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome.*" [Salen & Zimmerman, 2003, p. 80]

Ultimately one has to be aware though that this is just another attempt to formalize a very intangible concept, something which Salen & Zimmerman indicate to be conscious of themselves by ironically quoting David Parlett in their introduction who outlined that "[…] *the word* [game] *is used for so many different activities that it is not worth insisting on any proposed definition. All in all, it is a slippery lexicological customer, with many friends and relations in a wide variety of fields.*" [Parlett, 1999, p. 1] as cited in [Salen & Zimmerman, 2003, p. 71]

Social worlds

These worlds are a specialization of virtual worlds by providing "[…] *strong social tools and innovative content creation tools* […]" [Peachey, Gillen, Livingstone, & Smith-Robbins, 2010, p. xx] (in contrast to the focus on game play in game worlds). Compared to MMORPG's the reasons for the success of worlds falling into this category (such as Second Life or Entropia) are at first glance much more difficult to specify since social worlds do not offer any of those self-evident incentives originating from game-like mechanisms. On top of that these worlds are also much harder to govern. For example user created content will vary in quality and style impacting the in-world experience for other users.

Social worlds are interesting for this thesis because OWL, the target environment to host TEALsim, constitutes a social world in its current form. An interesting aspect about social worlds is the fact that, besides providing a few in real life currently unfeasible technologies like teleportation, they often try to mimic physical real-world limitations. For example most virtual social worlds comply with the laws of gravitation (or at least have them switchable). Another controversial example is OWL's audio feature which states that "*Distance attenuation and spatial audio provide a sense of immersion in the 3D space.*"[13] The term 'controversial' is used in this context because the benefit of this feature is questionable in a situation where you would like to promote social interaction on the one hand (by definition this is one of the goals of social worlds) but on the other hand restrict respectively decrease the quality of communication in some way.

2.5 Examples for Existing e-Learning Instruments

Since any application or process which delivers instructions by computational means may be an e-learning tool (as specified in chapter 2.3.1), obviously there are a lot of software products available which meet this definition. Therefore the subsequent list of e-learning tools respectively platforms is not based on criteria like an exceptional degree of public awareness or particular innovative and unique features, but rather tries to introduce to a handful of examples which are

[13] OWL feature list (accessed 11th July 2013)- http://openwonderland.org/about/features

already somehow related to the practical part of this thesis or could be potentially relevant in a further project continuing this work.

The mentioned examples of available real world applications also show that the categorization suggested in chapter 2.3.2 cannot be considered as a strictly delimited schema where each category is mutually exclusive. Instead real world implementations of e-learning software will almost certainly inherit characteristics of multiple categories.

2.5.1 The TEAL Simulation Framework

An auxiliary tool developed to be used within the broad educational concept of TEAL was TEALsim. This term is the name of a software library programmed in Java which should allow a rapid setup of virtual (physics) experiments. The framework is intended to expose a user-friendly and comprehensible API. While it is up to the creator of new content to combine a reasonable set of elements to create interesting experiments, it is the framework's task to hide away all of the underlying mathematics caused by the cross-influence of the simulation's elements and furthermore simulate and visualize the system's evolution over a given period of time.

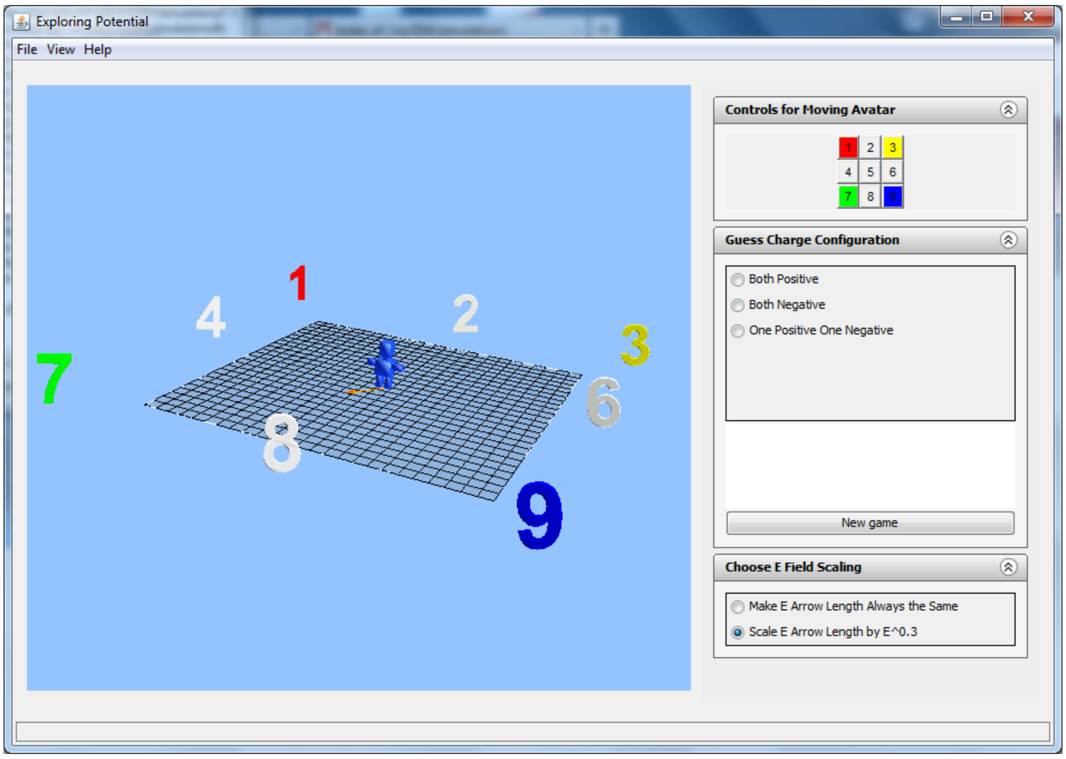
2.5.2 Academic Online Learning Programs Like edX

Currently most universities already established programs to evaluate possibilities to extend their educational services to the virtual space. In practice these projects are usually some sort of WBT platforms. For example MIT has been long known for its efforts in this area with their program called OpenCourseWare (OCW)[14]. Recently this commitment has been extended by the introduction of the even more interactive online courses of MITx, which are made available as part of the new joint-venture project run together with Harvard University called edX[15]. In this context MITx courseware is considered "[…] *as the 'movie' while edX is the 'theatre' in which the movie plays.*" [Sarma & Chuang, 2013, p. 11]

These kinds of initiatives are usually also referenced as massive online open courses (MOOCs). Putting figures for their popularity in relation to their host university's regular student base it becomes evident why this term was chosen. For example MIT has approximately 10 000 enrolled students but almost 155 000 people registered for the 6.002x MITx online course with 7 157 of them passing it in the end [Hardesty, 2012]. Likewise Stanford University offered a course for artificial intelligence which was attended by 160 000 students from 190 different countries across the globe [Zillner, 2012]. Despite the large fraction of registered students often failing these courses the absolute number of graduates can be, nevertheless, considered as a *massive*

[14] MIT OpenCourseWare – http://ocw.mit.edu
[15] edX online-learning platform - https://www.edx.org

educational success. For instance MIT's experimental 6.002x online course matches the capacities of a total of 40 years of regular on-campus education in this field [Hardesty, 2012]. However, courses on these online platforms are not exclusively intended to be used by a virtual audience but instead shall also "[...] *support parts of courses on campus, augmenting traditional teaching* [...]" [Bradt, 2012].



Guess Charge Configuration

- Both Positive
- Both Negative
- One Positive One Negative

Choose E Field Scaling

- Make E Arrow Length Always the Same
- [✓ Scale E Arrow Length by E^0.3]

Figure 2.2: TEALsim game-like application to explore electric potentials

2.5.3 Educational Games

Like outlined by Jantke [2010] in chapter 2.3.2 a multitude of games could be considered as educational games. However, of particular interest for this thesis are the already available applications in TEALsim which could be used for a multiplayer game. For example the 'Exploring Potential[16]' application (figure 2.2) lets users reveal an electric field piece by piece until they are able to guess the electric potential of the corresponding charge. This scenario could also be used in a competitive setup where multiple students compete for the lowest amount of hints to come up with the correct charge.

[16] TEALsim Exploring Potential – http://web.mit.edu/viz/EM/simulations/exploringpotential.jnlp

Another example is the electrostatic videogame[17] (figure 2.3) where a player has to adjust the charge of his own point charge accordingly to drag it through a maze. Again this could be extended to a multiplayer version where multiple players simultaneously race for e.g. the fastest time, maybe also with possibilities to interfere with other players' actions for increased competition.

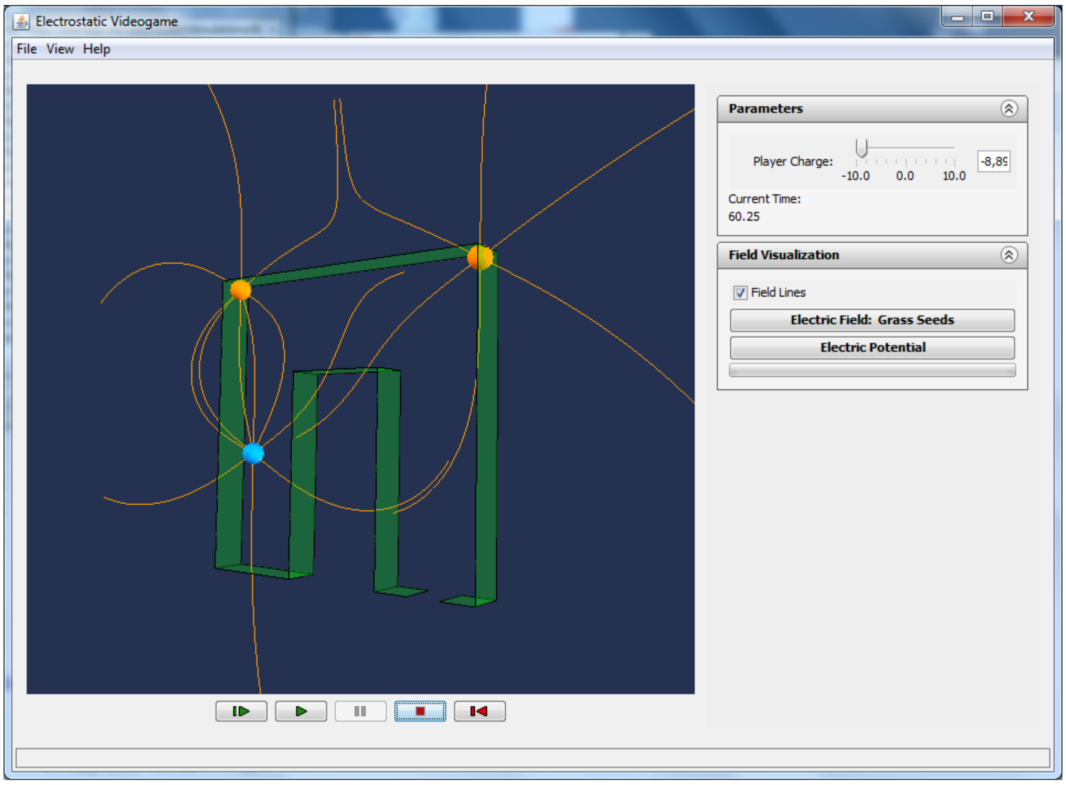


Figure 2.3: TEALsim electrostatic videogame

2.6 Thin Clients Versus Fat Clients

The concepts of thin and fat clients are outlined in this sub-chapter because for the practical part of this thesis it was of importance to find the right balance between these two extremes. Usually these terms are used within the context of distributed computer systems (that is, client-server architectures) to determine the role and powers of the client software. Fat clients are sometimes also called thick clients.

[17] TEALsim Electrostatic Videogame - http://web.mit.edu/viz/EM/simulations/videogame.jnlp

Generally speaking a thin client is a client who has to handle a minimum amount of application logic and focuses on the presentation of the program (therefore slow and cheap hardware may be used) whereas a fat client computes most of the application logic himself (instead of the server), with a desktop application constituting an ultimate fat client.

Hammerschall [2005] concisely describes the dilemma that existed with the desktop version of TEALsim by recapitulating that "*Under the assumption that a stand-alone, non-distributed application is secure and offers sufficient performance, the question for the programmer arises how to distribute the single components of this application to the nodes in a distributed system without introducing great deficits neither to performance nor to security.*"[18] [Hammerschall, 2005, S. 24] This general assertion is at the core of any evaluation leading to the decision how to design the distributed system.

The common understanding is that "*Compared with maintaining a central server, fat client TCO* [total cost of ownership] *also is higher, because of initial hardware and software requirements and the ongoing expense of supporting and updating remote client computers.*" [Shelly & Rosenblatt, 2011, p. 464] One has to be careful with this definition though since it assumes the whole distributed system to be maintained exclusively by a single entity (e.g. a company maintaining their workstations and server infrastructure). However in educational situations students often bring their own devices essentially taking over the expenses for client hardware and IT maintenance. Such scenarios may obliterate one of the major pros of thin client systems.

Even though the true advantages and disadvantages of each concept may be very situational as mentioned before, Hammerschall proposes a rough guideline for when to prefer one or the other paradigm:

	fat client	thin client
good network connection		✓
bad network connection	✓	
high complexity of application logic		✓
low complexity of application logic	✓	
high concurrency		✓
low concurrency	✓	✓
web application		✓

Table 2.2: Guidelines for utilization of either thin or fat clients [Hammerschall, 2005, S. 28]

[18] English translation from German origin:
"*Unter der Prämisse, dass zentrale, nicht verteilte Anwendungen sicher und performant sind, stellt sich für den Entwickler die Frage, wie bei einer verteilten Anwendung die Komponenten geeignet auf die Knoten des verteilten Systems zu legen sind, ohne dass zu große Verluste bei der Performance auftreten oder Sicherheitsrisiken entstehen.*"

2.7 Summary

This chapter introduced the various concepts which motivated to conduct this thesis in the first place. Figure 2.4 summarizes the interplay of these concepts with the most conceptual paradigm at the bottom and getting increasingly technical on the higher levels. In the long run the most interesting matter would be to evaluate the true efficiency of the entire illustrated system. However, this would require each level to be completely specified respectively implemented. This pre-condition holds true for the lower levels, since they have already been discussed and revised for a longer period of time, but not quite for the top levels.

Therefore, the rest of this thesis focuses almost exclusively on this top tier because all of the practical work has been spent in an effort to come up with a releasable software product and notionally 'complete this pyramid' to allow empirical studies.

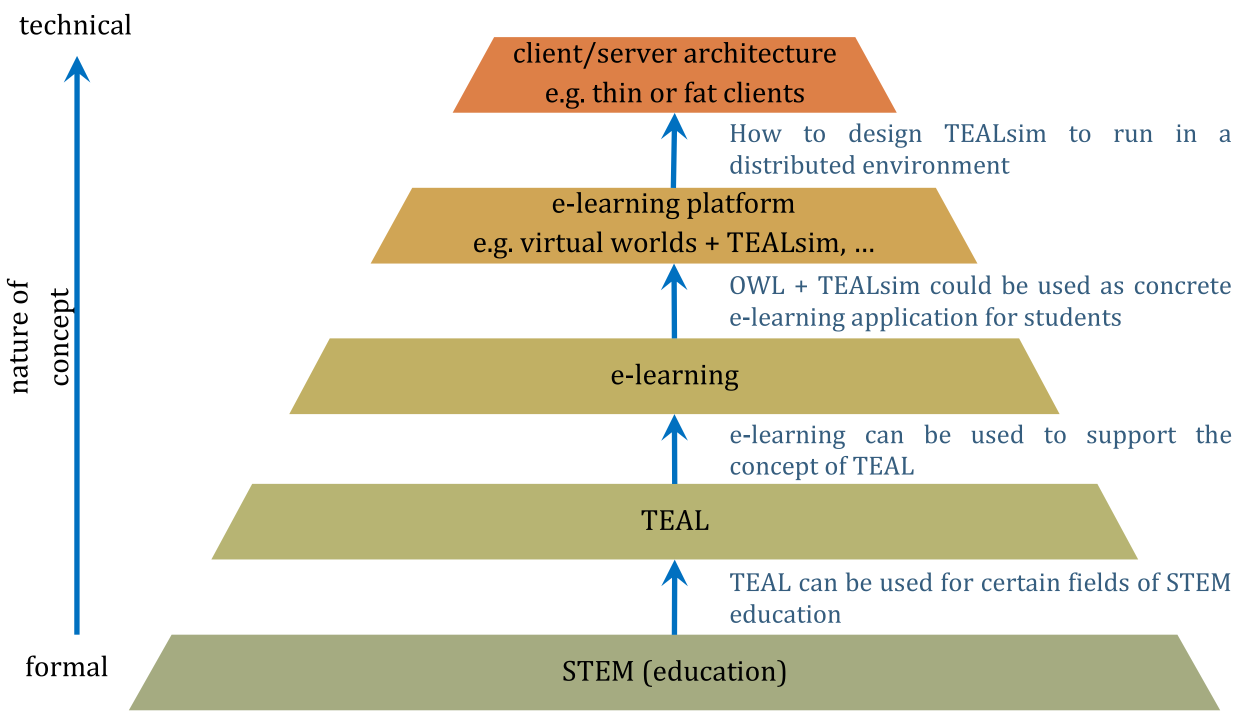

technical
nature of concept
formal
client/server architecture
e.g. thin or fat clients
How to design TEALsim to run in a distributed environment
e-learning platform
e.g. virtual worlds + TEALsim, ...
OWL + TEALsim could be used as concrete e-learning application for students
e-learning
e-learning can be used to support the concept of TEAL
TEAL
TEAL can be used for certain fields of STEM education
STEM (education)

Figure 2.4: Structure of this thesis' principles

Chapter 3

TEALsim in a Multi-User Environment

The first pages of this chapter introduce to some scenarios which are potentially interesting setups to use of TEALsim in the context of e-learning and collaboration of geographically spread students.

Subsequently, theoretical aspects related to the synchronization of an application (that is, a TEALsim simulation in this case) between multiple clients across a network are outlined (and in particular with OWL as host framework). Part of this discourse is a summary of the status quo which was available at the start of this thesis, a suggestion for a more or less completely alternative approach finishing with a comparison explaining why the new design was eventually introduced to the TEALsim framework.

In the end follows a confrontation of the principles and constraints of two distinct synchronization paradigms respectively their corresponding technical implementations. These were identified as the fundamental ways to synchronize user interactions for toolbox-like frameworks (like TEALsim) where less-versed programmers coming from other fields of science are confronted with the additional complexity induced by parallel, distributed execution of software.

3.1 Potential Host Environments for TEALsim

As mentioned in chapter 2.2 and 2.5.1 TEALsim is the tool of choice at the MIT to support their concept of teaching physics. Currently TEALsim consists of two principal areas with content of divergent dimensional level (sketched in figure 3.1):

a) an inner pane used to display the three dimensional visualization of the simulation flow
b) an outer pane used to accommodate all controls to influence the simulation (that is, two dimensional buttons, sliders, text fields, etc.)

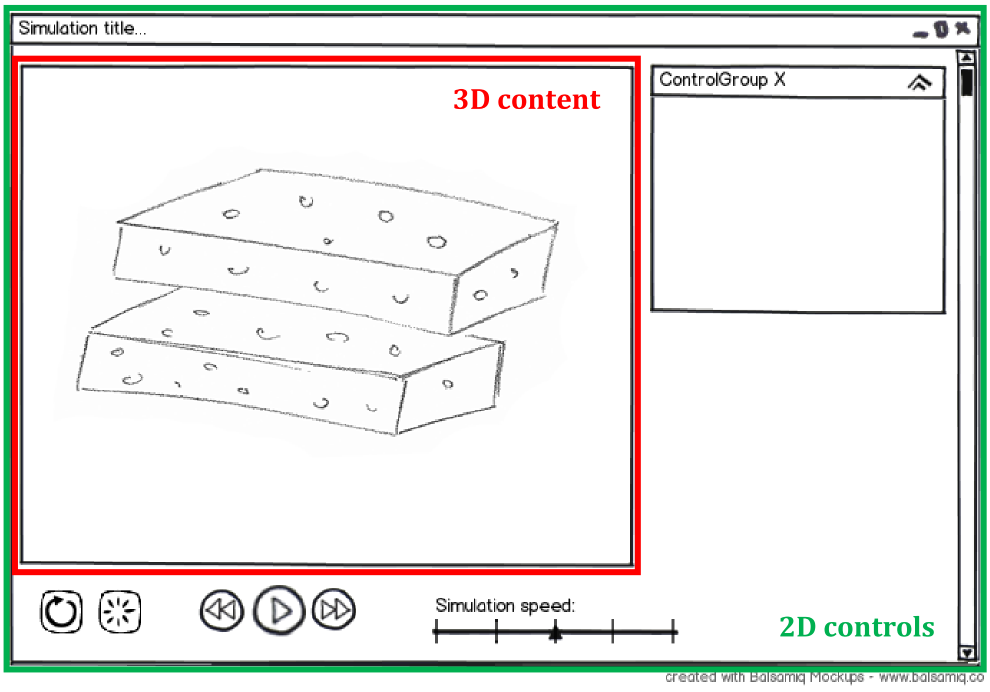


Figure 3.1: Sketch of TEALsim window highlighting areas of interest

Initially the software package was designed to run as a single, autonomous process on one computer. Given this objective, a stand-alone desktop version of TEALsim was provided as well as a specialized built to be used within a Java Web Applet[19]. Eventually projects have been conducted which explored ways to use TEALsim within a multi-user host environment (that is OWL - see [Scheucher, 2010], [Berger, 2012] and [Pirker, 2012]). However, besides OWL in particular, other multi-user setups would be conceivable as well. Depending on the concrete scenario there is a varying degree of features of TEALsim's current version which would fit out-of-the-box (e.g. the 3D content would be an ideal candidate to get integrated into a virtual world, but contrariwise the 2D controls would need a replacement in this case to provide an overall coherent user experience). Therefore, when planning to adapt TEALsim for a specific multi-user setup several aspects become relevant, such as:

- Degree the last stable, major release of TEALsim fits to the outlined use case
- Expectable complexity of code
- Dependency on 3rd-party projects
- Suitability to run on mobile devices
- Potential for interesting multi-user simulations

[19] Java Web Applet Tutorial – http://docs.oracle.com/javase/tutorial/deployment/applet/

- Requirements on network reliability, speed and responsiveness

To inspire the imagination in which context TEALsim could be used the following sub-chapters summarize some of the possibilities. It is up to the reader to evaluate to which degree the previously mentioned aspects are of relevance for the corresponding scenarios.

3.1.1 Stand-alone Version of TEALsim Without Network Features

While probably not up to one's expectations of a contemporary multi-user system, nevertheless, this scenario could be used to set up multiplayer applications. In fact this method was in place in real classrooms for the last couple of years. Students gathered around a single computer and mutually shared its controls.

3.1.2 Stand-Alone Java Application with Multi-User Component

Derived from the original TEALsim project this would be the most logical next step for any intentions aiming to add more comprehensive multi-user features to the framework. In this context TEALsim running as an independent process or as a Java Web Applet can be considered as an analog use case. Most convenient for clients would be a client-server architecture where an educational entity maintains the server side. However, a peer-to-peer based architecture would be feasible as well.

Exemplary use case: in a client-server architecture client A would start a multi-user application on his computer. During boot up client A connects with the server and finally waits for other users. Eventually client B starts up the same application on his computer and connects via the server with client A. Furthermore, they are able to mutually use the particular simulation via network (e.g. compete or collaborate).

For WBT initiatives like edX (see chapter 2.5.2) this configuration could be an option when coupled together with team collaboration solutions like Sococo[20] to create integrated working areas. Unlike virtual worlds explained in the succeeding chapter these environments would not provide a rich 3D experience focused on an avatar but would instead only provide means for communication (chat, audio, video, etc.), for exchange of information (like application sharing) and to organize teamwork (management of user permissions, managed communication channels, etc.).

[20] Sococo - https://www.sococo.com

3.1.3 TEALsim Embedded in a 3D Virtual World

This is the logically next advancement to a stand-alone networking-capable TEALsim version. TEALsim could be integrated in the virtual world e.g. via a plug-in system (like in OWL). Although in theory the virtual world could provide certain functions which are otherwise of no use for the stand-alone no-network version, in reality this single-user version has to be adapted to the principles of multi-user applications nevertheless because superimposing a client-server architecture on a single-user application will introduce serious design inconsistencies (as explained in the chapters hereafter). Additionally, new elements and concepts would have to be developed as well to truly take advantage of the possibilities a 3D virtual world offers. That said, while the implementation of certain features could supposedly be outsourced to external virtual world projects (e.g. the network protocol), overall it would require more work to create a releasable version compared to the scenario outlined in the previous chapter 3.1.2.

From a conceptual point of view, embedding TEALsim into a virtual world opens up various interesting use cases. Implemented properly, a 3D world definitely offers a more immersive experience compared to 2 dimensional environments. In an oral discussion one of the principal investigators of the TEAL project, John W. Belcher, for example expressed the desire of expanding the game-related capabilities of TEALsim. He liked the idea to tap the innate motivation of games – e.g. when shooting around point charges like projectiles in a first-person shooter – to unfold within the boundaries of an educational setting.

3.1.4 Mixed Mode

Another conceivable use case would be a mix of clients running TEALsim as stand-alone version alongside clients using TEALsim from within a virtual world. At first glance this does not make sense because chapter 3.1.3 outlined that additional elements would be required in TEALsim to truly take advantage of a virtual world's immersive potential. Then again these elements would most likely not be available for the stand-alone version which appears rather inconsistent. However, seen from a different perspective a mixed setup could also serve as transitional solution as long as it is unclear which scenario (see 3.1.2 and 3.1.3) to prefer.

In this context TEALsim could be used from within the virtual world identical to the way it is used from the stand-alone version. In fact this is how Scheucher [2010], Berger [2012] and Pirker [2012] integrated TEALsim into OWL, with the exception that they did not intend to provide outgoing connections to non-OWL clients.

3.2 Previous Network Architecture

The fundamental intention behind the formerly existing client-server architecture was to keep the main project of TEALsim as unchanged as possible, essentially leaving it as a real desktop application while packaging everything related to a network architecture into the separate TEALsim OWL module. Inherent to such an approach is the circumstance that for the development of the module in-depth knowledge about TEALsim's design is required because core components have to be rebuilt to enable the framework's executability in the distributed environment. Another aspect is the resulting complexity of the module which is owed to the fact that superimposing a client-server architecture on an underlying desktop system usually causes design inconsistencies and requires considerable hacks to make both worlds work together.

3.2.1 Principles of the Architecture and the Execution Sequence

For TEALsim's OWL module the simulation engine was outsourced from the client to the server. To understand the implications of this decision, one has to be aware that – while there are in general many threads running, like in any other modern-day application with a GUI – the simulation-part of the framework adheres to a single-threaded design. In simple terms there is only one thread which is responsible to calculate the state of all of the simulation's elements for the next frame. Upon completion of the calculation the simulation thread informs the rendering package (which can be considered as black-box consisting of an arbitrary amount of threads) to render the view based on the updated states of the elements. Since everything – simulation thread, rendering package, etc. – operates on the same set of states, the simulation thread has to wait for the rendering process to finish before resuming its duty and repeating the same procedure again.

Due to the fact that some possible (usually tricky) use cases were not specified (let alone being implemented – e.g. a satisfying solution to slider synchronization) inevitably no final, thorough explanation outlining every detail of the proposed synchronization mechanism can be given in this place. Nevertheless, for the purpose of understanding the mechanism's general concept this factor of uncertainty shall be disregarded in this place and any yet unspecified problems related to this design are assumed to be solvable hereafter.

Now, the basic idea was to achieve synchronization of the simulation elements amongst all clients by putting that part of the simulation logic onto the server side which was responsible for calculating the dependent values of these elements. Subsequently, these values were broadcasted to all clients who finished any outstanding, auxiliary calculations based on them before triggering the render process. As a result of this design the simulation had to be kept, more or less completely, in memory on both sides – the server as well as on the clients.

In the end this lead to a situation where clients were sort of streaming the simulation states like a video stream.

3.2.2 Performance Characteristics

To illustrate the effects of this design on bandwidth requirements table 3.1 summarizes the occurring network congestion for 3 basic simulations. Measurement of these figures was done by logging the size of *EngineMessages* being sent from the server to the client (that is, after the serialization of a message and before it was handed over to the OWL/RedDwarf Server infrastructure for transmission). For this reason the values may only be considered as a rough estimation since any further processing of the message on the layers below – possibly increasing or decreasing the total amount of bytes to transmit, e.g. by compressing the messages or adding meta information, etc. – is not reflected in the table. Besides, an important detail is the fact that the table relates to a single client receiving the stream of dependent values. Because OWL currently does not utilize any techniques like multicast [Kaplan, 2012], any successive client joining the simulation increases network congestion by the denoted figures. Additionally, the *EngineMessages* are only dispatched when the simulation is in a running state.

Table 3.1 serves as the basis to analyze three aspects of the discussed network design with regard to the visual appearance of TEALsim and its performance characteristics:

a) First of all the simulations used to run with a target frame rate of 20 frames per second (fps). *"Motion picture film originates at 24 frames per second."* [Poynton, 2002, p. 429] Wide spread standards for television broadcasting like PAL or NTSC clearly exceed the 20 fps level as well, usually yielding an effective frame rate of 25 fps respectively close to 30 fps [Poynton, 2002], whereas more recently developed systems like the Sony Playstation console seem to run with even higher target frame rates[21]. While the human eye is a sophisticated and complex device where – especially for a computer scientist – it seems difficult to come up with *one* final value outlining its capabilities in terms of distinctively perceivable frames per second, depending on the underlying test scenario various studies in this field of science uniformly report of a required frame rate greater than 20 fps. In [Dahm, 2005, p. 46] the value is estimated to 22 fps for example. Assuming this estimation to be applicable to most of the in practice occurring scenarios, the hard-coded maximum frame rate of 20 fps is suboptimal. Therefore table 3.1 contains two columns which show figures with extrapolations of the 20 fps measurements for a frame rate of 30 fps. This value was chosen as a target figure to reach in case of the event that the simulation engine gets revamped, since it seems to be a fair tradeoff between increased computational load and improved visual presentation.
b) The second aspect to mention is the last column of table 3.1 which is an indicator for the leeway to enhance TEALsim's simulation capabilities in the future based on the underlying network design. Considering that the simulation engine was capable of calculating at least 200 point charges simultaneously on a personal computer dating from September 2009

[21] John Carmack from Id Software and Cliff Bleszinski outlining their expectations on console target frame rates - http://www.tomshardware.com/news/30fps-John-Carmack-Next-Generation-Console-Framerate,19864.html

(Intel Core i5 2.66 GHz Quad-Core CPU, 4 GB RAM, Nvidia GeForce 260 GTX, Windows 7 64 bit), which would result in approximately 600 kb/s bandwidth utilization for each client streaming such a simulation, network performance can be considered as one of the major bottlenecks of the framework (unless bandwidth is considered to be unlimited, which could hold true for LAN-only scenarios). As a solution to this issue Berger [2012, p. 69] suggested three potential techniques to reduce bandwidth utilization, which are:

1) Change data for transmission from double values to float (reducing bandwidth utilization by 50%, but also reducing simulation accuracy to a certain extend)
2) Transfer only data relating to changed simulation elements
3) Transfer only those attributes of simulation elements which cannot be omitted to render the frame on the client side

c) Last but not least table 3.1 also outlines that not every simulation takes advantage of the available server-side processing power (and it is open if this could be changed in a way which makes sense). The Falling Coil simulation for example, which is a very resource hungry application in its current, low-optimized state, causes very little network traffic after all.

Simulation name	bytes per frame*	kb/s with 20fps	kb/s with 30fps	# point charges (PC)	resulting bytes per PC**	kb/s per PC with 30fps
Capacitor	1312	26	38	12	109	3
Capacitor	1888	37	55	18	105	3
Charge by Induction	1328	26	39	10	133	4
Charge by Induction	2288	45	67	20	114	3
Falling Coil	224	4	7	-	-	

* approximate values; based on the size of a message dispatched from server to client
** disregarding any overhead potentially caused by e.g. message header, etc.

Table 3.1: Bandwidth usage of OWL module for different simulations

3.2.3 Limitations Introduced by 3rd-Party Frameworks

Besides the already mentioned, expectable issues caused by superimposing a client-server architecture on a desktop system, the underlying RedDwarf Server[22] used by OWL introduced another set of limitations, further intensifying the difficulties. Following is a consolidated list of constraints which are of potential relevance for the TEALsim OWL module, taken from [Berger, 2012, p. 70]:

[22] RedDwarf Server homepage - http://www.reddwarfserver.org/

"Code run on PD [Project Darkstar] *has to follow several guidelines* [RedDwarf Server Application Tutorial, 2010]:

a) All objects must implement the serializable interface. Without that the mentioned atomicity of a task can not be provided. PD throws an exception if an object not being serializable.

b) A single managed object must not contain too much data. Otherwise the de-serialization and re-serialization process would take too much time and the task will be thrown away very often.

c) All inner classes should be static since the time taken for the serialization increases significantly if they are not.

d) Synchronization blocks must not be used among managed objects and their members. Since PD uses it's own locks those can conflict with the ones the user defined code uses. This can easily lead to a deadlock.

e) Static fields which are not constant vanish on re-serialization. Although this problem can be solved with Java semantics another problem with this fields appear. Such fields are specific to a single Java virtual machine. This behavior undergoes the feature of PD to run on more than one virtual machine.

f) Java's exception base class java.lang.Exception should never be caught. This is because PD uses its own exceptions which would in this case be caught by the user code. This is especially important for debugging and testing new functionality since the exception base class is often used together with such approaches.

g) No objects except managed objects themselves should be referenced by more than one managed object. After the first serialization process they will not be identical any more since a new object is created on re-serialization."[23]

3.2.4 Recapitulation

Evidently the design explained in this chapter comes with certain limitations which are difficult to overcome based on minor reiterations of respectively mere tweaks to the whole concept. For example none of the suggestions by Berger [2012] to reduce bandwidth utilization (as summarized in chapter 3.2.2) are straight forward to implement, where two of them yield an unpredictable net performance gain (item 2 and 3). It is even more likely that the idea to transfer only changed simulation data does not result in a significant performance improvement at all, since in a realistically simulated physics environment everything somehow interacts with everything else at every given point of time, and by doing so causing changes (and it does not

[23] This quote was cited as a whole because it is already a recapitulation of the RedDwarf Server Application Tutorial with a very high information content which would be very difficult to exceed on the one hand, and where any further summary would inevitably omit important pieces of information.

matter if the impacts caused by the mutual interactions may be just infinitesimal after all because in general a very tiny floating-point number has the same memory footprint compared to a large number).

Another side effect of the discussed design constitutes the fact that individual system requirements for simulations considerably diverge because certain simulations are capable of taking full advantage of the available 'server-side acceleration' whereas other simulations are denied this aid and therefore still require a decent computer to run with an acceptable frame rate. Put another way the design is inconsistent in its effort to establish a system requiring fairly 'thin' clients and a 'fat' server eventually enforcing an environment where both – the clients and server alike – have to be 'fat'.

Additionally complexity of the code turned out to be rather high, for which reason a different approach to handle synchronization will be discussed in the succeeding chapter 3.3.

3.3 Alternative Approach Derived from Video Games

Starting point for a reconsideration of the existing design is an analysis of what has to be synchronized. In its former state the TEALsim OWL module had to synchronize two things between server and client(s):

a) user inputs (such as button clicks or text field inputs)
b) simulation states (of the simulation elements – via *EngineMessages*)

Out of these two mentioned items, user inputs are always going to be unpredictable and will therefore require a mechanism for synchronization. Hence only simulation states remain as a candidate for potential optimization.

Techniques used to build multiplayer video games could help to find the right approach for this issue – after all simulations built upon the TEALsim framework can be regarded as a kind of game as well (particularly since ideas exist to evolve TEALsim into a direction where it allows for even more game-like simulations to be created). For a very popular title in this area that is, the first-person shooter Half-Life, its developer Valve Corporation[24] published a series of articles on its Developer Community web page[25], describing various aspects of their technology. Most interestingly their engine and network logic does not synchronize each frame, but rather expects a certain kind of determinism in the game flow (which equals to the 'simulation flow' with regard to TEALsim). This determinism usually only gets violated by user input. Due to this reason a lot of brainpower was invested to optimize methods for game flow prediction, e.g. by extrapolating and interpolating client states, with one overall goal being to decrease perceivable network latency

[24] Valve Corporation - http://www.valvesoftware.com/
[25] Valve Developer Community page - https://developer.valvesoftware.com

(subsumed under the term ‘lag compensation’ – see [Bernier, 2001]). Since a competitive multiplayer game is always prone to hacking attacks by individuals trying to gain an unfair advantage, their system comprises a supervising server instance computing the game flow in parallel to the clients, verifying that user events stay within the boundaries of possible. Still, communication between client(s) and server basically consists of user inputs influencing the game in a way which alter the predictable future, or in other words: only new impulses, changing the ‘direction’ the game converges to, are transmitted. This is possible because all of the involved parties (server and clients) share the same set of algorithms which calculate the same output based on the same input.

Apply best practices

Applying this paradigm to TEALsim creates the requirement for the simulation engine to become deterministic. Formally, the engine’s responsibility is to calculate the next, future state for all simulation elements (= output) based on their former states and a specified amount of time to forward the scenery (= input).

Naturally, computational intensive tasks from other, in parallel running processes on the same machine are able to delay the computation of new frames for TEALsim (by racing for computational resources). Previously, there existed no specified logic to ensure that the engine made up for the time it was lagging behind after such a delay occurred. Put simply, the most important thing that has to be changed to make the engine predictable is its behavior from calculating new outputs on a best effort basis to a stricter specification forcing it to produce defined outputs in certain intervals with the option to fill in supplementary outputs depending on available processing power (and furthermore skip in-between steps/frames as long as it is lagging behind). More details on how the engine is supposed to work will be given in chapter 4.

Assuming that this requirement for determinism can be met, in general no additional synchronization would be required on that score. Just like in the case of Half-Life synchronization could be restricted to mere user inputs, drastically reducing bandwidth utilization. Above all no imperative need for server-side calculations exists for the TEALsim framework because it is currently not intended for use in a real competitive environment hence the issue how to ensure that integrity is maintained is of no concern. For this reason the sever could resemble a lightweight broker service responsible to interconnect its clients, resolve race conditions occurring between clients trying to change the same simulation element within a short time interval as well as store an up-to-date version of the current simulation state (which could be queried from one of the connected clients) for persistence purposes .

Impacts on TEALsim’s design

As mentioned adoptions to the simulation engine are necessary to ensure its determinism. Beyond that the integral part of the proposed design is the shift of TEALsim’s design from desktop to client-server architecture, regardless of the actual environment the simulation is intended to run in. The background for this paradigm change is the notion that it is easier – and from a design point of view more consistent – to emulate a distributed environment for a desktop application than it is to superimpose a client-server architecture on a desktop application.

For technical details how this aspect was realized in TEALsim see chapter 7. In general the framework references a minimalistic connection interface for which an appropriate implementation gets instantiated depending on the prevailing host environment. The client-server architecture itself is of an authoritative nature, which means that the client (the TEALsim simulation) indispensably requires a server to authorize user inputs. Though, due to the abstraction of the connection the client does not care whether the server runs in parallel on the local machine (either as independent process or in the same Java virtual machine) or remotely – it is the concrete connection's responsibility to close the gap between client and server.

Besides the need for a package containing the network layer and changes to the simulation engine, the third mentionable effect of the new design on TEALsim is its requirement to slightly adapt all existing simulations. Similar to the concept of multi-threaded programming, where the Java language provides standardized constructs to synchronize concurrent threads on various levels of scope (e.g. via synchronized methods and blocks, reentrant locks, etc.), equivalent decisions have to be made when designing applications for distributed execution. To reduce the amount of work required to adjust the whole project to adhere to the new paradigm two major types of objects will be part of the network package (for detailed technical information regarding these properties see chapter 7.5):

a) A generic property which meets the requirement to set its value asynchronously only after server authorization. Furthermore, it will also be used to provide synchronization on block level.
b) A package of classes referencing Swing UI elements which can be synchronized across all other concurrent clients and furthermore linked to generic properties.

Recapitulation

The proposed alternative design is based on paradigms successfully used for proprietary multiplayer games. It will inevitably increase the amount of TEALsim's code base (and consequently also the complexity by a certain degree), since the network capabilities become integral part of the framework instead of being outsourced to secondary projects. In return any project aiming for distributed execution will become more comprehensible and more consistent in design by a magnitude. The optimization of the system for client-server architecture will provide considerably improved performance in this area while suffering – if at all – from negligible performance hits in the desktop scenario. Once adapted to the new architecture, simulations built on the TEALsim framework can be used with little to no adjustments in frameworks like OWL. Beyond that no noticeable additional work to create new simulations should derive from the new architecture due to the availability of simple-to-use properties which encapsulate the synchronization logic.

3.4 Comparison of Previous Design Versus Alternative Design

In this chapter the term '*previous design*' relates to the concept explained in chapter 3.2, whereas the term '*alternative design*' references the ideas depicted in chapter 3.3. Various general aspects which are of importance when it comes to adding network features to TEALsim respectively the associated OWL module get discussed hereafter. This analysis shall outline each concept's corresponding strengths and weaknesses.

3.4.1 Complexity of TEALsim Framework

Considering TEALsim's codebase isolated from all other related projects, the client-server architecture of the alternative design doubtlessly adds in a certain amount of complexity compared to the previous design. In this context it is important not to mix code and complexity reducing effects of the cleanup duties described in chapter 6.1 with the consequences to the framework's code caused by the new design. Instead the former activity has to be seen independently and it would have benefited both designs alike. Therefore TEALsim seen isolated from any derived project was certainly less complex based on the previous design.

3.4.2 Complexity of OWL Module

Different to the reduction of code in TEALsim's main project (as discussed in the previous chapter 3.4.1) is the decreased size and complexity of TEALsim's OWL module (see chapter 6.1) directly related to the introduction of the new client-server architecture. This is due to the downgrade of OWL's role to something resembling a media player.

In fact everything that was needed to make the TEALsim simulations run in OWL was to create a specific implementation of the connection system as well as provide adapted classes to allow embedding the viewer into the OWL world (which already existed for the module based on the previous design).

Currently there are still some classes with redundant/obsolete code left in the module; further work in this area with the goal to completely take away the requirement to know any TEALsim framework internal details would make it even easier to embed TEALsim in 3rd-party applications.

3.4.3 Complexity to Create New Content

Complexity to create new simulations should stay equal regardless of the underlying design of the TEALsim framework. The documentation of the previous design owes detailed answers to certain use cases which require synchronization. For this reason it is sensible to assume that eventually similar mechanisms like those described in chapter 7.5 would have to be implemented for the previous design as well to cover the fundamental types of synchronization (see chapter 3.5). The tools to synchronize user inputs introduced with the alternative design give content developers a fine-grained control over the things they want to be synchronized on an easy-to-use basis.

3.4.4 Network Congestion and Scalability

Network congestion caused by synchronization of user inputs should be almost identical for both designs. Beyond that the flexible operation mode of the alternative design (see chapter 4.3) requires a distinction of the expectable benefits depending on the situation:

a) Compared to the previous design, network traffic is reduced by a magnitude in the alternative design's standard mode, since synchronization amongst all clients is generally done implicitly by a deterministic simulation flow. For one (capable) client little bandwidth usage will arise from the server's subscription to receive regular updates of the simulation state (to share with connecting clients).
b) In this context the worst case scenario occurs with computational weak clients (like smartphones or tablet computers) who could switch into a mode of operation similar to the previous design (where they will receive a stream of simulation states to avoid having to compute the simulation flow themselves).

3.4.5 Code Extensibility and Autonomy

With the previous design TEALsim's network capabilities originated solely from the (tight integration into the) OWL framework. This situation could be described as a vendor lock-in because no easy integration of TEALsim in any other 3rd-party multi-user framework was conceivable.

Furthermore extension of the main project itself was challenging, not least due to the split engine design used for the OWL module. When trying to change the TEALsim framework this setup made it difficult to entirely comprehend and anticipate the consequences on dependent projects like the OWL module. In absence of extensive tests to verify the code's functionality this issue becomes especially severe.

The alternative design packages everything into one coherent project. As a result verification of the correct operation of all features becomes more centralized and consequently testing becomes easier and more reliable. Furthermore, the development of new features for TEALsim should become more rapid due to isolating changes to one (small) project instead of having to work with multiple interconnected projects, which all come with their own set of limitations and rules regarding e.g. debugging, compilation, etc.

3.4.6 Versatility

Not least due to its flexibility to switch between operating modes, the alternative design opens up additional possibilities for future developments and fields of operation for TEALsim. On the one hand focus could be put on the simulation core to cover more physics experiments or even other fields of science again (e.g. biochemistry). In line with such developments would be the tasks to add more sparkle and improve performance on the visualization side. This direction would especially benefit computational powerful computers like laptops or workstations.

On the other hand attention could be directed to work on use cases relating to the execution of TEALsim on slower devices streaming the simulation flow. Such devices often come with a unique set of constraints to mind, like reduced screen sizes which would need special adjustments to the user interface to allow for a good user experience.

While the previous design operated similar to the streaming mode of the alternative design and thus shares the same set of possible future fields of operation, its tolerance for more graphical brilliance is potentially limited by the available network capacity.

3.4.7 Server Hardware Requirements

Hardware requirements for the server are on its bare minimum for the alternative design. In its current conception this design does not offload any computational intensive tasks from the client side onto the server side. Instead the server merely has to decide if incoming requests are admissible. Consequently a server running the alternative design's service should be capable of handling a vast amount of clients and simulations simultaneously.

With the previous design the threshold for concurrent clients and simulations was much lower. For the maximum amount of clients the available network bandwidth was a major factor (see chapter 3.2 and in particular table 3.1) whereas the maximum amount of concurrently running simulations was mostly restricted by the available computing power. Finding a feasible set of simulations the server could handle was quite complex. Auxiliary tasks which were running in parallel had a much higher influence on the perceivable, visual experience for clients executing a simulation since any spike in CPU load by another process immediately delayed the computation of the subsequent simulation frames.

3.4.8 Client Hardware Requirements

In theory the previous design was based on the concept that computational intensive tasks – like calculation of the simulation flow – should be delegated from the client to the server, thereby relieving the client. In practice not all simulations were equally suitable for such a divided computation model (see table 3.1 and the related discussion), hence slow clients could only run a subset of the available simulations whereas fast clients were not always able to take full advantage of their computing power but were often forced to idle waiting for the server to provide missing parts of the calculation. In this context clients running TEALsim in the streaming mode of the alternative design obviously do not experience any difference compared to the previous design.

The true benefit of the alternative design lies in its better utilization of otherwise untapped resources for computational capable clients. In this regard the client hardware requirements could be considered to be higher, which becomes irrelevant though due to the possibility to switch over to the streaming mode if really necessary.

3.4.9 Recapitulation

Each design concept comes with certain – sometimes mutual – pros and cons. The following table gives a brief overview of the discussed aspects, and shall serve as a simple and quick reference to the conclusions which can be drawn from the explanations in the corresponding chapters. In this case a check means that one design could be considered superior to the other design for this particular trait.

aspect	superiority	
	previous design	alternative design
3.4.1 Complexity of TEALsim Framework	✓	
3.4.2 Complexity of OWL Module		✓
3.4.3 Complexity to Create New Content	✓	✓
3.4.4 Network Congestion and Scalability	(✓)	✓
3.4.5 Code Extensibility and Autonomy		✓
3.4.6 Versatility		✓
3.4.7 Server Hardware Requirements		✓
3.4.8 Client Hardware Requirements	(✓)	✓

Table 3.2: Pros and cons for the previous versus the alternative synchronization design

3.5 Synchronization of User Inputs

As explained, the alternative design for TEALsim assumes that the framework itself executes in a deterministic way, thus reducing synchronization requirements to events triggered by external entities that is, a human user in most cases. Referencing the example given in listing 3.1, the most manifest solution for this requirement would be to try to synchronize the call to the *actionPerformed(…)* method of the button's *ActionListener* instead of individually synchronizing each *TextField's* call to the *setValue(…)* method. Besides, considering that TEALsim is intended to be a toolbox which should be easy to use even for non-versed programmers (creating new applications), it would be handy to provide a very generic solution to this issue which operates more or less hidden to the application creators.

In this context attempts have been made e.g. with marker interfaces along with JDK 5's instrumentation[26] features to detect and dynamically augment corresponding events in need of synchronization. Even though certain related problems can be solved with this technique (and additional code in the simulation definition files is reduced to a minimum), a couple of unsolvable problems remained which eventually resulted in the requirement to create specific objects for every single Swing UI element (see chapter 7.5).

Listing 3.1: Pseudo code declaring a basic UI and the relation of its elements

```
Button btn = new Button("Increment");
TextField field1 = new TextField(0);
TextField field2 = new TextField(0);

btn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    field1.setValue(field1.getValue() + 1);
    field2.setValue(field2.getValue() + 1);
  }
});
```

3.6 Summary

This chapter introduced to the fundamentals related to the execution of TEALsim in a distributed environment. Initially several scenarios were outlined to give an idea about how to utilize TEALsim in various multi-user environments. Subsequently followed a theoretical disquisition on the software design used by former, related projects ([Scheucher, 2010], [Berger, 2012] and [Pirker, 2012]) in contrast to an alternative approach. After recapitulating each concept's mechanics subsequently several autonomous aspects were identified to highlight each design's pros and cons. In the end followed a short introduction to the principles of synchronizing user inputs.

After this chapter pointed out reasons why to adapt TEALsim to utilize a client-server architecture as a whole, the following chapter explains the ideas to attain a deterministic simulation flow. This discussion starts out specifying the general runtime characteristics of the simulation engine, leading to encountered issues on the road to meet these requirements and finally closes with details about the developed algorithm.

[26] Java instrumentation example - https://today.java.net/pub/a/today/2008/04/24/add-logging-at-class-load-time-with-instrumentation.html

Chapter 4

Designing a Deterministic Simulation Engine

This chapter familiarizes with the idea of a deterministic simulation engine and the problems which have to be dealt with. The initial simulation engine algorithm was not predictable respectively configurable enough to use it in an environment with the requirement to reach an exactly defined simulation state at an exactly defined point of real-world time to execute arbitrary tasks. Since the former algorithm additionally lacked thorough design-documents specifying the in place procedures a complete redesign of the simulation algorithm was preferred over an – as minimalistic as possible – adaption of the existing one.

After a very general summary of the new algorithm's concept and the encountered technical issues, the execution flow of the algorithm gets outlined in more detail explaining its different modes and particular steps of the execution.

4.1 Idea Behind a Deterministic Algorithm

In chapter 3.3 it was mentioned that – in simple terms – all clients should compute a simulation identical to avoid the need for explicit synchronization. Even though all clients will execute the same Java bytecode, and therefore they all might eventually compute the same result when hard-coding the execution flow, they will inevitably require a distinct amount of real-world time to do so. This is one reason that causes clients to run out of sync (until they hit this defined final state) without any other precautionary design decisions.

Now the main idea behind the new engine algorithm is to implement it in a flexible way which is capable of computing more or less sub-steps (so called *in-between frames*) depending on the underlying hardware capabilities. However a fixed minimum frame rate of so-called *key frames* shall be maintained, which serve as reference states to implicitly achieve synchronization across all concurrent clients. Formally there exist the following parameters:

a) ΔT is the amount of virtual simulation time to forward the simulation from one key frame to the next key frame.

b) *Δstep* is the amount of simulation time from one in-between frame to the next in-between frame. In general this value is used as the time factor in the equations calculating the physical correct behavior of all simulation's elements during the integration phase.

c) A *SimulationSpeed* value which serves as multiplier for ΔT. During runtime this value can be adjusted via the available slider (see figure 6.10). For the algorithm itself this value has no relevance because ΔT and Δstep are multiplied alike.

For a given simulation ΔT is either set to a specific value by the simulation creator or otherwise a standard value from the *AbstractEngine* is used. In other words, when setting up new simulations the creator may specify that two distinct simulations 'run' with a different pace even though the *SimulationSpeed* slider is set to the same value. In execution mode the algorithm (or 'engine') advances the simulation by Δstep for each calculation round.

Figure 4.1 visualizes the new algorithm's behavior in an exemplary distributed system. It shows 2 clients running in parallel. Client A is in general capable of running the simulation with 8 fps whereas client B only manages to compute 4 fps on average. Starting from an identical state both clients compute the next in-between frame(s). Once the 1st second is over both clients compute their first key frame (marked with a star and letter in bold) based on the initial simulation state. Subsequently both clients compute the next in-between frames. Figure 4.1 also indicates what happens when one client experiences unexpected difficulties to maintain the frame rate. When calculating the 6th frame after the 1st key frame client A faces a delay (e.g. a virus scanner process running in parallel to the TEALsim simulation used up a severe amount of processing power). Due to this delay client A does not manage to compute a 7th in-between frame anymore but instead immediately calculates the next key frame. This is the fundamental concept at the heart of the new algorithm. Only the most common use case was covered in this explanation, but there are several other situations which have to be considered as well. These situations and the algorithm itself are covered in more detail in the diagrams in chapter 4.4.

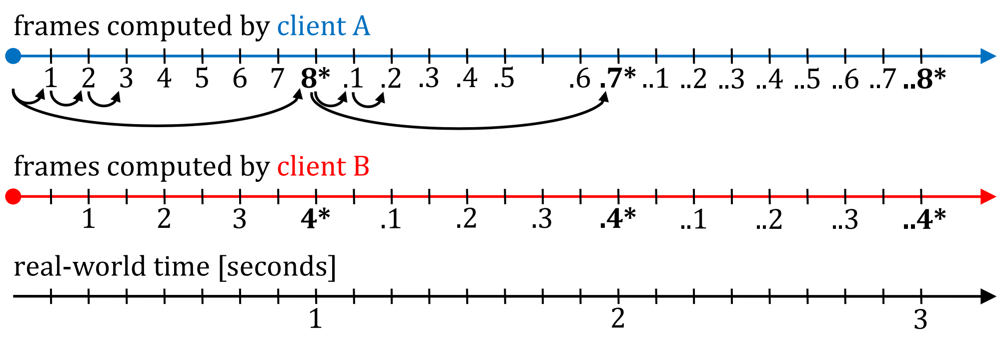


Figure 4.1: Schema of deterministic simulation engine

4.2 Issues Complicating an Implicit Synchronization Simulation Calculations

There are several issues which have to be considered when trying to achieve an implicit synchronization of concurrently running programs. Some of these problems are of logical nature whereas others originate from technical constraints. This sub-chapter introduces to the technical reasons why key frames were introduced as reoccurring states to ensure implicit synchronization, breaking the simulation flow up into a stream of in-between frame intervals.

4.2.1 Double Precision Divergence Across Multiple Clients

A serious challenge impeding the task to create a flexible but still deterministic simulation engine emanates from the definition of floating point numbers. Since computers can only handle two different types of values on their lowest level – zero and one – ultimately everything has to be mapped to a binary presentation. The same holds true for floating point numbers. Mathematical operations which appear easy to solve for a human in his head might effectively return unexpected results once executed on a computer. An example for such an issue is given in listing 4.1 [Stack Overflow, 2008], where two mathematical operations lead to results which are not representable double-precision values.

Listing 4.1: Code sample to demonstrate precision issue with floating point numbers

```
public class doublePrecision {
  public static void main(String[] args) {
     double total = 0;
     total += 5.6;
     total += 5.8;
     System.out.println(total); // prints 11.399999999999

     double three = 33.33333333333333 / 100;
     System.out.println(three); // prints 0.33333333333333326
  }
}
```

This problem intensifies when using multiplications and in particular with divisions since they are increased likely to require rounding. Therefore, two separated systems computing an anticipated identical end result with a different amount of sub-steps (each time causing small rounding to occur) will return divergent results. Listing 4.2 illustrates this issue by computing a value once in a single step and subsequently computing the supposedly same figure in a series of smaller sub-steps.

Listing 4.2: Demonstration of diverging results based on amount of calculation steps

```java
public class doublePrecision {
  public static void main(String[] args) {
    double distance1 = 0;
    double distance2 = 0;
    final double TIME = 1.0; // hours
    final double SPEED = 30.0; // km/h

    distance1 = SPEED * TIME;
    System.out.println(distance1); // prints 30.0

    final int STEPS = 7;
    for(int i = 0; i < STEPS; i++)
      distance2 += SPEED * (TIME / STEPS);

    System.out.println(distance2); // prints 29.999999999999996
  }
}
```

4.2.2 Calculation Divergence Based on Algorithm Step-Sizes

Besides rounding occurring due to floating point numbers which exceed the amount of information storable in their associated objects in computer memory, also the underlying algorithms describing physical effects may produce slightly varying results depending on the amount of sub-steps used to compute an aggregated end result.

An example for this is the Runge-Kutta method employed in TEALsim to numerically integrate ordinary differential equations. The algorithm splits the ‘distance’ between a starting point and an end point into an interval of intermediate steps [Weisstein, 2013]. Based on this interval an approximation of the true solution is calculated as well as an estimated accuracy for this result. The result’s accuracy is then compared with a target accuracy. If a certain threshold is not met an adaptive step-size control breaks up the interval and reruns the algorithm on these smaller intervals. This is a recursive process which decreases the intervals until all of their approximations meet the target accuracy. An approximation’s result will slightly vary depending on its starting point. For this reason when the simulation engine computes one frame based on a larger time delta the simulation's state will be unequal to the final state by way of computing two successive frames each with half the time delta.

4.3 Concept of the Configurable Simulation Engine

One design goal of the new simulation engine was to make it flexible in a way that it can adapt to the actual hardware available on the client side. Computational potent devices should be able to tap their capabilities whereas slow devices should receive aid from the server. Figure 4.2 outlines the implemented concept as a flow chart. In general there exist two possible strategies a client may pursue: either the simulation can be retrieved as a stream from the server (similar to the previous design) or alternatively it gets computed locally. Both strategies are self-contained cycles, although there are optional checks which allow switching from one to the other mode.

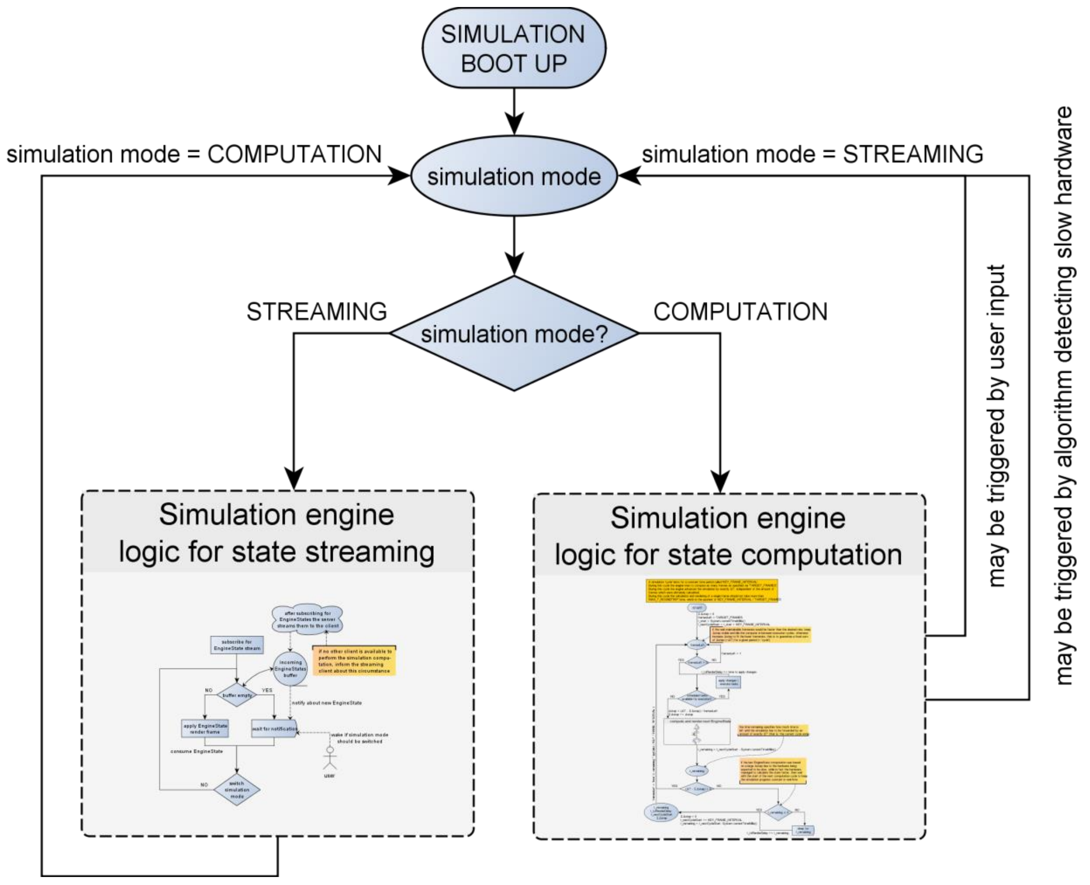

SIMULATION BOOT UP
simulation mode = COMPUTATION
simulation mode
simulation mode = STREAMING
STREAMING
simulation mode?
COMPUTATION
Simulation engine logic for state streaming
Simulation engine logic for state computation
may be triggered by user input
may be triggered by algorithm detecting slow hardware

Figure 4.2: Diagram of the concept of a switchable simulation engine

4.4 Design of the Algorithm for Local Simulation Calculation

Figure 4.4 outlines the algorithm in place to adapt the frame rate to the hardware capabilities. Variables suffixed with a '*t_*' are figures dealing with time values which are measured in milliseconds. The algorithm itself should be comprehensible without any further written explanation by studying the flowchart, possibly backed up by calculating one or the other cycle with pen and paper. Furthermore figure 4.5 shows that part of the algorithm in more detail which is responsible to compute the next simulation step, export the simulation state and update the currently achievable frame rate. One thing to keep in mind is that scheduled tasks are usually only executed immediately after a key frame as a method to deal with race conditions and ensure synchronization.

Figure 4.3 roughly outlines the concept where the server registers with a client to receive its calculation results of the simulation (to share with other clients requesting the simulation as a stream). This part has not yet been implemented in the prototype version.

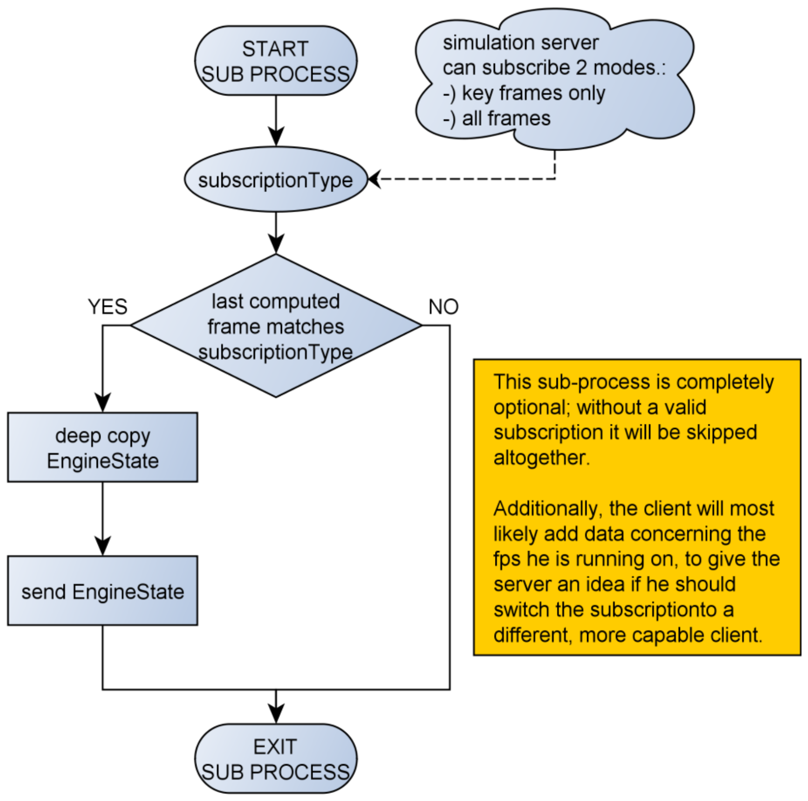

START SUB PROCESS
simulation server can subscribe 2 modes.:
-) key frames only
-) all frames
subscriptionType
last computed frame matches subscriptionType
YES
NO
deep copy EngineState
send EngineState
This sub-process is completely optional; without a valid subscription it will be skipped altogether.
Additionally, the client will most likely add data concerning the fps he is running on, to give the server an idea if he should switch the subscriptionto a different, more capable client.
EXIT SUB PROCESS

Figure 4.3: Logic to export engine states to the remote computers (most notably the server)

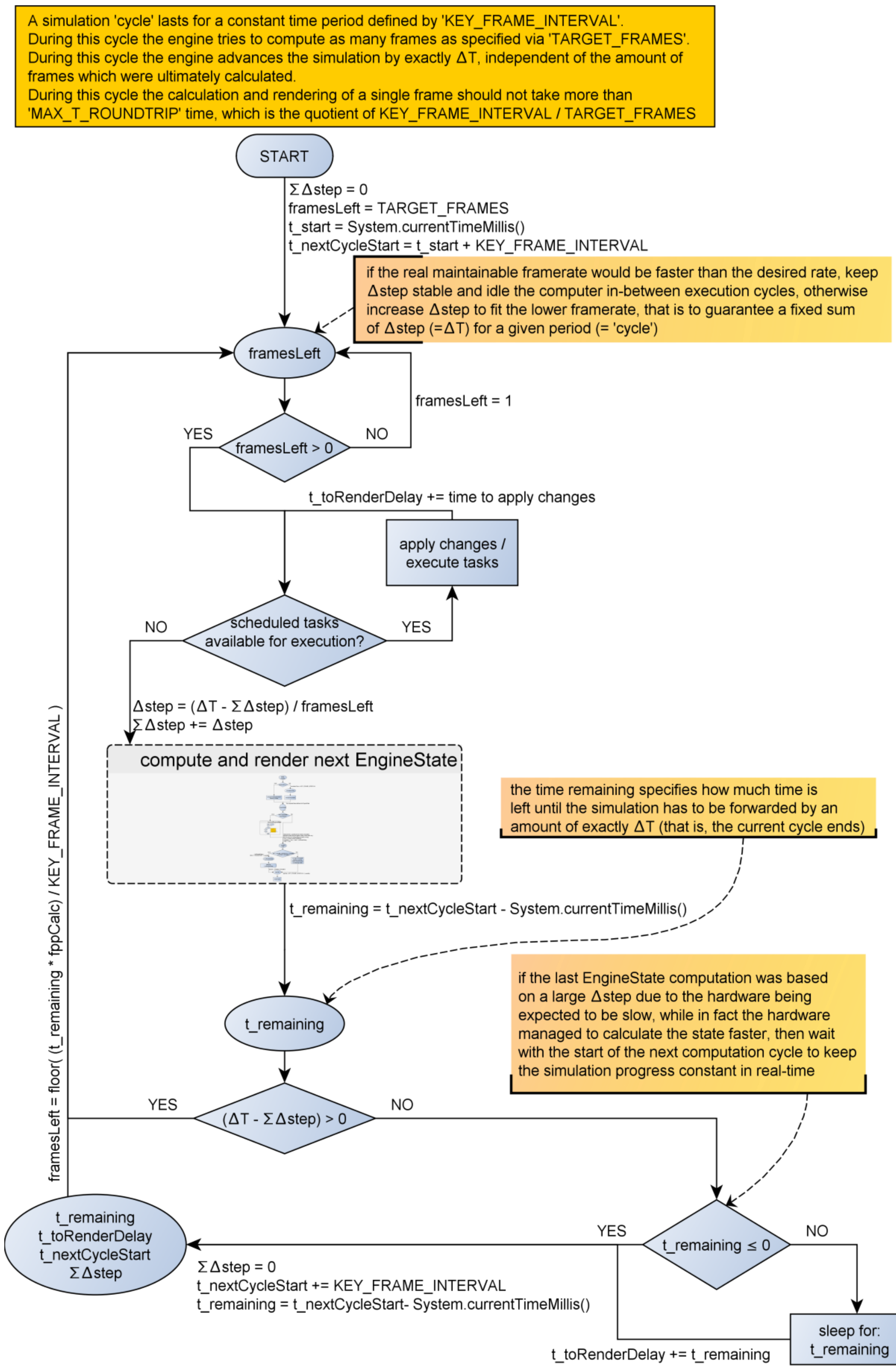

A simulation 'cycle' lasts for a constant time period defined by 'KEY_FRAME_INTERVAL'.
During this cycle the engine tries to compute as many frames as specified via 'TARGET_FRAMES'.
During this cycle the engine advances the simulation by exactly ΔT, independent of the amount of frames which were ultimately calculated.
During this cycle the calculation and rendering of a single frame should not take more than 'MAX_T_ROUNDTRIP' time, which is the quotient of KEY_FRAME_INTERVAL / TARGET_FRAMES
START
ΣΔstep = 0
framesLeft = TARGET_FRAMES
t_start = System.currentTimeMillis()
t_nextCycleStart = t_start + KEY_FRAME_INTERVAL
if the real maintainable framerate would be faster than the desired rate, keep Δstep stable and idle the computer in-between execution cycles, otherwise increase Δstep to fit the lower framerate, that is to guarantee a fixed sum of Δstep (=ΔT) for a given period (= 'cycle')
framesLeft
framesLeft = 1
YES
framesLeft > 0
NO
t_toRenderDelay += time to apply changes
apply changes / execute tasks
NO
scheduled tasks available for execution?
YES
Δstep = (ΔT - ΣΔstep) / framesLeft
ΣΔstep += Δstep
compute and render next EngineState
the time remaining specifies how much time is left until the simulation has to be forwarded by an amount of exactly ΔT (that is, the current cycle ends)
t_remaining = t_nextCycleStart - System.currentTimeMillis()
if the last EngineState computation was based on a large Δstep due to the hardware being expected to be slow, while in fact the hardware managed to calculate the state faster, then wait with the start of the next computation cycle to keep the simulation progress constant in real-time
t_remaining
framesLeft = floor((t_remaining * fppCalc) / KEY_FRAME_INTERVAL)
YES
(ΔT - ΣΔstep) > 0
NO
t_remaining
t_toRenderDelay
t_nextCycleStart
ΣΔstep
YES
t_remaining ≤ 0
NO
ΣΔstep = 0
t_nextCycleStart += KEY_FRAME_INTERVAL
t_remaining = t_nextCycleStart- System.currentTimeMillis()
sleep for: t_remaining
t_toRenderDelay += t_remaining

Figure 4.4: Overall design of the deterministic algorithm for local simulation calculation

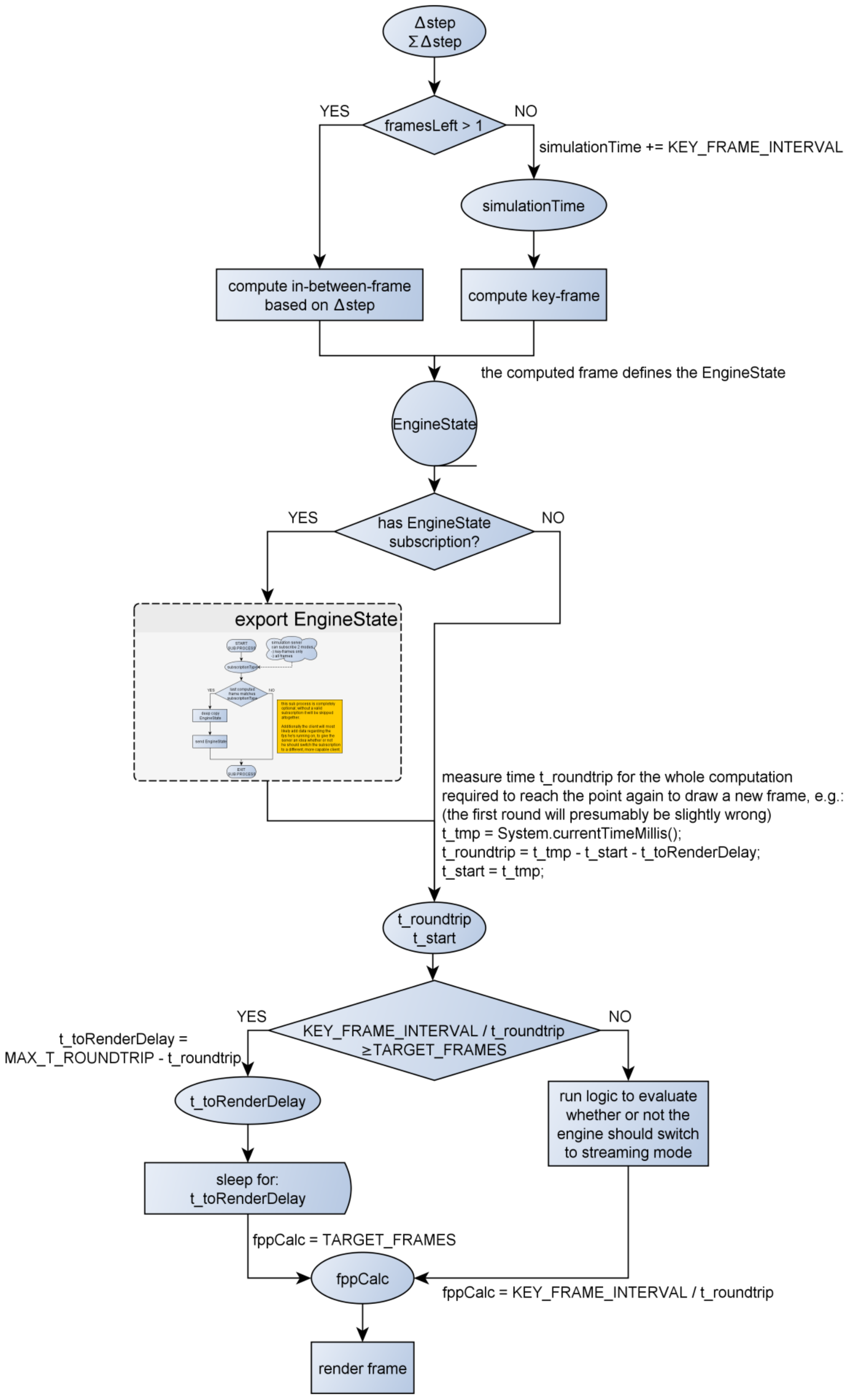

Δstep
ΣΔstep
YES
framesLeft > 1
NO
simulationTime += KEY_FRAME_INTERVAL
simulationTime
compute in-between-frame based on Δstep
compute key-frame
the computed frame defines the EngineState
EngineState
YES
has EngineState subscription?
NO
export EngineState
START SUB PROCESS
simulation server can subscribe 2 modes: -) key-frames only -) all frames
subscriptionType
YES
last computed frame matches subscriptionType
NO
deep copy EngineState
send EngineState
this sub process is completely optional; without a valid subscription it will be skipped altogether.
Additionally the client will most likely add data regarding the fps he's running on, to give the server an idea whether or not he should switch the subscription to a different, more capable client.
EXIT SUB PROCESS
measure time t_roundtrip for the whole computation required to reach the point again to draw a new frame, e.g.: (the first round will presumably be slightly wrong)
t_tmp = System.currentTimeMillis();
t_roundtrip = t_tmp - t_start - t_toRenderDelay;
t_start = t_tmp;
t_roundtrip
t_start
YES
KEY_FRAME_INTERVAL / t_roundtrip ≥TARGET_FRAMES
NO
t_toRenderDelay = MAX_T_ROUNDTRIP - t_roundtrip
t_toRenderDelay
sleep for: t_toRenderDelay
run logic to evaluate whether or not the engine should switch to streaming mode
fppCalc = TARGET_FRAMES
fppCalc
fppCalc = KEY_FRAME_INTERVAL / t_roundtrip
render frame

Figure 4.5: Computational part of the algorithm doing simulation calculation locally

4.5 Details of the Simulation Engine Running in State Streaming Mode

Figure 4.6 summarizes the execution flow when a client runs in the streaming mode. Like the steps illustrated in figure 4.3, this part of the algorithm has not yet been implemented in the prototype.

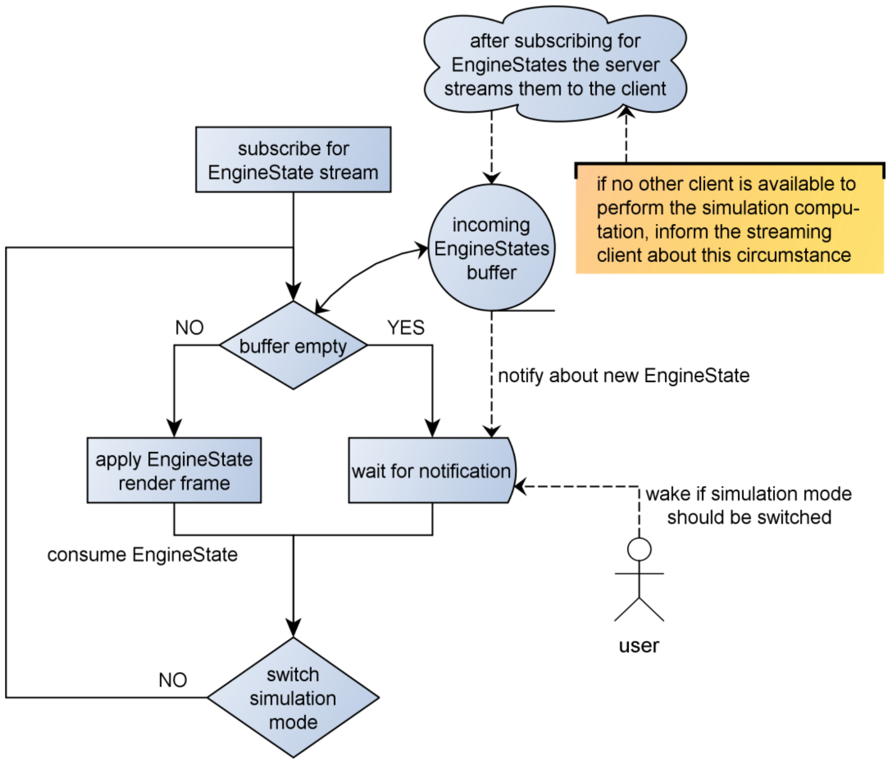

after subscribing for EngineStates the server streams them to the client
subscribe for EngineState stream
incoming EngineStates buffer
if no other client is available to perform the simulation computation, inform the streaming client about this circumstance
NO
buffer empty
YES
notify about new EngineState
apply EngineState render frame
wait for notification
wake if simulation mode should be switched
consume EngineState
user
NO
switch simulation mode

Figure 4.6: Execution flow for simulation engine when streaming simulation states

4.6 Summary

This chapter introduced to the algorithm utilized to run TEALsim's simulations in a deterministic way. It was kept on a rather formal level and did not cover any implementation related details (such as class diagrams). After explaining certain variables which can be used to influence the simulation's pace, based on a short example the concept which keeps simulations with varying frame rates in sync was explained. Furthermore, two reasons were discussed which required the introduction of key frames as an implicit mechanism of synchronization. Ultimately the chapter closed with various flowcharts which illustrated the algorithms execution flow.

Assuming that the mechanism outlined in this chapter removes the requirement to explicitly synchronize the simulation calculations, the second area requiring means for synchronization are user inputs. This topic is covered in the next chapter where multiple use cases specify the client-server architecture's expected system behavior.

Chapter 5

Synchronizing Concurrent Clients

A key benefit of the new client-server architecture is its reduced bandwidth utilization due to reducing network traffic to events triggered by human interactions for most scenarios. However, this requires these user inputs to be dealt with in an appropriate way to ensure that all clients apply these changes in exactly the same way (respectively at the same point of time). For this purpose the server has to act as authoritative instance to validate user inputs and to forward them to all concurrently running clients. The consecutive chapter specifies the expected behavior for this validation logic as a sequence of use cases.

5.1 Use Cases for Most Relevant Design Aspects

Unless explicitly specified in another way the expression '*client*' refers to the TEALsim client-side implementation whereas the expression '*server*' refers to the TEALsim server-side implementation.

The expression '*ControlState*' serves as a placeholder term for something which may be changed by the user but which has to be synchronized between all clients (and therefore has to be previously authorized by the server). An example for this would be the simulation's running state (running or paused). Furthermore, by using the term ControlState to describe something which has to be synchronized implies that changes to this element (after authorization) will also change all of its associated elements (e.g. if 'ControlState' relates to the value of a text field, which is used to specify the charge value of an arbitrary amount of point charges, changing this ControlState will subsequently also change the point charges).

'*SimulationTime*' is defined as the sum of |ΔT| the simulation engine progressed the simulation, hence a paused simulation does not change its SimulationTime.

Use case 5.1: Multiple clients running with out-of-sync SimulationTime	
Pre-condition	2 clients (A and B) opened the same simulation, connected (and therefore synchronized) with the server, and the simulation is paused
Story	Client A requests to start the simulation. Due to the authoritative role of the server, the request to start the simulation is forwarded to the server before anything may happen. Subsequently, the server verifies whether or not the requested ControlState change may be authorized (see the succeeding use cases for more details). Assuming that the change is authorized, all of the clients connected to the server are notified about the new state (that is, to start the simulation). Due to differing latencies between the server and its connected clients, all of the clients will ultimately start running their simulation at a different point of (real world) time.
Post-condition	All clients are running but ultimately they might be out of sync assuming an omniscient supervisor capable of tracking such marginal differences.
Remark	Assuming latencies to be in the area below 1000 [ms] for modern networks, having 2 clients next to each other connected e.g. via LAN to a local server, the overall difference of alternating SimulationTimes between both clients should not be visible with the human eye. While it probably may never be possible to achieve a 100% synchronization, certain techniques could be implemented to decrease the effective difference in SimulationTime amongst all clients. For example clients could be slightly fast forwarded at simulation start depending on their corresponding latencies.

Use case 5.2: Change of ControlState without user input collisions and running simulation	
Pre-condition	2 clients (A and B) opened the same simulation, connected (and therefore synchronized) with the server, and the simulation is running
Story	Client A causes a change to a ControlState (e.g. by user input). Client A logs the SimulationTime and sends a ControlState change request to the server. The server receives the ControlState change and verifies in its internal lookup table that the changed element was not changed by any other client at a later SimulationTime. Upon successful validation of this constraint the server broadcasts the ControlState change to all connected clients and updates its internal ControlState (that is, record the state and timestamp of the last change). The broadcasted ControlState change message contains a field telling the clients at which SimulationTime to execute the state change. For details regarding the calculation of this SimulationTime value, see chapter 5.2. All of the connected clients receive the broadcast message and react on it depending on which of the following two possible scenarios applies: a) For example if they have a fast connection to the server (that is, low latency) their internal SimulationTime might be before the timestamp in the ControlState change message specifying the point of time to update the simulation. In this case the particular client caches the received order and executes the change as scheduled.

	b) Contrariwise, if they would have a slow connection to the server their simulation will be beyond the point of time specified in the ControlState change message. In this case they have to record their current SimulationTime and roll back the simulation by negating ΔT until the point of time the ControlState change has to be executed. Then the change gets applied and the simulation is fast forwarded to the original SimulationTime.
Post-condition	All clients and the server share the same ControlState. At a global point of time X (that is, real world time) it is not guaranteed that clients display exactly the same rendered view because it is not guaranteed that their internal SimulationTime is exactly the same relative to point X. It is guaranteed though that at a given SimulationTime clients will eventually show exactly the same rendered view.

Use case 5.3: Change of ControlState <u>without</u> user input collisions and paused simulation	
Pre-condition	2 clients (A and B) opened the same simulation, connected (and therefore synchronized) with the server, and the simulation is not running
Story	Client A causes a change of a ControlState and sends a request to the server. The server receives the ControlState change, runs the verification process (which will always succeed – see use case 5.6 for additional details) and sends a broadcast message to all clients with SimulationTime set to: *(SimulationTime from change request)* All of the connected clients receive the broadcast message and apply the contained change.
Post-condition	All clients and the server share the same ControlState.
Remark	This use case is related to use case 5.2 – due to this reason some technical details have been omitted.

Use case 5.4: Change of ControlState <u>with</u> user input collisions – case 1 – trailing client B	
Pre-condition	2 clients (A and B) opened the same simulation, connected (and therefore synchronized) with the server, and the simulation is running
Story	Client A causes a change U of ControlState X. Client A logs the SimulationTime Y and sends a change request to the server. The server receives, (successfully) verifies and broadcasts the ControlState change to all connected clients. Client B causes a change V of ControlState X before receiving respectively applying the broadcasted change U from the server. Client B logs the SimulationTime (Y - Z) [where Z > 0] and sends a change request to the server. The server receives the ControlState change but the verification process fails causing the change not to be authorized (that is, not sent back). Eventually all (including client B) connected clients will receive the broadcasted

	change U originating from client A and update their simulation accordingly. Additionally client B might receive a notification that he is lagging behind compared to other clients which caused one change to be dropped.
Post-condition	All clients and the server share the same ControlState based on the change U caused by client A.
Remark	Constraint: in general, clients' SimulationTime is assumed to be not too far off amongst each other (see use case 5.1). This use case is related to use case 5.2 – due to this reason some technical details have been omitted.

Use case 5.5: Change of ControlState <u>with</u> user input collisions – case 2 – advanced client B	
Pre-condition	2 clients (A and B) opened the same simulation, connected (and therefore synchronized) with the server, and the simulation is running
Story	Client A causes a change U of ControlState X. Client A logs the SimulationTime Y and sends a change request to the server. The server receives, (successfully) verifies and broadcasts the ControlState change to all connected clients. Client B causes a change V of ControlState X before receiving respectively applying the broadcasted change U from the server. Client B logs the SimulationTime (Y + Z) [where Z > 0] and sends a change request to the server. The server receives the ControlState change and verifies it. In this case the server's verification process will succeed, causing it to save the new state V and issue a new broadcast to all clients with the state of client B. Eventually all clients will receive the broadcasted change U originating from client A and update their simulation accordingly. Subsequently, all clients will receive the change V originating from client B. Regular rules apply with regard to the way clients deal with simulation changes; that is, they will – if needed – roll back their simulation state to the timestamp value contained in the change message, apply the change and fast forward the simulation to the former point of time.
Post-condition	All clients and the server share the same ControlState based on the change V caused by client B.
Remark	Constraint: messages are expected to arrive in fixed order. This use case is related to use case 5.2 – due to this reason some technical details have been omitted.

Use case 5.6: Change of ControlState <u>with</u> user input collisions – case 3 – equal SimulationTime	
Pre-condition	2 clients (A and B) opened the same simulation, connected (and therefore synchronized) with the server, and the simulation is running
Story	Client A causes a change U of ControlState X. Client A logs the SimulationTime Y and sends a change request to the server.

	The server receives, (successfully) verifies and broadcasts the ControlState change to all connected clients. Client B causes a change V of ControlState X before receiving respectively applying the broadcasted change U from the server. Client B logs the same SimulationTime Y and sends a change request to the server. The server receives the ControlState change, runs the verification process and reacts on it depending on which of the following two possible scenarios applies: 1. The simulation is in the state 'running' with both clients causing a contradicting change at exactly the same point of SimulationTime, but for example client B's latency is greater than client A's. This issue is resolved by applying a first-come, first-serve paradigm on the server resulting in the rejection of client B's change and preferring the change arriving sooner. 2. The simulation is currently paused (therefore the SimulationTime is temporarily constant). Since successive changes will always have the same timestamp none of them may be dropped. Therefore any change will overwrite prior ones (resulting in new change broadcasts).
Post-condition	1. If the simulation is in running state all clients and the server will share the same ControlState based on the change U caused by client A. 2. If the simulation is in paused state all clients and the server will share the same ControlState based on which change was received last by the server.
Remark	This use case is related to use case 5.2 – due to this reason some technical details have been omitted.

Use case 5.7: Change of ControlState <u>with</u> user input collisions – case 4 – equal ControlState	
Pre-condition	2 clients (A and B) opened the same simulation, connected (and therefore synchronized) with the server, and the simulation is running
Story	Client A causes a change Y of ControlState X. Client A logs the SimulationTime and sends a change request to the server. The server receives, (successfully) verifies and broadcasts the ControlState change to all connected clients. Client B causes a change Y of ControlState X before receiving the broadcasted change from the server. Client B logs the SimulationTime and sends a ControlState change request to the server. The server receives the ControlState change and runs the verification process. Since the state of ControlState X on the server is equal to the requested change by client B, the request by client B gets dropped silently. Eventually all (including client B) connected clients will receive the broadcasted change originating from client A and update their simulation accordingly.
Post-condition	All clients and the server share the same ControlState based on the change caused by client A which is equal to the change requested by client B.
Remark	This use case is related to use case 5.2 – due to this reason some technical details have been omitted.

5.2 Details Regarding SimulationTime Calculation

Calculating the points of time to apply requested ControlState changes, which are used in the broadcast messages sent from the server to all clients, is a sensible task since inappropriate values will have a clearly visible negative feedback on running simulations. Too early figures will force clients with slow connections to excessively roll back the simulation and therefore cause a jerky visual experience. Too late time values will circumvent this particular issue. However, they will give the feeling that the whole system is unresponsive. In the end, supposedly a lot of brainpower and time could be invested to come up with a sophisticated formula to deal with this problem which was beyond the scope of this thesis. For this reason the following formula is suggested as an initial approach which can be improved at a later point of time:

	(SimulationTime from change request)
+	(originating client's 1-way latency)
+	(median 1-way latency of all connected clients)
=	SimulationTime to apply changes, rounded up to the next key frame

While this formula naturally causes some delay for user inputs, it may reduce the amount of simulation roll-back computation required on other clients (and therefore graphic 'stuttering'). Additionally, this design pays tribute to the idea to reward "high efforts" in a sense that clients can directly influence the perceived feeling of input lag by improving their own network connection and therefore reduce one of the delay factors.

The second factor which influences the delay of user interactions can be regarded as some kind of social parameter to improve the experience for slower clients. While theoretically this value could be abused for fraud (in a sense of disturbing the experience for other clients), its practical security relevance is supposedly minor since it requires approximately as much evil clients as those who are present with good ambitions.

5.3 Security

The system was designed to be used in a non-competitive environment which was assumed to be free of aggressors (that is, there are no incentives to cheat). Due to this reason security concerns did not influence the design of the implementation in general. Evident security risks were tried to be addressed as good as possible unless it involved a serious effort of redesign or increase of complexity. Throughout this thesis known security issues are mentioned purely for the sake of documentation. For the purpose of building a secure system an independent, in-depth analysis would be required which presumably would require a huge amount of work.

5.4 Summary

This chapter summarized the expected system behavior when simulation elements are changed by clients. In other words it is a high level specification outlining the decision algorithms required on the client side and the server side to ensure a coherent execution flow. For this purpose various likely use cases were covered on a formal level along with their proposed resolutions. The overall goal of the design is to achieve a smooth visual experience for users employing the software not under lab conditions but instead in a real, rather large, distributed network e.g. the internet. In general the design builds upon the idea that changes by users are not executed immediately but are slightly delayed. This delay comes from sending a change request to the server to get permission for the user input. If authorization is granted the server informs all clients about the change and orders them to apply this change at some point of time in the future. Calculating this future point of time is critical to ensure a smooth visual representation of the simulation. A first approach for this calculation is outlined, explaining the involved parameters. This chapter completed the formal introduction to this thesis' work. Subsequently more implementation specific chapters follow, with the next chapter explaining the less technical steps perform, which are nevertheless important for a large scale project. In this discussion topics are covered such as a summary of the quality level of TEALsim's code base or which auxiliary tools for version control, programming and logging were used.

Part II
Implementation Details

Chapter 6

State of the TEALsim Project

At the point of time when programming activities - as part of this thesis' scope - were stopped, TEALsim [Massachusetts Institute of Technology, 2012] was not in a releasable state. As a matter of fact this situation held true at the beginning of the work for this thesis and did not change until its completion due to various reasons. While there was certainly a point of time in history when the framework appeared to people – at least those who were not intensely engaged in the project – to work pretty well (see the binaries compiled with older code[27]), several factors negatively impacted the project and pushed the trunk of the source code step by step back into a state resembling a demonstration version instead of something which could be used reliably in a real teaching environment.

Amongst others the following reasons can be identified which lead to the current situation:

a) *Student projects* – over the course of the last couple of years a lot of the work to enhance TEALsim has been done in the form of student projects. Taking into consideration *output limiting constraints* like those listed hereafter it becomes evident that the intrinsic desire of students to come up with a maximum amount of interesting findings – to support an extensive thesis like this one – inevitably results in a conceivable low quality of the created source code. In other words, in the case of TEALsim students usually preferred the pursuit of ideas for new features (and thus documentable results) at the expense of software quality. Factors which limit the possible output of mature code are for example:
    - an approximate length of only 6 months for a student project
    - the considerable size of TEALsim paired with its lack of high-quality in-depth (design-) documentation

[27] TEALsim binaries based on older code - http://web.mit.edu/viz/EM/simulations/

- seriously complicated code deriving from the inconsistent software design and abandoned, broken or deferred features
- substantial amount of time required to properly test a software which shall be released to a public user base

b) *Lack of (paid,) continued work* on the framework – this item has been partially suggested in the former paragraph. The lack of paid and therefore continuing, dedicated work to funnel deviating feature branches, created by temporary collaborators, back into the trunk to match a unified style and design paradigm exacerbated the problem of diminishing software quality. While the absence of detailed documentation or system specification itself indicates that the framework's design was mostly defined on-the-fly (and therefore was condemned to eventually become inconsistent over the years – see the Lava Flow AntiPattern [Brown, Malveau, McCormick III, & Mowbray, 1998]), the missing authority to evolve and watch over the code ultimately amplified the lava-flow like situation.

c) *Changing client hardware/software* – naturally software and hardware is evolving rapidly over the years. Given the fact that the TEAL project was started in the year 2000 [Scheucher, 2010] where Java version 1.3 was the most recent library available, TEALsim has faced more or less comprehensive new client environments at least a couple of times throughout its lifetime. This circumstance implies the requirement for code maintenance activities of various scales, which in turn becomes an issue in absentia of dedicated workforce (see the former paragraph).

As mentioned TEALsim appears as a perfect example for a project which evolved over time matching the explanations for the Lava Flow AntiPattern given in [Brown, Malveau, McCormick III, & Mowbray, 1998]. To further stress the significance of this AntiPattern, following is a quote of the most evident consequences (that is, drawbacks in the case of an 'anti' pattern) with a subsequent check whether or not the particular assertion also held true for the TEALsim framework:

a) "*Lava Flows are expensive to analyze, verify, and test. All such effort is expended entirely in vain and is an absolute waste. In practice, verification and test are rarely possible.*"

b) "*Lava Flow code can be expensive to load into memory, wasting important resources and impacting performance.*"

c) "*As with many AntiPatterns, you lose many of the inherent advantages of an object–oriented design. In this case, you lose the ability to leverage modularization and reuse without further proliferating the Lava Flow globules.*" [Brown, Malveau, McCormick III, & Mowbray, 1998]

Brief check of existence of Lava Flow AntiPattern issues in TEALsim source code:

assertion	applicable to TEALsim?	description
a)	✓	Dozens, if not hundreds, of hours were spent analyzing program flow and object creation to trace bugs or find out a way to extend the software. Like mentioned in chapter 6.4 for example, prior to this thesis hardly any more comprehensive and centralized (J)unit tests were available.
b)	✓	A good example for this behavior was the rendering system which was fed a list of objects to render which contained every item twice.
c)	✓	Cascades of interfaces assigning roles to classes which made no sense and were not used, e.g. the *SimPlayer* ultimately was also a *TElementManager* thus forced to implement all its methods while in fact they were never used (or if so, it was an erroneous usage)

Table 6.1: Lava Flow AntiPattern issues found in TEALsim source code

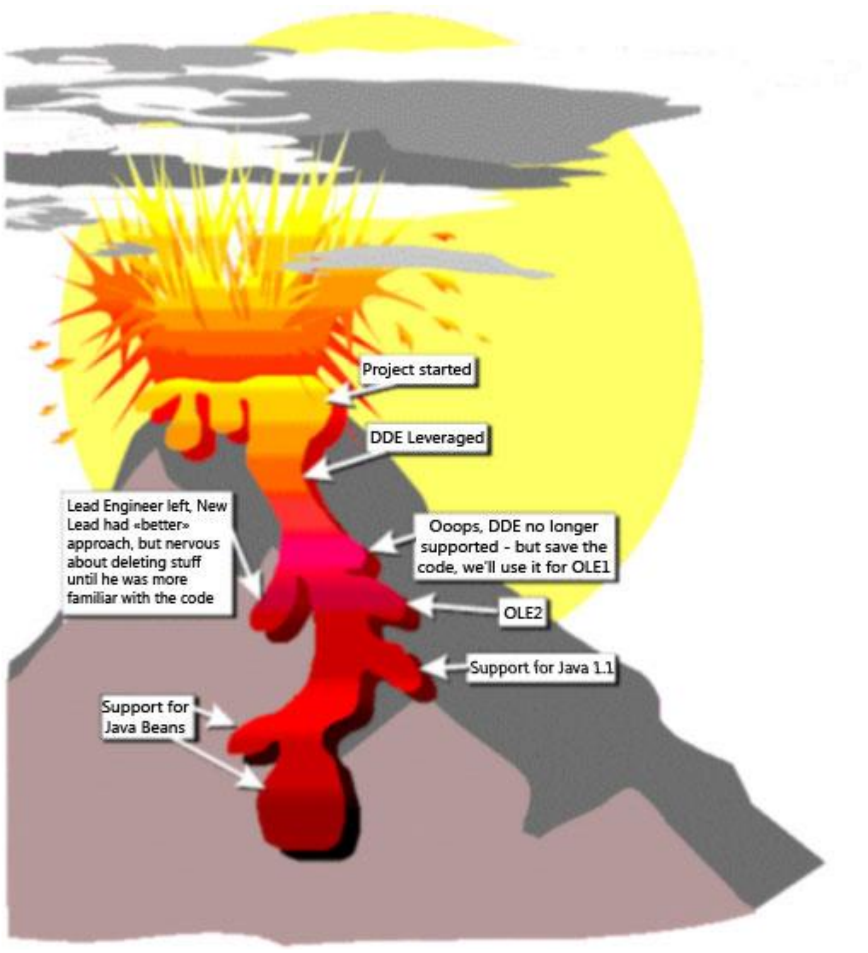

Project started
DDE Leveraged
Lead Engineer left, New Lead had «better» approach, but nervous about deleting stuff until he was more familiar with the code
Ooops, DDE no longer supported - but save the code, we'll use it for OLE1
OLE2
Support for Java 1.1
Support for Java Beans

Figure 6.1: Illustration of the Lava Flow AntiPattern – Source: [Brown, Malveau, McCormick III, & Mowbray, 1998]

Consequentially one part of the practical work for this thesis was to clean up as many dead 'features' and design inconsistencies as possible. To illustrate the results of this effort the next chapter 6.1. summarizes and comments the changes to the code base of TEALsim. Although this task arguably changed the code base to the better, the description for the Lava Flow AntiPattern in [Brown, Malveau, McCormick III, & Mowbray, 1998] also gives a hint why the framework seemingly worked worse after the end of this thesis than it did before:

"*As suspected dead code is eliminated, bugs are introduced. When this happens, resist the urge to immediately fix the symptoms without fully understanding the cause of the error.*"

In simple terms, the time spent coding was too short to reach the break-even point where the runtime behavior of the framework inarguably exposed apparent improvements compared to its former state.

6.1 Issues with Redundant and Obscure Code

As mentioned one of the big challenges encountered during the course of this project was to understand the 'design' of the TEALsim framework and track down its execution flow. Code artifacts of abandoned features and inconsistent design decisions bloated the amount of code to analyze. The following table 6.2 summarizes the work performed to clean up the framework's code based on approximate figures.

	files	blank lines	lines of comment	lines of code
feature branch of TEALsim with SVN revision 150	602	17629	31286	74852
network, command-line argument and test packages added after original check-in to SVN repository	46	772	1930	2654
resulting leftover from original check-in	556	16857	29356	72198
original check-in to SVN repository (revision 2)	582	17902	29534	76205
variance of leftover to original content of check-in	-26	-1045	-178	-4007

Table 6.2: Overview of evolution of LoC respectively amount of files of TEALsim project[28]

Most mentionable for table 6.2 is the reduction in lines of code by approximately 4 kLOC which would equal to ~5% of the whole project. Usually such code consisted of classes forced to implement unused methods due to them implementing inappropriate interfaces, or derived

[28] Figures calculated with the tool CLOC - http://cloc.sourceforge.net/

classes repeatedly re-implementing the same methods instead of calling into their base class. As reference for ‘new code submits’ the packages containing TEALsim’s new network capabilities, the command-line argument parser and the JUnit test suite were chosen, since these were definitely non-existent previous to this thesis. While a magnitude of changes to files in other packages were applied as well, taking them into exact consideration is arguably quite challenging, for which reason it was assumed that the measurable effects of these changes would even out each other.

	files	blank lines	lines of comment	lines of code	lines of comments per line of code
network and command-line argument pack-ages added after original check-in	42	645	1813	2102	0,862511893
resulting leftover from original check-in	556	16857	29356	72198	0,406604061

Table 6.3: Overview of amount of in-line documentation of TEALsim project[29]

To emphasize the improving quality of the framework’s code emanating from this thesis’ practical work, table 6.3 summarizes the ratio of lines of comments to lines of code for old code versus new code submits. Apparently a degree of documentation more than twice as comprehensive as prior outlines the improving software quality.

Despite the fact that at this thesis’ start it was of importance to provide means to effectively test the code base, eventually the attention shifted over to eliminate design flaws and implement new features (e.g. for network capabilities) which left behind the testing suite mainly as a solid starting point for future work in this area. Therefore code comprising the JUnit tests is rather unsophisticated in its current state, generally consisting of copied and pasted boilerplate code, which is the reason why the test package code was not included in table 6.3.

	files	blank lines	lines of comment	lines of code	lines of comment per line of code
feature branch of OWL TEALsim module with SVN revision 150	21	437	547	1475	0,370847458
original check-in of OWL TEALsim module	23	783	810	2731	0,296594654
change of project contents from check-in to rev. 150	-2	-346	-263	-1256	0,074252804

Table 6.4: Overview of evolution of OpenWonderland TEALsim module project[30]

[29] Figures calculated with the tool CLOC - http://cloc.sourceforge.net/

For the sake of completeness table 6.4 confirms that changes to the TEALsim framework did not come at the cost of increased complexity in related projects like the Open Wonderland module. Correlative to the developments in the TEALsim code base, the TEALsim OWL module code base was reduced in complexity while getting slightly more thoroughly documented alike. Effectively the changes turned the module into a real player-like wrapper, responsible to embed TEALsim into OWL without redefining single, technical aspects or even whole parts of the design of the simulation framework.

6.2 Move from CVS to SVN

One of the first activities was the migration from the formerly used CVS revision control system to the more recent SVN[31] system. Performing this step before anything else was a logic consequence, since experience has shown that the move from one revision system to another one often implies the loss of various information while additionally the necessary amount of time to accomplish the migration increases with the amount of information contained in the former system. Motivations for this task were amongst others:

- The extended functionality of SVN, which allowed for example to easily move folders (e.g. via drag and drop within Windows Explorer using TortoiseSVN[32]).
- A unified workflow while developing TEALsim in parallel to OWL and the corresponding module (since OWL used SVN as well).
- In general the broader personal experience with SVN compared to CVS of people involved in this project.

The SVN host of choice was Sourceforge where currently all of the source code is publically available[33].

6.3 Integration into Netbeans and Rework of the Build Script

Next on list was the migration of the TEALsim project from Eclipse to Netbeans. Motivation for this work was again the desire to create a more streamlined development workflow, since Open Wonderland itself (and all of its related projects created respectively maintained by the Open Wonderland Foundation), the TEALsim OWL module as well as one of its most important

[30] Figures calculated with the tool CLOC - http://cloc.sourceforge.net/
[31] Subversion revision control system - http://subversion.apache.org/
[32] Tortoise SVN client - http://tortoisesvn.tigris.org/
[33] TEALsim project hosted on Sourceforge - http://sourceforge.net/projects/tealsim/

frameworks – the MTGame Graphics Engine[34] – are also Netbeans projects. Unifying the amount of required integrated development engines (IDE) to a minimum not only decreases the amount of time required to setup a working station and get involved in the project, but also leverages productivity with the concrete IDE by allowing usage of advanced features such as step-by-step debugging including stepping into code of separated projects maintained within the same IDE.

Getting everything right to allow usage of all of Netbeans' features usually becomes a serious challenge without profound knowledge of Ant[35] build scripts in general and Netbeans' way to deal with things in particular. Adding into this consideration the complex structure of the former build script (see figure 6.3), which existed without any documentation or any formal system specification for the deliverables, a serious amount of time was required to achieve a tight and properly working integration with Netbeans. Figure 6.2 shows the final *build.xml* file used by Netbeans to build, debug and profile the TEALsim project.

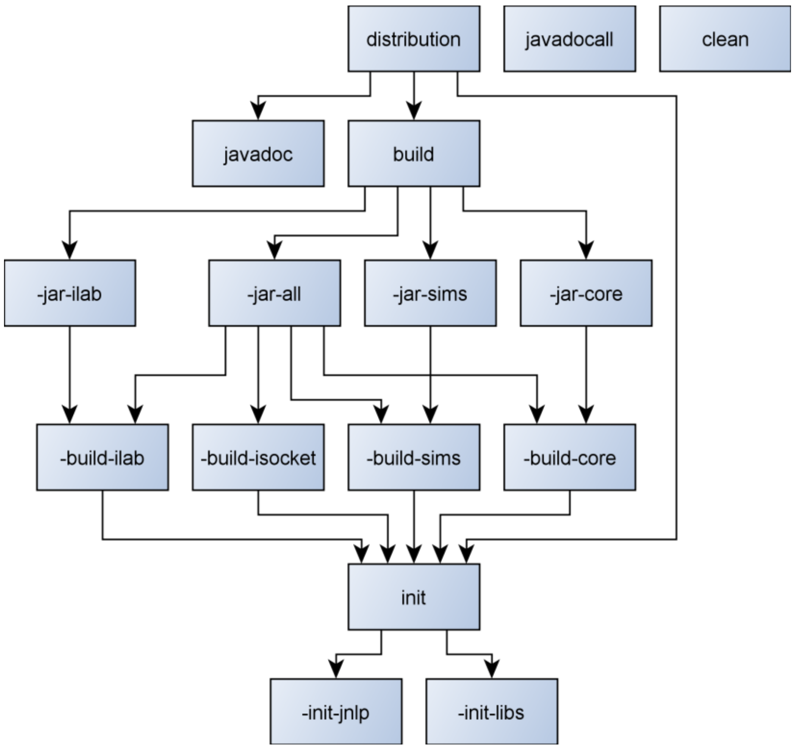


Figure 6.2: Dependency graph of the streamlined TEALsim build file

[34] MTGame Graphics Engine - http://code.google.com/p/openwonderland-mtgame/
[35] Ant software build tool - http://ant.apache.org/

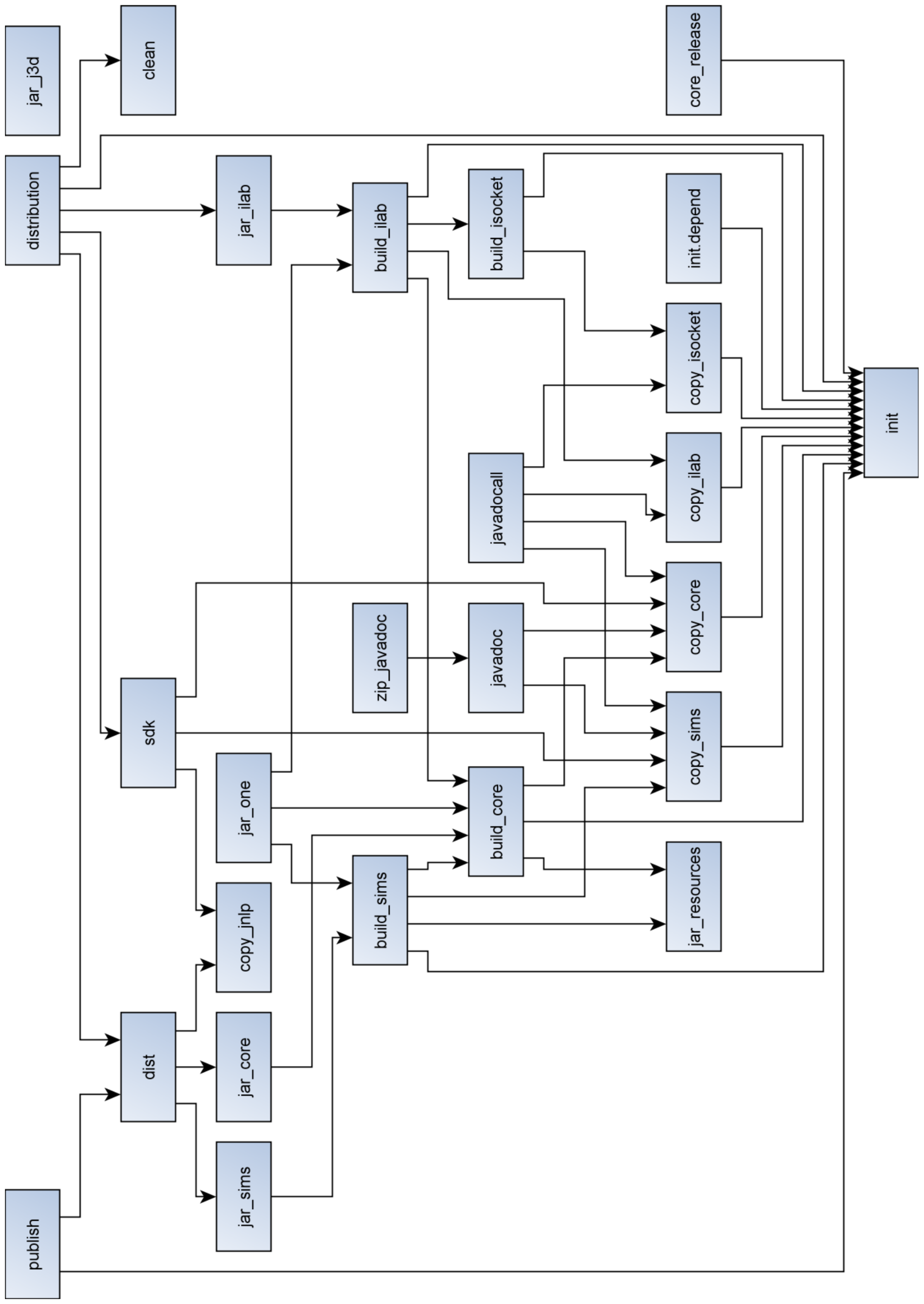


Figure 6.3: Dependency graph of the former TEALsim build script

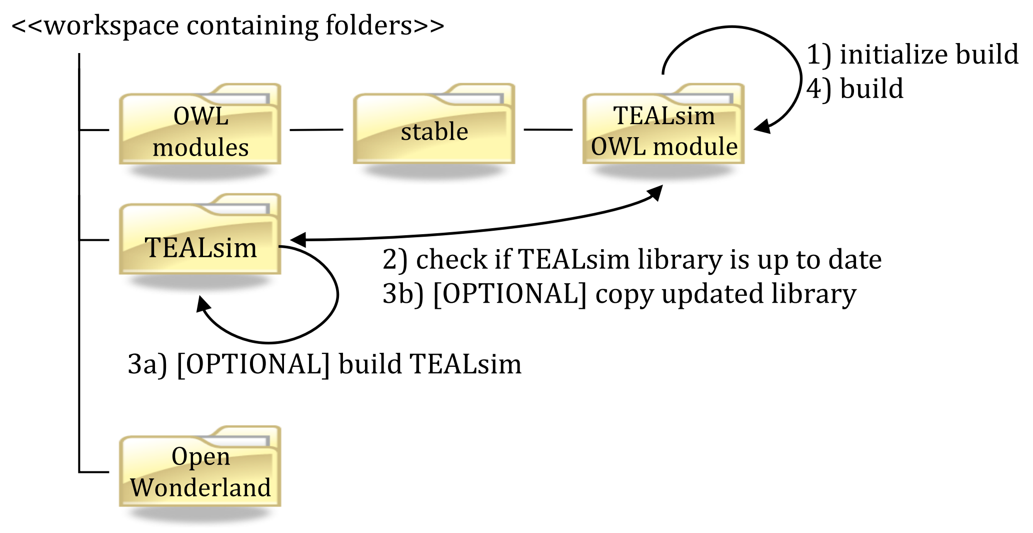

<<workspace containing folders>>
OWL modules
stable
TEALsim OWL module
1) initialize build
4) build
TEALsim
2) check if TEALsim library is up to date
3b) [OPTIONAL] copy updated library
3a) [OPTIONAL] build TEALsim
Open Wonderland

Figure 6.4: Illustration of TEALsim's OWL module build sequence

Besides the TEALsim project, also the TEALsim OWL module project was adapted slightly. Most mentionable the module's build script became considerably smarter by usage of a custom Ant-Contrib Tasks[36] build, which allows to check if the module uses the most recent TEALsim library file. In this process the script checks if compiling TEALsim would yield a more recent binary and if so triggers the build and fetches the new library. Figure 6.4 outlines this sequence based on the recommended folder structure to setup a working environment for development of all involved projects.

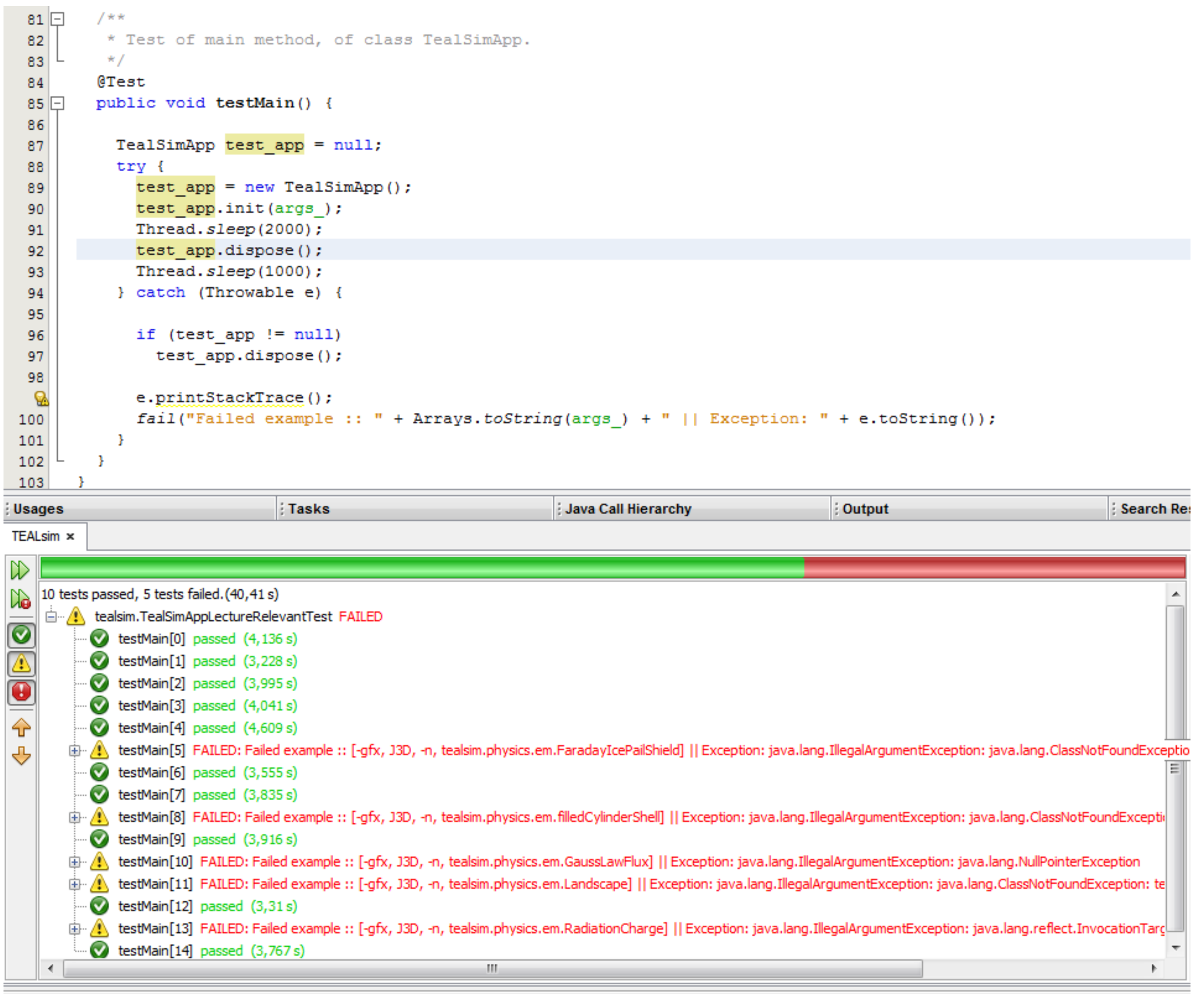
6.4 Addition of JUnit Tests

Integral part of any project but maybe very experimental R&D prototypes is the testing suite. Since TEALsim did not include any extensive package for this task prior to this thesis, an inevitable duty was to introduce a suitable framework for this purpose. The library of choice was JUnit[37] since it "[...] *is the de facto standard for Java unit testing.*" [Burke & Coyner, 2003, p. 59] Another advantage of JUnit is its tight integration into the Netbeans IDE (see figure 6.5 for an example) which allows for a smooth workflow while working on the TEALsim code. Due to the amount of time required for other areas of this thesis, unfortunately the testing aspect ultimately ended up being treated rather poorly again. One major issue currently limiting the usefulness of tests was a problem with the JME[38] graphics library which refused to close its canvas between single tests, making it impossible to test TEALsim with this particular graphics setting.

[36] Ant-Contrib Tasks - http://ant-contrib.sourceforge.net/
[37] JUnit project page - http://junit.sourceforge.net/
[38] JME graphics framework (see the Wonderland branch) - http://code.google.com/p/jmonkeyengine/

Presumably the root for this error is way down in the source code of the JME framework. Eventually search for this bug was postponed because of its complexity.

One of the currently available tests resembles a black-box style test[39], checking the proper, exception-free startup of TEALsim based on 2 lists of simulations: one comprehensive list, covering almost all available simulations and one subset of the former list consisting of a selection of important simulations used by John Belcher in his classes at MIT. A few other tests exist as well, checking the correct execution of command-line arguments or the determinism of the simulation engine, but definitely more work in this area would be required.


```java
    /**
     * Test of main method, of class TealSimApp.
     */
    @Test
    public void testMain() {

      TealSimApp test_app = null;
      try {
        test_app = new TealSimApp();
        test_app.init(args_);
        Thread.sleep(2000);
        test_app.dispose();
        Thread.sleep(1000);
      } catch (Throwable e) {

        if (test_app != null)
          test_app.dispose();

        e.printStackTrace();
        fail("Failed example :: " + Arrays.toString(args_) + " || Exception: " + e.toString());
      }
    }
  }
```

```
10 tests passed, 5 tests failed.(40,41 s)
tealsim.TealSimAppLectureRelevantTest FAILED
  testMain[0] passed (4,136 s)
  testMain[1] passed (3,228 s)
  testMain[2] passed (3,995 s)
  testMain[3] passed (4,041 s)
  testMain[4] passed (4,609 s)
  testMain[5] FAILED: Failed example :: [-gfx, J3D, -n, tealsim.physics.em.FaradayIcePailShield] || Exception: java.lang.IllegalArgumentException: java.lang.ClassNotFoundExceptio
  testMain[6] passed (3,555 s)
  testMain[7] passed (3,835 s)
  testMain[8] FAILED: Failed example :: [-gfx, J3D, -n, tealsim.physics.em.filledCylinderShell] || Exception: java.lang.IllegalArgumentException: java.lang.ClassNotFoundExcepti
  testMain[9] passed (3,916 s)
  testMain[10] FAILED: Failed example :: [-gfx, J3D, -n, tealsim.physics.em.GaussLawFlux] || Exception: java.lang.IllegalArgumentException: java.lang.NullPointerException
  testMain[11] FAILED: Failed example :: [-gfx, J3D, -n, tealsim.physics.em.Landscape] || Exception: java.lang.IllegalArgumentException: java.lang.ClassNotFoundException: te
  testMain[12] passed (3,31 s)
  testMain[13] FAILED: Failed example :: [-gfx, J3D, -n, tealsim.physics.em.RadiationCharge] || Exception: java.lang.IllegalArgumentException: java.lang.reflect.InvocationTarg
  testMain[14] passed (3,767 s)
```

Figure 6.5: Screenshot taken in Netbeans IDE showing the result of a JUnit test run

[39] *"A test that does not have access to the internals of the object being tested. Also known as a 'functional test.'"* [Hamill, 2005, p. 185]

Addition of a Mocking Framework

To enable very generic testing of certain components an additional framework providing mocking features was also added to TEALsim. The library of choice was EasyMock[40]. An example of its usefulness is the test which checks the soundness of the algorithm responsible to execute the *RuntimeArguments* according to the declared specifications (e.g. fail if a developer accidentally declared arguments depending on each other, creating a circular reference). In the concrete test an instance of each available type of *RuntimeArguments* gets generically created, and this set of arguments is fed to the execution logic. Since the algorithm will eventually execute every single argument – where certain arguments might try to interact with an environment unavailable within the particular test run – the called method on the *RuntimeArgument* has to be mocked to prevent exceptions breaking the test. A mocking library like EasyMock takes care of this task, providing e.g. functions to record calls to mocks. In the test mentioned above every argument has to be executed exactly once to pass the test. Adding mocking capabilities to the test suite enables very elegant, precise tests without the need to create a huge amount of hacks to instantiate an emulation of an expected environment hosting the elements to test.

6.5 Addition of a Logging Framework

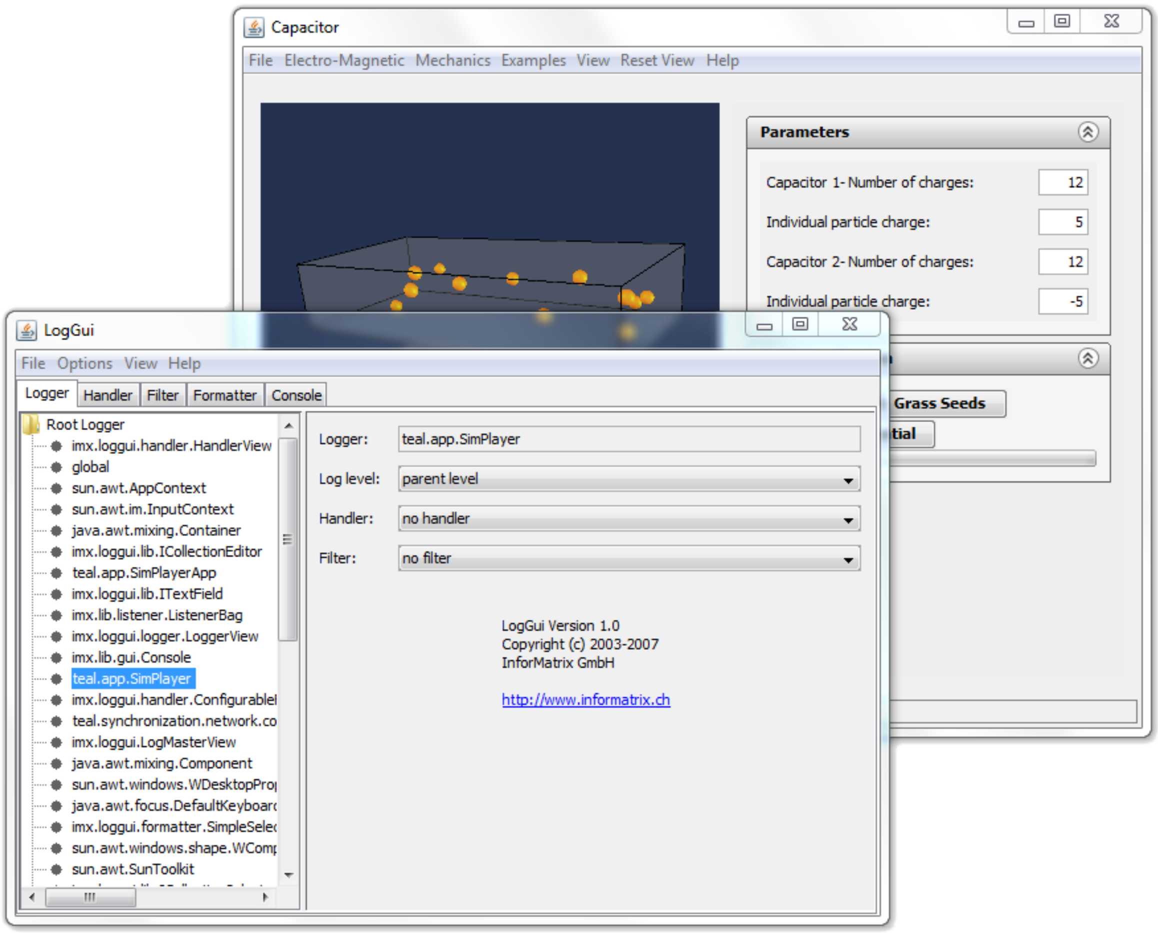
Another missing component indispensable for any larger project was a capable logging facility to allow dynamic adjustment of log settings via e.g. helper windows. Previously there existed a rudimentary class providing static methods which essentially allowed printing various types of objects to *System.out*. This approach suffered from several drawbacks:

a) Analogous to the rest of the project hardly any documentation of the available methods.
b) Similarity to old-style C code by parameterization via plain integers instead of e.g. enum types, making it even more difficult to guess the right usage in absence of (inline) documentation.
c) Due to the fact that everything related to logging constituted real source code, recompilation was required when changes to the log output should be made.

To solve these issues any new code written consistently utilized the standardized logging facilities included in the Java framework since JDK 1.4[41]. Many of the former debugging messages were changed to do so as well. In the face of the amount of code constituting the TEALsim framework unfortunately not all of the debugging messages utilizing the former debug system could be changed to use the JDK version – this should be done incrementally as further work is done to enhance the framework.

[40] EasyMock project - http://www.easymock.org/
[41] JDK logging facilities - http://docs.oracle.com/javase/6/docs/api/java/util/logging/package-summary.html

Apparently the JDK logging utilities tackle all of the three mentioned problems of the self-made debugging system, adding on top the benefits of outsourcing the whole logging package (thus no work for maintenance respectively further development to extend it has to be done) as well as being a widely recognized standard which allows to use other 3rd-party applications to plug into the logging system for even more sophisticated log analyses. As an example for this serves the currently included LogGui [InforMatrix GmbH, 2007] library, which opens up a separate logging window (see figure 6.6) to allow adjustments to the logging output at application runtime. For details on how to start TEALsim with the LogGui window see chapter 8.1.2.


Capacitor
File Electro-Magnetic Mechanics Examples View Reset View Help
Parameters
Grass Seeds
tial
LogGui
File Options View Help
Logger Handler Filter Formatter Console

Capacitor 1- Number of charges: 12
Individual particle charge: 5
Capacitor 2- Number of charges: 12
Individual particle charge: -5

Root Logger
imx.loggui.handler.HandlerView
global
sun.awt.AppContext
sun.awt.im.InputContext
java.awt.mixing.Container
imx.loggui.lib.ICollectionEditor
teal.app.SimPlayerApp
imx.loggui.lib.ITextField
imx.lib.listener.ListenerBag
imx.loggui.logger.LoggerView
imx.lib.gui.Console
teal.app.SimPlayer
imx.loggui.handler.Configurablel
teal.synchronization.network.co
imx.loggui.LogMasterView
java.awt.mixing.Component
sun.awt.windows.WDesktopPro
java.awt.focus.DefaultKeyboar
imx.loggui.formatter.SimpleSele
sun.awt.windows.shape.WComp
sun.awt.SunToolkit

Logger: teal.app.SimPlayer
Log level: parent level
Handler: no handler
Filter: no filter

LogGui Version 1.0
Copyright (c) 2003-2007
InforMatrix GmbH

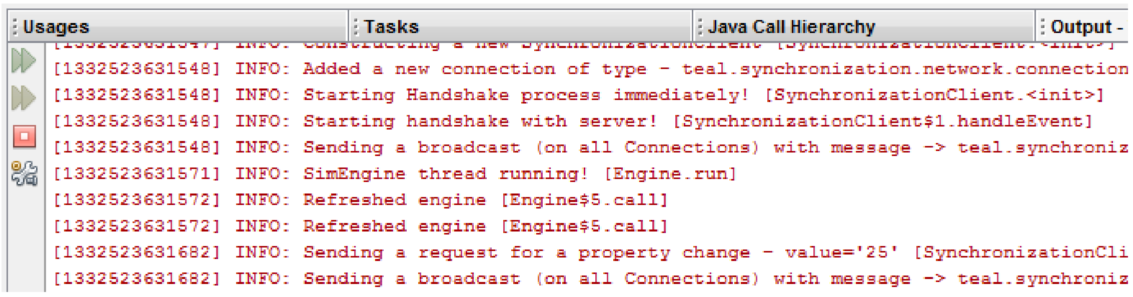
http://www.informatrix.ch

Figure 6.6: TEALsim being started up with a tool for adjusting debugging output at runtime

Besides the mentioned possibility to plug 3rd-party applications into the logging system, one of its other key features is the dynamic configurability without recompilation. In TEALsim's case this is currently achieved via a configuration file in the project root folder called *logging.properties*. An example for a couple of instructions can be seen in listing 6.1, with a sample of the corresponding logging output in Netbeans shown in figure 6.7.

Listing 6.1: Example for instructions found in *logging.properties* file

```
# Set the standard Loglevels
.level= INFO
# this tells the handler to use our custom formatter
java.util.logging.ConsoleHandler.formatter = teal.util.CustomLoggingFormatter
# you don't need to specify the next line, which demonstrates a very terse log
format
teal.util.CustomLoggingFormatter.format =[%t] %L: %m [%C.%M]
```



```
[1332523631547] INFO: Constructing a new SynchronizationClient [SynchronizationClient.<init>]
[1332523631548] INFO: Added a new connection of type - teal.synchronization.network.connection
[1332523631548] INFO: Starting Handshake process immediately! [SynchronizationClient.<init>]
[1332523631548] INFO: Starting handshake with server! [SynchronizationClient$1.handleEvent]
[1332523631548] INFO: Sending a broadcast (on all Connections) with message -> teal.synchroniz
[1332523631571] INFO: SimEngine thread running! [Engine.run]
[1332523631572] INFO: Refreshed engine [Engine$5.call]
[1332523631572] INFO: Refreshed engine [Engine$5.call]
[1332523631682] INFO: Sending a request for a property change - value='25' [SynchronizationCli
[1332523631682] INFO: Sending a broadcast (on all Connections) with message -> teal.synchroniz
```

Figure 6.7: Log messages from customized formatter

6.6 Streamlining the User Interface

Another part of the work comprised a slight redesign of the user interface to align certain functions of TEALsim with other everyday tools providing a similar functionality. For a screenshot of TEALsim running with its original UI see figure 6.9. The screenshot shows the Capacitor simulation running on a Microsoft Windows machine. Identical to almost all of the other available simulations for TEALsim, the fundamental interaction with this simulation is similar to the interaction with a movie; that is, basically it can be controlled via a set of playback controls familiar from media players (at this place neglecting special controls provided by e.g. *ControlGroups*, since they were not changed during the course of this project). Figure 6.8 shows a close-up of the previously used playback control (which was called '*EngineControl*') annotated with the purpose of the particular buttons.

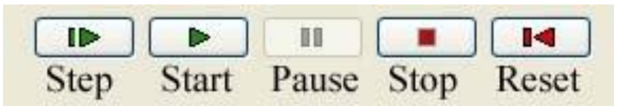


Figure 6.8: The former simulation controls and their associated properties – Source: [Belcher, McKinney, Bailey, & Danziger, 2007]

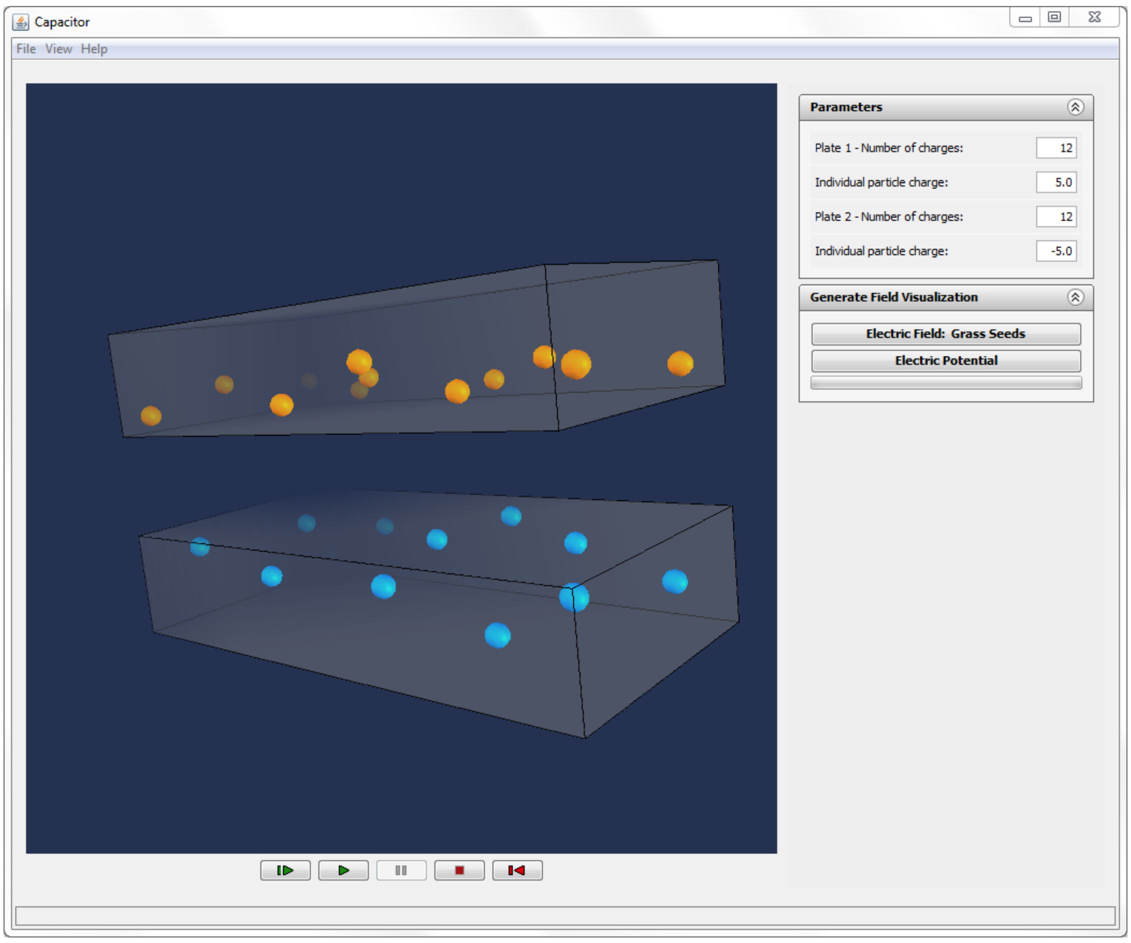


Plate 1 - Number of charges: 12
Individual particle charge: 5.0
Plate 2 - Number of charges: 12
Individual particle charge: -5.0

Figure 6.9: Screenshot of an older version of TEALsim showing the former user interface

For a sketch outlining the proposed look of the new UI for the stand-alone desktop version of TEALsim see figure 6.10. Consecutively is a brief break down of the ideas behind the new buttons:

a) Most evidently the new UI emulates modern media player's style to merge the play and pause button to one toggle button, since both states are mutually exclusive.

b) The underlying behavior of 'Reset' should change from its former concept of setting everything back to a random value to setting everything back to a defined state (thus repeated resets would always return to the same state). This would be more consistent with the common understanding of what a 'reset' in everyday life usually does. Inevitably this may create the need to provide the means for a real randomized restart for certain simulations. For this purpose a dedicated button could be placed on demand next to reset, clearly describing its effect.

c) The former 'Stop' button was removed since its main purpose was to pause the simulation and prevent it from being started again. For most simulations – e.g. in the case of the capacitor – this did not make much sense after all. Those few simulations which really need this kind of behavior could simply trigger pause and disable the controls themselves.

d) Because a button to forward the simulation by a predefined time value ΔT existed ever since, a logic consequence was to provide a button with an inverse behavior as well.

e) Last but not least a slider was added to give users control over the pace of the simulation, essentially multiplying the predefined time value ΔT.

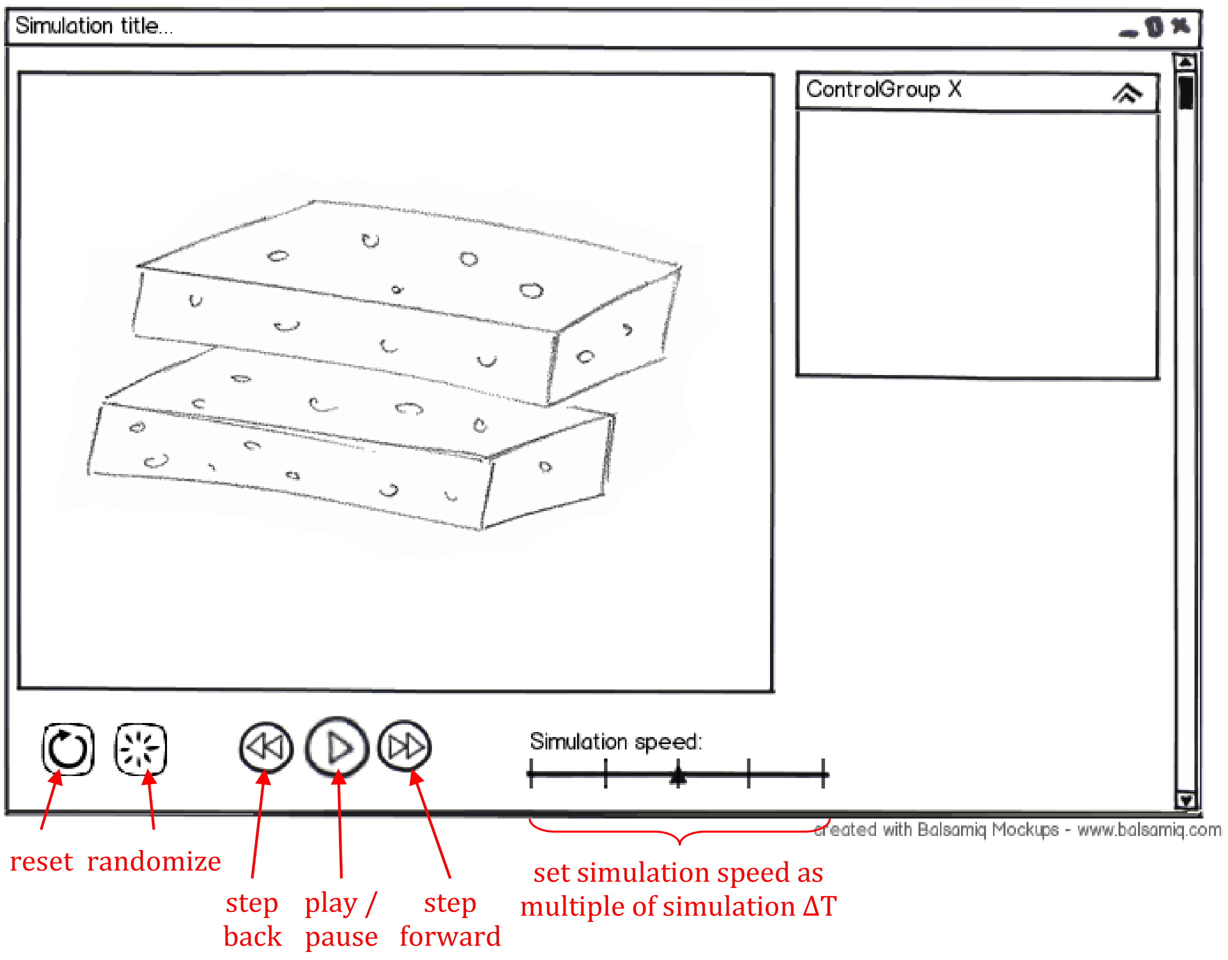

Simulation title...
ControlGroup X
Simulation speed:
created with Balsamiq Mockups - www.balsamiq.com
reset
randomize
step back
play / pause
step forward
set simulation speed as multiple of simulation ΔT

Figure 6.10: Sketch of the proposed refurbished user interface created with the free online version of Balsamiq Mockup tool[42]

[42] free online version of Balsamiq Mockup tool - http://builds.balsamiq.com/b/mockups-web-demo/

6.7 Summary

This chapter constitutes a documentation of all the ordinary tasks carried out as part of this thesis. It is rather intended to give long time contributors to the TEALsim project an overview of what has changed why, since novel contributors might take most of the integrated features for granted anyway (for large projects like TEALsim).

Initially the current state of TEALsim's code base was discussed along with some potential reasons which supposedly have led to this situation. This included some basic metrics highlighting the effects of the programming activities performed. The rest of this chapter was split into sub-chapters each highlighting a single feature introduced to respectively changed in the framework.

After this chapter summarized the very general tasks performed, the next chapter becomes more technical explaining the included network package in TEALsim. This discussion starts with a summary of the package's layer hierarchy leading to its general schema, which is analyzed based on the relations between the classes. Subsequently follows an exemplary startup sequence for a multi-user system. Furthermore, the different available connection types are explained as well. Finally the chapter closes with a documentation of the implemented classes which provide means for synchronizing objects across the network.

Chapter 7

The Underlying Network Layer

One of the fundamental differences between the software design previously used to embed TEALsim in 3rd-party applications like OWL (see chapter 3.2) and the design proposed in this thesis is the integration of a complete network layer in TEALsim. The following chapter outlines the various aspects and considerations of the implemented network package. It shall give a high level overview of the existing classes and their corresponding tasks and interactions without discussing concrete technical details like available methods or fields in-depth. For such details on such a low level see the inline documentation.

7.1 General Design of the Network Architecture

The whole network package added to TEALsim has to be seen as a hierarchy of software layers which are in general based on each other although sometimes the boundaries amongst them are not strict and somewhat debatable. Irrespective of this ambiguity there are basically 4 layers (see figure 7.1):

a) *application layer* -> On the client side this is more or less the former TEALsim framework (albeit enhanced with the deterministic simulation engine, UI elements adapted to synchronization, etc.). On the server side this layer would comprise the *SimulationSynchronizer* object. This *SimulationSynchronizer* encapsulates the whole logic and data required to keep multiple concurrent clients synchronous. For each distinct simulation the TEALsim server creates a new *SimulationSynchronizer* object – for this reason only one server instance is required to handle an arbitrary amount of different simulations with any amount of clients.

b) *message dispersion layer* -> This layer is responsible to select the appropriate *Connections* for transmission of outgoing messages respectively forwards incoming messages to the application layer. *SynchronizationClient* and *SynchronizationServer* are threads. The client thread is started when a simulation is created – every simulation has its own *SynchronizationClient*. During its construction the *SynchronizationClient* is responsible to perform the handshake with the server (that is, amongst other things fetch the current simulation status and pass the data to the simulation engine thread). The *SynchronizationServer* is currently a singleton (thread) which has to run within one TEALsim process, for which reason it is created as one of the first things when starting TEALsim as desktop application. This thread owns an arbitrary amount of *Connections* which can be used by TEALsim clients to connect with the server. Currently there exist ~3 different types of *Connections* (*InnerProcessConnection*, *SocketConnection* and *WonderlandConnection*).

An exemplary configuration could look like this: one TEALsim process with a *SynchronizationClient* (plus the host simulation) using an *InnerProcessConnection*; within the same process the *SynchronizationSever* thread running with an *InnerProcessConnection* and an additional *SocketConnection*. Another process running only with a *SynchronizationClient* (plus the host simulation) using a *SocketConnection*. Since the network package is currently designed as a client-server architecture (that is, peer-to-peer communication is not yet implemented), one *SynchronizationClient* instance communicates with the *SynchronizationSever* via the *InnerProcessConnection* whereas the other *SynchronizationClient* uses the *SocketConnection* to communicate with the *SynchronizationSever*.

c) *abstract Connection layer* -> As the name suggests, this layer keeps the concrete technical implementation of the real connection transparent to the *SynchronizationClient* and *SynchronizationServer* threads.

d) *concrete Connection layer* -> Concrete implementations of different point-to-multipoint connection types. As mentioned, currently exist 3 different types which differ in the degree of their technical sophistication. The *SocketConnection* (also see chapter 7.2) for example requires multiple threads to:

- listen for connecting clients on the server side
- maintain a channel for incoming data for each connected peer
- dispatch messages to remote peers

On the other hand the *InnerProcessConnections* (see chapter 7.3) is rather straightforward by only maintaining one worker thread to pass data forth to the connected peer (to avoid deadlocks).

As mentioned, the whole network package is currently designed and implemented as client-server architecture. In other words, communication between clients always goes through the server instance. In view of use cases where peer-to-peer communication would be an interesting option (e.g. one client streaming the simulation to another client to save server bandwidth) it seems feasible to extend the layer schema without the need for a complete rework.

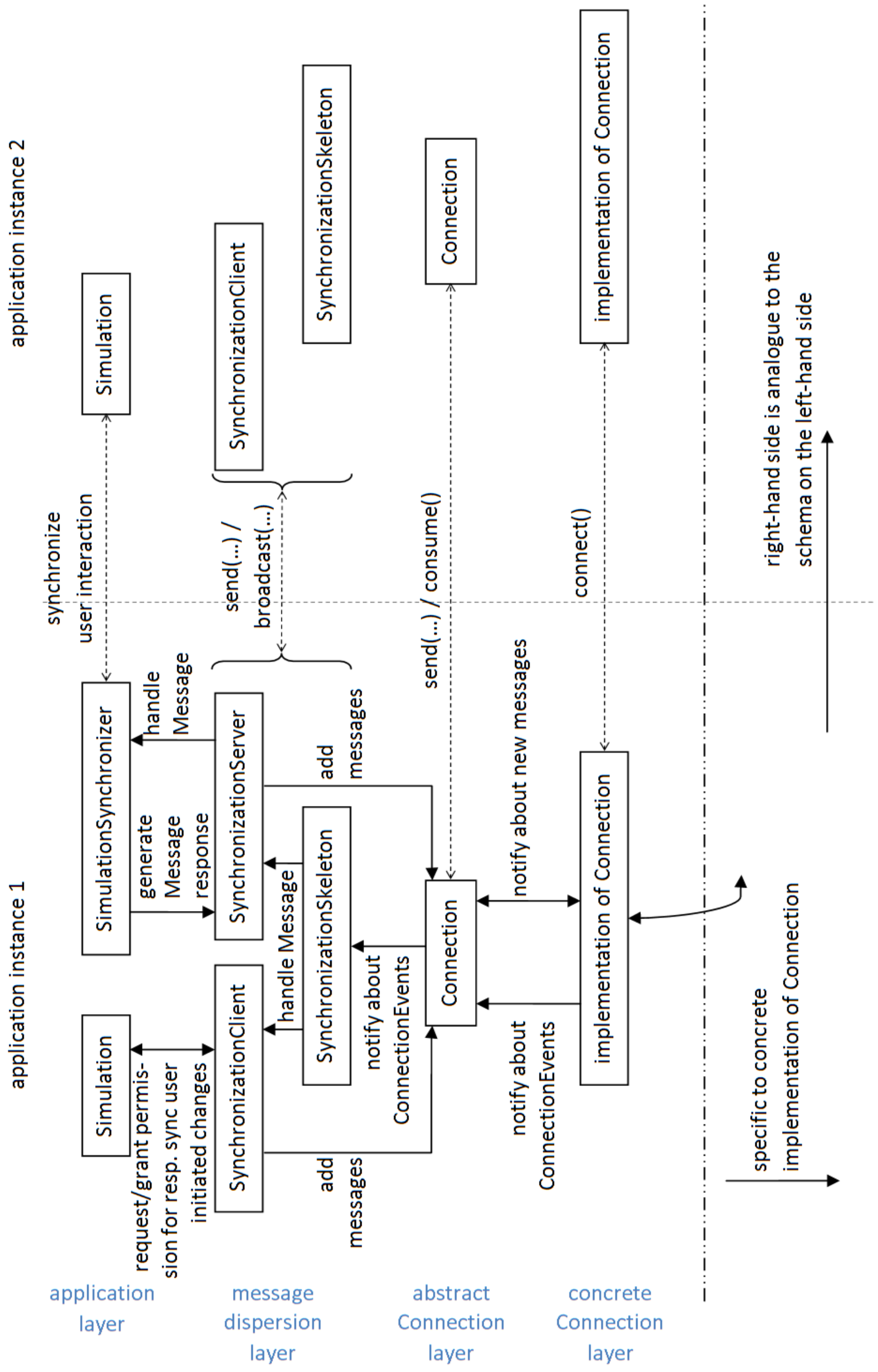

application instance 1
application instance 2
Simulation
SimulationSynchronizer
SynchronizationClient
SynchronizationServer
SynchronizationSkeleton
Connection
implementation of Connection
request/grant permis-sion for resp. sync user initiated changes
generate Message response
handle Message
synchronize user interaction
send(...) / broadcast(...)
add messages
handle Message
notify about ConnectionEvents
send(...) / consume()
notify about new messages
notify about ConnectionEvents
connect()
specific to concrete implementation of Connection
right-hand side is analogue to the schema on the left-hand side
application layer
message dispersion layer
abstract Connection layer
concrete Connection layer

Figure 7.1: layers of TEALsim's network architecture

Like explained in the paragraph recapitulating the message dispersion layer (see page 71), the *SynchronizationServer* and *SynchronizationClient* objects are threads responsible to coordinate the asynchronous communication between nodes in a network (that is, a client and a server node). As shown in figure 7.2 both threads derive from a common super class (the *SynchronizationSkeleton*), thus both sub-classes are in some measure individual strategies to deal with a remote node. In theory each *SynchronizationSkeleton* may maintain an arbitrary amount of *Connections*. However, clients will usually only instantiate one implementation of *Connection* to connect to the server.

In a TEALsim multi-user system each remote endpoint of a *Connection* is referenced via a *NetworkNode*. The concrete implementations of *Connection* are responsible to create and store the corresponding implementations of *NetworkNodes* when two distinct *Connections* connect with each other. For example will the *SocketConnection's* thread which is listening for new clients on the server side add a new *NetworkNode* (or *SocketNode* in this particular case) to the *SocketConnection's* list of connected nodes. Likewise will the client add a *NetworkNode* referencing the server to its list of connected nodes.

Naturally, messages exchanged between nodes in the network have to implement the *Serializable* interface. It is important to bear in mind that the fields in messages which are referencing *NetworkNodes* (source and target nodes) are not serialized and transmitted (-> they are transient fields) but have to be set by the receiving *Connection* on demand. This is logical since only the *Connection* receiving a message has access to the *NetworkNode* referencing the outgoing *Connection*.

To summarize the network design up this point: Besides possibly forwarding initial configuration parameters, TEALsim simulations basically do not know much about their underlying network architecture. When users change elements of the simulation (e.g. click a button or change a text field value) the simulation hands over the change request to the *SynchronizationClient* thread. This thread is responsible to further forward the message and issue new tasks for the simulation on incoming orders. The *SynchronizationClient* does in turn not care about specific technical details of the underlying network connections. For this duty the concrete implementations of *Connection* encapsulate all technical aspects and provide a slim API for the *SynchronizationSkeleton*. That said the TEALsim server side works approximately identical.

In case the network package shall be extended: apart from the fact that the current implementation is still a prototype version with many methods yet waiting to be properly finished and tested, the only components required when intending to implement e.g. a secure socket based connection are:

a) a concrete implementation of *Connection*
b) a corresponding implementation of *NetworkNode*
c) and <optional> a *ConnectionParameter* used to create instances of the related *Connection*

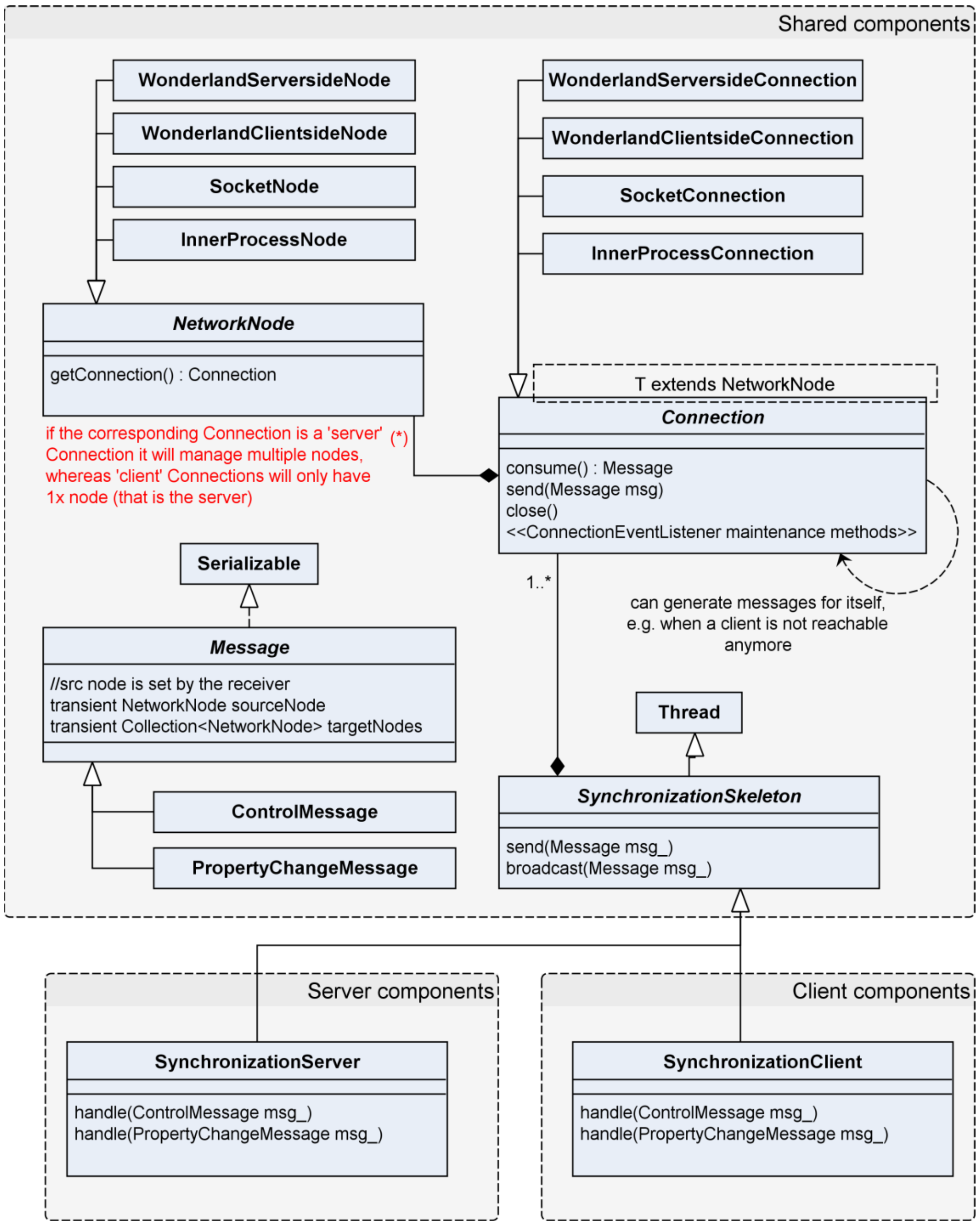

Shared components
WonderlandServersideNode
WonderlandClientsideNode
SocketNode
InnerProcessNode
NetworkNode
getConnection() : Connection
if the corresponding Connection is a 'server' (*) Connection it will manage multiple nodes, whereas 'client' Connections will only have 1x node (that is the server)
WonderlandServersideConnection
WonderlandClientsideConnection
SocketConnection
InnerProcessConnection
T extends NetworkNode
Connection
consume() : Message
send(Message msg)
close()
<<ConnectionEventListener maintenance methods>>
can generate messages for itself, e.g. when a client is not reachable anymore
1..*
Serializable
Message
//src node is set by the receiver
transient NetworkNode sourceNode
transient Collection<NetworkNode> targetNodes
ControlMessage
PropertyChangeMessage
Thread
SynchronizationSkeleton
send(Message msg_)
broadcast(Message msg_)
Server components
SynchronizationServer
handle(ControlMessage msg_)
handle(PropertyChangeMessage msg_)
Client components
SynchronizationClient
handle(ControlMessage msg_)
handle(PropertyChangeMessage msg_)

Figure 7.2: Class diagram of TEALsim's network layer

Figure 7.3 shows an exemplary sequence diagram outlining the most important activities which happen when TEALsim is started (with regard to network activities). After a user selected a simulation eventually the network layer on the client side has to initiate the handshake procedure with the server (sending "*Connect*" and receiving the "*ConnectionEstablished*"). Since the client in figure 7.3 is the first one to use the particular simulation, no further updates have to be sent from the server to the client. Indicated in parentheses is the case where a client connects to a running simulation and the server has to provide the current simulation state. After the connection is established the server registers with the client to receive newly calculated *EngineStates* when the simulation is running. This has to happen since the server does in general not compute the simulation itself but relies on the clients to do the calculations themselves. This way the server can provide the current simulation state to new clients which connect during simulation execution. Once the user clicked to start the simulation and the request was authorized by the server, the client with the *EngineState* subscription repeatedly sends the updated *EngineState* to the server until the simulation is paused.

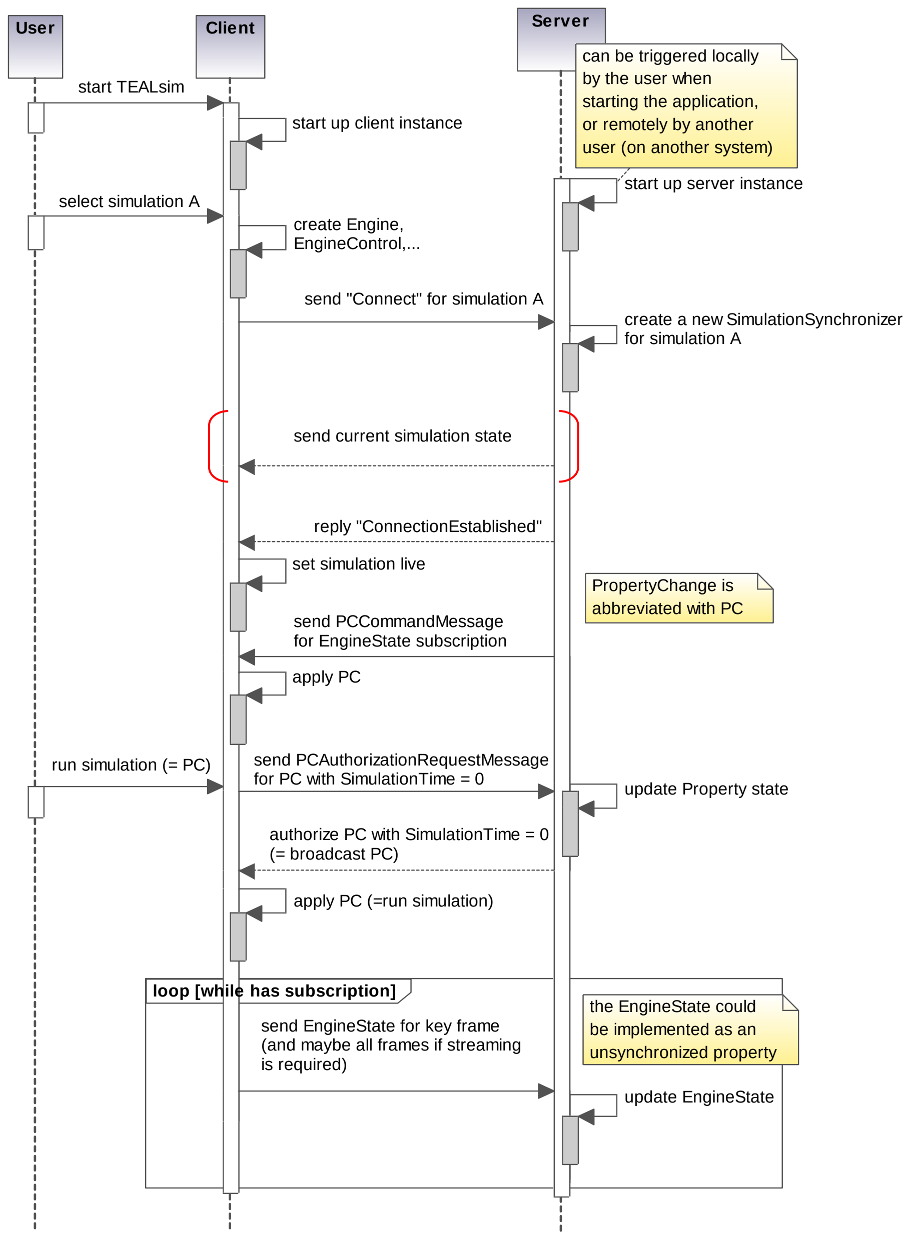
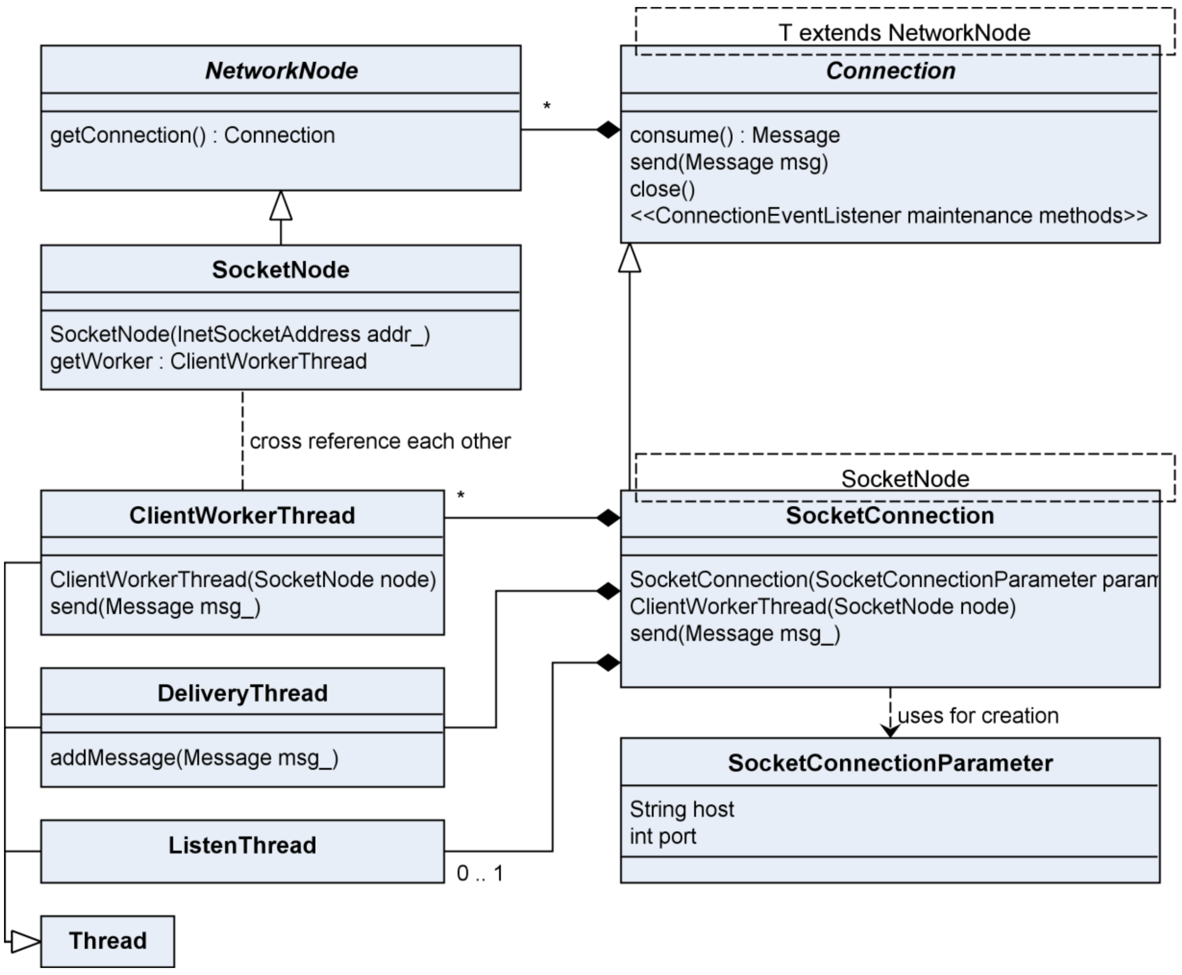

User
Client
Server
can be triggered locally by the user when starting the application, or remotely by another user (on another system)
start TEALsim
start up client instance
start up server instance
select simulation A
create Engine, EngineControl,...
send "Connect" for simulation A
create a new SimulationSynchronizer for simulation A
send current simulation state
reply "ConnectionEstablished"
set simulation live
PropertyChange is abbreviated with PC
send PCCommandMessage for EngineState subscription
apply PC
run simulation (= PC)
send PCAuthorizationRequestMessage for PC with SimulationTime = 0
update Property state
authorize PC with SimulationTime = 0 (= broadcast PC)
apply PC (=run simulation)
loop [while has subscription]
the EngineState could be implemented as an unsynchronized property
send EngineState for key frame (and maybe all frames if streaming is required)
update EngineState

Figure 7.3: Standard start up sequence of TEALsim's client-server architecture

7.2 Socket Based Implementation of Connection

The *SocketConnection* allows clients to communicate with a server via Java's sockets over a TCP connection. This covers use cases with network nodes connected via WAN, LAN or even multiple local processes using the operating system's loopback network interface '*localhost*'.

Figure 7.4 shows the rough schema of the whole package: The server-side *SocketConnection* is created with a port to listen for incoming connections. For this purpose a *ListenThread* is spawned. This thread spawns in turn new *ClientWorkerThreads* for each connecting client (which is represented via a *SocketNode* containing a field referencing the socket of the corresponding client). The client-side implementation creates the *SocketConnection* with a port and hostname. This will immediately create one *ClientWorkerThread* handling the connection to the server. Currently clients do not spawn a *ListenThread* because they are not intended to establish connections with anything but the remote server. However, *SocketConnections* on client and server alike spawn a *DeliveryThread* which is responsible to dispatch outgoing messages via each *SocketNode's* socket field.



T extends NetworkNode
NetworkNode
getConnection() : Connection
*
Connection
consume() : Message
send(Message msg)
close()
<<ConnectionEventListener maintenance methods>>
SocketNode
SocketNode(InetSocketAddress addr_)
getWorker : ClientWorkerThread
cross reference each other
ClientWorkerThread
ClientWorkerThread(SocketNode node)
send(Message msg_)
*
SocketNode
SocketConnection
SocketConnection(SocketConnectionParameter param
ClientWorkerThread(SocketNode node)
send(Message msg_)
DeliveryThread
addMessage(Message msg_)
ListenThread
0 .. 1
uses for creation
SocketConnectionParameter
String host
int port
Thread

Figure 7.4: Schema of the socket based implementation of *Connection*

7.3 Implementation of a Connection for Communication Within a Single JVM

One of the goals of this thesis was to maintain TEALsim's existing application spectrum, in this case the possibility to run simulations self-contained as a single-user process. For this purpose the *InnerProcessConnection* (IPC) exists which allows to retain the client-server architecture also for single-user scenarios. The *IPC* acts like a joint between the *SynchronizationServer* and *SynchronizationClient* threads running within the same JVM. Figure 7.5 is a schema of *IPC's* design. At construction time an *IPC* is set as server or client instance. This is the point of time when the client *IPC* establishes connectivity between both instances of *IPC* via the server *IPC's* static reference (thus the server *IPC* has to exist beforehand). Messages handed over to an *IPC* are buffered in the *IPC's* worker thread. Eventually this *IPCWorkerThread* hands over the message reference to the connected *IPC*.

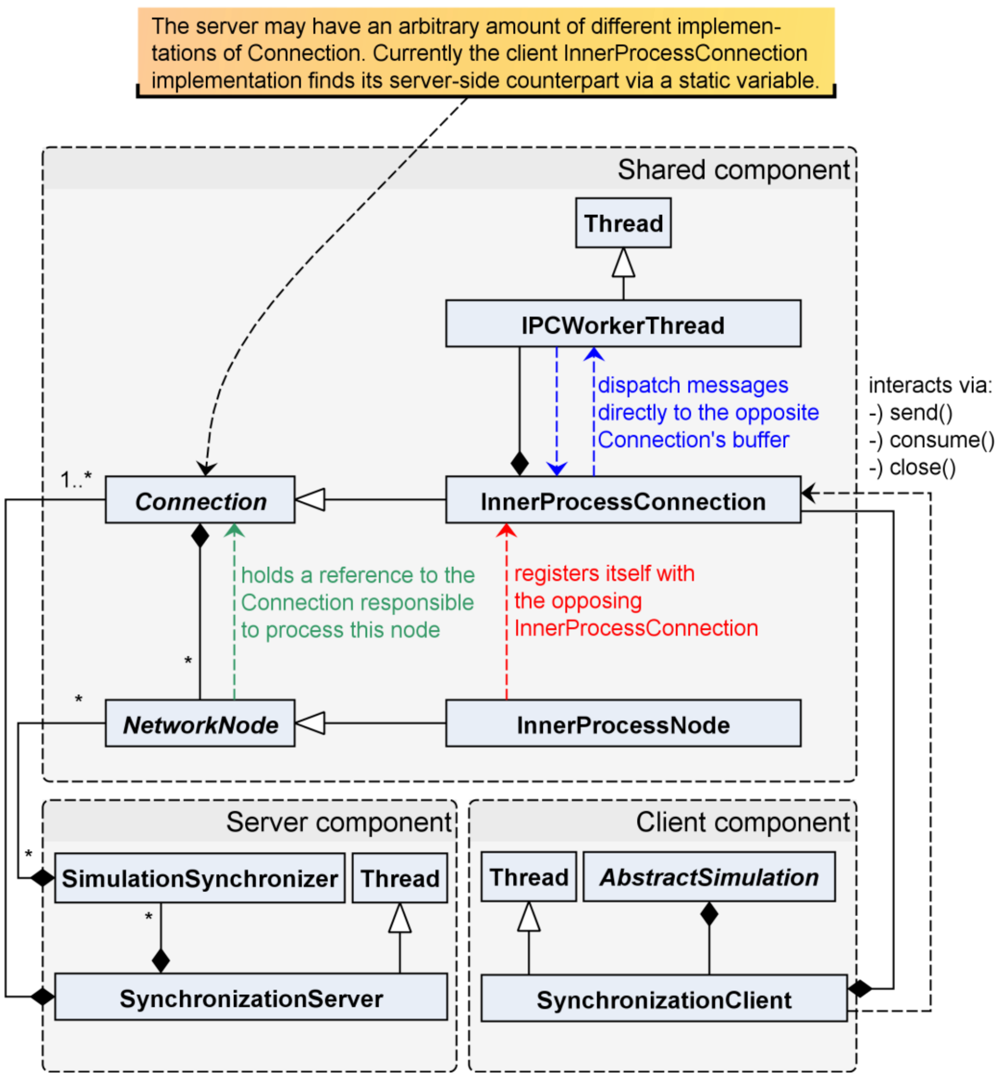

The server may have an arbitrary amount of different implementations of Connection. Currently the client InnerProcessConnection implementation finds its server-side counterpart via a static variable.
Shared component
Thread
IPCWorkerThread
dispatch messages directly to the opposite Connection's buffer
interacts via:
-) send()
-) consume()
-) close()
1..*
Connection
InnerProcessConnection
holds a reference to the Connection responsible to process this node
registers itself with the opposing InnerProcessConnection
*
*
NetworkNode
InnerProcessNode
Server component
Client component
*
SimulationSynchronizer
Thread
Thread
AbstractSimulation
*
SynchronizationServer
SynchronizationClient

Figure 7.5: Schema of the *Connection* used to run client and server within one JVM

7.4 Open Wonderland Specific Implementation of Connection

Besides the *InnerProcessConnection* and the *SocketConnection*, currently there also exists a 3rd implementation of *Connection* which can be used by TEALsim when it is embedded as module in OWL. This OWL specific *Connection* is part of the OWL module and as such is explained in detail in chapter 8.2.

7.5 Design of Synchronizable Elements

As mentioned, TEALsim is a toolbox framework with the goal to provide easy to use classes to enable a wide range of people to put together new and interesting simulations. Considering the goal of this thesis to enable TEALsim to run in distributed environments there is the need for an object which provides a mechanism to stay synchronized amongst all concurrent clients but hides away the complexity of the related synchronization process. This is one of the issues tackled with the *SynchronizableGeneric* class.

Another key feature of *SynchronizableGeneric* is the possibility to link multiple instances together. This feature was previously provided by a concept called '*Routes*'. An advantage of *SynchronizableGenerics* over *Routes* is the compile-time type safety because *SynchronizableGenerics* do not reference each other via *String* ID's but instead use 'real' references to the particular objects.

7.5.1 General Schema of SynchronizableGeneric Classes

Figure 7.6 shows the general schema of the *SynchronizableGeneric* class. First thing to mention is that each instance holds a reference to the running simulation. Via this reference the simulation's *SynchronizationClient* instance can be accessed to dispatch change requests to the server. In this context it becomes evident that *SynchronizableGeneric's* type has to extend *Serializable* because the type's value field is transferred to the server. Since the whole concept to ensure synchronization amongst concurrent clients builds upon a deterministic execution flow, changes to a value are not immediately set but have to be approved by the server (see chapter 5 for more details). Due to this authoritative principle the *SynchronizationClient* might in effect set the new value at a later point of time.

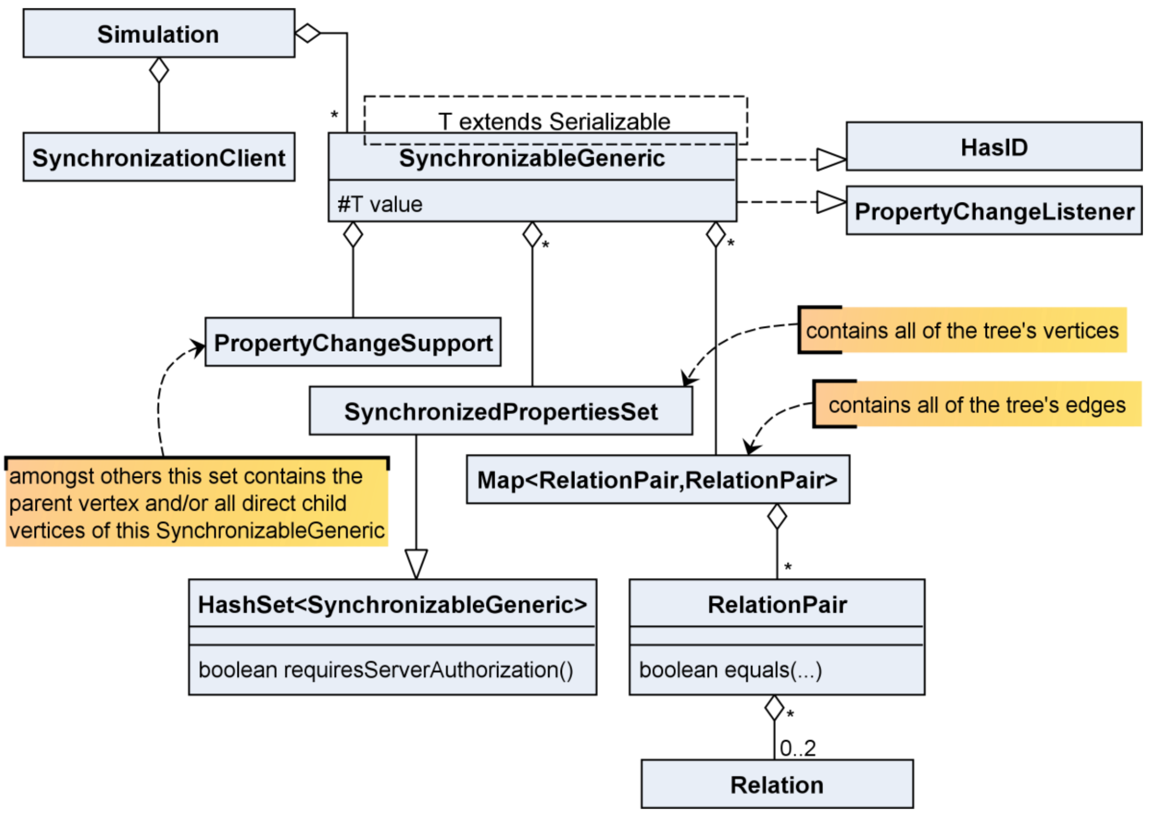

Simulation
SynchronizationClient
T extends Serializable
SynchronizableGeneric
#T value
HasID
PropertyChangeListener
PropertyChangeSupport
SynchronizedPropertiesSet
contains all of the tree's vertices
contains all of the tree's edges
amongst others this set contains the parent vertex and/or all direct child vertices of this SynchronizableGeneric
Map<RelationPair,RelationPair>
HashSet<SynchronizableGeneric>
boolean requiresServerAuthorization()
RelationPair
boolean equals(...)
0..2
Relation
*

Figure 7.6: Schema of synchronizing values in TEALsim

The mechanism to link multiple *SynchronizableGenerics* together is based on a tree[43] data structure. Altering one of the tree's vertices will change all other vertices as well. Changes will propagate based on a snowball schema. Initially all vertices directly connected to the source of the change are notified (if existent the parent vertex and all direct child vertices). This process is reiterated until the whole tree is updated with the new value. Since in almost all reasonable use cases linked properties are in some way distinct from each other, there exists the possibility to add *Relations* which allow conversion from one property's value to the other (and the other way around). To maintain the tree-like data structure 3 fields exist:

a) a *PropertyChangeSupport* object which holds references to all objects interested in changes of the corresponding *SynchronizableGeneric*. While any object (that is, *PropertyChangeListener*) may register itself to receive notifications about changes, this *PropertyChangeSupport* object will also reference

- all direct child vertices
- the parent vertex

[43] *"A graph without cycles, or acyclic graph, is called a forest. A connected forest is called a tree* [...] *Thus, a forest is a family of trees and a tree is a connected graph without cycles."* [Kheyfit, 2010, p. 110]

b) a *HashSet* containing *all* vertices of the *tree*. This *HashSet* object is shared between all linked *SynchronizableGenerics*. When linking new *SynchronizableGenerics* to a tree this *HashSet* makes it easy to verify that the new elements are not already part of the tree in some other place. Having the same *SynchronizableGeneric* twice (or more times) in the tree data structure would imply circles, resulting in a simple graph data structure instead of a tree!

c) a *Map*, also shared between *all* linked *SynchronizableGenerics*, containing all edges (that is, *Relations* between vertices) of the *tree*. It is possible to completely omit a relation between two vertices in which case a direct relation is assumed (one value is identical to the other value). Another option is to only set a one-way relation (e.g. a *TextField* linked with the property specifying the color of point charges in an application; a user may type 'blue' into the *TextField* and the *Relation* object takes care to convert this string into something usable by the point charge object). When creating one-way relations it is important to ensure that the tree of linked *SynchronizableGenerics* never gets traversed in a way where missing relations are encountered (e.g. referencing the former example -> there is no other way to change the color of the point charges which would then require the linked *TextField* to be changed as well).

To avoid *PropertyChangeEvents* (PCE) to echo back from notified *SynchronizableGenerics*, causing an infinite loop, the source of a *PCE* is deregistered from the *PropertyChangeSupport* object before firing a new *PCE* and reregistered afterwards.

7.5.2 Synchronizable UI Controls

Based on the fundamental functionality provided by *SynchronizableGenerics*, custom objects were introduced to TEALsim to deal with the different available controls included in the Java Swing toolkit. Currently not for every Swing control a corresponding *Synchronizable.XYZ.UI.Element* exists though (in this place '*XYZ.UI.Element*' is used as a placeholder term to reference any member of the related package). The idea behind the whole concept is that each *javax.swing.XYZ.UI.Element* has an internal state of some sort. The associated *Synchronizable.XYZ.UI.Element* mirrors this state and implements methods to update the Java Swing control respectively intercept changes to this control by the user. Since most Swing controls are very different from each other there has to be a specialized *Synchronizable.XYZ.UI.Element* for most of them. Figure 7.7 illustrates this design.

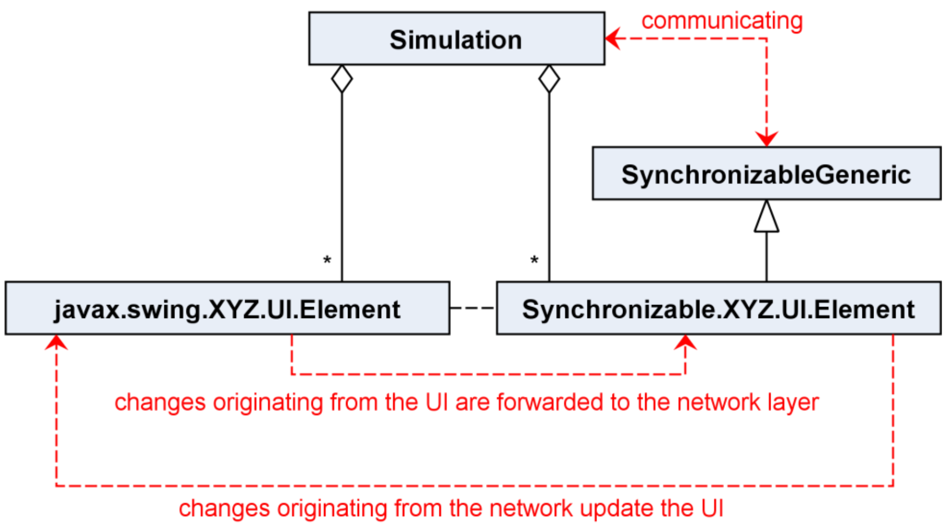

Simulation
communicating
SynchronizableGeneric
*
*
javax.swing.XYZ.UI.Element
Synchronizable.XYZ.UI.Element
changes originating from the UI are forwarded to the network layer
changes originating from the network update the UI

Figure 7.7: Generic schema of TEALsim's UI synchronization model

7.5.3 Usage in TEALsim Applications

Using the *Synchronizable.XYZ.UI.Elements* requires little additional code. For example listing 7.1 is a snippet taken from *teal.sim.engine.EngineControl.java*. The code lines 1 to 15 already existed before the introduction of the network layer to TEALsim. A slider is created there with 5 ticks, where the first tick label is set to $2^0 = 1$ and for each successive tick label the exponent is incremented.

Line 17 and 18 create a new *SynchronizableSlider* object which is constructed with a reference to the Java Swing *JSlider* object. This *SynchronizableSlider* object keeps in sync with the state of the *JSlider*.

Since the slider is supposed to do something the *SynchronizableSlider* object is linked with the *SimulationSpeed* object of the simulation engine on line 26. For demonstration purposes the engine is cast to an *AbstractEngine* without any further checks. However, linking the slider's value to the *SimulationSpeed* requires a *Relation* because the internal value of the *JSlider* (and therefore also the mirrored value in the *SynchronizableSlider*) is different to its labels (1 to 5 – see line 1). For this reason a *Relation* is created on lines 20 to 25 converting from an integer value to an integer value, however taking the input integer as the exponent to a base of 2.

In this case a *Relation* converting from the *SimulationSpeed* to a value usable for the slider is omitted. Therefore the programmer has to ensure that no changes to the *SimulationSpeed* may originate from anything else but the slider!

Listing 7.1: Usage of *SynchronizableGenerics* in TEALsim applications

```java
JSlider speed = new JSlider(1, 5, 3);
speed.setName("SimSpeedSlider");
speed.setMajorTickSpacing(1);
speed.setMinorTickSpacing(1);
speed.setPaintLabels(true);
speed.setPaintTicks(true);
speed.setSnapToTicks(true);
//Create the labels which are 2^(i-1), that is they start with 1x and
//increase exponentially
Dictionary labelTable = new Hashtable();
for (int i = 1; i <= 5; i++) {
  labelTable.put(new Integer(i),
                 new JLabel((int) Math.pow(2, ( i - 1 )) + "x"));
}
speed.setLabelTable(labelTable);

final SynchronizableSlider ss;
ss = new SynchronizableSlider(parentSimulation, speed);

Relation<Integer, Integer> rel = new Relation<Integer, Integer>() {
  @Overrride
  public Integer convertFrom(Integer fromValue_) {
    return (int) Math.pow(2, ( fromValue_ - 1 ));
  }
};
ss.addLinkedProperty(
((AbstractEngine) parentSimulation.getEngine() ).simulationSpeed, rel);
```

7.6 Summary

This chapter introduced to the network architecture added to the TEALsim framework. After outlining the various network layers an exemplary start up sequence for a TEALsim multi-user environment was analyzed. Furthermore, some details regarding the implementations of *Connection* were discussed which are part of the TEALsim package (with a 3rd implementation, which is part of the OWL TEALsim module, being discussed in the succeeding chapter). Finally this chapter closed with an analysis of the implemented tools to synchronize user controls respectively arbitrary properties. In this regard this chapter constitutes a reference point for further developments on the network layer as well as efforts to create new, multi-user enabled TEALsim applications. In the end, this chapter concluded the documentation of technical aspects related to the changes done to the TEALsim framework. Therefore the succeeding chapter covers rather configurational aspects of the TEALsim project as well as an insight into the independent project maintaining the TEALsim OWL module.

Chapter 8

Starting and Using the TEALsim Framework

The programming work done in the course of this thesis introduced several new possibilities to start up TEALsim's desktop version respectively changed the way the framework may be embedded in other projects. For this reason the command-line arguments for the desktop version are outlined at first, with some more detailed explanations following afterwards. Finally the TEALsim OWL module is explained to serve as reference point for similar projects.

8.1 Options to Start the Desktop Version of TEALsim

When starting TEALsim's desktop version the following arguments can be used to alter the applications behavior:

a) ***-a*** ⟶ populates the window's menu bar with a comprehensive set of simulations
b) ***-n*** *<Simulation>* ⟶ starts up TEALsim with the specified simulation (and no other options in the menu bar; e.g. "-n *tealsim.physics.em.Capacitor*")
c) ***-gfx*** *<Engine>* ⟶ specifies which graphics engine to use; currently "*J3D*" and "*JME*" are supported
d) ***-debug*** ⟶ starts up TEALsim together with a 3rd-party debugging framework (see chapter 8.1.2)
e) ***-noserver*** *<Hostname>* *<Port>* ⟶ starts up TEALsim without server instance and makes it connect to the server running on at the specified address (via a TCP connection that is, an instance of *SocketConnection*). See chapter 8.1.1 for more details.
f) ***-server*** *<Port>* ⟶ starts up TEALsim with a server instance running locally, listening for remote clients on the specified port (using a *SocketConnection*). See chapter 8.1.1 for more details.

8.1.1 Configuring the Client and Server Component

By setting the corresponding command-line arguments TEALsim's desktop version may be started in different ways. Especially the behavior when no server instance is configured to run in the JVM may seem odd at first glance but it is owed to the way the *RuntimeArgument* mechanism is implemented. Basically the possible ways to start TEALsim's desktop version are:

a) *with no particular command-line arguments at all*

   This will cause TEALsim to start with a local server instance running with exactly one associated *Connection* of the sub-type *InnerProcessConnection*. Instances of simulations will create *SynchronizationClient* instances utilizing themselves a *Connection* of sub-type *InnerProcessConnection*. This behavior ensures a minimum amount of conflicts which may be caused for example by firewalls when the socket based implementation of *Connection* would be used for desktop use of TEALsim. Furthermore, the *InnerProcessConnection* performs much better compared to the socket based *Connection* type due to the fact that messages are exchanged directly between client and server package by handing over pointers.

b) *with arguments for the server part of TEALsim but no special arguments for the client part*

   This will cause TEALsim to start with a local server instance running with all of the specified *Connections* plus one additional of the sub-type *InnerProcessConnection*. Instances of simulations will create *SynchronizationClient* instances utilizing themselves a *Connection* of sub-type *InnerProcessConnection*. Remote clients may connect to the local server by means of any of the additionally instantiated *Connection* types.

c) *with a 'no server' argument for the server part of TEALsim and a specific server for the client part*

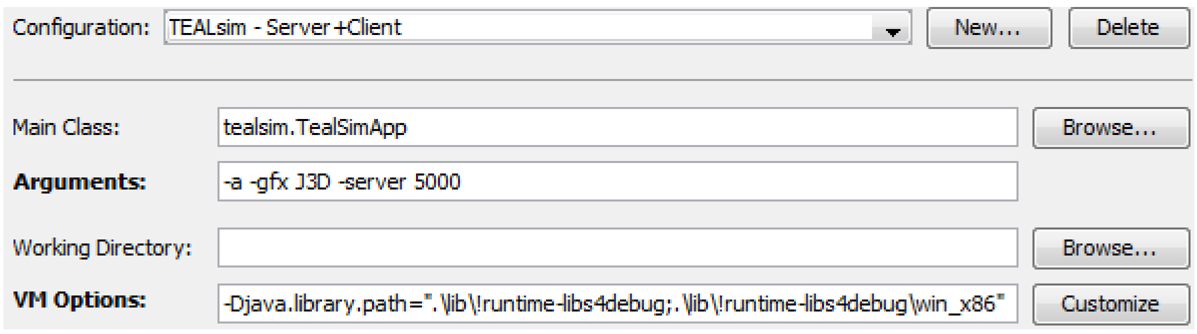
   This will cause TEALsim to start with a local server instance running with an instance of *InnerProcessConnection*. After this instance is created the command-line argument will cause the server to shut down. Instances of simulations will create *SynchronizationClient* instances utilizing a sub-type of *Connection* matching the given command-line argument (currently only socket based connections are used in the desktop version of TEALsim).

Condensing these configuration possibilities, TEALsim's behavior can be summed up with the following two statements:

a) If not explicitly specified in a different way the client side of TEALsim (that is, the *SynchronizationClient*) will use a *Connection* instance of the sub-type *InnerProcessConnection*.

b) Unless specified via the 'no server' argument a local server instance will be created with at least one *Connection* of sub-type *InnerProcessConnection*.

For example if someone would like to start a multi-user demonstration system of TEALsim's desktop version with 2 TEALsim clients running on one computer and an additional TEALsim client on a different machine, the first thing to start would be an instance of TEALsim providing the server functionality. Therefore the run configuration in NetBeans would look similar to figure 8.1. This configuration starts TEALsim running with J3D as graphics engine for the simulation client and an additional *SocketConnection* for the *SynchronizationServer* listening for remote clients on port 5000.



Configuration: TEALsim - Server+Client

Main Class: tealsim.TealSimApp

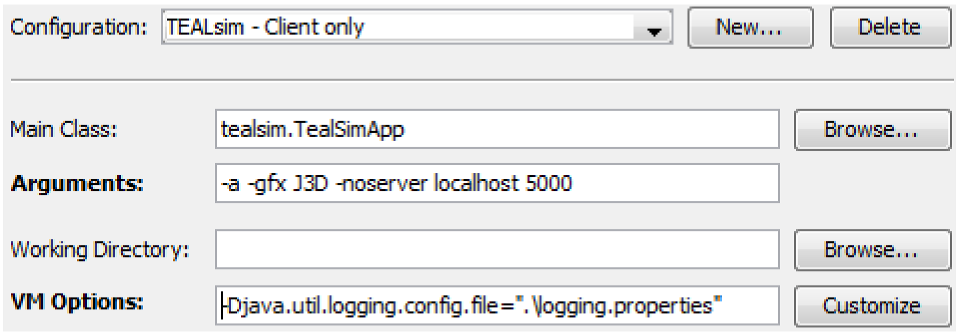
Arguments: -a -gfx J3D -server 5000

Working Directory:

VM Options: -Djava.library.path=".\lib\!runtime-libs4debug;.\lib\!runtime-libs4debug\win_x86"

Figure 8.1: Exemplary run configuration for a TEALsim instance with client and server

Next to start are the TEALsim clients who would like to mutually use the server instance running inside the previously started TEALsim process. The run configuration in NetBeans for these clients would look something like figure 8.2. Again, TEALsim clients would render the simulation with the J3D graphics engine. In figure 8.2 the TEALsim instance would connect to a server instance running on the same machine (therefore 'localhost' as host name) via port 5000. On a remote computer within the same network 'localhost' could be replaced e.g. with the IP address.



Configuration: TEALsim - Client only

Main Class: tealsim.TealSimApp

Arguments: -a -gfx J3D -noserver localhost 5000

Working Directory:

VM Options: -Djava.util.logging.config.file=".\logging.properties"

Figure 8.2: Exemplary run configuration for a TEALsim instance connecting to a remote server

8.1.2 Configuring the Log Settings

Different ways to adjust the logging output have been introduced to the TEALsim project. The basis for all these methods constitutes the continuous utilization of the standardized JDK logging framework.

a) The first method to change the log settings is to change the logging properties file. Figure 8.2 shows the switch for the VM options in NetBeans to set a specific configuration file (*-Djava.util.logging.config.file=".\logging.properties"*). Changes to this text file change the resulting logging output without the need to recompile the whole project.

b) The second method to alter the logging output is to change the formatter referenced in the logging properties file. Currently a custom formatter is used (*teal.util.CustomLoggingFormatter*) which prints each log message on a single line (see figure 6.7 for a screenshot of the resulting log messages).

c) The last method to influence the logging output is to start up TEALsim along with an additional logging framework. For this purpose the *'-debug'* switch may be set as TEALsim command-line argument (without any additional parameters) which – for demonstration purposes – will launch an instance of LogGui to provide means to adjust the logging output during application runtime (see figure 6.6 for a screenshot).

8.1.3 Configuring TEALsim for 32bit or 64bit Environments

When starting TEALsim from NetBeans the IDE has to be told where to find the required native library files. Different library bundles have to be used for the various existing host systems (currently only Windows 32bit and Windows 64bit are available). However, this configuration is not set dynamically yet. For this purpose the *-Djava.library.path* property has to be configured to point to the correct folder containing the native libraries (figure 8.1 shows a configuration running with a 32bit Windows environment). The two available bundles are:

a) *\lib\!runtime-libs4debug\win_x64* **->** for Windows 64bit Systems

b) *\lib\!runtime-libs4debug\win_x86* **->** for Windows 32bit Systems

8.2 Using TEALsim Framework in a 3rd-Party Application like OpenWonderland

The current OWL module, which allows embedding TEALsim in OWL, is a quick adaption of the formerly used module. In this regard it is far from being optimized design (and code) but shall demonstrate that with the new client-server architecture it became easier and more comprehensible to integrate TEALsim in 3rd-party applications.

OWL comes with a comprehensive set of restrictions for external modules (see chapter 3.2.3) which forced this former module to be very complex since it built upon a powerful server component. TEALsim's new client-server architecture, outlined in this thesis, comes with a much more light-weight server component which is essentially only brokering between the clients, but – most importantly – is in its current state not computing the mathematical complex simulations. However, this behavior could be changed in future versions to deal with situations where no computational capable client is available. In such scenarios the server could spawn a thread to run the simulation itself and send the stream of *EngineStates* to all clients.

8.2.1 Adapting the Client Package

Figure 8.3 shows the general schema of the OWL module's client package. Generally speaking, most of TEALsim remains unchanged and currently only a few components have to be replaced with OWL compatible versions (that is, classes related to the visualization of TEALsim). On the whole, the new TEALsim design turned OWL into just another type of container to display the simulations, not very different to the scenario where they run as stand-alone application in a Java Swing *JFrame*.

As it stands, the JME version of TEALsim – which is required for OWL, since JME is used to render the virtual world as well – is very immature code. It is very likely that with a clean and consistent implementation in TEALsim the following required components would probably be unnecessary. Disregarding this issue, the currently adapted components for OWL are mainly:

a) a *viewer* – used to hold respectively render the 3D visualization of the simulations; for a schema of the regular TEALsim rendering system see figure 12.6
b) a *GUI* component – which takes the 2D controls
c) a *player* container – which contains all simulation components

Naturally an OWL compatible version of *Connection* has to be provided as well which gets explained hereafter.

The OWL specific implementation of TEALsim's *Connection* builds on top of OWL's channel system to exchange data between client and server. For this purpose a message receiver has to be registered with the TEALsim cell's channel. This so called *ClientTealMessageReceiver* intercepts all of the *TealMessages* transferred over this particular OWL channel. The *TealMessage* class is itself just a wrapper class containing the actual TEALsim framework message since any message sent over OWL's channel system has to extend the abstract class *CellMessage*. The *WonderlandClientsideNode* contains a reference to the cell containing the TEALsim simulation. Via this cell's utility routine (*sendCellMessage(…)*) messages are transmitted to the server. Figure 8.4 visualizes this design with the most important components involved and annotated with their corresponding interactions. In this figure also the *ChannelActivityMessage* is mentioned which is a current workaround for the issue that the underlying framework does not offer a solution based on the Observer pattern[44] to register for channel activities.

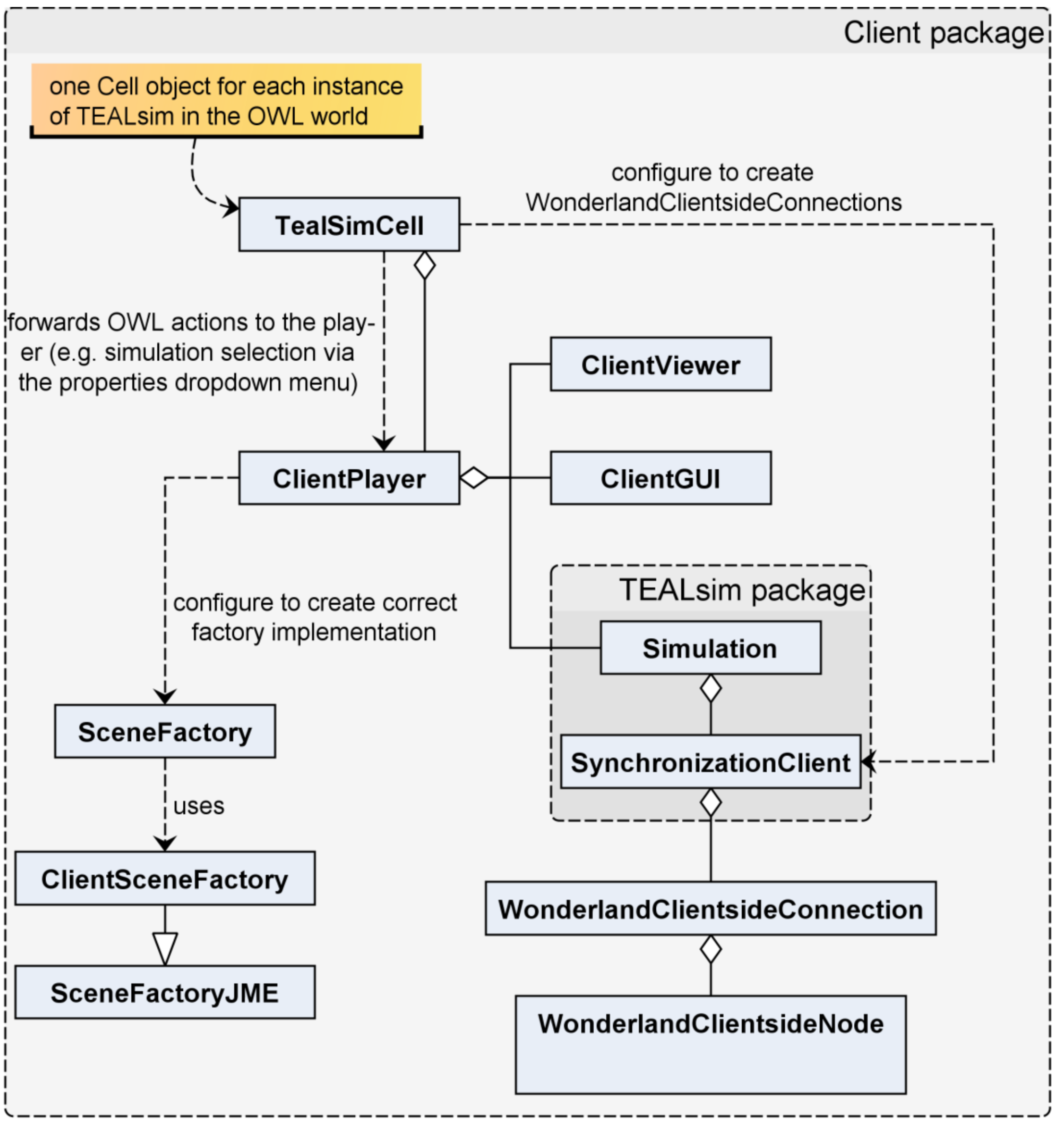

Client package
one Cell object for each instance of TEALsim in the OWL world
TealSimCell
configure to create WonderlandClientsideConnections
forwards OWL actions to the player (e.g. simulation selection via the properties dropdown menu)
ClientViewer
ClientPlayer
ClientGUI
TEALsim package
configure to create correct factory implementation
Simulation
SceneFactory
SynchronizationClient
uses
ClientSceneFactory
WonderlandClientsideConnection
SceneFactoryJME
WonderlandClientsideNode

Figure 8.3: Schema of required client-side components to integrate TEALsim into OWL

[44] Intent of Observer pattern: “Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.” [Gamma, Helm, Johnson, & Vlissides, 2004]

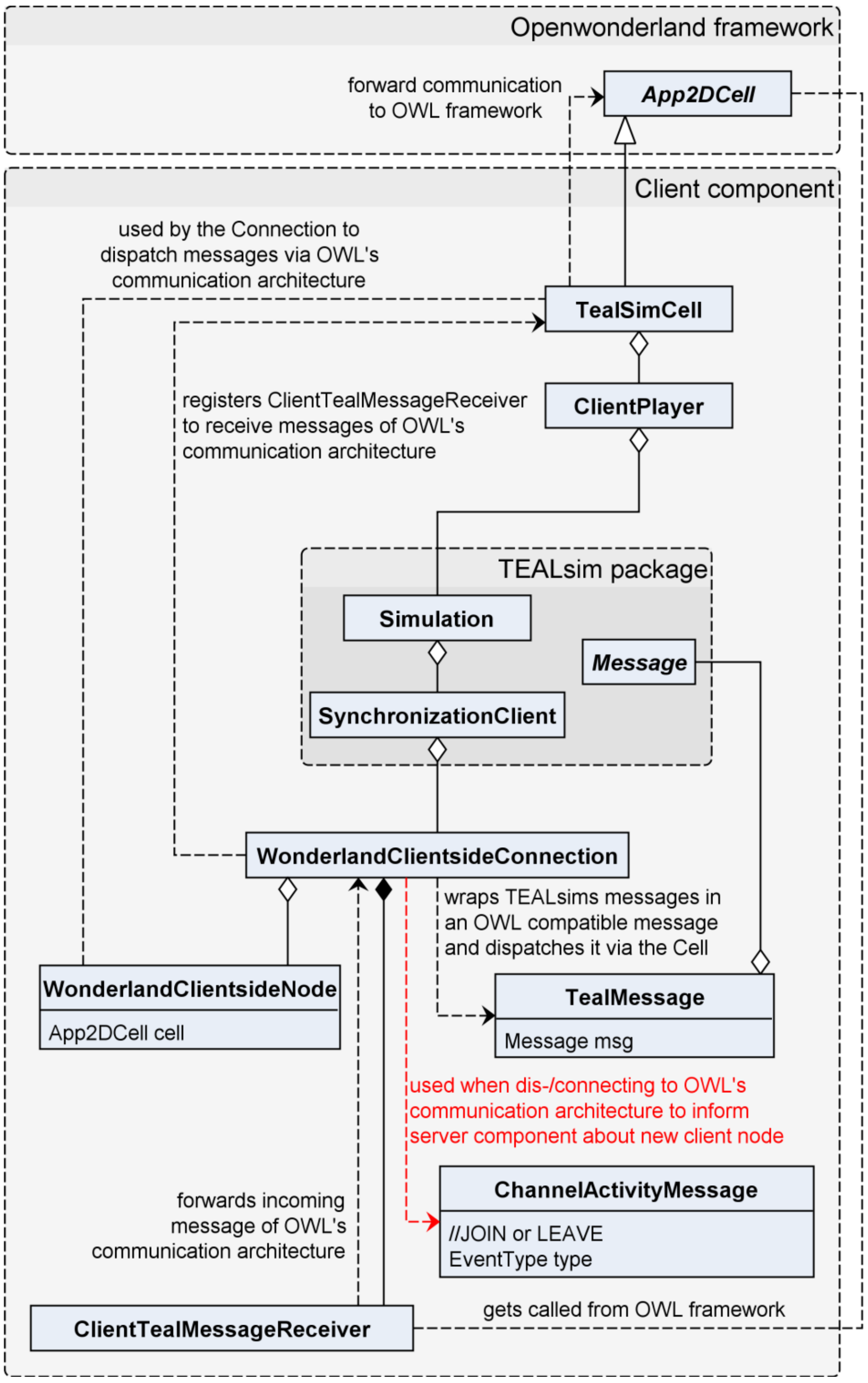

Openwonderland framework
forward communication to OWL framework
App2DCell
Client component
used by the Connection to dispatch messages via OWL's communication architecture
TealSimCell
registers ClientTealMessageReceiver to receive messages of OWL's communication architecture
ClientPlayer
TEALsim package
Simulation
Message
SynchronizationClient
WonderlandClientsideConnection
wraps TEALsims messages in an OWL compatible message and dispatches it via the Cell
WonderlandClientsideNode
App2DCell cell
TealMessage
Message msg
used when dis-/connecting to OWL's communication architecture to inform server component about new client node
ChannelActivityMessage
//JOIN or LEAVE
EventType type
forwards incoming message of OWL's communication architecture
ClientTealMessageReceiver
gets called from OWL framework

Figure 8.4: Schema of the client-side implementation of *Connection* for OWL

8.2.2 Adapting the Server Package

For the time being OWL module's server package is implemented as *CellMO*. A potential (better) alternative would be to implement it as service, however, this task turned out to be beyond the remaining resources for this thesis. The RedDwarf Server infrastructure used by OWL ensures that processing power is split evenly amongst all available modules. As technical consequence of this paradigm all objects used in the server package must implement the *Serializable* interface (see chapter 3.2.3). In general this technique becomes an issue when classes shall be used which are not intended to be serialized (e.g. sockets or threads).

A manifest solution for this issue for the *SynchronizationServer* thread was to manually run the *SynchronizationServer* runtime logic (*executeRuntimeLogic()*) each time the *CellMO* received a new message via OWL's channel system. In the end, this workaround mimics the original *SynchronizationServer* behavior very accurately since the only thing the thread's *run()* method does is to consistently call the runtime logic and idle in absence of messages to process. On the other hand no practical solution was found to enable sockets in *CellMOs*, for which reason e.g. *SocketConnections* are currently not available in the TEALsim OWL module.

Figure 8.5 shows the general schema of the module's server package. Basically no changes to TEALsim's design are required besides extending the *SynchronizationServer* to implement the *Serializable* interface.

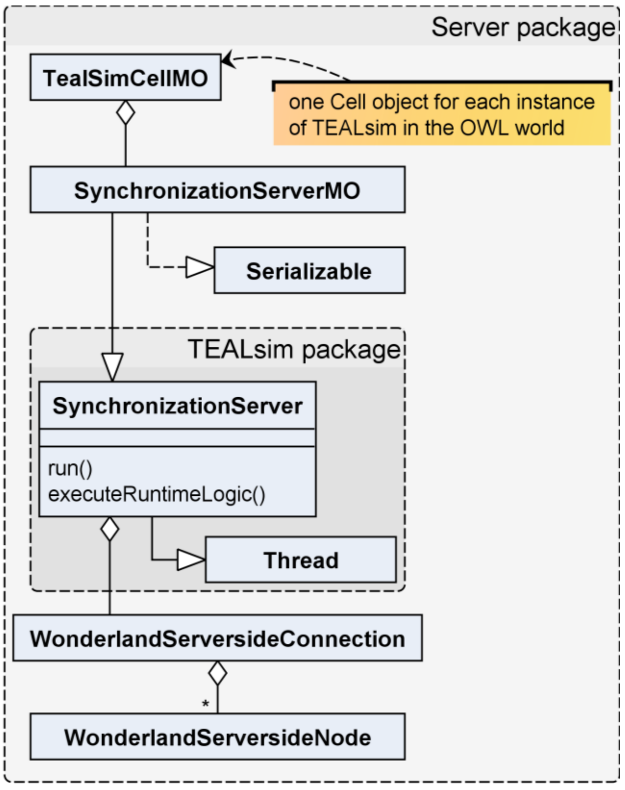


Figure 8.5: Schema of required server-side components to integrate TEALsim into OWL

Like the client-side *Connection* explained in the previous chapter 8.2.1 the server *Connection* also registers a *TealMessageReceiver* to listen for *TealMessages* on the cell's channel. The *MessageReceivers* hold a reference to the *SynchronizationServer(MO)*. Each time a message is received via the cell's channel a *MessageReceiver* sets the source node in the message (which is a transient field – see the explanation to figure 7.2) and calls the *executeRuntimeLogic()* in the *SynchronizationServer*. See figure 8.6 for a graphical schema of this design.

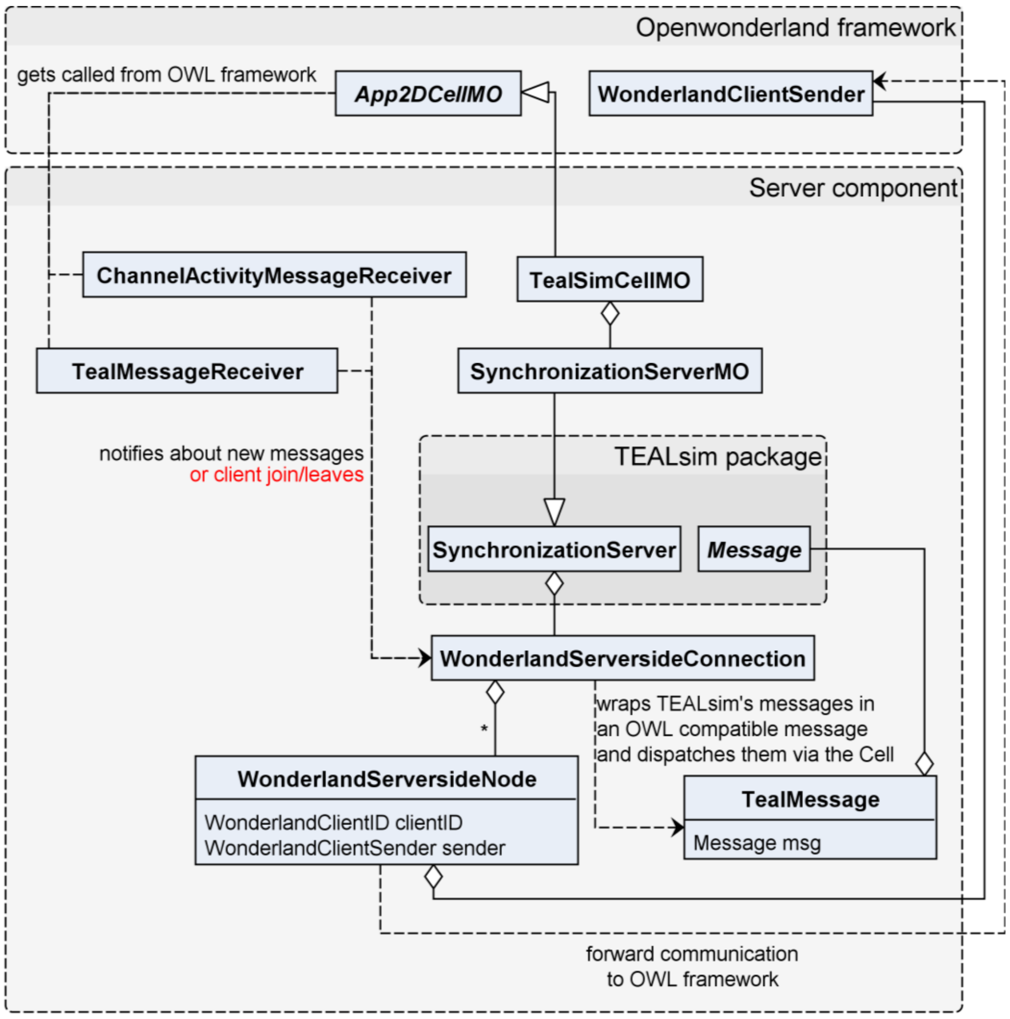

Openwonderland framework
gets called from OWL framework
App2DCellMO
WonderlandClientSender
Server component
ChannelActivityMessageReceiver
TealSimCellMO
TealMessageReceiver
SynchronizationServerMO
notifies about new messages
or client join/leaves
TEALsim package
SynchronizationServer
Message
WonderlandServersideConnection
*
wraps TEALsim's messages in an OWL compatible message and dispatches them via the Cell
WonderlandServersideNode
WonderlandClientID clientID
WonderlandClientSender sender
TealMessage
Message msg
forward communication to OWL framework

Figure 8.6: Schema of the server-side implementation of *Connection* for OWL

8.3 Summary

Currently there are two possible ways to run TEALsim. Either the framework can be started as a standalone process or it can be used via the available OWL module (within OWL). Running the simulations in OWL currently does not allow for any specific user customizations. However, the general concept of the TEALsim OWL module was explained to serve as starting point for further developments in this package. As it stands TEALsim appears and works in OWL the same way as the desktop version which uses a Java *JFrame*. In this regard the module does not return any additional advantages itself but solely benefits from any auxiliary features the virtual world provides.

For the desktop version of TEALsim various options are available to customize its behavior. Most important are switches to adjust the client and server features, the logging output and the graphics engine, which were explained in detail to allow setting up a demonstration system. When parameters are used to start the application a tokenizer creates the appropriate commands including any given parameters. The command-line argument parser (and executor) is quite comprehensive itself. For example dependencies can be defined between arguments. However, documentation of these features relies on the inline documentation of the corresponding classes.

Since this chapter finished documenting a large fraction of the work performed and how to use it, what remains for the subsequent chapters is a personal résumé of the experiences gained during the course of this thesis as well as recommendations where future projects could tie in.

Part III
Findings

Chapter 9

Lessons Learned

This thesis was conducted in multiple stages. It began with initial research into potentially relevant topics (which was conducted in Austria), and then moved on to the programming phase, which occurred during a 6-month stay at MIT. The final phase was the composition of the present document. Although a detailed account of the many lessons learned by the author in the course of the rather prolonged process of researching and composing this thesis is beyond the scope of the present work, the following sub-chapters summarize some of the more significant findings based on different perspectives and phases of the project.

9.1 Conducting Preliminary Research

Finding relevant information for this thesis proved to be challenging. Although the idea to use TEALsim in a distributed environment (along with the concepts outlined in this thesis to achieve this goal) was not a brand-new idea, literature on the relevant topics is sparse. Part of the problem is that, in general, there is no comprehensive, one-size-fits-all solution for multi user systems because each software project is somewhat customized. In addition, the applicability of any existing guidelines is also limited because such guidelines normally assume that a project is starting from scratch. However, this prerequisite does not apply for this thesis, since it is based on an existing, complex physics simulation project of considerable scale, which introduced various predetermined constraints.

Furthermore, it turned out to be quite difficult to gather information about the current status of TEALsim in order to determine the most reasonable next steps to achieve the present objectives. Other than some older documents describing the last major, official release of the framework, the only source of information beyond the in-line documentation was a limited number of other student theses. Naturally, experimental code is not a very accessible source of information. This experience demonstrated that proper documentation is not simply a desirable “extra”, but rather an essential component of a good product.

9.2 Design and Implementation

As the reader can see, there is a difference between the thesis' conclusion (see chapter 10) and the initial objectives (see chapter 1.2). The reason for this is that, lacking in-depth knowledge of the available software tools, objectives were defined based on assumptions that proved to be incorrect. Although the concept initially appeared to work quite well, deeper examination revealed that the framework was in many cases quite experimental and included many design aspects that were unable to meet the defined goals. For this reason, as work on this thesis progressed, the focus gradually shifted from the higher-level objectives to lower level fundamentals. In essence, the research work for this thesis ended up getting bogged down in a wide variety of specific problems that no individual could hope to address within the scope of a master's thesis. From this experience, the author learned that technical concepts are often much more complicated than they initially appear, and it is better to define small deliverables rather than grand visions, since such visions can lead to situations where many problems are identified, but few, if any, are solved.

9.3 Usability

One of the main driving forces behind this thesis was to enable the use of virtual worlds in the context of e-learning. Ultimately, however, one sobering key question remains: Does deploying the TEALsim framework within a virtual environment provide sufficient measurable benefits in terms of improved teaching to justify the significant work required to properly adapt TEALsim for such an environment. Unfortunately, a large-scale test run of the prototype implementation presented here with real students was beyond the scope of this paper. Since this was recognized at an early stage, the focus here was on maintaining maximum software architecture flexibility, so that it could eventually be run in any given distributed host environment (e.g. a 2D-browser-based learning platform, virtual worlds).

Even though new technologies that enable previously unfeasible technical implementations may seem very alluring to subject-specific experts, one important thing to bear in mind is their ultimate accessibility true benefit for less versed end-users. Belanger and Jordan nicely summarize this principle specifically for the context of distance education by asserting that "*Distance teaching* [...] *involves delivering education or training material while not being physically present at the same location as the students. Distance learning, on the other hand,* [...] *is closely tied to distance teaching, but learning may not occur in the distance environment if barriers exist from the learners' point of view, such as difficulty in using the technology, or lack of instructor interaction when answering questions.*" [Belanger & Jordan, 2000, p. 9]

Chapter 10

Outcome

The following two chapters present the final results. The overall status quo is first summarized in a few paragraphs, and the subsequent chapter then lists remaining open issues and questions which could be tackled by future projects.

10.1 Conclusion

The practical work performed for this master's thesis resulted in a prototype that demonstrates how TEALsim can be altered to allow it to run in distributed environments. To achieve this objective, it was necessary to become acquainted with many different scientific fields in order to identify and implement solutions for issues in areas such as the design of graphics engines, software tools to synchronize concurrent computer programs, and network infrastructures.

The major initial objective, the deployment of TEALsim in a 3D virtual world and an investigation of this combination's potential for e-learning, eventually shifted to adapting TEALsim's design so that it could not only cope with such a task, but would also feature the characteristics that are generally associated with the concept of "good software design", such as:

- Consistency
- Comprehensibility
- Scalability
- Extensibility
- Making effective use of available resources (e.g. network bandwidth or CPU load)

However, in the end, the possibility to embed TEALsim in a virtual world like OWL was the inevitable side-product of a well-thought-out design. In the long run, the extensive changes necessary to adapt the simulation framework resulted in a list of open issues, since the concepts outlined in this thesis went beyond what could be fully implemented within the scope of this master's thesis. For this reason, the subsequent sub-chapters summarize several topics that will require additional investigation and development.

10.2 Suggestions for Further Work

This sub-chapter describes other projects that could supplement the work done for the current thesis and thereby finish the prototype implementation described herein.

Important issues to be resolved

- The network protocol defining the communication between clients and server has to be specified and implemented in greater detail. Currently, only a rudimentary handshake procedure exists, whereby the client says '*hello*' to the server and is thereafter regarded as being connected. However, many more use cases have to be considered, such as the need for clients connecting to running simulations to receive the current state, or a routine which ensures that all clients run with an approximately identical SimulationTime (see use case 5.1).
- A concept is needed to synchronize the use of randomized values in the TEALsim framework.
- An analysis is required to identify all low-level parts defining a simulation's state. The set of all of these values is the variable referenced as '*EngineState*' in this thesis. TEALsim needs some means to read and write the *EngineState*. On the server side, the *EngineState* has to be stored with the corresponding SimulationTime so it can be provided to newly connecting clients. In the work done by Berger [2012], the workaround for this requirement was to serialize the whole simulation. However, this approach leads to serious issues in multi user systems in which peers rely on locally installed libraries (e.g. the JRE). With regard to the JRE, it becomes difficult to ensure that all peers' libraries provide the same versions of the classes used in simulations. For example, it happened that a client used a JRE with a different update version than the server, which caused the client to fail when deserializing the simulation data.
- Once *EngineStates* can be read and written, the simulation engine has to be adapted to store the states for key frames and use them as a basis for computing subsequent key frames.
- While stepping forward in a paused simulation works, the inverse action of stepping backward currently does not work.
- A bug in the JME library seems to prohibit the canvas from being closed. This is particularly problematic because it prevents effective unit testing of the TEALsim desktop application.
- The JME version of TEALsim is generally in need of maintenance to meet the same level of visual quality and stability the J3D version offers.

- An analysis is required to evaluate how the concepts outlined in this thesis conform to any constraints imposed by the Google Web Toolkit[45]. This is important because the MITx: 8.02x Electricity and Magnetism course uses cross-compiled TEALsim simulations to support its curriculum.

Issues in need of a redesign

The issues outlined below are not as severe as the topics discussed in the previous section. However, resolving the following problems would definitely make the prototype implementation more comprehensible and efficient:

- The *SimulationEngine* algorithm would need improvements to become more stable in situations where the required computing power exceeds the available hardware capacities. The current logic has the drawback that in situations where the hardware is unable to calculate at least the key frame within the given KEY_FRAME_INTERVAL, period delays are accumulated. This causes the algorithm to remain unresponsive for an increased amount of time, even if, for example, the simulation was changed by the user to a state which could be calculated faster (e.g. the amount of elements in the simulation was decreased).
- The simulation engine has a single-threaded execution flow which computes the next frame more or less on demand (that is, when it is required for rendering). A better solution would be a consumer/producer scenario where *EngineStates* are calculated (that is, “produced”) in advance and eventually “consumed” by the rendering engine. This would presumably also help to change the engine’s design to a true multi-threaded execution model, which would in turn yield increased performance on virtually any hardware.
- ID generation of *SynchronizableGenerics* currently has to be performed manually by content developers. It would be interesting to determine if there is a generic (automatic) way to do this, or if it must always be up to the user to set unique IDs.
- Currently, TEALsim still contains a significant amount of inconsistent design, which creates a redundant, bloated codebase. For example, factory methods that could be configured to produce the right type of objects are not utilized throughout the TEALsim (and its correlated OpenWonderland module) project. In this context, the *ClientPlayer* class in the OWL module might be cleaned up considerably – potentially up to the point where it would become a featureless shell. Other examples include classes such as *teal.ui.swing.JTaskPaneInfo* or *teal.ui.swing.JLinkButtonBeanInfo*, which are not used at all.
- The TEALsim OWL module is currently implemented as a *CellMO* object. It would be worth investigating whether or not it would be more efficient to run it as an OWL service.

[45] GWT framework – http://www.gwtproject.org/

List of “nice-to-have” features

During development of the prototype implementation, the following topics were recognized as helpful features, which, however, have no immediate impact on the concepts outlined in this thesis:

- Selecting, picking and dragging simulation elements via the mouse.
- Verification of simulations’ soundness via unit tests.
- Formal specification of Ant build scripts deliverables.
- Changing the remaining parts of TEALsim to utilize the *java.util.logging* framework.
- Fixing the broken unit test which searches for circular dependencies in the set of existing derivations of *RuntimeArgument*. Since such derived classes are no longer inner static classes of *RuntimeArgument*, the technique to query for all declared classes in *RuntimeArgument* obviously no longer yields results.

Part IV

Appendix

Chapter 11

References

American Society for Training & Development. (2013). ASTD Glossary. Retrieved June 13, 2013, from http://www.astd.org/Publications/Newsletters/Learning-Circuits/Glossary

Anger, C., Geis, W., & Plünnecke, A. (2012). *MINT – Frühjahrsreport 2012.* Institut der deutschen Wirtschaft Köln, Köln. Abgerufen am 21. June 2013 von http://www.iwkoeln.de/_storage/asset/86135/storage/master/file/1898328/download/MINT_Fr%C3%BChjahrsreport_2012-05-20.pdf

Belanger, F., & Jordan, D. H. (2000). *Evaluation and Implementation of Distance Learning: Technologies, Tools and Techniques.* London: Idea Group Publishing.

Belcher, J. W. (2001). Studio Physics at MIT. *MIT Physics Annual 2001*, pp. 58-64. Retrieved May 29, 2013, from http://web.mit.edu/jbelcher/www/PhysicsNewsLetter.pdf

Belcher, J., McKinney, A., Bailey, P., & Danziger, M. (2007, December 16). TEALsim: A Guide to the Java 3D Software (Version 1.1). Cambridge, Massachusetts, USA.

Bell, M. W. (2008, July). Toward a Definition of "Virtual Worlds". *Journal of Virtual Worlds Research, 1*(1). Retrieved June 1, 2013, from http://journals.tdl.org/jvwr/index.php/jvwr/article/view/283/237

Berger, S. (2012, April 12). *Virtual 3D World for Physics Experiments in Higher Education*(Master's Thesis). Graz, Austria: Graz University of Technology. Retrieved August 6, 2013, from http://www.iicm.tugraz.at/thesis/Berger,_Stefan_Masterarbeit.pdf

Bernier, Y. W. (2001). *Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization*. Retrieved April 14, 2013, from Valve Developer Community: https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization

Bradt, S. (2012, November 20). *Sanjay Sarma appointed as MIT's first director of digital learning*. Retrieved August 15, 2013, from MIT News Office:

http://web.mit.edu/newsoffice/2012/sanjay-sarma-director-of-digital-learning-1120.html

Brown, W. J., Malveau, R. C., McCormick III, H. W., & Mowbray, T. J. (1998). *Anti Patterns.* New York: John Wiley & Sons, Inc.

Burke, E. M., & Coyner, B. M. (2003). *Java Extreme Programming Cookbook* (1st ed.). Sebastopol: O'Reilly & Associates, Inc.

Bybee, R. W. (2010, August 27). What Is STEM Education? *Science, 329*(5995), 998-1112. doi:10.1126/science.1194998

Center for Educational Computing Initiatives (CECI). (2012, April). *TEALsim Project at MIT*. Retrieved from Massachusetts Institute of Technology: http://web.mit.edu/viz/soft/visualizations/tealsim/index.html

Dahm, M. (2005). *Grundlagen der Mensch-Computer-Interaktion* (2te ed.). Pearson Studium.

Domjan, M. (2009). *The Principles of Learning and Behavior* (6th ed.). Wadsworth Inc Fulfillment.

Dori, Y. J., & Belcher, J. (2005). How Does Technology-Enabled Active Learning Affect Undergraduate Students' Understanding of Electromagnetism Concepts? *Journal of the Learning Sciences, 14*(2), 243-279. doi:10.1207/s15327809jls1402_3

Dori, Y. J., Belcher, J., Bessette, M., Danziger, M., McKinney, A., & Hult, E. (2003, December). Technology for active learning. *Materials Today, 6*(12), pp. 44-49. Retrieved May 23, 2013, from http://web.mit.edu/edtech/casestudies/pdf/teal2.pdf

Ericsson. (2012). *Traffic and market report - On the pulse of the networked society.* Stockholm: Ericsson AB. Retrieved June 23, 2013, from http://www.ericsson.com/res/docs/2012/traffic_and_market_report_june_2012.pdf

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2004). *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education.

Hamill, P. (2005). *Unit Test Frameworks* (1st ed.). Sebastopol: O'Reilly Media, Inc.

Hammerschall, U. (2005). *Verteilte Systeme und Anwendungen: Architekturkonzepte, Standards und Middleware-Technologien.* München: Pearson Studium.

Hardesty, L. (2012, July 16). *Lessons learned from MITx's prototype course*. Retrieved August 15, 2013, from MIT News Office: http://web.mit.edu/newsoffice/2012/mitx-edx-first-course-recap-0716.html

Holub, A. I. (2011, September 26). *Allen Holub's UML Quick Reference*. Retrieved April 14, 2013, from Hollub: http://www.holub.com/goodies/uml/index.html

InforMatrix GmbH. (2007). *InforMatrix LogGui.* Retrieved April 14, 2013, from http://www.informatrix.ch/informatrix/loggui/index.html

Jantke, K. P. (2010). Toward a Taxonomy of Game Based Learning. *2010 IEEE International Conference on Progress in Informatics and Computing. 2*, pp. 858-862. Shanghai: IEEE. doi:10.1109/PIC.2010.5687903

Juneidi, S. J., & Vouros, G. A. (2005). Engineering an E-learning Application Using the ARL Theory for Agent Oriented Software Engineering. *2005 AAAI Fall Symposium* (pp. 87-92). Arlington, Virginia: The AAAI Press. Retrieved June 13, 2013, from http://www.aaai.org/Papers/Symposia/Fall/2005/FS-05-08/FS05-08-014.pdf

Kafai, Y. B. (1996). Gender differences in children's constructions of video games. In P. M. Greenfield, & R. R. Cocking, *Interacting with video* (p. 218). Norwood: Ablex Publishing Corporation. Retrieved June 18, 2013, from http://www.gse.upenn.edu/~kafai/paper/pdfs/GenderDifferences.pdf

Kaplan, J. (2012, March 28). *Open Wonderland Forum: Utilization of Multicast in OWL*. Retrieved April 14, 2013, from Google Groups: http://groups.google.com/group/openwonderland/browse_thread/thread/5a6cdec9ca72fd75

Kheyfit, A. (2010). *A Primer in Combinatorics.* Berlin: Walter de Gruyter GmbH & Co. KG.

Massachusetts Institute of Technology. (2005). *Educational Transformation through Technology at MIT - TEAL*. Retrieved August 14, 2013, from MIT Educational Technology: http://web.mit.edu/edtech/casestudies/teal.html

Massachusetts Institute of Technology. (2012, March 1). *TEALsim Project at MIT*. Retrieved April 14, 2013, from http://web.mit.edu/viz/soft/visualizations/tealsim/index.html

Mencke, S., & Dumke, R. R. (2007). *Agent-Supported e-Learning.* Magdeburg: Otto-von-Guericke-Universität Magdeburg.

Miranda, L. C., & Lima, C. A. (2012, May). Trends and cycles of the internet evolution and worldwide impacts. *Technological Forecasting and Social Change, 79*(4), 744–765. doi:10.1016/j.techfore.2011.09.001

Morella, M. (2012, Jule 26). *U.S. News Inducts Five to STEM Leadership Hall of Fame.* Retrieved June 21, 2013, from U.S. News & World Report: http://www.usnews.com/news/blogs/stem-education/2012/07/26/us-news-inducts-five-to-stem-leadership-hall-of-fame

O'Brien, D. (2010). A Taxonomy of Educational Games. In *Gaming and Simulations: Concepts, Methodologies, Tools and Applications* (Vol. 1, p. 2164). London: Information Science Reference.

Open Wonderland Foundation. (2012, October 16). *Home: Open Wonderland*. Retrieved April 14, 2013, from Open source 3D virtual collaboration toolkit | Open Wonderland: http://openwonderland.org/index.php

Parlett, D. S. (1999). *The Oxford History of Board Games* (1st ed.). Oxford: Oxford University Press.

Peachey, A., Gillen, J., Livingstone, D., & Smith-Robbins, S. (2010). Editors' Introduction: The Physical and the Virtual. In A. Peachey, J. Gillen, D. Livingstone, & S. Smith-Robbins (Eds.), *Researching Learning in Virtual Worlds* (pp. xv-xxviii). London: Springer. doi:10.1007/978-1-84996-047-2

Pirker, J. (2012, December). Design and Implementation of Virtual Three-Dimensional E-Learning Scenarios of Physics Simulations. (Master's Thesis). Graz, Austria: Graz University of Technology.

Poynton, C. (2002). *Digital Video and HDTV.* Morgan Kaufmann.

RedDwarf Server Application Tutorial. (2010, March). Retrieved April 14, 2013, from RedDwarf Server Project: http://sourceforge.net/apps/trac/reddwarf/attachment/wiki/Documentation/RedDwarf%20ServerAppTutorial.odt

Salen, K., & Zimmerman, E. (2003). *Rules of Play: Game Design Fundamentals.* Cambridge, Massachusetts: The MIT Press.

Sarma, S. E., & Chuang, I. (2013, May / June). The Magic Beyond the MOOCs. *MIT Faculty Newsletter, XXV*(5), pp. 1; 10-12. Retrieved August 15, 2013, from MIT Faculty Newsletter: http://web.mit.edu/fnl/volume/255/fnl255.pdf

Scheucher, B. (2010, March). *Remote Physics Experiments in 3D*(Master's thesis). Graz, Austria: Graz University of Technology. Retrieved April 14, 2013, from http://www.iicm.tugraz.at/thesis/MA_%20Bettina_Scheucher.pdf

Shelly, G. B., & Rosenblatt, H. J. (2011). *Systems Analysis and Design* (9th ed.). Boston: Course Technology.

Soh, L.-K., Miller, L. D., Blank, T., & Person, S. (2004). *ILMDA: Intelligent Learning Materials Delivery.* Computer Science and Engineering. University of Nebraska-Lincoln. Retrieved June 18, 2013, from http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1104&context=csetechreports

Stack Overflow. (2008, November 27). *Retain precision with Doubles in java*. Retrieved August 8, 2013, from Stack Overflow: http://stackoverflow.com/questions/322749/retain-precision-with-doubles-in-java

Trevey, M. T. (2008, October). STEM Education - The JASON Project is One Innovation. *State News, 51*(9), 34-35. Retrieved June 23, 2013, from http://www.csg.org/pubs/capitolideas/capitolideas_archive/statenews_archive_pdfs/sn_2008/sn0810.pdf

Unesco. (1987). *Distance Learning Systems and Structures: Training Manual : Report of a Sub-regional Training Workshop, Colombo, Sri Lanka, 5-18 July 1984.* Regional Office for Education in Asia and the Pacific and Asia and the Pacific Programme of Educational

Innovation for Development. Bangkok: Unesco Regional Office for Education in Asia and the Pacific.

Weisstein, E. W. (2013). *Runge-Kutta Method*. Retrieved October 28, 2013, from Wolfram MathWorld: http://mathworld.wolfram.com/Runge-KuttaMethod.html

Wittgenstein, L. (1958). *Philosophical Investigations* (2nd ed.). (G. E. ANSCOMBE, Trans.) Oxford: Basil Blackwell Ltd. Retrieved June 1, 2013, from http://ebookbrowse.com/ludwig-wittgenstein-philosophical-investigations-pdf-d71272821

Zhong, Z.-J. (2011, November). The effects of collective MMORPG (Massively Multiplayer Online Role-Playing Games) play on gamers' online and offline social capital. *Computers in Human Behavior, 27*(6), 2352-2363. doi:10.1016/j.chb.2011.07.014

Zillner, C. (2012, May). Editorial. *Falter Heureka*(21/12), p. 3.

Chapter 12

Remarks

12.1 Diagram legends

The diagrams, schemas, charts, etc. in this thesis do not strictly adhere to any specific standard (like the UML standard[46]). Basically there are 2 reasons for this decision:

a) The chapters of this thesis related to TEALsim and the TEALsim Open Wonderland module do not intend to serve as a full technical documentation in the narrow sense, but shall rather give an overview of the performed programming work and the ideas and concepts behind the implemented features.

b) Almost all diagrams were created manually (instead of automatic generation from source code), not least because no suitable, affordable tools were available during the course of the project.

For this reason the diagrams sometimes mix different sets of graphical notations, even though effort was taken to stick with familiar and wide spread norms. After all, the careful selection of content by a human being – to keep the drawings as minimalistic as possible while maximizing their pictured information – should outweigh the loss of standardization by far. Subsequently follow descriptions of the most commonly used notations in class diagrams and flow chart diagrams.

[46] Specification of the UML standard - http://www.omg.org/spec/UML/

12.1.1 Class Diagrams

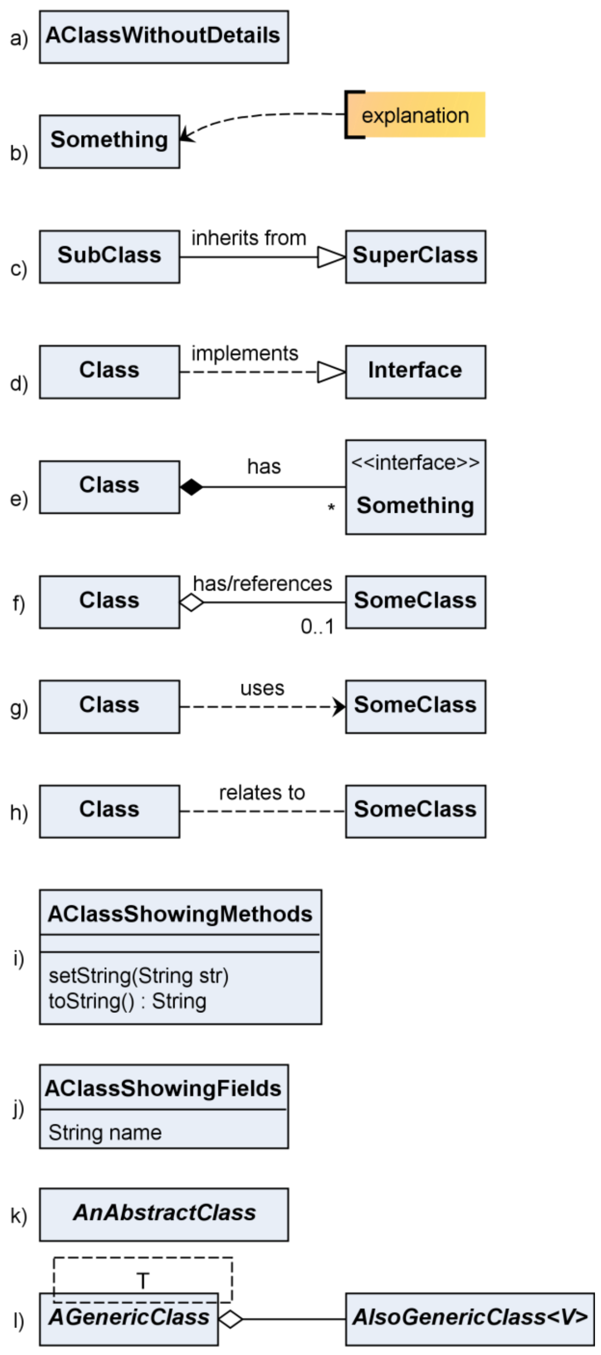


Figure 12.1: Drawings with UML class diagram like content

a) Usually class/interface/etc. representations (during the course of this sub-chapter referenced as ‘*elements*’) are kept as minimalistic as possible, hiding most or all methods/fields/etc. (during the course of this sub-chapter referenced as ‘*properties*’).

Therefore a simple box without any other information but its name does not imply that there is no functionality contained in this element, but it shall rather be seen as a black box fulfilling all of the tasks which can be logically derived from the name. If a particular property may be of special interest or may further help to clarify the purpose of the element, it may be included in the drawing.

Usually access levels of properties are omitted, but generally one can assume that all properties included in drawings are publically accessible, either directly or via access methods.

b) If something may not be immediately evident out of the drawing but might be necessary or useful for its understanding (respectively to understand the context of the pictured elements), a box with an orange to yellow gradient was used to add more details in a textual form.

c) Inheritance of one element from another is indicated with a solid line and an arrow, usually without the textual hint attached to the line.

d) A class implementing an interface is indicated with a dashed line and an arrow, usually without the textual hint attached to the line. In some places a stereotype notation is used to indicate that a box constitutes an interface (for example if a class relates to other classes via an aggregation and requires the related classes to implement a certain interface – for instance the next item e)).

e) Since composition and aggregation symbols are very often controversially defined in literature, little effort was taken to align all diagrams with one final, strict rule. Usually a composition in this thesis indicates that the part-element may (or at least should) not exist without the owner-element. Due to the blurry difference between composition and aggregation, without doubt one notation may be substituted by the other occasionally.

Additionally, in this thesis a specialty of Java – Inner Classes – are usually connected to their parent class with a composition. While there exist various recommendations for non-standardized symbols like [Holub, 2011], for the sake of simplicity this approach was chosen.

If omitted, then multiplicity is assumed to be 1

f) As indicated in item e) aggregations usually outline related elements which may exist independently. Again, if omitted, then multiplicity is assumed to be 1.

g) Shows a class which uses another class in some way. Usually additional textual information is available indicating the purpose of this interaction.

h) Shows a class which relates to another class in some way. Usually additional textual information is available indicating the purpose respectively nature of this relation.

i) Shows an element where at least two methods seem to be of particular interest for its understanding. While the *setString(…)* method has a void return type, the *toString()* method returns an object of type 'String'

j) Shows a class where at least the 'name' field seems to be of particular interest for its understanding.

k) Classes showing an italicized name are definitely abstract.

l) Shows two associated, abstract generic classes. Usually the representation style of the AGenericClass is preferred, with the parameter in a separate box above the class. However, the second representation form of AlsoGenericClass might be used analogously as well.

Additionally group nodes were used in various places to give a hint about how classes belong together in a broader sense, but this notation should be self-explanatory.

12.1.2 Flow Chart Diagrams

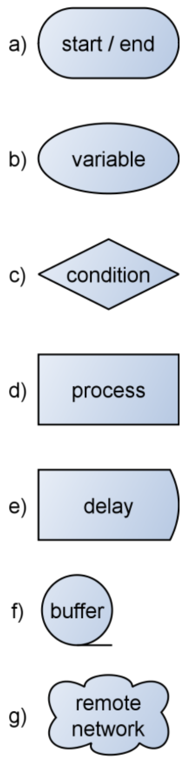


Figure 12.2: Flow chart diagrams

a) Termination symbols are boxes with round corners. They are used to mark the start and end point of an algorithm.

b) Ellipses are used to outline important variables which are changed. Sometimes multiple variables are contained within one ellipse to shrink the diagram size.

c) Diamonds indicate a condition check. *Usually* this is a true/false evaluation where true conditions are connected to the diamond’s left corner and false conditions connect to the right corner.

d) Simple actions are usually annotated directly next to an arrow leading from one symbol to another. If a more comprehensive task has to be executed then a square box is used briefly describing what shall happen in this place (like a black box).

e) Specifies steps which cause the algorithm to idle for a specific time.

f) Shows buffers which contain multiple instances of the denoted variable.

g) Clouds indicate a remote entity like other computers in network.

12.2 System Diagrams

This chapter covers various technical aspects of TEALsim which did not particularly fit to any of the main chapters of this thesis but might be interesting for someone wanting to get involved with TEALsim and continue its development.

12.2.1 Design of the Revamped Simulation Engine

The former simulation engine was able to enter 5 different states and switch forth and back amongst them, resulting in a rather complex state diagram. A byproduct of the work on this thesis was the reduction of possible states to two as shown in figure 12.3.

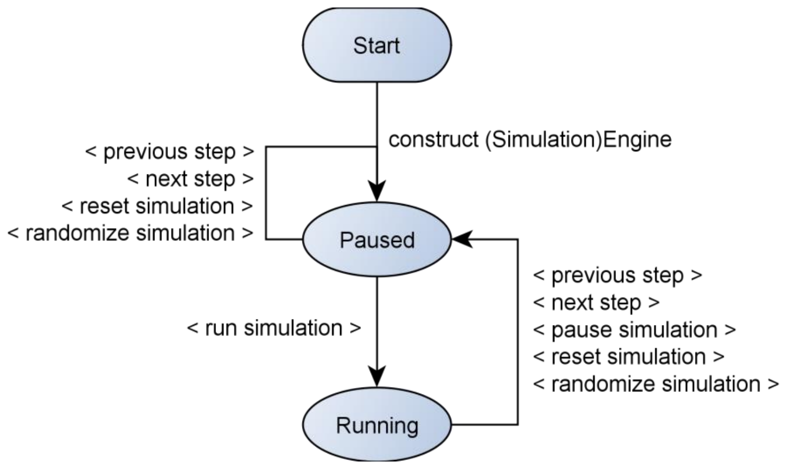


Figure 12.3: State diagram of the deterministic simulation engine

12.2.2 System Design and Object Ownership in TEALsim

Figure 12.4 visualizes the execution flow and object ownership of the TEALsim framework in place at the start of this thesis. The mutually entangled object references already indicate the absence of clear rules of conduct for object ownership which rendered the whole system very sensible to changes in its design. For this reason substantial changes were introduced leading to the design visualized in figure 12.5 (automatically generated class diagram).

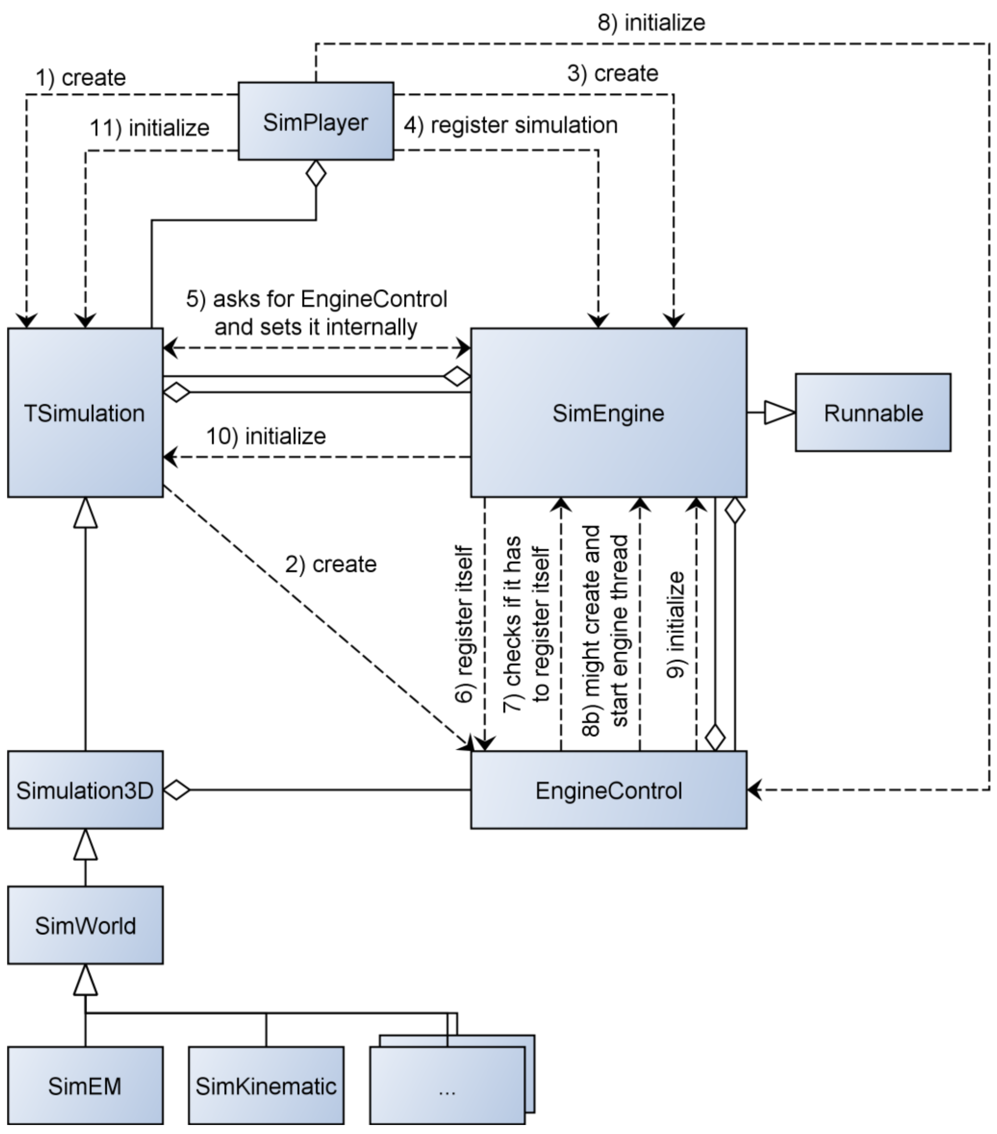


Figure 12.4: Execution flow and object ownership of former simulation construction

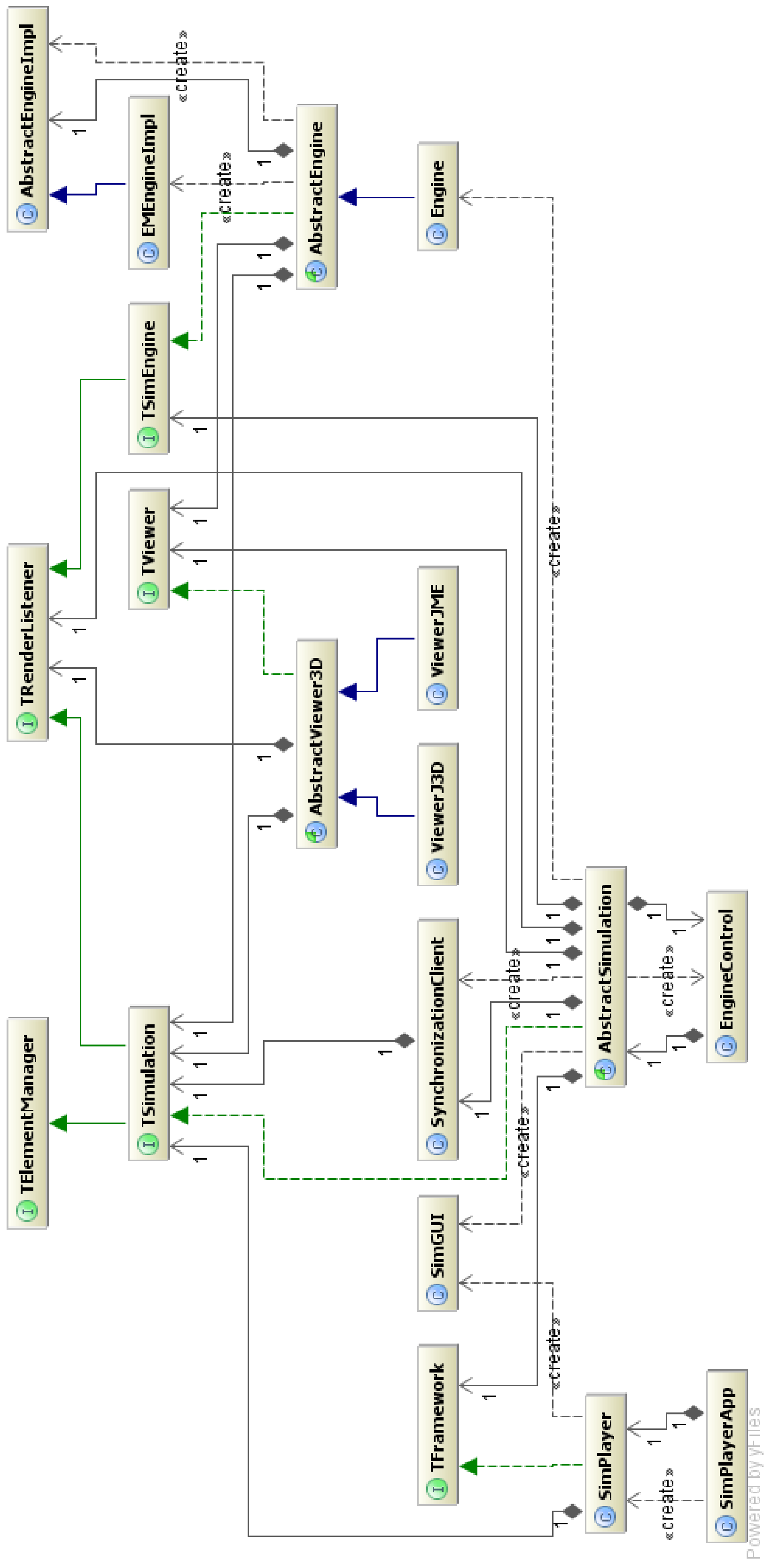


Figure 12.5: Design of TEALsim (based on client-server architecture)

12.2.3 Application Flow of the Rendering Process

To better comprehend TEALsim's design of the rendering process figure 12.6 visualizes the most important components involved in this procedure.

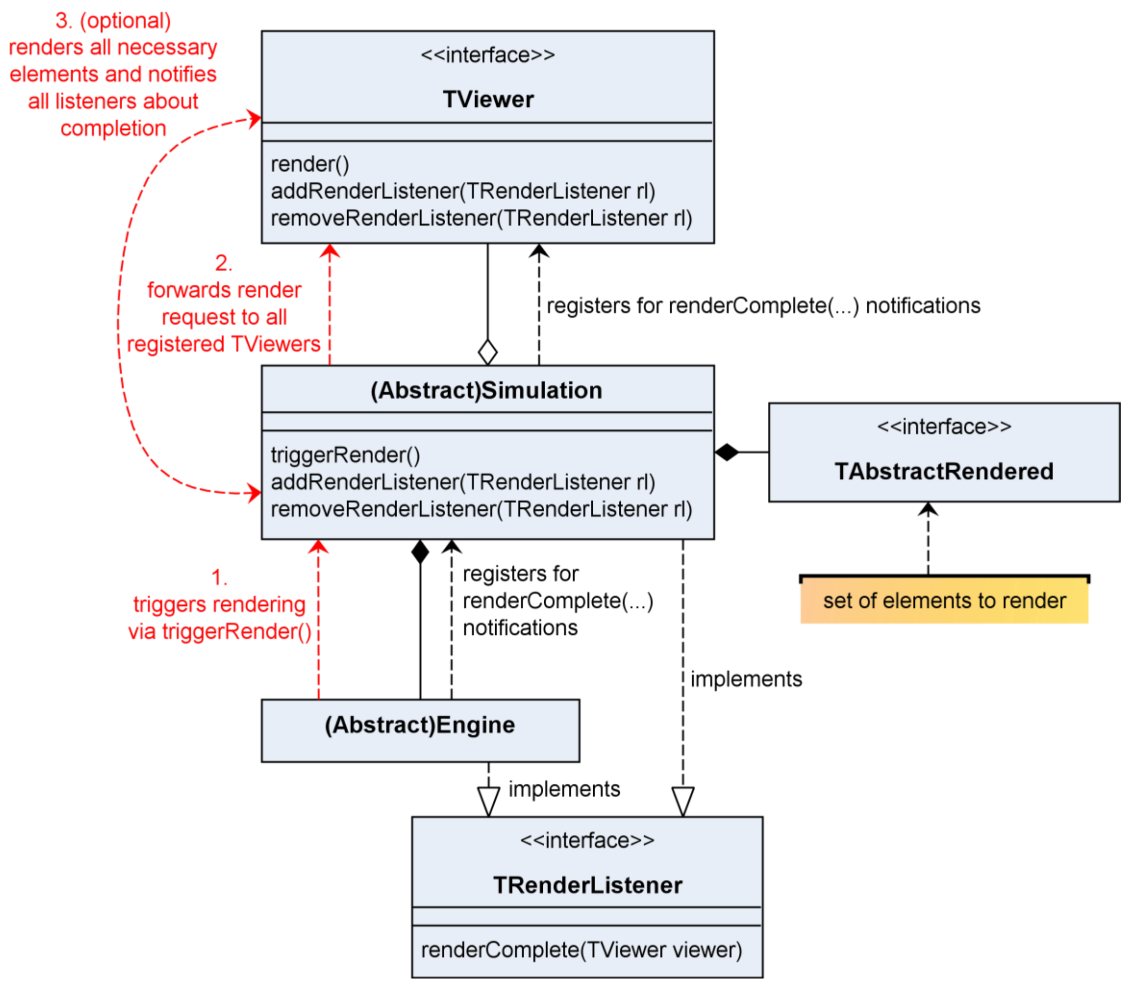

3. (optional) renders all necessary elements and notifies all listeners about completion
<<interface>>
TViewer
render()
addRenderListener(TRenderListener rl)
removeRenderListener(TRenderListener rl)
2. forwards render request to all registered TViewers
registers for renderComplete(...) notifications
(Abstract)Simulation
triggerRender()
addRenderListener(TRenderListener rl)
removeRenderListener(TRenderListener rl)
<<interface>>
TAbstractRendered
set of elements to render
1. triggers rendering via triggerRender()
registers for renderComplete(...) notifications
implements
(Abstract)Engine
implements
<<interface>>
TRenderListener
renderComplete(TViewer viewer)

Figure 12.6: Schema of the rendering process in TEALsim

Chapter 13

Listings

13.1 Table of Figures

13.2 Table of Tables

13.3 Table of Listings

13.4 Table of Use Cases

Chapter 14

Index

14.1 Glossary

API	–	application programming interface
ASTD	–	American Society for Training & Development
CECI	–	Center for Educational Computing Initiatives
CVS	–	Concurrent Versions System or Concurrent Versioning System
fps	–	frames per second
GWT	–	Google Web Toolkit
IPC	–	InnerProcessConnection
IT	–	Information technology
JRE	–	Java Runtime Environment
JVM	–	Java virtual machine
kLOC	–	lines of code x 1000
LAN	–	Local area network
MIT	–	Massachusetts Institute of Technology
MMORPG	–	Massive Multi Player Online Role Playing Games
NTSC	–	National Television System Committee
OWL	–	Open Wonderland
PAL	–	Phase Alternating Line
PCE	–	PropertyChangeEvent
PD	–	Project Darkstar
STEM	–	science, technology, engineering, and mathematics
TCO	–	total cost of ownership
TEAL	–	Technology Enabled Active Learning
TEALsim	–	TEAL Simulation Framework

UI	–	user interface
WAN	–	Wide area network
WBT	–	Web-based training

14.2 Table of the Most Important Software Tools Used

TEAL Simulation Framework
http://web.mit.edu/viz/soft/visualizations/tealsim/index.html
http://sourceforge.net/projects/tealsim/

Open Wonderland
http://www.openwonderland.org

Netbeans IDE
http://www.netbeans.org

IntelliJ Idea Ultimate (evaluation version)
http://www.jetbrains.com/idea/

yEd Graph Editor
http://www.yworks.com

Balsamiq Mockups
http://www.balsamiq.com

CLOC
http://cloc.sourceforge.net

EasyMock
http://www.easymock.org

Ant-Contrib Tasks
http://ant-contrib.sourceforge.net

InforMatrix LogGui
http://www.informatrix.ch/loggui/index.html

TopThreads JConsole plug-in
http://lsd.luminis.nl/top-threads-plugin-for-jconsole/

14.3 Digital assets