Virtual 3D World for Physics Experiments in Higher Education

Implementation of Physics Experiments in a Virtual World

MASTER THESIS

Author:
Stefan Berger,
Graz University of Technology

April 12, 2012

Supervisor:
Univ.-Doz. DI Dr.techn. Christian Gütl,
Graz University of Technology

Co-Supervisor:
Associate Director V. Judson Harvard,
Massachusetts Institute of Technology

Physikexperimente für Hochschulen in einer virtuellen 3D-Welt

Implementierung von Physikexperimenten in einer virtuellen Welt

MASTERARBEIT

Autor:
Stefan Berger,
Technische Universität Graz

12. April 2012

Betreuer:
Univ.-Doz. DI Dr.techn. Christian Gütl,
Technische Universität Graz

Associate Director V. Judson Harvard,
Massachusetts Institute of Technology

Massachusetts
Institute of
Technology

Abstract

E-learning software has the ability to enhance students performance significantly. The improvements in hard- and software over the last decade enables developers to implement extendible real-time simulations to increase the learning effect. Not only e-learning software does benefit from the advancement of technology, but also virtual worlds. Especially high-speed Internet which is broadly available nowadays is essential for the use of virtual 3D worlds with complex functionalities.

Only recently has research in the field of e-learning discovered the potential of virtual worlds. For teaching it has several advantages over the conventional "real-world" approach. The range of applications is manyfold; students can, e.g. meet in-world for course preparation or studying at any time. Teachers, on the other hand, can hold lectures attended by students from any geographic location. In other words, virtual worlds can be used as a distance education tool.

This thesis presents a prototype leveraging the potential of virtual worlds combined with computer simulations and video games in e-learning. An existing educational physics software implementing simulations and games is ported to a three-dimensional virtual world. The prototype combines the advantages of its components and features concepts such as collaborative learning, distance education and computer-based training. Furthermore, the resulting prototype is customizable with little, or no programming skills, depending on what to customize. This behavior allows teaching personnel to adapt the solution towards their needs.

By analyzing the theoretical background of collaborative learning, simulations and games in education, and virtual worlds together with their current application the conceptional requirements for the prototype are defined. After discussing the implementation details the usage of the prototype is described. Finally, a summary is given with possible improvements and future work suggestions.

Kurzfassung

E-learning Software kann die Lernleistung von Studenten markant erhöhen. Die Verbesserungen bei Hard- und Software des letzten Jahrzehnts erlauben es Entwicklern, erweiterbare Echtzeitsimulationen zu implementieren. Diese wiederum können verwendet werden, um den Lerneffekt zu erhöhen. Aber nicht nur E-learning Software profitiert vom Fortschritt der Technologie, sondern auch Virtuelle Welten. Vor allem das heute weitreichend verfügbare schnelle Internet ist erforderlich, um virtueller 3D Welten mit komplexer Funktionalität zu verwenden.

Noch vor nicht allzu langer Zeit hat die Forschung im Bereich E-learning das Potential virtueller Welten für sich entdeckt. In der Lehre haben diese Welten viele Vorteile gegenüber dem konventionellen Ansatz. Das Anwendungsgebiet ist vielfältig. Zum Einen können sich Studenten in den Welten unabhängig von der Tageszeit zur Kursvorbereitung treffen. Andererseits können Vortragende in der virtuellen Welt Vorlesungen halten, also sie zum Fernunterricht verwenden.

Diese Arbeit beschreibt einen entwickelten Prototypen, der das Potential von virtuellen Welten in Verbindung mit Computer-Simulationen und Videospielen auszunützen versucht. Eine bestehende Physik-Simulationssoftware, die Simulationen und Spiele implementiert, wird dazu in eine dreidimensionale virtuelle Welt portiert. Der Prototyp vereint die Vorteile seiner Einzelkomponenten. Konzepte wie gemeinschaftlichem Lernen, Fernlehre und Computer-Based Training werden dabei thematisiert. Der implementierte Prototyp ist, ohne, oder mit wenig Programmierfähigkeit anpassbar. Dies ist vor allem für Lehrpersonal wichtig um die Lösung auf ihre Bedürfnisse anpassen zu können.

Mit der Analyse der Theorie über gemeinschaftliches Lernen, Simulationen und Spiele in der Lehre und virtuelle Welten, sowie deren derzeitigen Einsatz werden in dieser Arbeit die konzeptionellen Anforderungen an den Prototypen aufgestellt. Nach der Beschreibung der Implementierungsdetails wird die Verwendung des Prototypen erklärt. Als letztes folgt die Zusammenfassung, die mögliche Verbesserungen und Vorschläge für zukünftige Er-

weiterungen enthält.

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.

April 12, 2012	
date	signature

Acknowledgments

Although the stay at the MIT was not my first longer stay in a foreign country, this experience was new with many respects. I want to thank all the people who supported me in the US, making my stay a lot more comfortable. Especially to mention, Fabio Ricardo and Josh Schuler, who pretty much organized my housing, and Meg Westlund at CECI who not only helped me out with all the organizational staff, but also with every-day life problems. Generally, I want to thank the CECI stuff for their support. I really enjoyed the friendly atmosphere at the institute.

Many thanks also to Christian Gütl who was giving me the opportunity to work at MIT in the first place. He also gave good advice to improve the contents of this thesis.

Thanks also to my family and friends for their support during my stay, and during the writing of my thesis. Various coffee breaks with them helped me to keep a clear head during the thesis writing phase. Special thanks to my girlfriend Susi who did not only visit me in the United States, but also gave me a warm welcome when I was back in Austria.

Last, but not least, I want to give special thanks to those people who took the long journey to visit me in the United States. Those are Johanna and Martin, Sabine and her sister, and Birgit and Matthias.

Thank you!

Contents

Chapter 1

Introduction

Virtual worlds are a technology which has been evolving for about twenty years. Within these worlds many people represented by their in-world avatars can work together or meet for other purposes. The technology was firstly used by multi-player games such as Doom and Quake in the 1990s. When the computational power as well as the connection speed between the peers increased more complicated and realistic applications such as Second Life (SL)[1] were feasible (Messinger et al., 2009). Due to this improvements in technology web-based e-learning systems are becoming an interesting field in computer education (Li & Zhao, 2008).

1.1 Motivation

In Scheucher, Bailey, Gütl, and Harward (2009) a prove of concept showed that MIT's internet-accessible physics experiments (iLabs) can be integrated within a virtual 3D world. In order to visualize a 3D model of the chosen experiment in-world parts of MIT's TealSim physics e-learning software where used. TealSim is a physics simulation software which aims to improve the students' performance of well-visited courses. It does so by providing a wide range of simulations the students can study prior to the classes. Teachers can adapt the software by defining simulations specifically for their courses without sophisticated software development skills. This behavior of TealSim turns out to be beneficial for this thesis' approach; to run simulations and games within a virtual world, creating an efficient collaborative learning environment. Such a collaborative online learning system can increase the learning performance of students significantly compared to conventional collaborative learning (Mukti, Razali, Ramli, Zaman, & Ahmad, 2005).

[1] `http://secondlife.com/`

For the stated reason, TealSim was used as simulation and game framework for this work. In contrast to Scheucher et al. (2009) not only a single simulation was defined and placed into a virtual world, but the whole framework was ported to run within the world. During the implementation process, TealSim itself was parly refactored and improved in terms of object-oriented design. Additionally, support for the 3D back-end of the chosen virtual world was implemented.

As target environment to run the simulations on, a virtual world with three-dimensional graphics support is required. The choice fell on Open Wonderland (OW)[2], version 0.5. It is free open-source software and is highly extendible using a module concept. It runs on a variety of platforms and already implements many useful features for teaching, such as blackboards, 3D-spatial audio, and so forth. Elements such as images and 3D models can be added to the world simply by drag-and-drop, making the world easily usable even for unexperienced users.

1.2 Thesis Structure

This work is split into two main parts, as well as a Chapter bridging between the two parts. The first part contains the theory behind the prototype including the used terms, some history as well as advantages and disadvantages of the used technologies. The second part consists of practical issues including implementation details and the usage of the software.

In Chapter 2 the theoretical foundations of collaborative learning are explained. After the description of the term "collaboration" different types of collaborative learning are described briefly. Subsequently, an overview of the history of the topic is given. Lastly, studies about the effect of collaborative learning on students are summarized.

Chapter 3 investigates computer simulations and video games in the field of education. First, different definitions and characteristics of simulations and games are given. Subsequently, differences between these two terms are pointed out. The history of video games and simulations in education was highly influenced by the history of psychology. Thus, the history Section of this Chapter contains relevant foundations of this discipline as well. Finally, the benefits and problems with games and simulations in e-learning are discussed.

In Chapter 4 virtual worlds are discussed. A definition of the term is given through distinctive characteristics. After stating the most important

[2] `http://openwonderland.org/`

facts of the virtual worlds' history the impact on education is summarized, raising the main advantages of virtual worlds in education. In order to show the possibilities of this technology in education some similar projects making use of it are described briefly.

Chapter 5 is the bridging Chapter to the practical part of this thesis. It investigates how to combine a virtual world with computer simulations and video games in order to create a collaborative environment. The concept is outlined and the reasons for using the specific frameworks are stated. Additionally, both frameworks are described with respect to their functionality and their software design.

The practical part of this thesis starts with Chapter 6. It describes how the prototype was implemented in terms of design and implementation. Changes in the TealSim software are discussed as well as possible alternative approaches. Chapter 7 describes the resulting software from the user's perspective; how to install it and how to use it. For better understanding the Chapter is supplemented with screen shots. While Chapter 8 states some lessons learned by stating general problems with the authors solutions, Chapter 9 gives a final conclusion on the project. This contains the outcome as well as future work suggestions.

Chapter 2

Collaborative Learning

In this Chapter the foundations of collaborative learning including the psychological and technical research are explained. Both, the importance of collaboration in e-learning and the problems will be discussed.

2.1 Terms and Definitions

Due to the various usages of the term *collaborative learning* Dillenbourg (1999) does not recommend any specific definition. However, he gives a broad definition stating that collaborative learning *"is a situation in which two or more people learn or attempt to learn something together"*. Similarly, Gokhale (1995) sees in collaborative learning *"an instruction method in which students work in groups toward a common academic goal"*. The latter definition focuses rather on teaching. Additionally, the term "two or more" is replaced by "a group of students" making the definition more specific towards the need of this thesis.

Thagard (1997) focuses on the term *collaboration* itself putting it into four contexts. However, only three of these contexts are relevant to learning:

- *Teacher-apprentice.* One characteristic of this type of collaboration is an asymmetry of knowledge by the participants. The apprentice who in case of teaching is the student aims to gain knowledge from the teacher.

- *Peer-similar.* Contrary to teacher-apprentice collaboration, the collaborators have a similar, but not necessarily identical knowledge of the subject. The term peer-similar refers rather to collaboration in research. People working on a similar subject are working together. However, the term can be easily adapted to collaborative learning where the peers are typically represented by students in a similar grade.

- *Peer-different.* This context occurs when researchers of different fields collaborate in order to achieve a common goal. In learning this type of collaboration happens among students of different grades or knowledge.

The differentiation of collaboration into these contexts is useful in learning. However, a distinctive categorization with respect to the contexts is not always possible.

Collaboration is often used synonymously with the term *cooperation*. There is no agreement on the difference of these two terms (Dillenbourg, Baker, Blaye, & O'Malley, 1996). However, Roschelle and Teasley (1995) define differences of collaborative and cooperative problem solving. Cooperation means splitting work in portions where each member of the group executes one portion. With collaboration the focus lies on *"the mutual engagement of participants in a coordinated effort to solve the problem together"*.

The term *computer-supported collective learning* (CSCL) can be defined as a *"joined intellectual effort by a learning community"* which *"is supported by means of computer programs and media"* (Gütl et al., 2012). In various resources the term *computer-enabled collective learning* (CECL) is found as well. It will be used synonymously with CSCL in this work.

2.2 Foundations

Early research in collaboration focused on the individuals and their functioning within a group. During the 1970s and 1980s this trend was supported by cognitivism which is individual-centered (for details on cognitivism see Section 3.2). Later the focus was put on the group itself. The studies on collaborative learning tried to find correlations of parameters such as group size, communication media or type of the task and their learning impact. Since this parameters correlate with each other the studies could not be successful in creating a link between one of the parameters and the effect of collaboration. Subsequently, the focus was put on the influence of the mentioned variables to mediating interaction (Dillenbourg et al., 1996).

The general focus of studies evolved in time from the individual to the group (see Figure 2.1). Subsequently, the three approaches described in Dillenbourg et al. (1996) are listed and explained briefly:

- The *socio-constructivist approach* suggests social interactions among the people within a group to increases the learning performance. This is due to differences in the knowledge or perspectives of the participants. To come to a common solution the learners have to argue with each other. This "socio-cognitive conflict" does not necessarily lead to

Figure 2.1: This Figure contains the three research approaches of collaboration studies stated by Dillenbourg et al. (1996). The approaches are ordered by focus which is equal to the order of historical appearance.

better performance. However, another aspect, namely the generation of communication among learners beneficial.

- The *socio-cultural approach* descends sociocultural theory which was mainly developed by Vygotsky in the 1920s and 1930s (John-Steiner & Mahn, 1996). In the socio-cultural approach the focus lies *"on the causal relationship between social interaction and individual cognitive change"*. With this individual cognitive change of the participants their own understanding is improved. As the socio-constructivist approach this approach also requires differences in knowledge of participating learners (Dillenbourg et al., 1996).

- Lastly, the *shared cognition approach* is not concerned about individual learners, but about the group as a whole. Thus, it focuses much more on the social aspects around the learners. For this approach no difference among the participants in terms of knowledge is necessary.

All of these approaches can be beneficial depending on the task and the participants. The socio-cultural approach, e.g., is predominantly used for adult-child pairs whereas the socio-cognitive requires a similar developmental level (Dillenbourg et al., 1996).

From the technological perspective impulses to establish CSCL appeared in the 1980s. Previously, learning software had focused on the individual and thus, had isolated the learner (Stahl, Koschmann, & Suthers, 2006). Later, the idea of solving problems in collaborative learning with networked computers became attractive for people working on learning applications (Hoadley, 2010). Predecessors of today's collective learning environments (CLEs) such as ENFI, CSILE, and the 5th dimension started facilitating chats and other communication methods. This way they provided technology to foster meaning-making processes by organizing social activities. In the

mid-1990s CSCL systems began to emerge following the socio-constructivist approach. The benefits of providing communication channels became more obvious; students who are brought together this way are keen to construct knowledge. Additionally, guidance to support the task is provided (Stahl et al., 2006). With enhanced network technology, most notably the world-wide web, the CSCL tools have become more and more powerful. CLEs use this technology and are nowadays a widespread technology.

2.3 Studies on the learning impact

Several studies have contributed to the knowledge about collaborative learning. Most of them see positive effects to the learning performance. Ellis, Klahr and Siegler, e.g., found that a group of students has the potential to perform better than its best individual alone (as cited in Stevens et al., 2005). However, different studies use different approaches for investigation. As described in Crook (1998) early studies typically consisted of a pre-test, the group work and a post-test. This approach, however, is criticized for some reasons. Firstly, for simplicity reasons the studies only investigated small groups. Thus, the question if collaboration does occur within small groups is often not considered at all. Secondly, the number of utterances of the group members is taken as a direct indicator for the degree of collaboration. This assumption hides the dynamics of collaboration. Lastly, the degree of considering the cognitive structures of the participants is rather low. In order to overcome this problems, Crook suggests focusing on three features. Those are intimacy, supply of external sources and previous quality of interpersonal relations of participants.

Another interesting approach is given by Nova, Wehrle, Goslin, Bourquin, and Dillenbourg (2007) who investigate so-called awareness tools in collaboration. The term "awareness" refers to information about other, distant peers. Such information includes presence, identity, location and so forth. This is very important because an individual has to have some idea of the partners viewpoint in order to collaborate well. This way the learning process itself is enhanced since *"it forces the learner to reason more deeply on the domain and to perceive the task from a second viewpoint"*. The study used a video game platform for psychological experiments called "Spaceminers". With using a game the authors see advantages in engagement of the participants. The players were put into collaborating pairs. Questionnaires during the game as well as open interviews after the game were used to gain results for three postulated hypothesis. Firstly, providing awareness tools should improve the participants performance significantly. This hypothesis

was proven to be true by an increased achieved game score. Secondly, the accuracy of the mutual modeling should be increased. Mutual modeling refers to the shared understanding achieved by sharing the game experience of both partners. Tests did *not* show any improvements with respect to this issue. Lastly, the mutual modeling accuracy should increase with time. Every pair played the game three times. Although there is a slight improvement the last assumption could not be proven to be correct. However, awareness tools seem to help the participants anticipating the partners intents.

In a similar study Gutwin and Greenberg (1999) examined the effect of awareness tools in real-time distributed groupware. There are specific problems which can be tackled by awareness tools. Those occur especially if the screen cannot show the whole workspace at once and different users see different parts of the environment. The study mainly focuses on some usability aspects. Those are:

- The *task completion time* as a performance measure.
- The *communication efficiency* evaluated by number of words participants speak.
- The *participant's perceived effort* as a subjective measure.
- The participant's *overall preference* as a more general subjective measure.
- *Strategy use* focuses on how the task was carried out.

The results of the study showed partly improvements when using awareness tools and partly not. Significant improvements are gained especially with respect to the task completion time and the communication efficiency on some tasks.

Challenging problems of today's collaborative learning include the time consuming implementation of tools as well as collaborating groups very different levels of knowledge or skills. Such participants tend to become passive when collaborating with others and thus, do not commit to the positive learning outcome of the group (Gütl, 2011).

2.4 Summary

In this Chapter different aspects of collaboration and cooperation are explained. This topic benefits from psychology research as well as from the improvement of technology in recent years. As described by Thagard (1997),

one can distinguish collaborative learning among students only and, more similar to traditional teaching, collaboration with the help of a teacher. Both types can be beneficial in education.

The improvement in technology with respect to computer connectivity and computer performance enables the utilization of CSCL, which has become a powerful tool in education. A multitude of studies lead to improvements in this field. As an example, awareness tools can easily be integrated in collaborative learning tools and enhance the learning outcomes.

A problem to be tackled in collaborative learning is the passiveness of weaker students. Enhancing engagement may be a proper way but is depending on the topic and, more influenceable, on the CSCL environment. A utility known to be highly engaging are games. Those, together with simulations are worth investigating and will be explained in the following Chapter.

Chapter 3

Simulations and Games in Education

Computer simulations and video games play an important role in education. This introductory Chapter presents basic background of the topic and describes relevant terms used throughout this thesis. Beneficial features of the use of games and simulations in education will be pointed out and discussed.

3.1 Terms and Definitions

Whereas a *simulation* is often referred to as an abstraction or simplification of real-life (Akilli, 2007), this thesis uses the term with respect to computer simulations. These can be defined as *"programs that contain a model of a system, or a process"* (de Jong & van Joolingen, 1998) or, rather process-oriented, as *"a method for studying complex systems that has had applications in almost every field of scientific study"* (Winsberg, 2010). These definitions apply well to the field of education. Since this thesis focuses on simulations in physics, the term *physics experiment* refers to simulations rather than to real-world experiments.

For the term *game* many definitions can be found. For Dempsey, Lucassen, Haynes, and Casey (1996) it is *"a set of activities"* which is *"rule-guided and artificial in some respects"* and involves competition. According to Narayanasamy, Wong, Fung, and Rai (2006) a game can also be defined as *"a goal-directed and competitive activity that involves some form of conflict, conducted within a framework of agreed rules"*. Akilli (2007) adds fun and creativity to the definitions above. It is *"a competitive activity that is creative and enjoyable in its essence, which is bounded by certain rules and requires certain skills"*. In Juul (2003) a more formal approach is used for defining

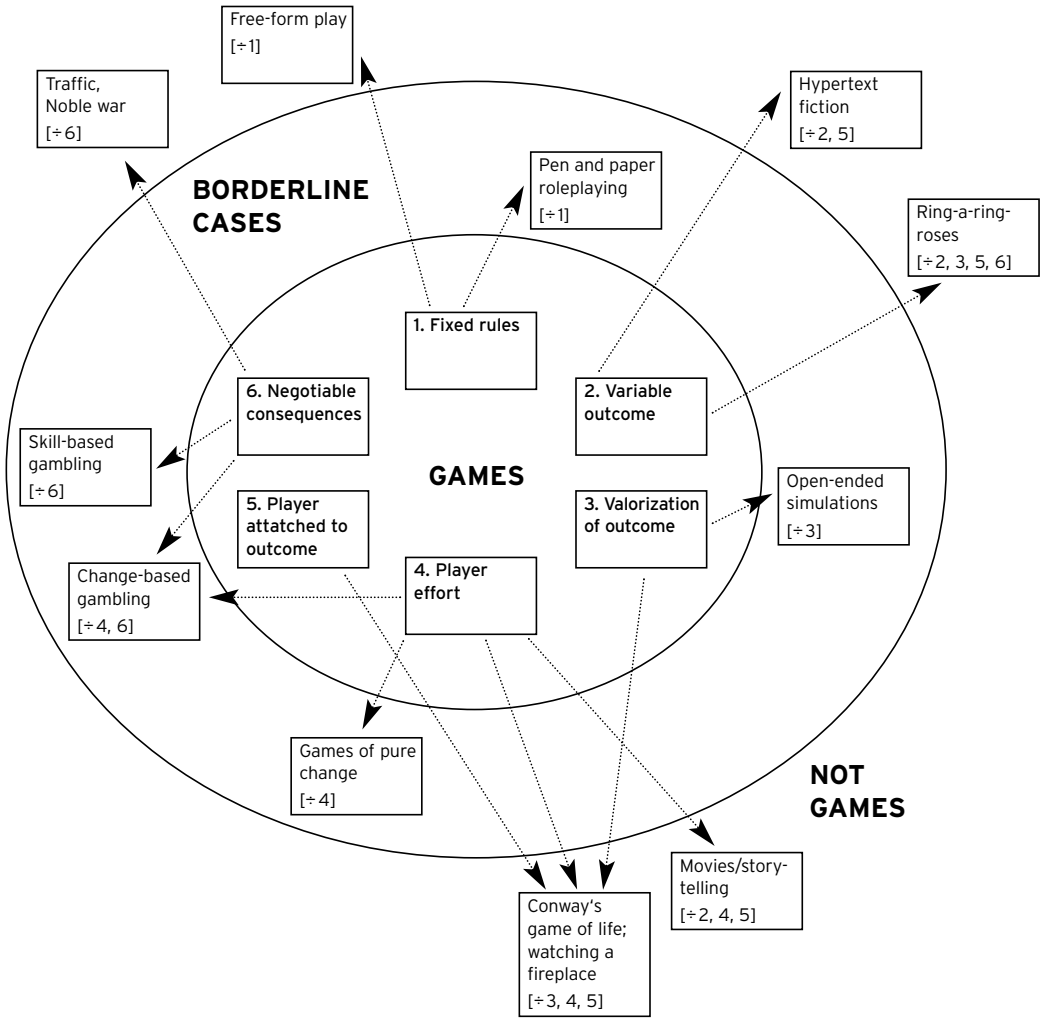

Figure 3.1: This image shows the game diagram with its six game features according to Juul (2003). Some borderline cases and some non-games are stated. Those lack in one of the features (indicated by the arrows).

games. Six features essential for games are identified (see Figure 3.1):

1. *Fixed Rules.* The rules of a game have to be unambiguous and must be clearly defined.

2. *Variable and Quantifiable outcome.* There must be different outcomes possible in a game. The goal must be clear and beyond discussion.

3. *Valorization of the outcome.* The value of an outcome for the player can be better or worse. A higher number of points is, e.g., usually

a better outcome. In general it is harder to achieve better outcomes which makes a game challenging.

4. *Player effort.* The effort of a player influences the outcome. A higher effort mostly leads to a better outcome. Similarly to the valorization of the output this feature corresponds to making games challenging.

5. *Attachment of the player to the outcome.* The players feeling corresponds to aspects of the outcome. A player may feel happy when winning or sad when losing.

6. *Negotiable consequences.* Real-life consequences can be linked to a games outcome. Non-negotiable consequences include e.g., injuries in sports. The time effort when playing a game can also be counted to non-negotiable consequences. If a game lacks all negotiable consequences this feature is not fulfilled.

The features involve the previously mentioned aspects for defining a game. Additionally separating games from non-games is possible in a formal way.

Compared to simulations, games have additional structural elements. Those are, e.g. play, a goal or competition (Prensky, 2001). This applies for physics simulations as well. By adding such elements, physics experiments become games which can have several advantages over plain simulations. Another property of a Game is the linearity of the event sequence. In most games a player solves a challenge and advances to the next exercise. This does not change when the game is played repeatedly. The order of the events remains the same, no matter *how* the player advances. With simulations however, a player's decision can have an impact on the subsequent behavior of the whole simulation. Different branches of possible outcomes can be reached in one and the same simulation (Gredler, 1996).

3.2 History

Using games or simulations in education is not a new approach; it dates back to Chinese games in 3000 B.C. (Dempsey et al., 1996). The rising popularity in the 1950s declined later when the basic-skills movement appeared (Gredler, 1996). Over the last 20 years however, the importance of simulations and games has risen significantly. The research in that field was strongly influenced by the progress in the field of psychology. Paraskeva, Mysirlaki, and Papagianni (2010) mention the most important psychologic frameworks for simulations and games in e-learning during the last decades

namely behaviorism, cognitivism and constructivism. Subsequently, these frameworks with their influences on educational games and simulations are described briefly.

Behaviorism was introduced as early as 1913 by John B. Watson in his article "Psychology as the Behaviorist Views It". It criticized the main method in psychology, namely introspection, for being too subjective and thus not scientific. Instead of being a science of consciousness psychology should be a science of behavior (Baum, 1994). The focus is put on investigating relevant stimuli and their response rather than self-observation. With learning in games behaviorism sees the player matching questions (stimuli) and answers (responses). As soon as the player finds a correct match, learning has occurred (Paraskeva et al., 2010). This involves playing sequences repeatedly until finding a solution. The fact that most players do so increases the learning effect (Egenfeldt-Nielsen, 2006).

Later increasing criticism of behaviorism lead to the cognitive counter-revolution. *Cognitivism* was established in the field of psychology (Miller, 2003). It suggests that all processes in mind, learning included, are cognitive. Problem solving is achieved by gathering facts, constructing hypothesis and finally making inferences (Dreyfus, 1991). Hence, cognitivism focuses on the process between stimulus and response. When used in education the focus lies on structuring and organizing data to make it easier for a student to gain information (Dark & Winstead, 2005). For educational games the cognitive approach suggests building intrinsic motivation. This is done by challenging the player's knowledge with game experience. Learning material is presented in different forms. This eliminates one main concern with behaviorism, namely training one single solution of a problem while not knowing any schemas behind. The need of extrinsic motivation in behaviorism games is also a critical point avoided by the cognitive approach (Egenfeldt-Nielsen, 2006).

Constructivism has its roots in cognitivism. It postulates *"that each individual mentally constructs the world of experience through cognitive processes"* (R. A. Young & Collin, 2004). Thus, the focus is put on construction rather than on discovering reality (Smith & Ragan, 1999). cognitive processes are mainly the interaction with content, other people or ideas. The constructed meaning differs with the learner because of different previous knowledge, attitude, beliefs, and so forth (van Eck, 2007). In educational simulations following a constructive approach the learner will build a simulation in order to gain knowledge. In *discovery learning*, which is constructivist based, e.g., the process of learning is achieved in four phases, namely the generation of hypothesis, the design of experiments, the interpretation of the outcoming data, and the regulation of learning (de Jong & van Joolingen,

1998). The major difference to the cognitive approach is the design of the experiments by the learner. With games, the constructive approach lets both, the player and the game construct the game experience (van Eck, 2007). This requires an educational game to ensure that the player is constructing knowledge. From the game developers perspective this is not easy to realize (Egenfeldt-Nielsen, 2006).

3.3 Benefits and Problems

Various studies have contributed to the effects of computer games and simulations in e-learning. While many studies see positive effects on the learning outcome, Russell (1999) devoted a whole book to prove the opposite. "The No Significant Difference Phenomenon" contains a collection of reports, all concluding that no technology increases the learning effect significantly (as cited in Molenda & Sullivan, 2003). Different fields in educational science provide different explanations for such results. Subsequently, some representative cases are mentioned.

In discovery learning de Jong and van Joolingen (1998) find some characteristic problems. Those can, however, be overcome by problem-specific support to the learner. Many learners have, e.g., problems with the step of finding hypothesis. This is often due to a lack of basic knowledge in the subject. To solve that issue the learner can be supplied with additional information within the simulation environment. Some authors even suggest to give a set of hypothesis to choose the right one from. Other problems are addressed by giving feedback to the learner. Overall, applications using discovery learning need to be aware of shortcomings which, however, can be avoided by adding additional help.

A similar solution is given by Yeo, Loss, Zadnik, Harrison, and Treagust (2004) for the research field of interactive multimedia, which did unexpectedly not increase the learning effect of students. Again, this problem was overcome by giving additional guidance. The effect of providing different degrees of such guidance is inspected in a study by González-Cruz, Rodríguez-Sotres, and Rodríguez-Penagos (2003). In an undergraduate course on the subject of biochemy three groups of students were using a simulation tool supplementary to their practical. The degrees of guidance in the groups were detailed, intermediate and minimal, respectively. The students wrote reports and took an exams. The results of the study were different for the reports and the exams. Student groups with higher degree of guidance performed best with the reports while a lower degree of guidance rather lead to better exam marks. The authors conclude that the effect on the exam is

due to the feedback the students received on their reports prior to the exam. The understanding on the other hand increases with the degree of guidance.

Despite the problems with simulations and games in education there are many studies showing positive effects. When examining several studies with games, Egenfeldt-Nielsen (2006) states that *"findings on [the] learning outcome are positive and promising"*. In an overview of studies within the last ten years given by Rutten, van Joolingen, and van der Veen (2012) the studies are grouped in four categories. In the first category the outcome of simulation-based enhancement of traditional instruction is investigated. Rather than comparing different simulation-supported teaching systems with each other this studies are trying to find differences in teaching when simulations are added to the teaching system. Studies investigating the impact on traditional instruction in general are investigated as well as simulations as preparation to laboratory activities. The fields are manyfold; chemistry, biology, and physics, including mechanics, electronics and optical physics. All of the studies found positive outcomes on the learning performance, although one study could not detect a positive long-term effect. When used for preparation to laboratory students with learning deficiencies can profit more than others, which can be valuable in teaching.

Beside the impact on the learning performance simulations and games also address other important improvements for learners. Stelzer, Brookes, Gladding, and Mestre (2010) investigated the introduction of a prelecture simulation in a physics course on electricity and magnetism. A high improvement of the perception of the course was achieved which lead to a positive attitude of the students to physics in general. Additionally, the difficulty of the course was rated much lower after the introduction of the simulation although the demands of the course did not change. Similarly, Durán, Gallardo, Toral, Martínez-Torres, and Barrero (2007) find a positive influence on the students' satisfaction when using Matlab/Simulink simulations for an electrical engineering course. This improvement has also positive effects to the cognitive domain. By providing intrinsic motivation to learners (Hodhod, Cairns, & Kudenko, 2011) simulations and games seem to meet the expectations stated by Prensky (2001) who sees in them *"a fabulous way to learn"*.

3.4 Summary

Simulations and video games only differ by some structural elements as well as with the linearity of the event sequence. Thus, both can technically be realized in similar ways. The Chapter has summarized the two terms, espe-

cially with respect to their history which includes the underlying psychological frameworks.

According to the studies previously mentioned video games and simulations are not only commercially successful but can also be a powerful learning tool. They not only have the ability to increase learning performance, but also seem increase the students' interest in the given subject. However, games and simulations do not automatically perform well when used in education. Guidance and help has to be provided to the student. A promising approach is to combine collaborative learning and games. This way the guidance can be given by other participants.

Technically a link between CSCL and games can be established with virtual worlds. This technology provides a collaborative environment and can also be used to play games or to run simulations. Virtual worlds can be seen as a collaboration tool advanced in technology. From this perspective they are worth further investigation.

Chapter 4

Virtual Worlds

The field of e-learning often leverages newly emerging technologies. Virtual worlds have been proven to be an applicable technology. This Chapter introduces the field of virtual worlds by giving definitions and giving some background of their history. Subsequently, the current features of virtual worlds with their implications on education are pointed out. Lastly, examples of educationally used virtual worlds are given and explained briefly.

4.1 Terms and Definitions

The two words of *virtual world* are described separately in Bartle (2004). According to this a *world* is seen as *"an environment that its inhabitants regard as being self-contained"*. The term *virtual* is somewhere between *real* and *imaginary*. It is described as *"that which isn't, having the form or effect of that which is"*. The definition of both words combined given by Bell (2008) is *"a synchronous, persistent network of people, represented as avatars, facilitated by networked computers"*. The various terms of this definition are described as follows:

- A *synchronous* communication is necessary in order to maintain the concept of "common time" among the participants. This common time is essential for social activities within the virtual world.

- The *persistence* of a virtual world lets the world keep functioning even while a participant is not in-world.

- The central term *network of people* indicates the need of participants communicating and affecting one another in a virtual world.

- The *avatar representation* of real people can be graphical or textual. It can perform actions initiated by a human agent.

- A last a very important property of virtual worlds is the *facilitation by networked computers*. There is no other way to perform the complex actions required for the data management of a virtual world.

According to this definition all the terms above must apply in oder to be a virtual world. Such terms do not need to be assembled to a definition in a formal way. More generally a set of characteristics can be given. Choi and Baek (2011) summarize the characteristics presented in previous studies. Table 4.1 states the characteristics by author. Some studies shown address

Researchers	Distinct media characteristics
Whitelock, Brna & Holland(1996)	Representational fidelity, Immediacy of control, Presence
Brna (1999)	Representational fidelity, Immediacy of control, Presence, Social fidelity (including social familiarity and social reality), Immediacy of discourse, Social presence
Book (2004)	Shared Space, graphical user interface (GUI), Immediacy, Interactivity, Persistence, Socialization/Community
Dickey (2005)	Illusion of 3-D space, Avatars, Interactive chat environment
Lehdonvirta (2006)	Numerous users, Real-time interaction, Geometric space, Avatars, Persistency

Table 4.1: Characteristics of Virtual Worlds (Choi & Baek, 2011)

virtual 3D worlds rather than virtual worlds in general. 3D functionality can be seen as an additional feature two-dimensional virtual worlds do not provide. Thus, definitions will only add 3D-specific characteristics and all non-3D terms will apply to virtual worlds in general.

4.2 History

Appearing in the 1970s, one of the first systems fulfilling the definition of a virtual world to a high extend were *multi-user dungeons (MUDs)* (Bartle, 2010; Damer, 2008). Those accept connections from multiple users and provide a “world” with “rooms” stored in a database (Curtis & Nichols, 1994).

One of the first feature added to the earliest versions of MUD was an open-ended world increasing the players "personal freedom". In the late 1980s the avatars in MUDs became capable of adding content to the world. In order to add functionality to the content scripting functionality was added in "MOO" (MUD, Object-Oriented). Other important innovations in the early 1990s' MUDs were event triggering and non-player characters (NPCs). During that time a division in two categories of the worlds occurred. One group focused on providing a game-play environment and the other one on social purposes. With the appearance of the world wide web commercial game-play MUDs became profitable. The introduction of graphics was perhaps the last important evolvement step of MUDs to current virtual worlds. The first graphical MUDs appeared in the mid-1990s and were developed by extending earlier text-based MUDs by graphic elements (Bartle, 2010). Such worlds are not any more referred to as MUDs. Whereas for the game-play worlds the term *massively multi-player online role-playing game (MMORPG)* is used, social worlds are now called *virtual worlds*.

Although many MUDs were used as *video games* and the history of both overlap video games in general can be seen as another root of virtual worlds. The history of video games starts with coin-operated arcade games in the early 1970s. Many features of current virtual worlds were added later. Whereas in early games the player had to follow a specific path the freedom for the player to move around and create elements in the world started increasing with "Populous" in 1989. The next step was to connect player and thus, creating multi-user environments. Firstly, LAN-games appeared. They started with ego-shooters in the early 1990s. The players could move their avatars in the games' world and play against each other. However, with LANs it is necessary for the players to meet physically. With the increased data rate of the Internet this was not necessary as the first games utilizing this new media appeared in the mid-1990s. In "The Sims" and its sequels the ability of creating the world was combined with Internet connectivity to create a multi-user environment. Features as choosing the avatars skin color or decorating the avatars home were available. With the emerge of MMORPG, as e.g., "World of Warcraft", millions of player started playing in virtual environments. This evolution of games set many foundations for nowadays virtual worlds (Messinger et al., 2009).

Despite having several characteristics of virtual worlds earlier systems such as PLATO are not seen as direct predecessors to virtual worlds. They did not influence the authors of later predecessors (Bartle, 2010). In contrast Messinger et al. (2009) see in social networks a root of virtual worlds. Some features influencing virtual worlds are given:

- User profiles including personal text and image data can be created.
- Friends can be added giving them additional permissions compared to normal users on the social network.
- A multitude of new media elements such as chats, friend invitations and so forth is provided.
- As with today's virtual worlds, different social networks are made for different purposes (e.g., professional introductions).

An overview of the history of social networks is given by Boyd and Ellison (2007). According to them the first social network site was SixDegrees.com launched in 1997. It implemented features such as creating profiles, listing friends and surfing friends lists. However, it is considered to have been ahead of its time and the site was closed in 2000. Additional functionality was added by subsequent sites such as LiveJournal and LunarStorm. This functionality included guest books, diary pages and friend recommendations. With the business network Ryze.com a new wave of social networks started in 2001. Due to the high number of new launches the term "Yet another social networking service" appeared. Many of these services focused on specific types of media (e.g., YouTube on video broadcasting) or types of user interests. The latter specialization is also found in virtual worlds.

4.3 Virtual Worlds in Education

In this Section the influence of today's virtual 3D worlds to education is examined. Unique properties of such world lead to specific benefits which are later described.

The effects of 3D virtual learning environments to students is investigated by Dalgarno and Lee (2010). Their model of learning in such environments is shown in Figure 4.1. Unique properties distinguishing 3D virtual environments from other learning resources are summarized into two groups, namely *representational fidelity* and *learner interaction*. These properties lead to a *construction of identity* by the player and a *sense of presence* within the environment. Another effect supporting the learner is the interaction with other people (*co-presence*). Through learning tasks which make use of the advantages of the environments several learning benefits are identified. Subsequently, those benefits are mentioned and explained shortly:

- *Spatial knowledge representation.* This point refers to the quality of the representation of an object within the virtual world. Virtual 3D

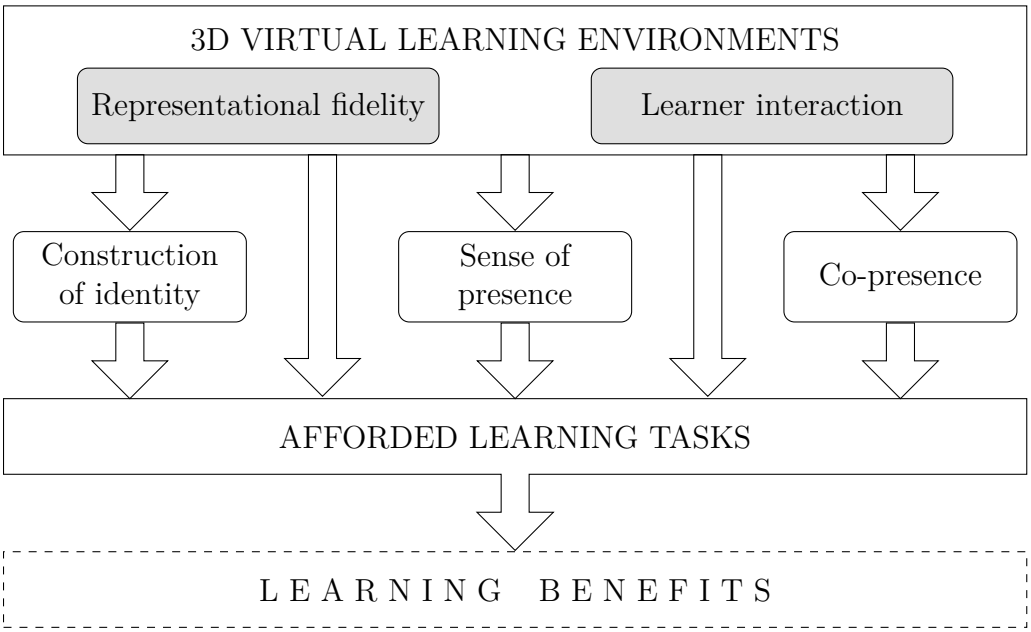

Figure 4.1: Learning process in 3D virtual worlds (Dalgarno & Lee, 2010).

models, e.g., are closer to real-world objects than 2D models. This can be used in many fields of science education (Dalgarno & Lee, 2010).

- *Experiential learning.* This term does *not* refer to the experiential learning theory, but to experiential learning tasks impossible to accomplish in the real world. Studies exploring this property can, e.g., deal with nuclear power plants or astronaut training (Dalgarno & Lee, 2010).

- *Engagement.* Engaged learning means *"that all student activities involve active cognitive processes"* and the *"students are intrinsically motivated to learn due to the meaningful nature of the learning environment and activities"* (Kearsley & Shneiderman, 1999). Engagement can be enhanced by allowing users to create content, solve problems, make decisions and reflect those (Dickey, 2005). This features are all available in virtual worlds. Engagement itself leads to a positive attitude towards the subject in the learner (Iqbal, Kankaanranta, & Neittaanmäki, 2010).

- *Contextual learning.* Since virtual worlds with their avatars can model real-world situations almost one-by-one a context is easier recognizable in-world. Divers, e.g., can recall more of their skills when they are underwater. A situation which can be built in a 3D virtual world

realistically. Thus, the transfer of knowledge and skills is increased (Dalgarno & Lee, 2010).

- *Collaborative learning.* The benefits of collaborative learning in education as well as in general are described in Chapter 2.

4.4 Examples of virtual worlds in teaching

4.4.1 The River City project[1]

River city is a multi-user virtual environment (MUVE) for learning purposes developed at Harvard university. The main idea is to put the learner from the 21st century into a city from the late 1800s called "River City". The participant is thus equipped with 21st century's knowledge which is to be used in the earlier time period. The purpose of the river city project is to teach scientific inquiry. Topics are mostly regarding health and biology. The project is especially designed to improve the performance of students who have problems with learning in middle school. Such students are often disengaged and can hardly be motivated. MUVEs are similar to the entertainment and communication media the students use in their free time. With the river city project it is investigated if this fact helps in teaching students with under-average learning performance. (Dieterle & Clarke, 2007; Ketelhut, Dede, Clarke, Nelson, & Bowman, 2007)

In Figure 4.2 the user interface of river city is shown. Participants can communicate with each other through the chat window at the bottom. On the right side a workspace window including help functionality is provided. The contents of this window changes depending on the situation of the avatar in the virtual world. The world itself is inhabited not only by the users' avatars but also contains computer-based agents as well as digital media objects. (Ketelhut, B. C. Nelson, Clarke, & Dede, 2010)

A beneficial way to accomplish inquiry learning within the world is described in Ketelhut, B. C. Nelson, et al. (2010). The students are put into small groups where investigate water-borne, airborne and insect-borne illnesses in town. After collecting data the groups postulate their hypothesis. Those are tested and a lab report has to be written. Finally, the groups compare their results among each other discussing causal relationships within the environment.

The river city project was shown to have a positive influence to the students engagement in inquiry process. An indicator for this is the use of differ-

[1] `http://muve.gse.harvard.edu/rivercityproject`

Figure 4.2: The River City user interface with its fields (Source: "River City Interface," 2007)

ent information sources used by the students as well as a continually increasing *"commitment to the activities of inquiry throughout the data-gathering period"*(Ketelhut, B. C. Nelson, et al., 2010). Additionally, problems with low attendance of students to a course and disruptive behavior can decrease (B. Nelson, Ketelhut, Clarke, Bowman, & Dede, 2005). In the subject of biology a significant increase of knowledge is achieved as well (Ketelhut, B. C. Nelson, et al., 2010). Overall, this MUVE shows many advantages of such worlds for education.

4.4.2 Quest Atlantis

Quest Atlantis $(QA)^2$ is a 3D MUVE for children focusing on educational activities. The players are confronted with a story-based context, namely *"the social, cultural, and environmental decay of the mythical world of Atlantis"*. The players goal is to save Atlantis. To achieve that the players can navigate their avatars through different worlds representing Atlantis in terms of unity,

[2] `http://questatlantis.org/`

culture, health and ecology, respectively. Different tasks have to be fulfilled to achieve the games goal, e.g. investigating facts about an animal in order to save the environment. (M. Young, Schrader, & Zheng, 2006)

As with many other MUVEs, the user interface of QA consists of several windows (see Figure 4.3). Among other features information about other

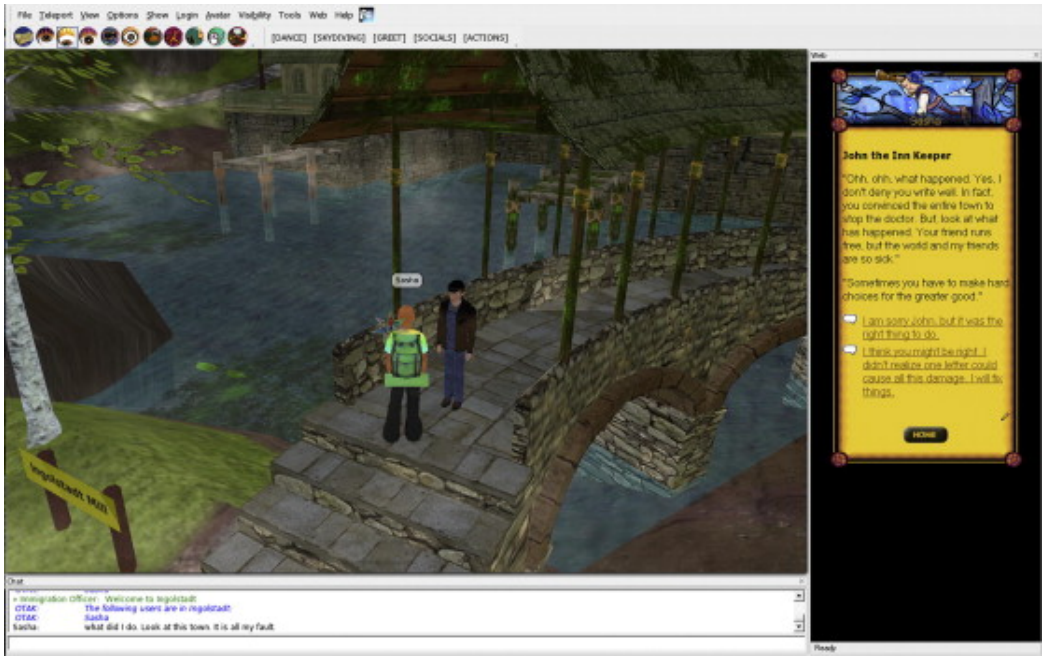

Figure 4.3: The QA user interface with the 3D view of the world on the left, the chat window at the bottom, and the conversation window with the game character on the right. (Source: Barab, Pettyjohn, Gresalfi, Volk, & Solomou, 2012)

players within the same world can be viewed. The user interface can be customized by the user to a high extend.

In Barab, M. Thomas, Dodge, Carteaux, and Tuzun (2005) the three basic design foundations of QA are described. The world itself is seen *"at the intersection of education, entertainment, and social commitment"*. The design has to take these three aspects into account. The education aspect is followed by involving the students in domain-related activities rather than only using other results for summary. The entertainment aspect refers mainly to engagement of the participants which can be easily achieved in MUVEs. Lastly, the social commitment ensures that all users are able to direct their own activity.

The used principles within QA lead to positive effects shown in many studies (e.g., Barab, Pettyjohn, Gresalfi, Volk, & Solomou, 2012; M. K.

Thomas, Barab, & Tuzun, 2009). Not only learning performance can be increased, but also the enjoyment of the students. Problems that can occur with QA are security issues and lack of implementation support (technical and social).

4.4.3 Meta-Institute for Computational Astrophysics

Meta-Institute for Computational Astrophysics (MICA)[3] is a *"scientific organization based entirely in virtual worlds"*. It is not restricted to a specific virtual world and has used Quaq[4] and SL. In this virtual worlds MICA holds frequent seminars and lectures (see Figure 4.4). The project intends mainly

Figure 4.4: Screenshot of a weekly held MICA astrophysics seminar. (Source: Djorgovski et al., 2010)

to demonstrate the advantages of virtual worlds in scientific research and to develop new tools to be used for such research. Although MICA focuses on astrophysics the technology can be adapted for use in other fields (Djorgovski et al., 2010).

As stated in Hut (2007) experiments concerning astrophysics can often not be done in a laboratory. This is due to the spatial dimensions as well as the amounts of energy this science has to deal with. Figure 4.5 shows a simulation with some objects which are attracted to each other by gravity. In Farr, Hut, Ames, and Johnson (2009) this simulation is explained in more

[3] `http://www.mica-vw.org`

[4] Now OpenQuaq. Site: `http://code.google.com/p/openqwaq/`

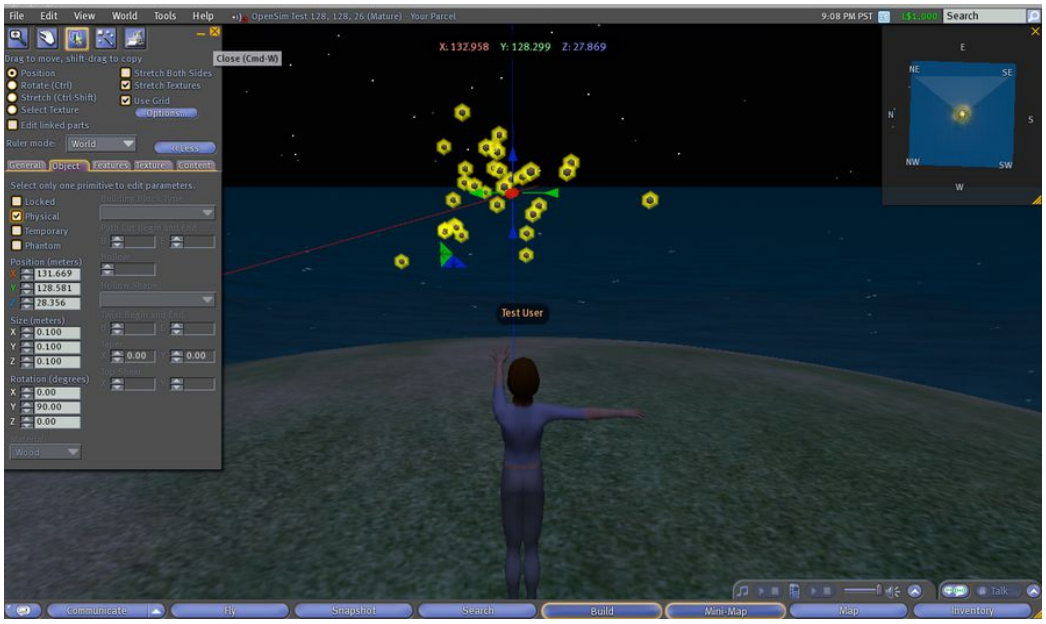

Figure 4.5: Screenshot of a gravity simulation. (Source: Farr, Hut, Ames, & Johnson, 2009)

detail. The simulation itself is implemented in OpenSim[5], a virtual world which implements the SL protocol and thus, can be used with a SL client. The world comes with a physics engine which was adapted in order to fit the requirements of the gravity simulation. Users can collaboratively place point mass objects in the virtual world and then run the simulation. After watching the simulation the users can discuss the outcomes with each other.

Problems with the simulation were detected with respect to scalability. With many objects the simulation cannot be run in real-time. Although experiments often exceed the feasible number of objects Farr et al. (2009) see a powerful tool in this application in a virtual world.

4.5 Summary

Virtual worlds are closely linked to collaborative learning as well as to simulations and video games. They can be used as a tool for collaborative learning. This Chapter has focused on the technology virtual worlds use as well as with their unique properties. The latter let virtual worlds provide functionality which can foster the learning process. Virtual worlds come with all concep-

[5] `http://opensimulator.org/`

tual advantages of earlier collaboration tools and use technology such as high fidelity audio and 3D graphics to become more realistic.

Despite the promising features a lot of research on virtual worlds is yet to be done. Persistent worlds may lead students into an immersive environment enabling them to perform tasks otherwise only possible in real world. However, technologies such as virtual worlds and educational video games are not beneficial for themselves; they only provide features which have to be used properly. The next Chapter will introduce the idea of this thesis based on the mentioned technologies.

Chapter 5

Conceptual Model

Based on the three previous introductory Chapters this Chapter introduces the idea of this thesis–an approach to enhance teaching in the subject of physics using the previously described technologies. To implement the approach two frameworks were used. These are described later in this Chapter.

5.1 Combination of a Simulation Framework and a Virtual 3D World

The combination of a virtual world as collaborative learning environment, computer simulations and video games aims at improving the students' learning performance as well as their satisfaction with learning physics. All of the stated technologies have the ability to satisfy both aims.

As stated in Section 2.3 the benefits of collaborative learning environments on the students' learning performance are backed by several studies. Further improvement can be achieved by awareness tools giving the user information of other users. As described by Nova et al. (2007), Gutwin and Greenberg (1999), they enhance task completion time and communication efficiency which leads to a higher learning performance. Such tools are often already included within virtual worlds.

Computer simulations and video games, on the other hand, can provide intrinsic motivation to enhance the learning performance. As shown in several studies (e.g., Durán et al., 2007; Stelzer et al., 2010) the attitude of the students towards physics can be enhanced. Problems with games and simulations in e-learning can be overcome by providing problem-specific support to the learner. Virtual worlds have the capability to show materials. Additionally, multiple users including teachers can provide feedback and guidance to the learner without being physically at the same place. As explained in

Section 3.3 feedback and guidance are required to achieve good learning performance.

Two of the benefits of virtual 3D worlds mentioned by Dalgarno and Lee (2010) (see Section 4.3) also play an important role, especially in physics. The first is the spatial knowledge representation in 3D, which can be necessary to understand the topic properly. Secondly, elements invisible in real-world allow enhanced experiential learning. With electromagnetism, e.g., otherwise tiny simulation elements can be scaled to any size. Additionally, concepts such as field lines, invisible in real-world experiments, can be shown.

The thoughts above lead to an approach of merging a virtual world with a physics simulation software. The approach is shown in Figure 5.1. A virtual

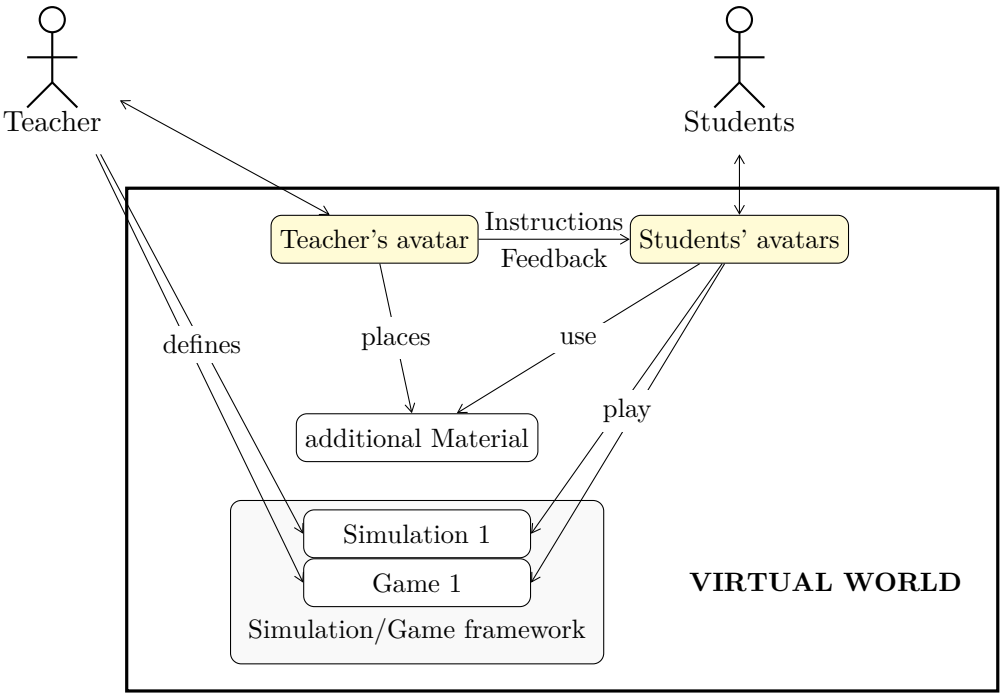

Figure 5.1: Concept for using a simulation framework inside a virtual world. Simulations and games defined by the teacher are placed within the virtual world together with additional material providing problem-specific support to the students.

world hosts physics simulations and simulation games. Additional material can be added to the world in order to guide the students and give them information of the simulations and games. The virtual world also provides communication channels such as audio and text chats. This allows the students to collaborate among each other through their avatars. Additionally,

a teacher can provide feedback, give instructions, or discuss outcomes of a simulation with the students.

With this approach a teacher is free to customize the learning environment within the virtual world. Problem-specific data in form of materials can be chosen by the teacher to be placed near the experiments. Also, tools such as a blackboard can be added. Some virtual worlds already provide such functionality. Thus, no additional work needs to be dedicated to introduce it. However, defining simulations and games needs to be reasonably simple to be made by teachers of physics. Particularly, no advanced programming skills can be assumed. Thus, the framework to be used needs to provide a simple interface for defining games and simulations.

A critical point with teaching software is usability (Squires & Preece, 1999). The software should be practical to use for both, teachers and students. The physics simulation framework used for this work is highly customizable by teachers; simulations may be written with little programming skills. Thus, a teacher in physics can adapt a simulation without the help of a software developer.

The frameworks used for the implementation are described subsequently. They are both freely available and open source.

5.2 TealSim

TealSim is a part of MIT's Technology-Enabled Active Learning (TEAL)[1] project. This project was launched in 1994 (Belcher, 2001) and defines a learning structure for courses with larger numbers of students. The project aims to improve the students' understanding by using simulation and visualization software (Dori & Belcher, 2003). TealSim is such a simulation software for the field of physics in higher education capable of running various simulations or games. It provides a simple interface for defining new simulations. This allows people with little experience in software development to implement simulations with a variety of features, e.g. the use of external 3D models. Thus, teaching personnel can construct simulations for their courses by themselves with little effort. On the other hand TealSim is extendible for programmers whenever new functionality of physical objects are needed. In Figure 5.2 the user interface is shown. A user can choose a simulation from the menu bar and change parameters on the right-hand side. The start of the simulation is triggered by pressing the "play"-button at the bottom. Simulation parameters can be changed while the simulation is running. The window on the left-hand side shows the 3D representation

[1] `http://web.mit.edu/8.02t/www/802TEAL3D/`

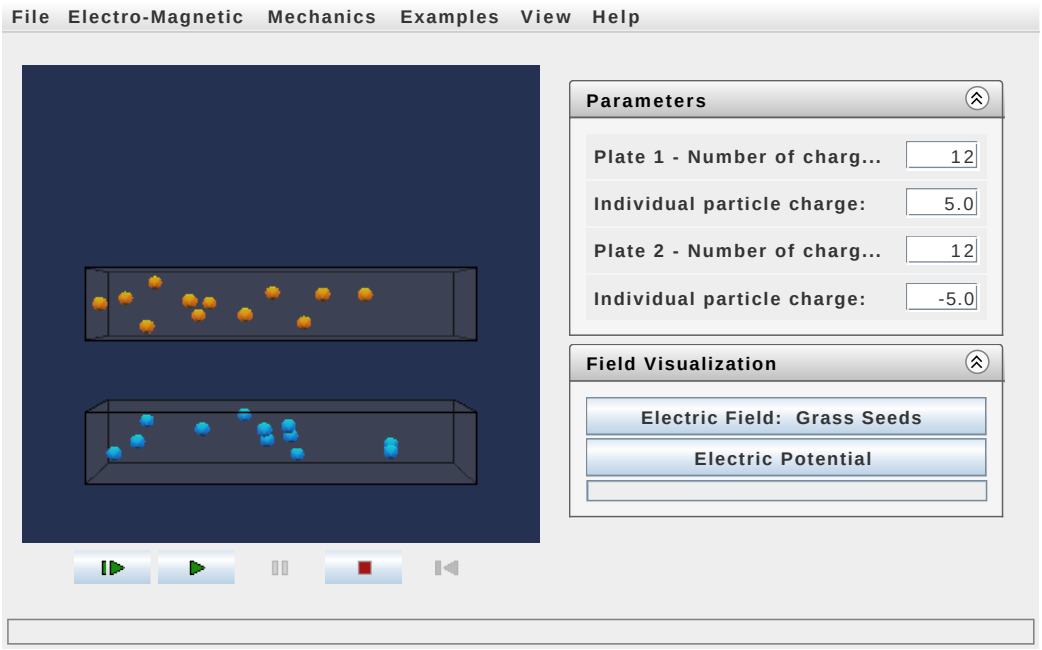

Figure 5.2: The GUI of TealSim with the "Capacitor" simulation.

of the simulation objects. With most simulations the user can change the point of view by dragging the 3D model. Such mouse interaction behaviors can be defined when defining a simulation. Additionally, TealSim provides valuable features to analyse the ongoing simulation, e.g. showing electric or magnetic fields. Some of those features as e.g. field lines are calculated in real-time while the simulation is running. When such elements are shown during the whole simulation a better understanding of the physics behind the simulation can be achieved.

5.2.1 Software Architecture

The implementation of TealSim follows the Model-View-Controller (MVC) design pattern. With this pattern the software is logically split into three components (Burbeck, 1992). Those are:

- The *model* representing the visual elements as descriptive data. It also defines the behavior of the elements, i.e., the proper reactions to inputs.
- The *view* is responsible for rendering the elements and is usually directly part of the user interface.
- The *controller* is the glue between the components. The user input goes

through the controller which then reacts properly by updating model and view.

As shown in Figure 5.3 TealSim consists of several components. The *simulation* makes the model within the MVC pattern. Whenever a concrete simulation is implemented this element is the only object to be defined. It

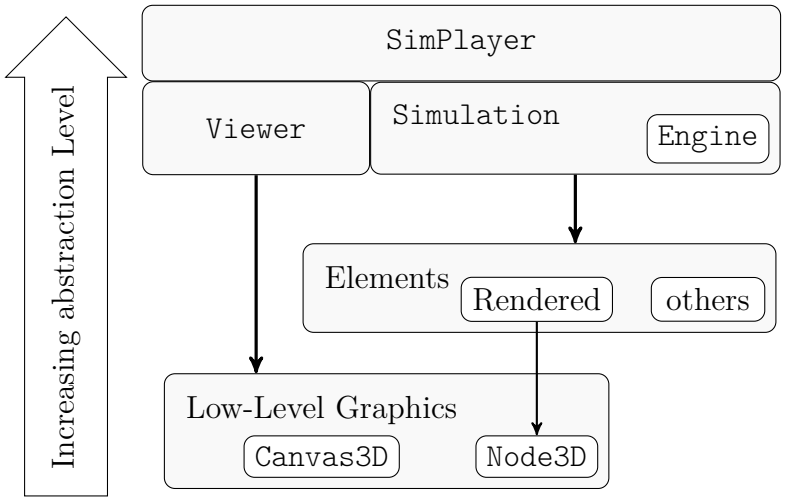

Figure 5.3: This Figure shows the main components of the TealSim software. At the highest abstraction level the three elements of the MVC pattern are shown. In a simulation various simulation elements can be defined. At the lowest abstraction level the 3D graphic elements represent the scene-graph as well as the canvas needed for rendering.

creates all the simulation elements; e.g. the 3D elements, the user interface and the type of the simulation engine needed. This is the part of TealSim which has to be easy to extend since simulations should be definable by teachers or lecturers. In order to achieve simplicity a simple interface is defined and a lot of code is already provided for standard cases.

The *simulation engine* is responsible for all the physics calculations. In order to compute a simulation step it has to know about all the objects in the simulation in order to retrieve their physical parameters. Within the engine all the low-level physics are implemented. Currently, the basic simulation engine and an electromagnetic simulation engine is provided. However, new engines with additional capabilities can easily be added by TealSim developers. This can be necessary when new simulations with new physical behaviors are used.

A simulation is loaded by the *SimPlayer* representing the controller in the MVC model. It creates all the components including the simulation engine and the simulation itself and also represents the user interface. GUI elements defined in a simulation are put to the right place of the actual user interface.

The last major element is the *viewer*. It displays the 3D simulation elements on a canvas using the Java3D[2] rendering library. During the implementation of this work the support of an additional rendering library was added (see Section 6.2). Another responsibility of the viewer is to report user interactions on the 3D elements of the currently loaded simulation.

5.3 Open Wonderland

OW is an open source virtual 3D world entirely written in Java programming language (Vani & Mohan, 2010). It is runnable on every computer running with either Windows, Linux, Mac OSX or Solaris operating system. This is a major advantage over proprietary virtual worlds such as SL. OW provides a full immersive environment with many features, e.g. high fidelity audio. In order to describe the key features the role of different types of users can be pointed out:

- *Content developers* are the constructors of the virtual world. They can add static 3D models by simply dragging and dropping files into the world (Slott, 2010a). By using a *cell*-concept static models are handled the same way as implemented content (e.g. animations) which makes it easier to place elements in world.

- Ordinary *users* log on to the virtual world by starting a Java Web Start application with a click on the server's web page. The virtual world can be explored with a highly customizable avatar giving each user a unique appearance. Users can communicated with each other by immersive audio, but also by interacting with elements in the world (e.g. blackboards) or a provided chat window.

- A OW server is usually configured by a *server administrator*. The configuration is accessible by a web interface covering all administrative functionality. This functionality includes enabling GUI-Applications installed on the server in-world, saving and restoring snapshots of the world, enabling security restrictions and uploading modules. Such modules are extensions to the virtual world. Many of them can be found

[2] `http://j3d.org/`

in the module warehouse[3].

- The last important group of users are *Software developers*. Those can add new functionality by implementing modules. OW provides an API which makes it fairly easy to develop a module. These extensions can extend nearly every implemented feature making the application very flexible. Most modules, however, add at least some visible elements together with interaction functionality to the virtual world.

The client-server architecture of OW allows the user to run the application without an explicit installation. With the Java Web Start technology updates are made automatically and resources can be loaded dynamically while the application is running.

5.3.1 Software Architecture

OW consists of a set of base components. In Figure 5.4 an overview of this components is given. On the server side a Glassfish application server[4] is

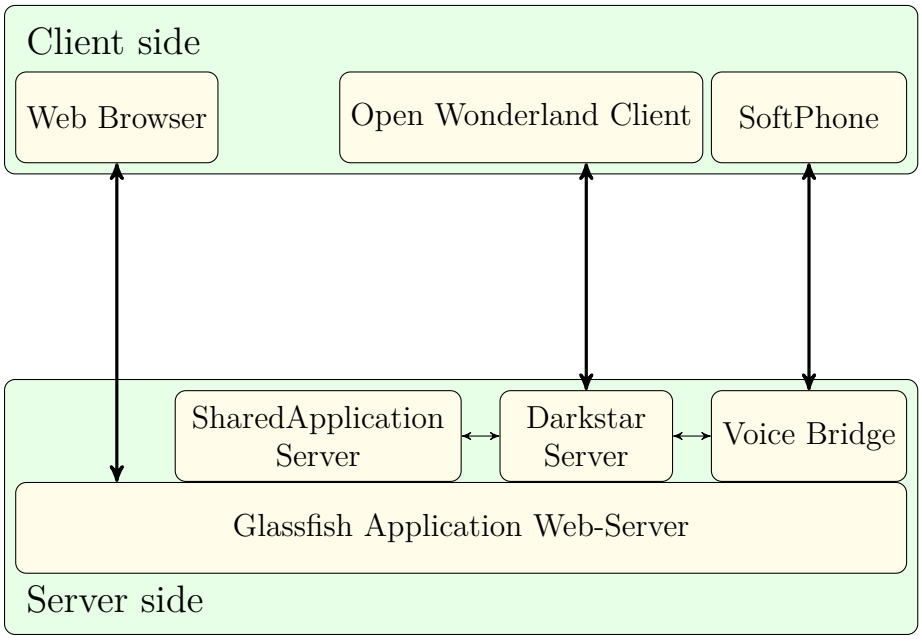

Figure 5.4: Client and server components of OW

responsible for managing the other three server-side services. The client can

[3] http://openwonderland.org/module-warehouse/module-warehouse
[4] http://glassfish.java.net/

start services, stop services, or change parameters using the web interface provided by Glassfish. The shared application server lets OW show applications started on the server within the virtual 3D world. Such applications can then be rendered within the virtual world on a two-dimensional pane. The voice bridge is a jVoiceBridge[5] audio mixer which supports high-fidelity stereo sound. It is used for sharing the voice among the in-world avatars. It also supports connections to remote soft phones.

The Project Darkstar (PD)[6] server is the middleware OW runs on. This high-performance game server with high scalability provides a communication interface to exchange messages to the clients as well as an API for the application running on a server. The server-side components of OW make use of the shared application server and the voice bridge. The OW client is a Java application establishing a connection to the server during a log-in process. In oder to realize the 3D graphics the JMonkeyEngine (JME)[7] rendering library is used. Unfortunately it lacks multi-threading support. For that reason MTGame runs on top of it in order to add the required functionality. Compared to earlier versions of OW which were using Java3D the combination of JME with MTGame provides a significant increase of scalability and performance.

One of the most powerful features of OW is its extendibility. To extend the functionality so-called "modules" are used. Those can easily be added or removed through the Glassfish web interface by server administrators. Because modules are a quite powerful in OW many built-in features are implemented as modules, hence do not need to be added to the core of the software (Slott, 2010b). In order to be noticed by OW a module has to fulfill requirements, such as implementing OW interfaces.

5.4 Summary

This Chapter intends to describe the concept of this work. The goal is to make TealSim runnable within OW. If this is achieved, advantages for teaching of both, virtual 3D worlds and simulations can be combined. Additionally, the users can benefit of unexpected advantages. Since TealSim allows teachers to define games, a multi-player game can be implemented and played by many participants at the same time in-world.

Although Scheucher et al. (2009) showed a prove of concept with an older version of OW and TealSim, this previous work lacks in scalability and

[5] `http://java.net/projects/jvoicebridge/`

[6] Now continued as RedDwarf Server (`http://www.reddwarfserver.org/`)

[7] `http://www.jmonkeyengine.com/`

uses only one single simulation. In this thesis porting the whole TealSim framework is attempted. The subsequent Chapters will describe the implementation and the results of this attempt.

Chapter 6

Implementation of the components

This Chapter describes the implementation details done for this thesis. In order to understand this Chapter properly some software programming knowledge is required. Some basics in Java programming language and in software architecture are recommended.

6.1 Requirements on an abstract level

In order to have as many simulation running within OW the TealSim framework will need to be ported to run within the virtual world. When this goal is achieved the resulting software can be used for a variety of fields within physics. From the perspective of software development the requirements can be defined as follows:

- As many TealSim simulations as possible should be runnable in OW with little special for single simulations. Most of the features of the simulations should be supported.

- Defining a simulation, e.g. by a teacher of a physics course, in TealSim should not become more complicated with the adapted code. Furthermore, the changes to this simulation-defining interface should be kept minimal since teachers are already used to the old interface.

- Neither the software design nor its performance should be influenced negatively. Possible improvements to this measures should be implemented.

- The software must be easily to install on every OW server. Thus the software should be packed into a OW module which can be uploaded by the web interface as shown in Chapter 7.2.
- TealSim should still be runnable as desktop version after the changes.
- The advantage of having TealSim run in a virtual world should be pointed out by defining a new simulation.

These requirements lead to several implementation steps applied to TealSim and OW:

1. Since the 3D output of TealSim should later be rendered by the virtual world, OW's graphics back-end has to be added to TealSim.
2. In OW TealSim will have to run in a client-server mode. Thus, the simulation functionality has to be split into according two parts. Since TealSim should still work as desktop version as well, this implementation step will lead to a TealSim version where components can run in either client-server mode or single-user desktop mode.
3. A OW module has to be created instantiating the needed components in client-server mode from TealSim. All OW-specific code-parts will have to be added to the module.

In all of these steps the previously mentioned requirements are to be kept in mind. The subsequent Sections contain explanations of the steps in more detail.

6.2 Additional 3D support for TealSim

As described in Section 5.2.1, TealSim uses Java3D for the three dimensional graphics output. In contrast, OW uses MTGame on top of the JME library. The latter was developed for OW but can also be used by other software. The concepts in Java3D are quite different from the ones in a JME/MTGame application (see Table 6.1). The scene-graph is built up with different objects. Fortunately, TealSim uses abstract elements to represent scene-graph elements. Thus, low-level back-ends for these abstract elements can be implemented for different graphic engines, including the JME/MTGame system.

In order to do so TealSim has to be prepared for the additional graphics engine. This includes mainly (a) providing the framework with the needed libraries including all dependencies and (b) implementing a factory in order to

	Java3D	JME
OpenGL back-end	built-in	JOGL or LWJGL
3D-model import	implementable	built-in
Scene-graph	deep	flat
SG object-geometry relation	has-a	is-a

Table 6.1: Differences of Java3D vs. JME

be able to select the desired graphic output (Java3D or JME). Additionally, the simulation definitions of TealSim need to be refactored wherever Java3D code has been used directly instead of the element abstractions.

For point (a) the source of MTGame needs to be downloaded via subversion[1] from the MTGame repository[2]. Before compilation all dependency libraries including native-code assemblies. The needed libraries are:

- A slightly adapted version of JME,
- the physics library JBullet[3],
- the JOGL[4] OpenGL front-end for Java,
- and the Java real-time library javolution[5].

As stated in Table 6.1 LWJGL can also be used as OpenGL back-end instead of JOGL. However, for this work, JOGL was used.

In order to use both, the Java3D and the JME library, and to be able to select between those, a factory method design pattern according to Gamma, Helm, Johnson, and Vlissides (1994) is applied. Through the factory, shown in Figure 6.1, the needed library-specific objects are instantiated. This approach is easily extendible to more additional back-end libraries. The graphics output is initially set to a default value (e.g. Java3D output), but the factory provides functionality to set the needed value. This will be needed particularly when TealSim is used in OW since JME output is mandatory here.

6.2.1 Implementation of JME Primitives

As subsequent step of the implementation of a scene factory the JME primitives have to be implemented as well. Since those primitives must be usable

[1] `http://subversion.tigris.org/`
[2] `http://openwonderland-mtgame.googlecode.com/svn/trunk`
[3] `http://jbullet.advel.cz/`
[4] `http://kenai.com/projects/jogl/`
[5] `http://javolution.org/`

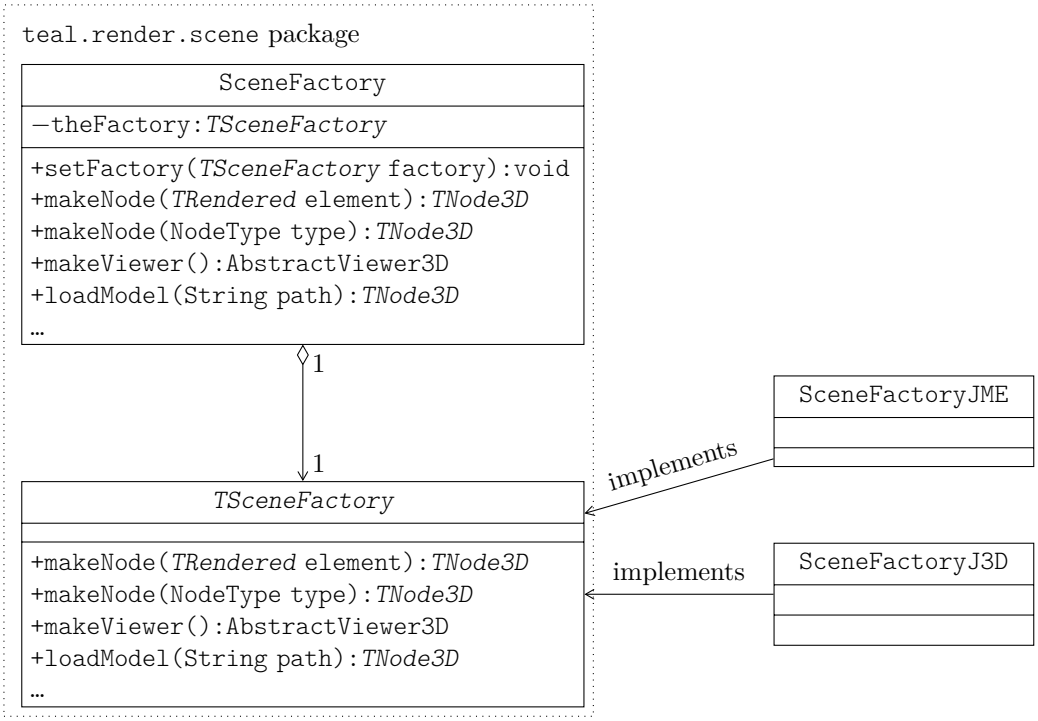

Figure 6.1: A simplified class diagram for the factory responsible for instantiating either Java3D or JME objects. The SceneFactory class holds an instance of one of the concrete factories (to the right) which is actually constructing the needed element.

in the same way as the Java3D version they have to implement the same interface. Because of differences between Java3D and JME the implementations of those two libraries also differ.

Figure 6.2 shows the Java3D scene-graph as it is implemented in TealSim. All drawable simulation elements in TealSim have to build such a scene-graph when they are to be drawn for the first time. The inner nodes of the graph contain mainly transforms, i.e. scale, rotation, and translation of the branch. The leaf node represents the object to be drawn. TealSim does not use the node classes of the Java3D library directly, but wraps its own classes around those. A base node called `Node3D` holds the Java3D-native inner nodes for applying two transforms and to change the node's visibility. Every subclass of `Node3D` adds the needed leaf nodes to the scene-graph.

In Figure 6.3 a class diagram of the JME scene-graph is shown together with the added Java interfaces. These interfaces have to be used by TealSim's simulation elements rather then the actual JME or Java3D node classes. Ideally, only one single interface is defined for all nodes in TealSim. This inter-

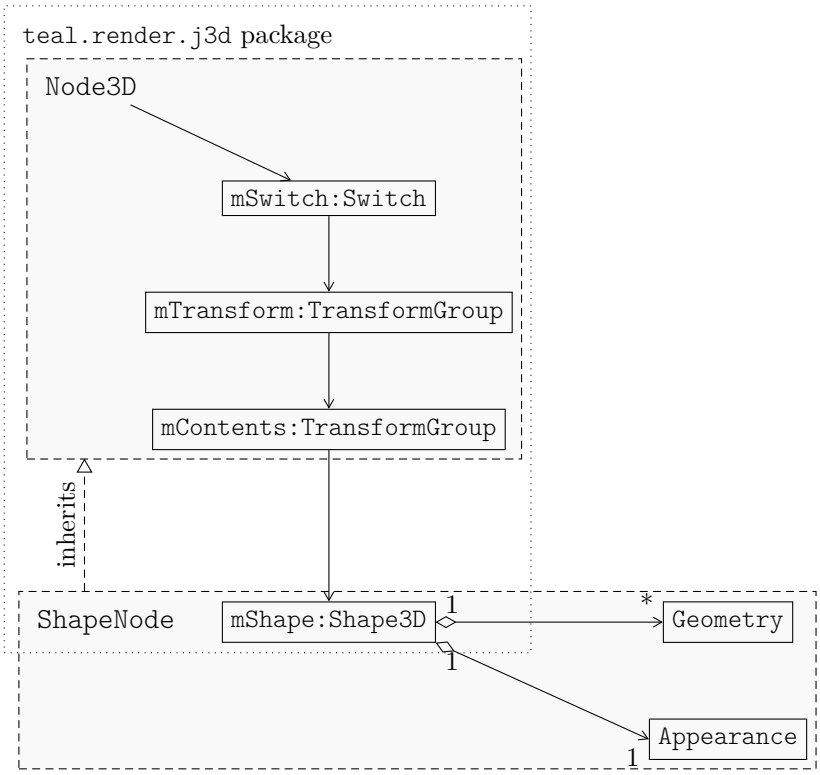

Figure 6.2: The Java3D low-level scene-graph with the surrounding TealSim object (dashed). The switch node is purely responsible for the visibility.

face will declare methods for setting and getting transforms, colors and so forth. The previously introduced factory could then return an object implementing such an interface for all possible nodes. However, some nodes such as field line nodes require special treatment and therefore have to implement extended interfaces. Therefore a hierarchy of interfaces has to be defined and applied to both, the JME and Java3D node classes.

The implementation of the nodes must result in the same appearance as the Java3D implementation on TealSim's 3D-window. As in Java3D the `Node3D` class is derived from the inner node class of the library. On order to be conform with all simulations the class hierarchy is kept the same as in Java3D. Singe JME scene-graph nodes are more powerful than the ones of Java3D; inner nodes are e.g. capable of setting their own visibility reducing the depth of the graph by one element. Additionally, transforms can not only be applied to inner nodes but also to leaf nodes, reducing the depth further. This leads to a JME implementation of `Node3D` which contrary to the Java3D

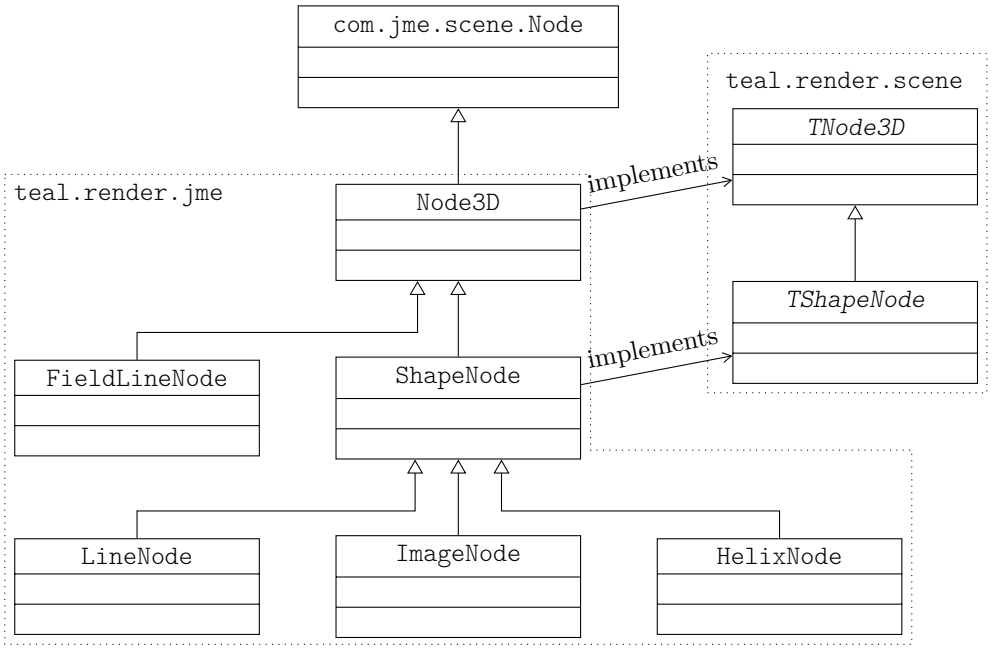

Figure 6.3: Node class hierarchy in JME part (incomplete) including the general interfaces (top right).

implementation only consists of a single JME scene-graph object. Thus, there is no need for keeping references to multiple JME objects. The interface of a TealSim node is implemented by forwarding calls to the JME node class. Visibility and transform are applied directly to `Node3D`. Subclasses then add geometry as needed for the specific object. Since JME comes with many predefined shapes, such as a sphere, a box and so forth, only few objects need to be constructed vertex by vertex manually. In contrast to Java3D where geometries are referenced by the scene-graph leafs, the JME scene-graph leafs *are* geometries. This makes sharing one geometry instance among more than one leaf node impossible.

Although most spatial elements provided in TealSim are fairly easy to port to JME, some need special treatment.

Field line nodes

The `FieldLineNode` class is the 3D representation of a field line. In TealSim field lines are symmetric around an axis (mostly the y axis). Thus, field lines can be defined by a single line geometry and a number of clones around the symmetry axis. The clones are distributed evenly within the 360 degree field around the axis. Figure 6.4 shows a simulation with 3 field lines. Each of

them has 25 clones.

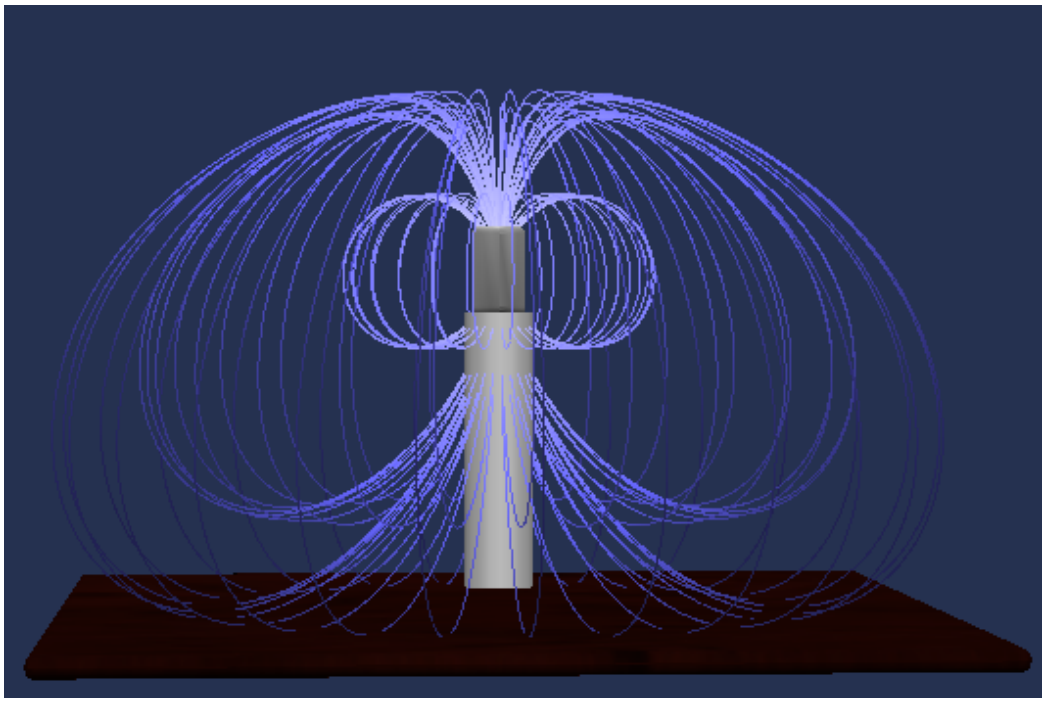

Figure 6.4: Field lines with clones

The main advantage of referencing a line compared to having an object for each line instance is the lower memory consumption and the higher performance. The vertices of the line are stored and the referencing nodes change only their rotation around the rotation axis. Without making use of this feature the computationally intensive drawing of the field lines would take too much time.

Both Java3D and JME contain a mechanism for sharing nodes. The Java3D scene-graph is shown in Figure 6.5. A scene-graph branch to reference as well as a link to this branch for every clone are constructed. With an additional transform element a different rotation for every clone can be applied. With JME this behaves slightly differently (see Figure 6.6). The link to the template branch must be compatible with the referenced template branch. This is mainly due to the incompleteness of the JME library. Otherwise, a single inner node object with attached leaf nodes could be taken in conjunction with a linking shared node object. However, in JME link nodes are only implemented for triangle mesh geometries. In computer graphics field lines are represented by lines. In order to be able to link to such lines a `SharedLine` class was implemented. As with Java3D the different rotations

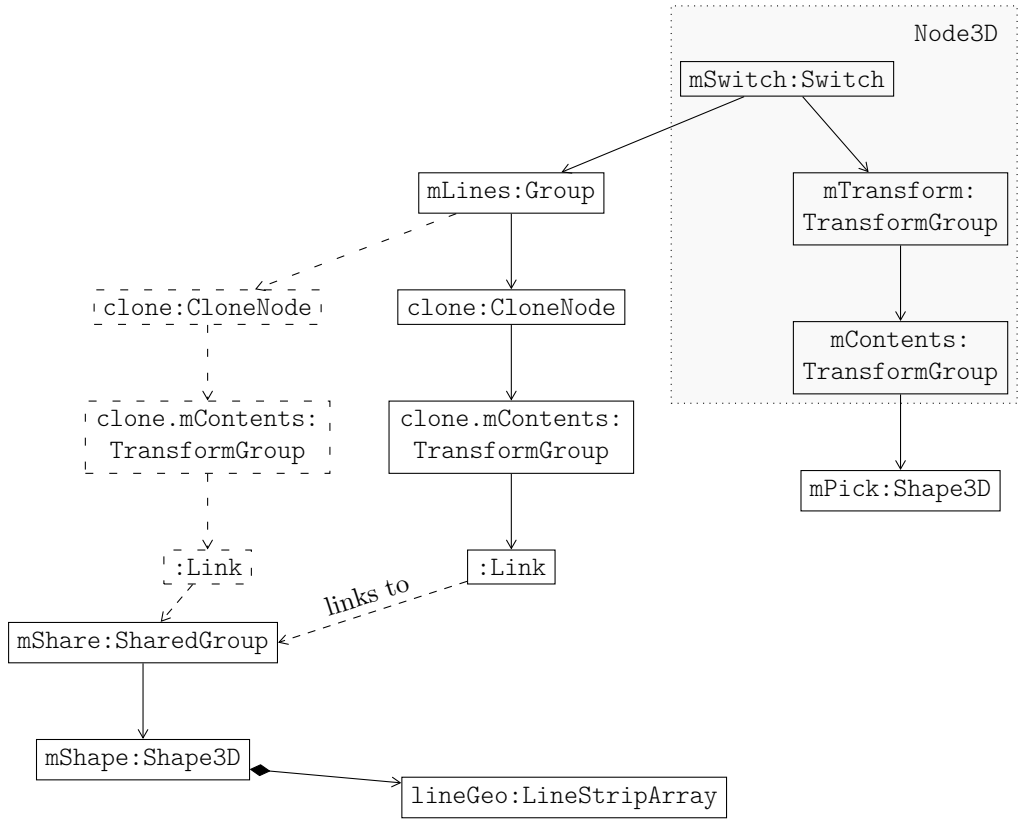

Figure 6.5: The filed line node scene-graph for Java3D. Transforms do not apply, since the branch is attached to the top switch-node. The shared group at the bottom contains a branch representing a field line. The `Link` node which will be created for each field line clone references to this template.

are applied by an inner node further up in the scene-graph.

In TealSim field lines are defined by a number of vertices representing the template line, per-vertex color information and the number of clones to be displayed. Since this parameters differ from other graphics objects a new interface to be used by the scene factory has to be defined. In order to keep the Java3D code mostly unchanged, its methods are given by the Java3D field line node. The method to set the geometry takes a float array with the line vertices and the per-vertex colors. For JME this arrays have to be converted to Java NIO-buffers. Since Java3D works with 3-dimensional colors as opposed to JME's 4-dimensional colors, the interface only support three dimensions. Thus, the color buffer of the JME line has be converted as well.

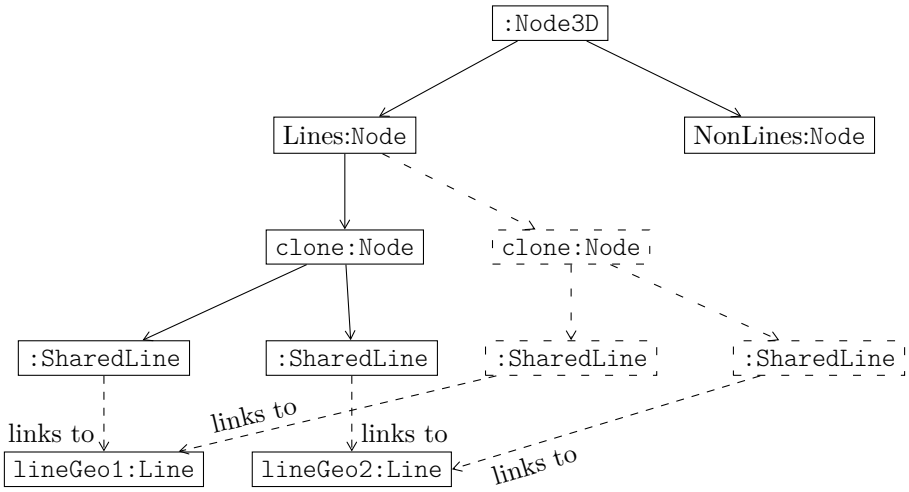

Figure 6.6: The field line node scene-graph for JME defines the line geometry template in the `Line` objects at the bottom. The clone nodes link to these objects via a `SharedLine` object.

While a simulation is running the line geometry often changes with every single calculation step. Therefore, synchronization has to be facilitated to avoid concurrency problems. In Java3D this is done by using the `synchronized` keyword provided by Java. In our case this locks the method for changing the geometry with the node object as lock. In MTGame any change to the part of the scene-graph which is currently displayed has to run in the so-called renderer thread. In order to minimize the load of this thread the vertex data in the geometry update method has to be prepared first. Then it will be passed to the renderer thread. This process does need any additional synchronization mechanisms such as locks, since the critical code is always executed by the same thread, namely the renderer thread.

Array nodes

These elements are capable of handling an array of TealSim scene-graph nodes. This is mainly used for field direction grids shown in Figure 6.7. Similarly to the field line nodes an interface to array nodes has to be defined. This interface has to be implemented by both, the Java3D and the JME node allowing polymorphic usage of both implementations. TealSim needs the array node to have access to every single element of the array. This is done by indexed access as well as by an iterator.

In Java3D the implementation of the array node uses a collection provided by Java to store all elements. Since a JME node is already capable of

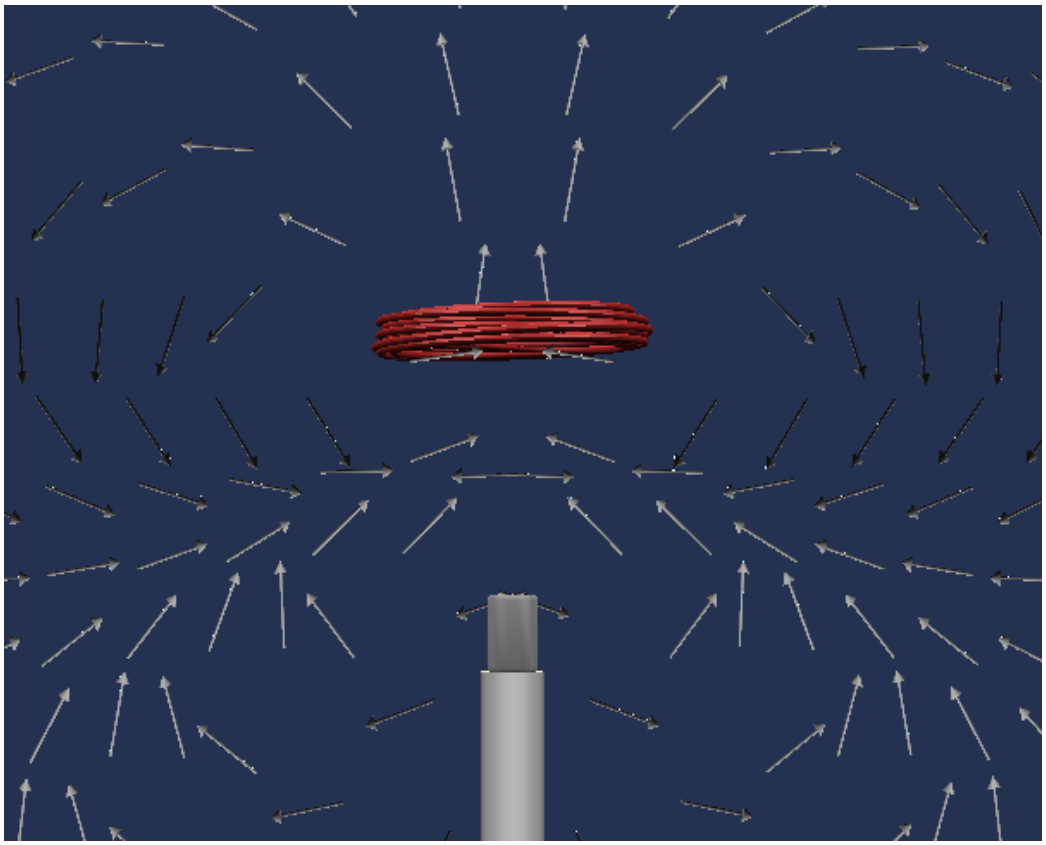

Figure 6.7: A field direction grid internally consists of a number of arrows held by an array node. The arrows point into the direction of the field.

handling scene-graph nodes as children. For that reason no additional data members are needed in the JME implementation. Thus, the implementation of the interface is simple. The calls can be forwarded to the JME node and an iterator class with bounds checking has to be defined. Since for the application of a field direction grid every member of the array is accessed frequently while the simulation is running, some thread-safety features need to be added. Similarly to the field line nodes this is done by delegating critical code to the renderer thread.

External 3D models

TealSim must be capable of adding 3D-models created by 3DStudio-max[6]. As an example, the wires shown in Figure 6.7 are such an externally created

[6] `http://www.3dstudio-max.com/`

model. The 3DS format supports various features which makes writing an importer complicated. Fortunately, JME already comes with such an importer. Additional information about the model are needed to load a model correctly:

- The path to the model file.
- A position offset vector needed since coordinate origins within the 3DS file may differ from the JME or Java3D system.
- A scaling vector.
- Optionally a path to the textures of the model. If this data is not given, the textures are assumed to be in the same directory as the 3DS file.

This information is stored within a container class. The model file must be accessible. When TealSim is later used within OW access to the artwork must be provided. Some additional tweaks such as rotating the model by 270 degrees around the x-axis need to be applied. These are adaptions to compensate conceptual differences between JME and the 3DS format.

6.2.2 Descriptive data types

The concepts in computer graphics often allow different data types to describe the appearance of the scene. This includes colors, materials, transforms and bounding volumes. Java3D uses different data types and classes for these features as JME. Various design and implementation decisions were made to be compatible with both graphic libraries. Those are described subsequently.

Colors are used in TealSim not only to make the scene colorful, but also to indicate values, e.g. the strength of the field at a certain point of a field line. Colors can be represented in different ways. Different color models can be distinguished. TealSim uses a RGB-based color model which is supported by both, Java3D and JME. A transparency valued is then added to the color model. Furthermore, the color of an object is defined by four colors; ambient, diffuse, specular, and emissive. At data type level TealSim uses four three-dimensional color vectors, one for each of the four colors. The three dimensions store the value of the red, green, and blue color component, respectively. Additionally, a single float is used to represent the transparency. JME used four-dimensional color vectors exclusively. The additional value holds the transparency value, one for each of the four object colors.

Colors are only one of the definable material properties in TealSim, the others being shininess, cull mode and face mode. The shininess is represented as float. TealSim uses values between 0 and 1. Since JME uses a range

between 0 and 128 the values have to be converted. With the culling mode faces can be hidden. Front-face culling, e.g., allows an object to be visible only from the backside. A similar concept is the face mode. Its effect on the rendering is shown in Figure 6.8. Spatial object are represented by vertices;

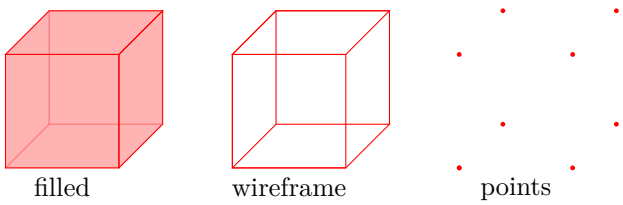

Figure 6.8: Different face modes shown with the example of a cube.

a number of consecutive 3-dimensional vectors. The face mode specifies if these vertices should be rendered as points, as line segments connecting the vertices in order, or as areas filling the area between the line segments. Both Java3D and JME only support the latter two modes. A points mode can only be achieved by workarounds. Since no TealSim simulation makes use of it currently it is not implemented yet.

To be able to use a common data structure for all mentioned material data a class was introduced. It stores four-dimensional color vectors for ambient, diffuse, specular and emissive color, respectively. Additionally, the shininess is stored as float and the culling mode as well as the face mode as integer mask values. The material class can be used externally by simulations as well as internally by rendered objects. Setter and getter methods are provided to be compatible with both, three-dimensional colors with a single transparency values and four-dimensional colors. Simulations can set a material object on a scene-graph node in order to apply the material. For Java3D nodes the material is stored internally within an `Appearance` object which is referenced by the leaf nodes of the scene-graph (see Figure 6.2). In JME materials are applied directly to any scene-graph node, including inner nodes by using state classes. Nodes deeper down the scene-graph can overwrite states of upper node objects. By applying a default material state to the top node Java3D's default material settings can be applied. This assures the scene to be in the same appearance when no material is explicitly defined by the simulation.

Another issue where Java3D and JME differ are transforms. These are used for rotating, scaling and translating scene-graph nodes. Figure 6.9 shows an object diagram indicating how transforms are applied in Java3D. Transform groups are inner nodes of the scene-graph used for applying transforms. The actual transforms are stored in a container class called `Transform3D`. Internally a single matrix is used to store scale, translation and rotation. In

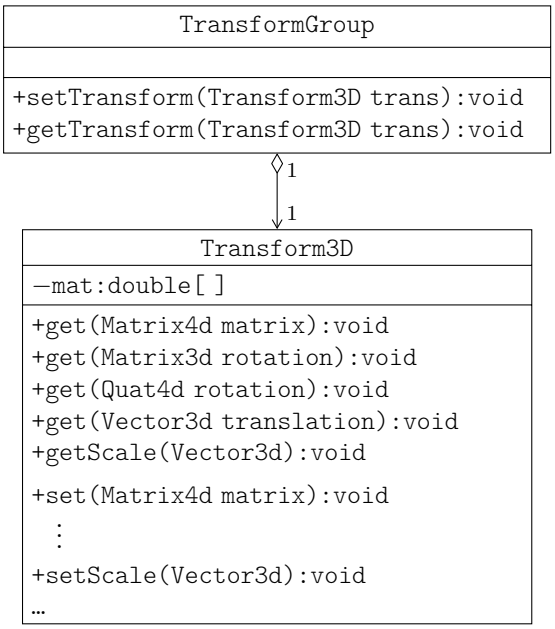

Figure 6.9: Object diagram: Java3D's transforms

JME those three transforms are stored separately (see Figure 6.10); translation and scale in a 3D vector, and the rotation in a quaternion. The getter and setter methods for all transforms are defined in JME's base class for scene-graph elements. Thus, transforms can be applied not only inner nodes, but also leaf nodes.

In TealSim transforms can be set to the most abstract scene-graph node class `Node3D` which wraps around Java3D or JME scene-graph classes. To be compatible to the previous Java3D-only implementation, the transforms are passed as Java3D transform objects. The JME implementation has to convert this transform object by obtaining translation, rotation and scale. When the node's transform getter method is called the conversion is performed the other way by constructing a Java3D transform object out of the node's transforms.

The last remaining data types to be usable by both, Java3D and JME are bounding volumes. This volumes mark the outer bound of a displayed object. It can then e.g. be used to determine if the object is within the field of view and thus, has to be rendered. The following bounding volumes are provided by Java3D and/or JME:

- A *bounding sphere* (Java3D and JME) defined by a center point and a radius.
- An *axis-aligned bounding box* (Java3D and JME) defined either by two

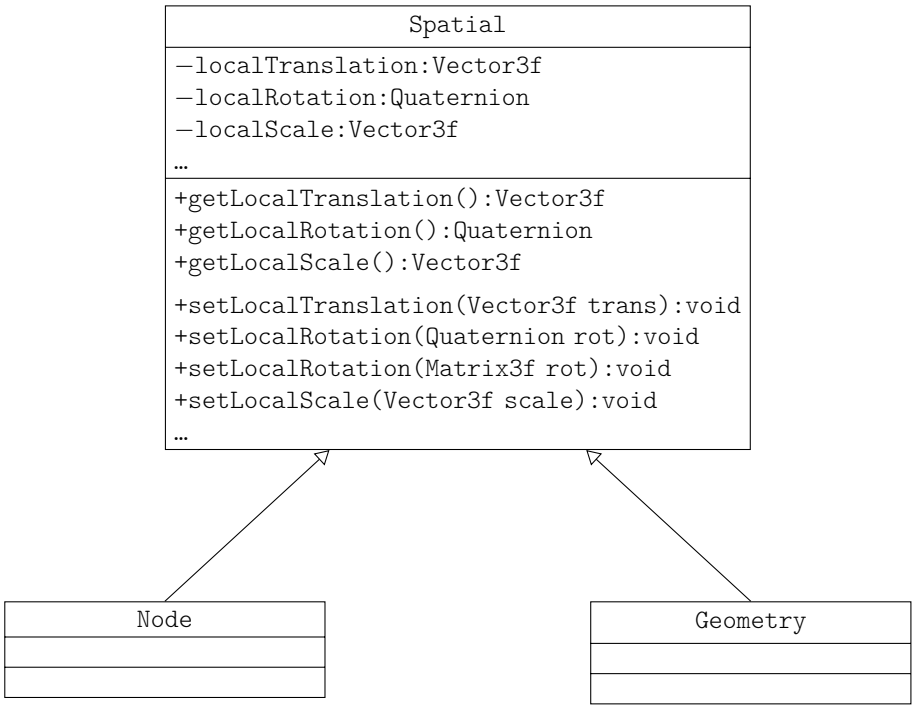

Figure 6.10: Class diagram: JME's transforms

opposite corner points, or by a center point and dimensions to all three axes.

- A *bounding polytope* (Java3D only) defined by surrounding half-spaces.
- A *bounding capsule* (JME only) defined by a center point, a line segment and a cap sphere radius.
- An *oriented bounding box* (JME only) defined by the eight corners of the box.

TealSim simulations only uses bounding volumes available in Java3D. The bounding sphere is represented with the same parameter in both, the Java3D and the JME library. This is not the case with the axis-aligned bounding box. While in JME this volume is defined by a center point c and an extend e to each axis, Java3D uses two opposite corner coordinates u and l of the volume. In Figure 6.11 those two approaches are shown. Since the extend e is a vector from the volume's center to the upper point, the center-extend representation can be converted to the upper-lower-point representation by simple vector additions with the three-dimensional vectors:

$$\vec{l} = \vec{c} - \vec{e}$$

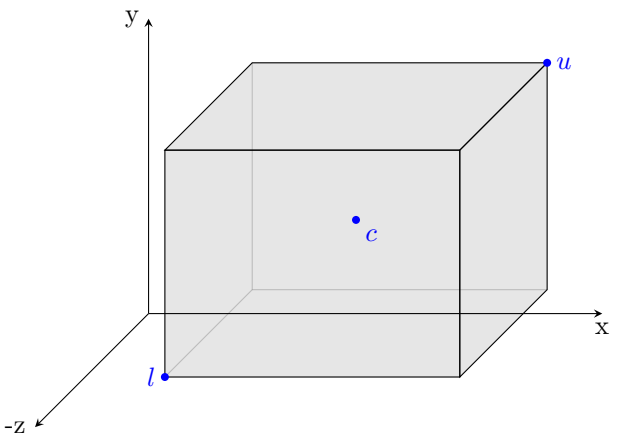

Figure 6.11: Bounding box coordinates

$$\vec{u} = \vec{c} + \vec{e}$$

Bounding capsules are not implemented by JME. Thus, capsules, when used by simulations, are converted to surrounding bounding spheres when JME is used.

With respect to software design bounding volumes are implemented by using an abstract class inherited by the specific bounding volume classes (see Figure 6.12). Such a hierarchy is needed in TealSim as well. The nodes

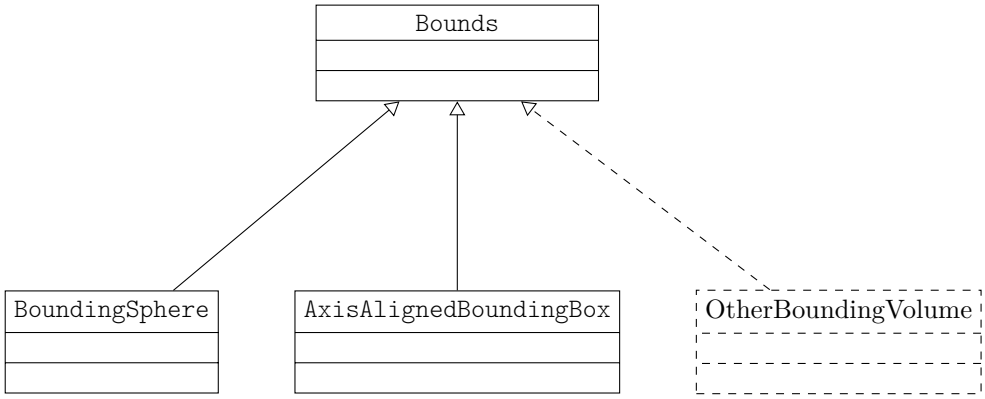

Figure 6.12: Class hierarchy of bounding volumes

have an interface to set and get the current bounding box. The new bounds data type uses the center-extend model for axis-aligned bounding boxes. The JME representation needs to be converted as shown above. The reason for choosing this model is not to change much in the simulation implementations. Previously, Java3D's bounding volumes were used directly. They have to be

replaced by the newly defined TealSim bounding volumes. The Java3D and JME-specific code handles the conversions. This allows TealSim to extend to other graphic libraries in the future.

6.2.3 The Viewer

TealSim's GUI is implemented with Java swing components. It shows up some control panels and a 3D panel showing the 3D graphics. The class responsible for this is the *viewer*. Since it is a swing panel it can fill the provided space for the 3D window in the GUI. That viewer also holds the scene-graph and the lights as well as the canvas where the 3D scene is drawn to. It is only used on the desktop version of TealSim and not needed in OW.

As with the scene-graph object the viewer is created by the scene factory (see also Figure 6.1). This way the needed implementation is returned.

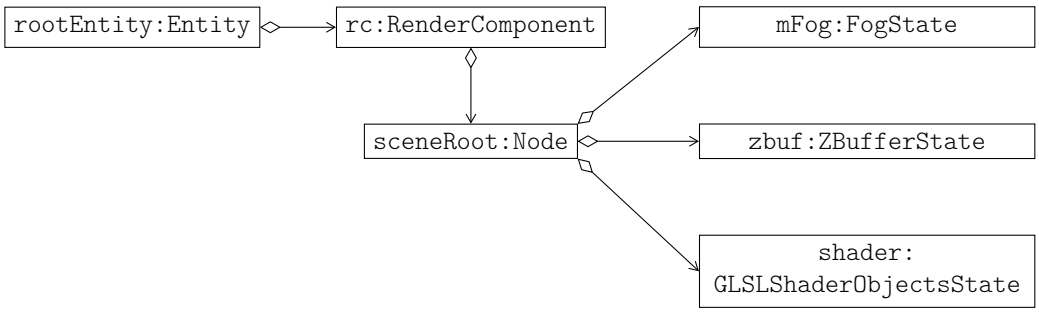

Figure 6.13: Object diagram of the viewer's root entity. Properties of the scene such as fog and the z-buffer state are added to the scene-graph's root node.

In order to implement such a viewer for JME more detailed information about MTGame is needed. In MTGame a scene is based on entities representing an element or a group of elements in the scene, and their components. Entities are important for the thread management of MTGame (Twilleager, 2008). The JME implementation in TealSim uses two entities, namely a root entity and a camera entity. An object diagram of the root entity and its linked objects is shown in Figure 6.13. In order to show a scene a render component has to be attached to the root entity. This component is capable of holding a scene-graph to be rendered as well as nodes for the light. The viewer implements methods to add and remove scene-graph objects which will be added to the root node of the scene-graph. To become part of the scene entities have to be registered at the so-called "world manager". Since only one of these objects should be available the class was implemented as

singleton (see Gamma et al., 1994). However, for the use in OW the instance of the world manager must be settable.

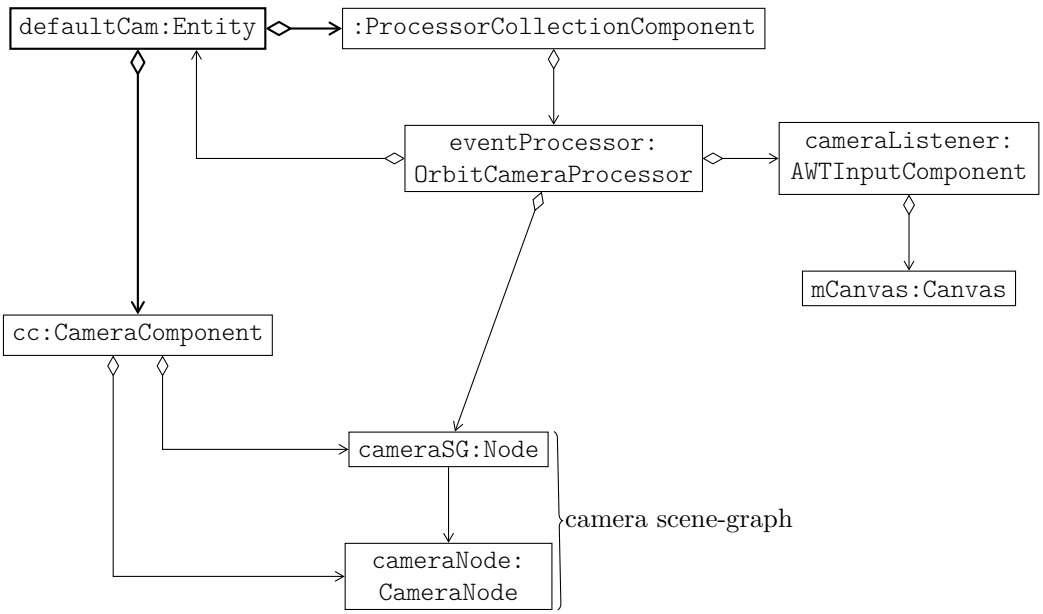

Figure 6.14: Object diagram of the camera entity

The second entity needed is the camera entity (see Figure 6.14). A camera component holding the scene-graph of the camera is attached to the entity. In TealSim the 3D model is static, i.e. it does not move or rotate. If the user wants to zoom or to see the scene from different perspectives the camera is moved. This requires a camera scene-graph of at least two elements; an inner parent node and the camera node itself. Additionally a processor to move the camera is linked to the entity by a corresponding component. Processors are used in MTGame to fulfill specific tasks. The orbit camera processor manages the user input with the help of an input listener. It moves the camera accordingly. The input listener needs a link to the canvas which must be created as stated in Twilleager (2008); through the render manager obtained by the world manager.

The camera component needs information regarding the field of view, the aspect ration, clipping distances and so forth. The parameters have to be set such that the view is nearly equal to the view with Java3D implementation. Some of the parameters can also be influenced by the simulation. The viewer interface includes setter methods for those.

6.3 Preparing TealSim for Client-Server use

Since OW consists of client-side code and server-side code, TealSim will have to run on OW as client-server software. Although the OW-specific code will be put into the OW module (see Section 6.4), some TealSim has to be prepared for that. This Section describes the changes made for this step. Possible alternative approaches are discussed as well. The communication between client and server is also an issue addressed in this Section. The OW module should eventually scale up well and use as little bandwidth as possible. On the other hand all the shared data should be perfectly synchronous among all clients.

6.3.1 Synchronization of the 3D Objects

A part of TealSim which has to be synchronized among all clients and the server are the three dimensional objects. Simulations use representative classes, so-called "rendered objects". This objects hold all physical properties and create the according scene-graph nodes for the 3D representation using the scene factory described in Section 6.2. Thus, rendered objects store all the relevant data the simulation will need to run.

Since some parts of the simulation will run on both, the client and the server side, the rendered elements will also be needed on both sides. Because nothing will be drawn on the server side, scene-graph objects do not need to be stored there. However, any changes applied to the rendered objects during the simulation on the server will have to be sent to the client. This could be avoided by running the whole simulation on the client side, which is impractical due to synchronization issues (see Section 6.3.2).

Having all of the rendered objects on both the server and the client side has the advantage of being able to do parts of the calculations on the client and parts on the server. This is necessary since the data representing some elements cannot be transferred whenever the data changes. Field lines are the most obvious case. In most simulations a single field line consists of 200 data points and possibly as many color specifications. One point requires three floats. This sums up to 1600 bytes needed for a single field line. While a simulation is running, the field line data has to be calculated with every frame, i.e. 20 times a second. This leads to more than 31 kilobytes per second needed to be transferred during the simulation to each client for one single field line. Some simulations contain more than five field lines and whenever there are many users logged in, the traffic on the server increases to an intractable amount. A field direction grid also produces a lot of data since the length and rotation will have to be transferred for every single arrow.

Since all the rendered objects can be found on the client side, the field lines and the field direction grid can be calculated and displayed on the client side only. This overcomes the problem of the need to transfer all the data points of a field line. The needed computational power is also divided since the client and the server can do different tasks.

The approach requires to serialize the rendered objects in order to send them to the client. If calculations are to be on the server they have to be serializable anyway. PD needs this behavior. However, transferring and distributing the serialized rendered objects with every change is not possible due to the big bandwidth utilization. Instead the objects can be transferred this way after they are created and be synchronized with update messages.

The dependence on the underlying graphic output primitive classes completely vanishes with this solution. The actual scene-graph objects only need to be created on the client side. If an other graphic library is used the scene-graph objects can be implemented for the desktop version of TealSim. Those objects can be used on the client side with the client-server version.

6.3.2 Splitting the Simulation Engine

The simulation engine is capable of all calculation in the simulation. This Section describes the changes made on the simulation engine part of TealSim to prepare it to work as client-server version. First the engine prior to the changes is explained. Subsequently, the changes made are discussed. Alternative approaches are also stated and discussed.

Engine functionality

The simulation engine is capable and responsible for the simulation calculations. It is given a list with all elements it needs from the simulation. This elements called “simulation rendereds”are marked by an interface. When the simulation is running the internal data of the simulation rendered elements is calculated 20 times a second. Several types of forces are taken into account for that. The simulation runs in four steps with the according methods implemented:

- `doReorder`,
- `doDynamic`,
- `update`,
- and `doRefresh`.

The *doReorder* step resolves all the collisions occurring between the objects at the beginning of the step. The objects rearranged are those who have a so-called "collision controller" attached. With the help of this controller those elements are repeatedly reordered until there is no collision any more. Some of the elements need to be told if the positions have changed after the collision resolving step. For this purpose a call-back method is implemented. Usually this call-back is used to set internal properties, e.g. the position, to a previously calculated shadow value. Further information about this values are given in the doDynamic step paragraph.

In the *doDynamic* step the actual integration of the objects occurs following the laws of physics. Only so-called "integratable" objects are affected by this step. These objects place the internal values, such as velocity, position, mass, charge, and so forth, into a single double array. This array can then be obtained by the engine and the values are used for the calculations. Additionally, the object need to provide derivative values to the engine. The calculations are done by the engine in several iterations. After each iteration the values are written back to the simulation rendered object. Those objects store the values as shadow values. After the last iteration, the objects are informed of the end of the doDynamic step.

To tell the integratable objects to apply the shadow values to the actual ones, the *update* phase calls a method to each relevant object. In order to save runtime the objects need to check if the dependent value is different from the actual before setting it.

The last step is the *doRefresh* step which consists of two phases. The first phase only effects specific object, called "spatials". These elements need to perform post-step calculations. An example are field lines which are computed after each step with the updated positions, charges, and so forth, of the objects influencing the field line. In the second phase the viewer is instructed to render the whole updated scene. For that purpose the viewer will forward the rendering call to all the objects to be rendered.

A part of the class hierarchy of TealSim prior to the changes is shown in Figure 6.15. Two engine types are implemented, a general simulation engine and a derived electromagnetic engine capable of handling electromagnetic simulations. The simulation itself contains the information which engine will be need. Besides the electromagnetic engine, different other types, such as a biochemical or a kinetic engine are declared. Previously, each of the types related to a specific engine class. However, the current implementation of the simulation engine covers all, but the electromagnetic simulation functionality.

The engine is created according to the type specified in the simulation and is then referenced by the *engine control*. This object connects the user interaction with the simulation engine, i.e., it provides buttons to start,

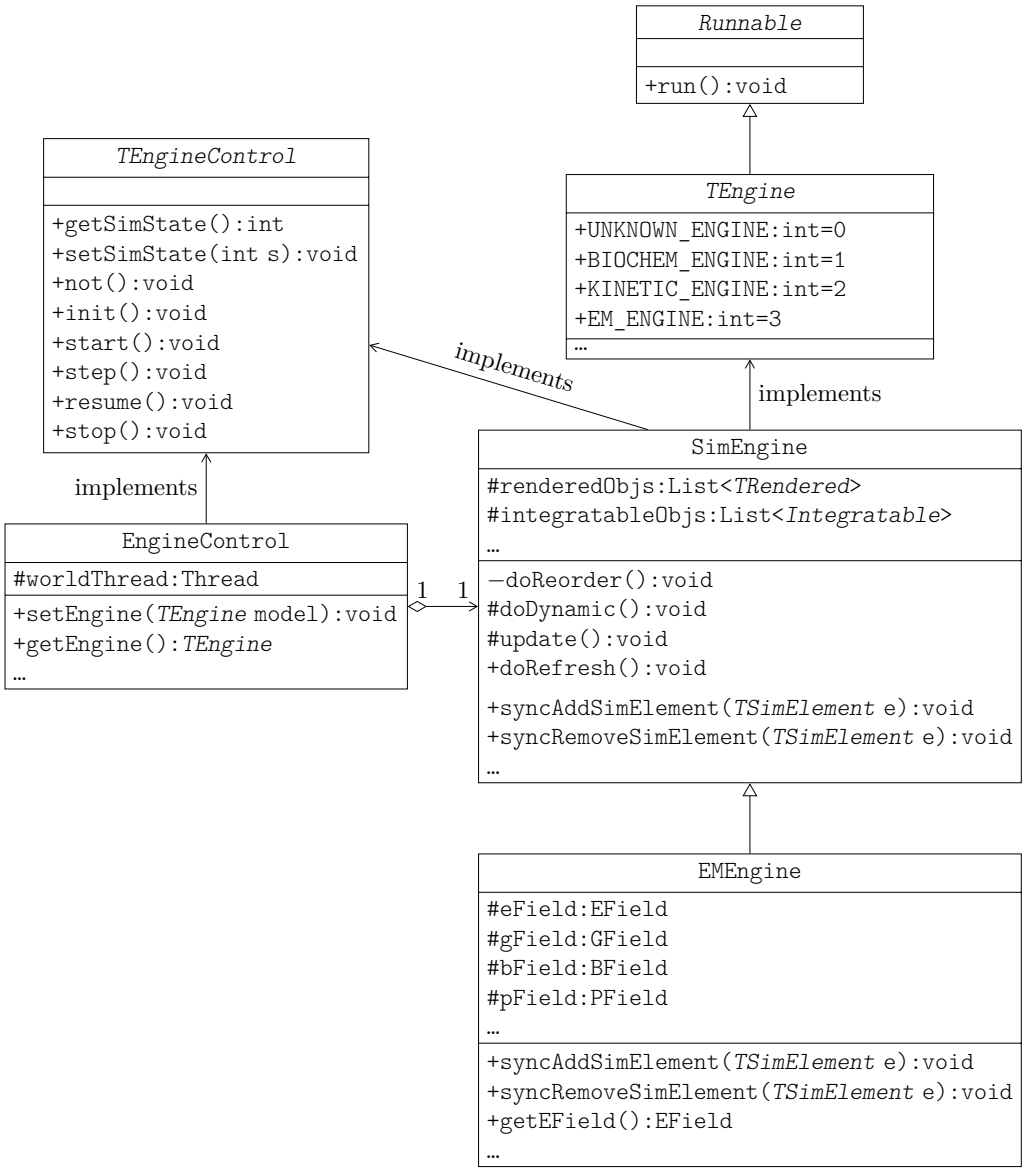

Figure 6.15: Class diagram: previous simulation engine

stop, pause the simulation, and so forth. In order to provide the buttons the engine control needs to be a swing object which can be integrated to the TealSim's GUI. On play-button click, the engine control starts the simulation by running the engine in its own thread. This way the engine is set to running mode and will run the simulation by performing the four simulation steps repeatedly. The different protection states of the four methods for each simulation step indicate previous poor design of code changes. This issue

has to be resolved when the engine is adapted. When the engine is not in the running state only the doReorder step and the doRefresh step is executed.

Every step should take around about the same amount of time. This time can be specified by the simulation. By default it is 50 milliseconds. If there is still time left after the four simulation steps the rest of the time is waited with a sleep call on the thread. The engine contains methods to add new elements. Those can be called while the simulation step is executed. In this case the elements are stored in a temporary list and are then added or removed after the calculations. This avoids concurrency problems with the element lists.

The electromagnetic simulation engine adds functionality to the simulation engine top class. It stores some data in its members so they can be accessed with additional methods. Mostly, these are field objects. Four types of fields can be handled by the electromagnetic engine:

- electric field,
- gravity field,
- magnetic flux field and
- Pauli field.

The engine does not need to provide much functionality for these fields since the calculations occur in the simulation objects itself and in the field objects. The latter store references to all field-creating simulation elements. They are capable of calculating the field force at every coordinate when needed.

The electromagnetic engine's role is only to provide quick access to these elements specific to electromagnetism. It keeps the elements in internal lists the application can step through during the simulation step. The lists are built when the engine is initialized and whenever an object is added to the simulation.

Design of a new class hierarchy

The design of TealSim allows to add new types or subtypes of engines by inheriting from the existing ones. However, if it should be used on a client-server system this becomes tricky. If some parts of the engine should run on the server it would run on PD. This requires some restrictions for the code (see also Section 6.3.3). Synchronization among PD's managed objects, e.g., is done exclusively by PD itself. If the engine contains synchronization mechanisms deadlocks can occur. Such code restrictions requires TealSim

to provide both, desktop-specific and client-server-specific code to run the engine.

Figure 6.16 shows a possible class hierarchy if the client-server functional-

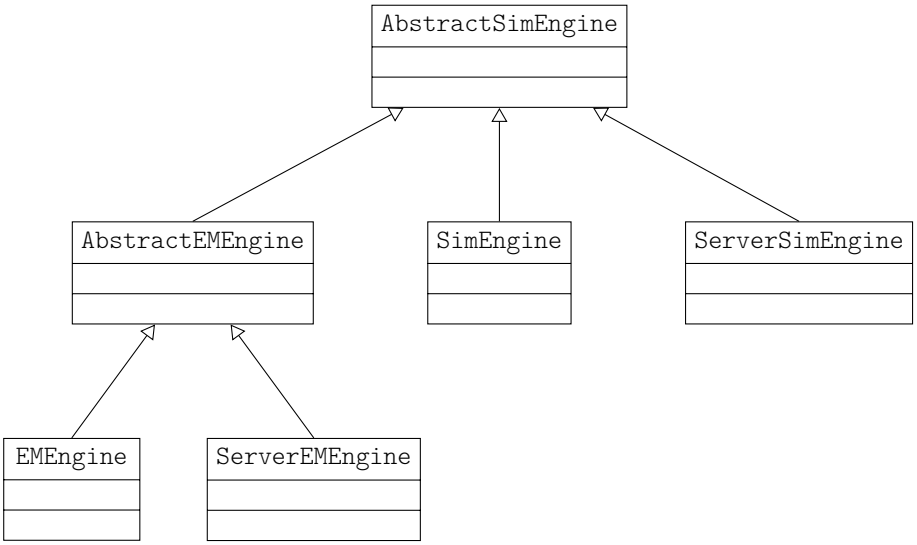

Figure 6.16: Extending desktop version by inheritance. Three instances of each, the simulation engine and the electromagnetic engine are declared. This results in problems such bad extendibility.

ity is added with additional inheritance classes. Code needed in the desktop version as well as in the client-server version will be placed in two abstract classes, namely one general and one electromagnetic abstract engine. Both of these two classes can be extended by classes specific to the desktop version and the client-server version. With this approach the client-server-specific engine classes could be implemented within an OW module whereas the desktop version of the engine comes with TealSim. Thus, TealSim would provide the same functionality as before, but with enhanced extendibility used by the OW module.

However, this approach comes with many disadvantages. First, it is not easily extendible. If a client engine is needed two different classes would have to be added. One for the general simulation engine and one for the electromagnetic simulation engine. This code is very likely to be very similar if not identical. Thus, the code would be copied from one class to the other having a bad influence on the maintainability. If changes are to be made on that copied code they would have to be applied to both added classes. Adding a new engine type, such as a kinetic engine leads to the same problems.

A closer look on the problem reveals two abstraction levels in the code hierarchy; the engine type (general and electromagnetic) and the application

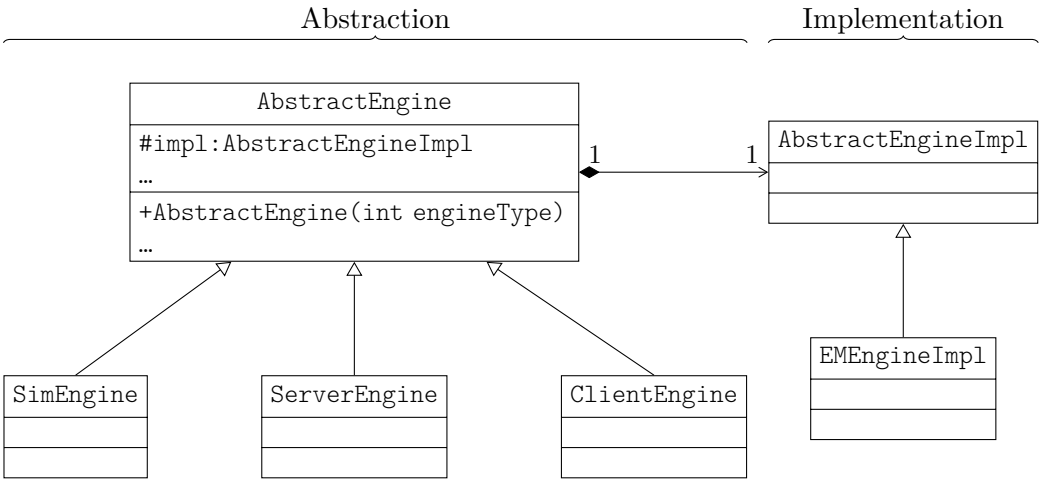

Figure 6.17: The engine class hierarchy using the bridge design pattern. On the left the class hierarchy for the application types (desktop, client-server) are shown. The implementation (right) implements the different engine types (general and electromagnetic).

type (desktop version and OW server-side version). In Gamma et al. (1994) exactly this problem is addressed. The solution to this is to build two class hierarchies and to use one of them as the abstraction and one as implementation hierarchy. Figure 6.17 shows the resulting class hierarchy. The design patter used is called “bridge”. The abstraction hierarchy holds the classes for different application types, while the implementation builds a hierarchy of engine types. The abstraction maintains a reference to the implementation. Compared to the pure-inheritance approach shown in Figure 6.16 a client engine for the client side of the client-server version is added to the abstractions. Only one additional class is needed for that. This way, duplicated code is avoided. In order to separate the code of TealSim and OW the server engine and the client engine will be put into the OW module.

The implementation is instantiated when the abstraction is constructed. This is achieved by using the existent engine type integer bit masks. Such an integer value is passed to the constructor of the abstraction class which then instantiates the implementation accordingly. The abstraction itself has all the functionality which is not engine-type-specific. The only functionality remaining in the implementation are to add, remove and obtain specific elements. This can, however, change in the future. Then additional functionality may become engine-type-specific and will have to be put from the abstraction to the implementation.

Adding and removing elements must be part of the abstraction’s interface,

which can be accessed from other parts of the application. The implementation has to be informed of every element which should be added or deleted. Hence, it can store the element if relevant. A getter method is harder to implement. Previously, the electromagnetic engine was equipped with getter methods for relevant objects, e.g. the fields. Since those are now managed by the implementation class, a unified getter functionality is implemented. This prevents the engine from having to add a method to the abstraction for every single element (e.g., for each of the four fields). Instead, simulation elements can be obtained by type. For that purpose getter methods are provided. Not only engine-type-specific elements, but also types from the abstraction can by obtained by such methods. Two versions are currently implemented for single objects and collections, respectively:

- Retrieving the object by a class type object. With the help of Java's reflection an object representing the class type is given as parameter to the getter method. The method uses Java's generics and replies the required object of the parameter class.

- Getting the object by an enumeration type. With this approach an enumeration of all gettable objects must be defined. The getter method takes the enumeration type as parameter.

The prior approach has the advantage of returning the needed data type. Neither type casts with the returned object, nor a declaration for every possible returned object is necessary. Especially for fields this approach proves to be simple and effective. With basic data types which are used more often, however, stating the type may be insufficient. The enumeration-approach overcomes this problem. However, the advantages of the first approach vanish. Currently, both methods are implemented. Internally the gettable objects are stored in maps with class type or enumeration type as key. This provides $\mathcal{O}(\log n)$ runtime with class types and $\mathcal{O}(1)$ runtime with enumerations. For class types this is quicker than using nested if-clauses. Type maps are contained by both, the abstraction and the implementation. If an object is requested the abstraction will first try obtain the object from its own list and then from the implementation's list.

There is also an other approach of using the bridge pattern which is logically better. The engine type could be the abstraction and the application type the implementation. However, this is hardly possible in our case because of synchronization issues with PD. Since the server parts of the code should not contain any synchronization behavior, synchronization would have to be part of the implementation. Since a lot of code is affected by synchronization

issues in the desktop version, the separation between abstraction and implementation would leave a lot of engine-type-specific code in the abstraction. With this behavior the bridge pattern does not have any advantages over the pure-inheritance approach.

Separating Server and Client engine

The functionality of the engine has to be balanced properly among client and server side. If the whole simulation is processed by the client, only little simulation data has to be transferred, because it is generated on each client. However, this approach would require enormous synchronization efforts during the simulation run. The engine state with all simulation objects needs to be the same on each client. The user interactions would have to go through the server to be applied on all clients synchronously. This would already require some adaption to the engine code. On the other hand, if the whole simulation is calculated on the server the synchronization among the client is reduced to a minimum. Only the results have to be sent to the clients. However, as discussed in Section 6.3.1 this is not possible because of the high network bandwidth utilization with the field lines and the field direction grid. Those elements have to be computed on the client side anyway. They will not need to be perfectly synchronous among all clients, because as representations of the field they do not interact with other simulation elements. Additionally, synchronizing the field-creating elements will keep the field lines synchronous automatically. Calculating some parts on the client also splits up the computational power between the client and the server. This saves CPU utilization on the server. All the code running on the server side has to run on PD. Therefore it must be adapted as described in Section 6.3.3. There is also the possibility of writing a PD service which would overcome that problem. However this approach comes with many disadvantages. Running a service enables running code "parallel" to the tasks in PD. In this case the thread must be synchronized by hand. The scalability provided by PD is also compromised because the service may not be restricted in the use of computational power on the server. In case of OW the whole virtual world could be slowed down. For that reason this approach was not followed. Thus, all the engine code running on the server has to be made compatible with PD. One calculation step of the server-side engine can be processed as a PD task. If the engine is in the running state the task can be scheduled repeatedly. PD provides functionality to schedule tasks in such a way. The time between the executions as well as the waiting time until the task is executed for the first time can be specified. The task is invoked when PD calls a method required by a responsible interface. This interface, and

thus the method, have to be implemented by the server-side engine.

In order to estimate how much engine code needs to be run on the server a closer look on some specific simulations is necessary. Unfortunately some of them are not deterministic. The "Capacitor" simulation for example starts with randomly distributed charges. During the simulation charges can be added or removed. The less deterministic simulations are the harder is it to synchronize them among clients. To avoid that problem the engine has to run mostly on the server. As mentioned above, it is not a problem to put the field line and direction grid calculation to the client. It is even possible to omit the calculation of these field-depending shapes synchronously to the other calculations. This might be necessary if the client is too slow for such frequent calculations.

The calculations which have to be done on the client happen inside the *doRefresh* step. As mentioned previously, this is the last phase of the engine calculation. The spatial objects affected by this step are mainly the field directions grid and the field lines. Subsequently the rendering is initiated. These steps are needed on the client side, i.e. the doRefresh step will happen there.

The integration done by the *doDynamic* step performs the main calculation phase of a whole engine step. For synchronization reasons, this will have to happen on the server. The *doReorder* step is a preparation step for the integration and will therefore happen where the doDynamic step happens; on the server. The *update* step is needed to perform updates to the simulation objects. These updates concern mostly internal descriptive values which are replaced by shadow values stored temporarily during the simulation step. This replacement has to happen prior to the doRefresh step since this phase may need the values. Because the doRefresh phase does not need to be executed on the server the update phase can also be skipped here. After the integration the clients have to receive the values and can then perform the update and doReorder step. Fortunately, the data needed to be sent to the client is already there and used in the doDynamics step. All the changed data is contained by the so-called "dependent values". A simple call on the engine can retrieve all the shadow values as a single array of doubles, this array will be sent to the clients. The client can then update all their object by setting the retrieved dependent values on the simulation objects.

The order of the integratables simulation elements in the according list in the engine plays a role in this case. Since this order corresponds to the array with all the dependent values, the order on the clients' integratable list must be the same as on the server. This can be accomplished by sending the order to the clients. After setting the shadow values of the client engine the *update* step can be performed to set the actual property values of the elements to

the shadow values. Subsequently, the client engine is ready to perform the *doRefresh* step in order to calculate the field lines or perform other actions done in this step. Lastly, the client engine triggers the rendering.

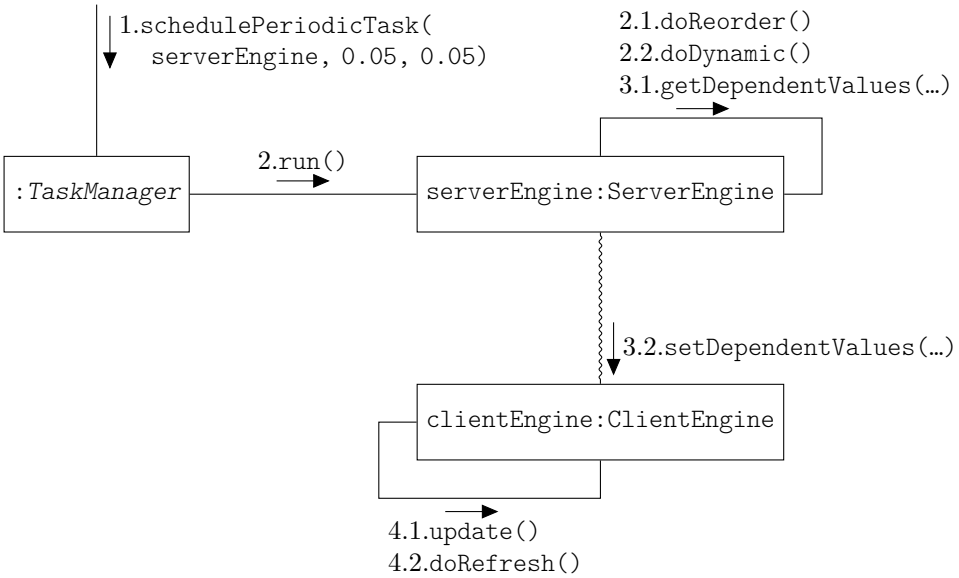

Figure 6.18: Communication diagram for the engine execution step. Network communication is indicated by the ~~ line segment for simplification.

The whole process of a simulation step is shown in Figure 6.18. First the engine is registered at the task manager of PD. With this registration the task manager also gets information of when and how frequently the task should be performed. The task manager then triggers the run method of the engine periodically. The server-side engine can now do the server-side calculation step and retrieve the dependent values to be sent to the clients. On the client side the relevant objects are updated with the dependent values and the remaining calculation steps are performed.

With the discussed method *all* dependent values are transferred to each client every simulation step. This can lead to a high bandwidth utilization at the server. In the capacitor simulation, e.g., there are 24 point charges per default. Each charge has got three dependent values for the position and the velocity, respectively, and one for the charge. Since all of these seven values are 64 bit doubles the transferred data is more than 10 kilobits per step and client. With 20 steps per second this value multiplies up to more than 200kbps, which is not always accessible. In order to reduce this amount some methods are possible:

- Using floats instead of doubles for the dependent values. This would reduce the amount of data which needs to be transferred to 50%. If the engine still uses doubles internally the double array would have to be converted into a float array which would take too much time. Another possibility is to use floats in the engine instead of double. This can possibly lead to rounding errors. Additionally, many other elements in TealSim would have to be changed to use floats.

- Transferring only the changed data. Unfortunately, it is not easy to determine the values which have changed. This can possibly only be done by comparing each value to the previous one. Another problem would be that additional information would be needed about what values are skipped for transfer. Simply setting all depending values on the client side would not be possible any more since some of the values are in the array and some are not. Another similar possibility is to transfer only value differences and to compress the data. Since several numbers will not have changed, a number of zeroes will occur in the data which compresses well. However, difference transmission relies on all data to be transferred in order to be synchronous.

- Not sending values which are not needed on the client side. There are several unneeded dependent values. With the point charges for example those are the velocities. The position is needed for rendering and the charges are needed if field lines have to be calculated. It is not easily to determine if a value is needed on the client. This is sometimes simulation-specific and would therefore need specific adaptions for single simulations. As it is with the transferring of only changed data it would not be possible to set the clients dependent values with a simply call.

The three methods could be combined arbitrarily. Because of their drawbacks in terms of performance and the increased complexity none of these methods were implemented. A test with several clients and different applications to test the bandwidth was successful.

6.3.3 Preparing TealSim for the PD server

All the code running on the server needs to be ready for PD execution. This forces the code to fulfill some additional requirements explained subsequently.

PD's first priority is to make client-server applications scalable. For this purpose the execution is split up into tasks. Every task has got a certain amount of time to be executed. If it has not finished after that time it is

interrupted and rescheduled. Splitting the execution into tasks avoids having a single task blocking the whole execution. A task always runs as an atomic state change. If it is interrupted the state prior to the task execution is reestablished. That functionality is implemented in PD using serialization. The objects are arranged to logic groups, each of them represented by a so-called "managed object" class object. In PD a marker interface exists for such objects. Classes implementing this interface are serialized and de-serialized at once with all of their members. PD holds all managed objects in their serialized form. If a task needs any data of them this objects will be de-serialized. Synchronization between the managed objects is done by PD. If a task tries to write to a managed object while another task accesses it the task will be closed and rescheduled for later execution. Reading one managed object by more than one task is allowed. After a task has successfully processed to its end the used managed objects are stored and are replace the previous versions of the objects. If a task cannot be completed the previous versions of the managed objects are kept and can be de-serialized again whenever another task (or the same rescheduled task) require it. Code run on PD has to follow several guidelines ("RedDwarf Application Tutorial," 2010):

- All objects must implement the serializable interface. Without that the mentioned atomicity of a task cannot be provided. PD throws an exception if an object not being serializable.
- 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.
- All inner classes should be static since the time taken for the serialization increases significantly if they are not.
- Synchronization blocks must not be used among managed objects and their members. Since PD uses its own locks those can conflict with the ones the user defined code uses. This can easily lead to a deadlock.
- 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.
- 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.

- 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.

Most of the problems are relatively easy to overcome in TealSim. The Java's exception base class should be used very rarely in Java code anyway. Non-constant static fields are sometimes used for singleton objects. For such purposes PD provides functionality to bind an object to a name, i.e., a Java String. However, for compatibility reasons PD code should not be put into TealSim. Such code parts can be replaced in OW with classes defined in the OW module. Fortunately, this was not necessary since all the non-constant static fields used on the server side could be removed.

To pull synchronization functionality out of the code is rather difficult with existing software. Most of the synchronized blocks needed on the server side are placed in the original engine. With the new engine class design explained in Section 6.3.2 those synchronization code-blocks needed for the desktop implementation are put into the desktop version of the engine. The base class, which the server-side engine extends, does not contain any synchronization code.

Non-static inner classes can access the members of the parent class. This is the case because a reference is held internally. The non-static inner classes can be replaced by static ones where these reference to the parent object is maintained manually. This makes the code a little more complicated but makes the serialization needed on the PD server much quicker. The process of replacing the non-static inner classes does not effect the definitions of the simulations at all and is therefore not problematic.

Since no dependencies on PD must not be added to TealSim, the interface for managed objects must not be used there. PD provides a wrapper class for the purpose of converting any object to a managed object. This will later be used within the OW module.

To make all the needed code serializable usually forces a lot of code changes. However, with TealSim the effort is acceptably low because most of the classes were already designed to be serializable. This is mostly due to TealSim's functionality of saving and reloading the current simulation state, which requires the elements to be serialized. The swing components for the user interface are serializable anyway. One problem with serializability occurs with the bounding volumes. Those are not only needed in the

low level graphics but also within the simulations and at several other parts of the code. Originally the Java3D bounding volumes were used, but those do not implement the corresponding interface to be serializable. For that reason manually implemented bounding volumes are replacing the Java3D ones outside the low level graphic packages. Further information about the implemented bounding volumes is given in Section 6.2.2.

With another class of Java3D used in TealSim the same problem was addressed differently. The three-dimensional transform class occurs only two times in TealSim outside the Java3D-specific package. Therefore it was kept and the serialization was done by Java's object serialization customization. Normally, Java stores all non-transient member objects of an object recursively on serialization. This default behavior can be overwritten by implementing two methods in the class to be serialized manually. These two methods will then be called by the virtual machine on serialization and de-serialization, respectively (Greanier, 2000). The serialization method triggers the default serialization first. Then the data representing the transform is written to the object stream responsible for the actual serialization. Transforms can be represented by a 4×4 matrix. In Java, these 16 double values are passed to the object stream. On de-serialization they can be retrieved and a transform object according to these values is constructed. The transform member has to be marked transient, since a non-serializable object leads to a runtime error when serialization is attempted. In some cases members do not need to be re-serialized at all. This happens mostly when the member is used for caching.

6.4 The OW Module

After having the desktop version of TealSim able to run with JME and MTGame an OW module can be implemented. As mentioned in Section 5.3.1 such modules, when constructed and implemented, they can be loaded during run-time via a simple web interface. Eventually, such a module consists of a single jar file. In our case this file will contain all needed TealSim components as well as the OW-specific files. Generally, a module consists of the following components:

- Server-side code put into a `server` package,
- client-side code put into a `client` package,
- code to be distributed to both, client and server put into a `common` package

- and artwork put into a separate directory within the module, called "art".

The module components are zipped into a single jar file consisting of a server-side jar and a client-side jar. The prior archive contains all classes in the `server` package and in the `common` package, whereas the client-side archive needs to contain the `client` and the `common` package, as well as the artwork. For the TealSim module the needed TealSim classes and their dependencies are needed as well. To add them a new "lib" folder is added to the module. TealSim is put to this directory as a jar file including all needed classes.

OW as well as its modules use apache ant[7] to build. Prior to any coding the build configuration file has to be adapted to make use of the TealSim archive. The environment path has to be adapted such that this archive is found and for both, the client and the server parts, the archive is put into the dependency list. That list has also to be extended by other modules needed by the modules, such as the "AppBase" module. Additional functionality to compile and pack the TealSim archive was also added to the ant file. This is especially needed when code changes in TealSim relevant to the OW module are necessary.

Artwork in TealSim mainly consists of 3D models in `3ds`-format. In TealSim they are packed into an archive. For simplicity reasons the models are copied unpacked to the OW module's "art" directory. When the module is uploaded via the web interface the artwork is placed in the OW filesystem. It can be accessed there by the module code via an URL resolver built into OW. Within the TealSim module this artwork is only needed by the scene factory and its functionality to load external 3D models (see Section 6.2.1). In the module a new scene factory is defined by inheriting the TealSim version. It only extends the functionality by resolving the path according to the OW filesystem. Since all displaying functionality is needed on the client side, the created factory is placed into the `client` package of the OW module.

6.4.1 Simulation Selection Functionality

One module should be capable of showing different simulations (but only one at a time) within the virtual 3D world. For that purpose a mechanism needed to be implemented to select a simulation. From the user's perspective, the functionality can be provided by the module properties. The user can simply right-click on the module's graphics and select the according properties window. A drop-down menu with all available simulations will appear. The

[7] `http://ant.apache.org/`

user can now choose the desired simulation. With a module such a functionality can be added by implementing a special interface recognized by OW. This interface declares a number of methods responsible for the needed functionality.

Firstly, the name of the property panel needs to be retrievable. OW provides localization functionality in order to support different languages. Text strings displayed to the user are represented by keys. The translated text for each key is stored in text files with their filenames suffixed by a two-character language code. OW provides functions to read the translated values by file name. The correct suffix of the filename is chosen according to the clients operating system settings. In order to support different languages the name of the properties panel is replied using the described language functionality. For that purpose language bundle files are provided for English and German language. Extending to other languages is possible by simply adding a new bundle file with the language-specific suffix and the translations stored in it.

Another important method the properties factory has to implement is used by OW to hand a `CellPropertiesEditor` object. It is needed by the module properties class to retrieve the current server state class and to make changes on it. Without this functionality it would be hardly possible to apply changes with the use of the properties dialog. A second purpose of the cell properties editor is to indicate changes the user made in the properties GUI. This will affect provided buttons, such as the "Apply" and the "Cancel" button. They will be enabled or disabled depending on whether the properties were changed or not. A click on a button will also trigger a call to corresponding methods declared in the properties window interface. Of these methods, only the one to apply changes is used. It sets the chosen simulation on the server state, retrieved from the properties editor.

Lastly, a Java swing panel to be shown to the user needs to be retrievable by OW. This panel will then be shown to the user. In our case the panel needs to hold a drop-down list with all available simulations, together with a short description text. As shown in Figure 6.19 the properties class is implemented such that it *is* the panel. Thus, it will return itself when asked for the panel object. The available simulations are represented internally as an array of strings and will be instantiated using Java's reflection functionality. The advantage of using a string array is the low memory utilization as well as the simple extension. By adding a new string with the simulation name to the array any new simulation in TealSim runnable in client-server mode can be added to the dialog.

The last functionality of the properties class is to act as an action listener to changes to the properties panel. It registers itself as listener to the

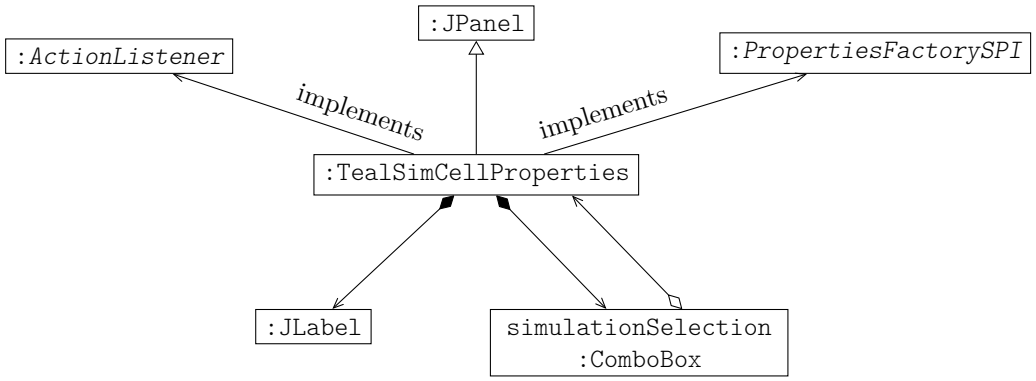

Figure 6.19: Object diagram of the properties dialog. The implemented properties class (center) implements the interface for showing a properties window in OW. It also serves as action listener to react on user interaction and as panel showing up.

drop down list. This allows the properties object to react whenever the user chooses another simulation. The properties editor will be used to change the behavior of the buttons to apply, cancel or restore the changes.

6.4.2 Creating a Simulation

As shown in Figure 6.20 the process of loading the simulation is rather complicated. The according components of TealSim need to be informed about the other main elements. Most of them hold references to each of the others. The `SimPlayerApp` class represents the entire user interface in the desktop version. First it creates the framework, in this case the so-called "player". Subsequently, the simulation is created. This happens whenever the user chooses a simulation. It is then given by a string which directly corresponds to the package and class name of the simulation. This way the simulation object can be loaded with Java's reflection functionality. The created simulation contains all the elements and parameters needed. It is passed to the player's initialization. For the OW version both, the client and the server side will need their own player. In a first step the player creates the engine. The needed engine type is obtained from the simulation object. The engine can then be instantiated as described in Section 6.3.2. The last main component needed is the viewer. With the desktop version the proper scene factory class instantiates the viewer. On the client side a client-specific viewer is created directly by the player. Since the server does not display anything all the steps including the viewer are skipped on OW's server side. The viewer is then added to the engine. This is necessary with the desktop version and

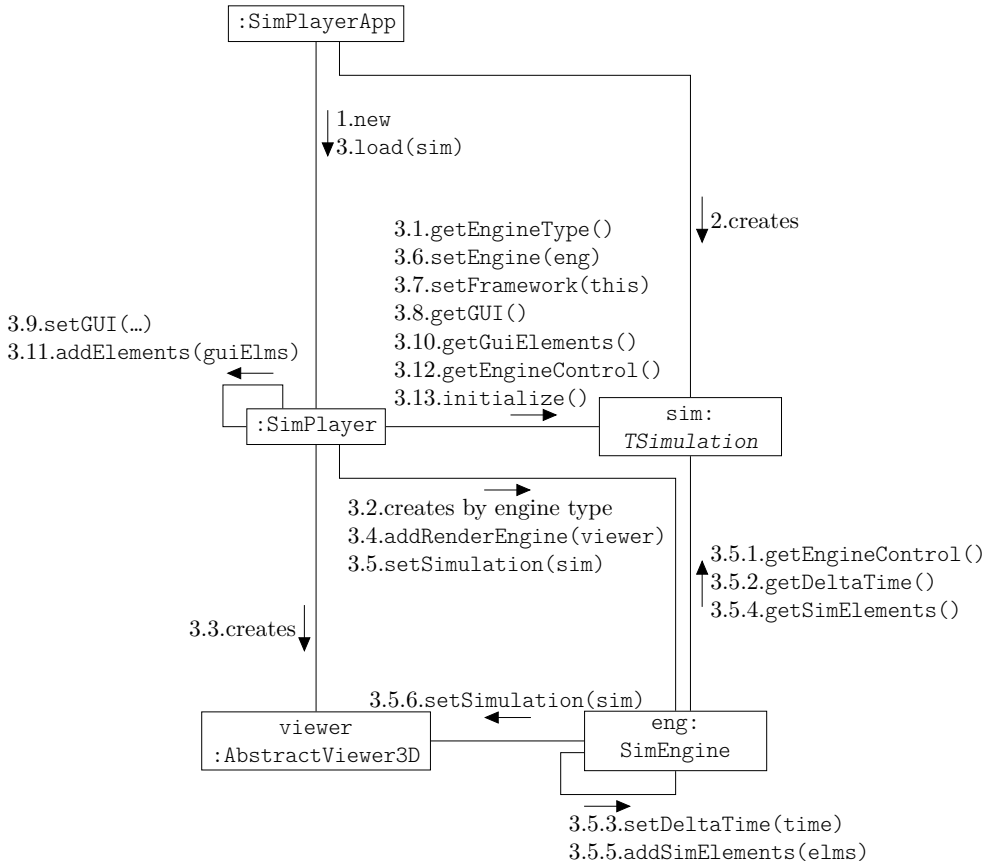

Figure 6.20: Communication diagram of the creation of a Simulation with TealSim's Desktop version (slightly simplified)

the OW module's client side. There is the possibility of adding more than one viewer to the engine. However, this is not completely implemented in TealSim yet.

The next step is to hand a reference of the simulation to the engine. This allows the engine to obtain all the needed values for the simulation step. This includes e.g. the time a simulation step should require. Most importantly the engine obtains all the simulation objects which are needed for the simulation. The engine can then hand the simulation to the viewer which grabs all drawable 3D-objects from the simulation. Since there is no viewer on the server side, this step is skipped there. Passing the simulation to the viewer via engine rather than directly adds a lot of unnecessarily complicated code. This will have to be changed in TealSim in the future.

After the engine and the viewer are aware of the simulation a reference *to* to engine as well as to the player needs to be established in the simulation.

This is necessary because some simulations are not only containers for all the elements, but do also contain functionality influencing the actual simulation. On the desktop version the player then obtains a GUI object which specifies how the user interface looks like. Figure 6.21 shows how the frames are

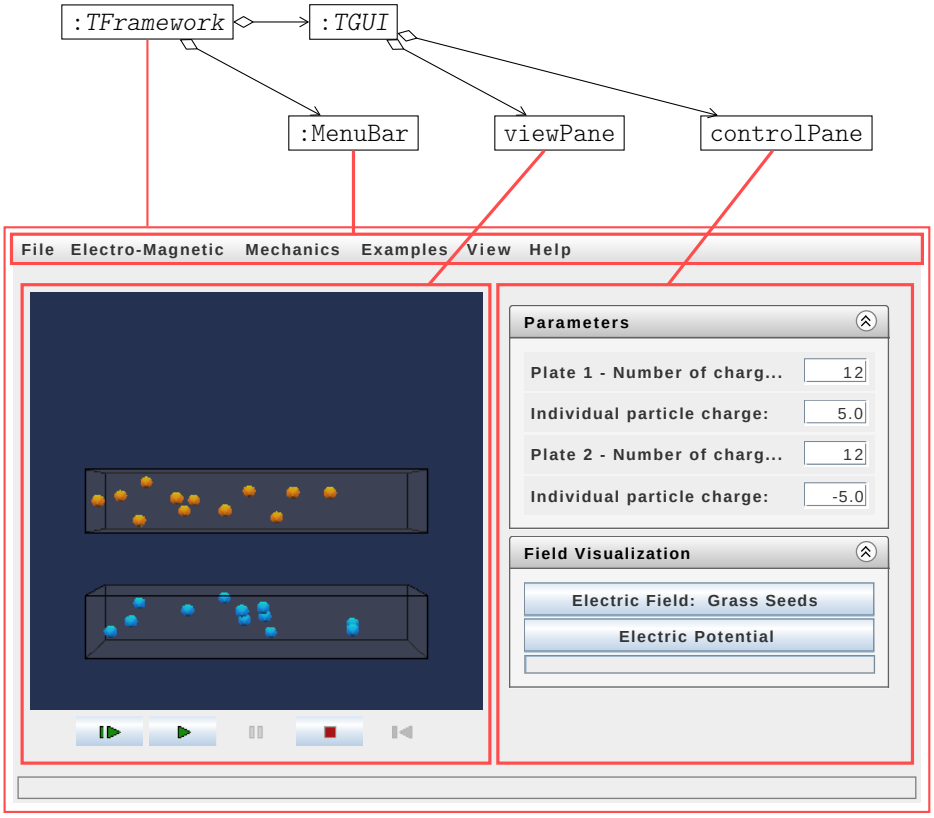

Figure 6.21: The elements in TealSim's user interface with their object hierarchy and there place of display (incomplete). The player of class interface *TFramework* holds TealSim's GUI object (interface *TGUI*). Their descendents in terms of the Java swing object hierarchy are arranged to form the user interface.

arranged with this GUI in the desktop version of TealSim. To the left there is the view pane displaying the three dimensional output and the engine control buttons. The control pane on the right contains elements the user needs to influence the simulation. This control pane is wrapped by a scroll pane which is not shown in the Figure. It is needed whenever the simulation needs too many elements to be shown in the control pane at once. All of the elements in the Figure are of Java sing classes and are nested as shown. The GUI object is responsible for placing its underlying elements. By implementing the *TGUI*

interface defined in TealSim a new arrangement can simply be accomplished and is done within the OW module. This newly implemented class does not have to show the 3D-window because the 3D simulation elements are shown directly and rendered by OW. Thus, the view pane, originally holding the engine control elements as well, is replaced by an engine-control pane put beneath the control pane.

With the next simulation initialization step the elements to be shown in the GUI are obtained from the simulation object by the player. Subsequently, they are added to TealSim's `GUI` object. The engine control then needs to be set to its initial state. On the desktop version this implicitly starts the engine thread and thus the main simulation loop. However, there are no calculations made yet. Additionally to starting and stopping the engine, the engine control does also represent the panel with the "play", "resume", "pause" buttons and so forth. On OW's server side no engine control is needed since the engine's states are set mainly by the user interactions on the client side. Whenever an according message is retrieved from the client the engine's state is changed directly. On the client side an engine control is used similarly to the desktop version. However, it does not start the engine thread on initialization. It has additional responsibilities as sending informations about user clicks on its buttons to the server. The enabling and disabling of the buttons are synchronized by the client's engine control (see Section 6.4.4). Because the functionality is very similar but extended to the desktop versions engine control the client-side engine control will be derived directly from TealSim's desktop version.

The last step of the simulation loading is to initialize the simulation by calling a corresponding method. Some elements need the engine to be instantiated to be initialized properly. Previously, the engine was created directly within the simulation's constructor. This implementation could not handle different engines on the client side and on the server side. For flexibility reasons the engine was late-bound to the simulation. With this solution the simulation only indicates what type of engine is needed. Elements who need the engine to be already created can now use the simulation's initialization method to perform their engine-specific initialization. This step needs some changes in TealSim's simulation definition, which is not desirable. However, the changes are minor and were thus permitted.

Instantiating a simulation can be time consuming. The main components as well as all simulation objects and swing components need to be created. Three possible ways for instantiating the whole simulation can be found for the OW module:

1. The simulation can be created on each client when it logs on. This

leads, however, to immediate synchronization problems since many simulations contain e.g. code with randomized initial values. The clients must therefore synchronize the simulation objects after creation. Since there are many different simulations, decisions would have to be made which client overrules the others. It seems to make more sense to create only one simulation object.

2. If the simulation is instantiated on the server the randomized parts of a simulation do not raise a synchronization problem. Once the simulation is created it can be distributed to the clients. Both, the server player and the client player can then do the initialization in their own way. PD already provides functionality to transfer an initial state whenever a client logs on. This functionality could directly be used to transfer the simulation to the client. The main problem with this approach is the long time it takes to create all the objects. Some tests made clear that for some simulations PD is not able to create the objects within the time a task can run. Increasing this time on the OW server would break the requirement of being able to run the module on every OW server without code changes.

3. To overcome the problem with PD the simulation can be instantiated on a client and then be sent to the server. The server then distributes the simulation to all the clients. Delegating the creation to one particular client is not possible because this client possibly disconnects before sending the simulation to the server. For that reason the server tells all the clients to create the simulation as long as it does not receive a simulation from a client. The first simulation received is then used and sent to all clients.

In Figure 6.22 the instantiation process of a simulation is shown. When the second client logs on the first one has not yet instantiated the simulation. For that reason the server tells the second client to create the simulation as well. In a similar case, namely when the simulation is changed while more than one client is logged on, the server will tell all the clients to create a simulation. When the server has received the simulation for the first time from a client it omits simulations received subsequently. The first simulation is distributed to all clients. Whenever a new client logs on the current, it receives the current copy. Then, the client's player can initialize the simulation as described previously.

The described behavior requires some extra code and makes the module slightly more complicated. However, for previously mentioned reasons some simulations could not be created purely on the server. In future versions more

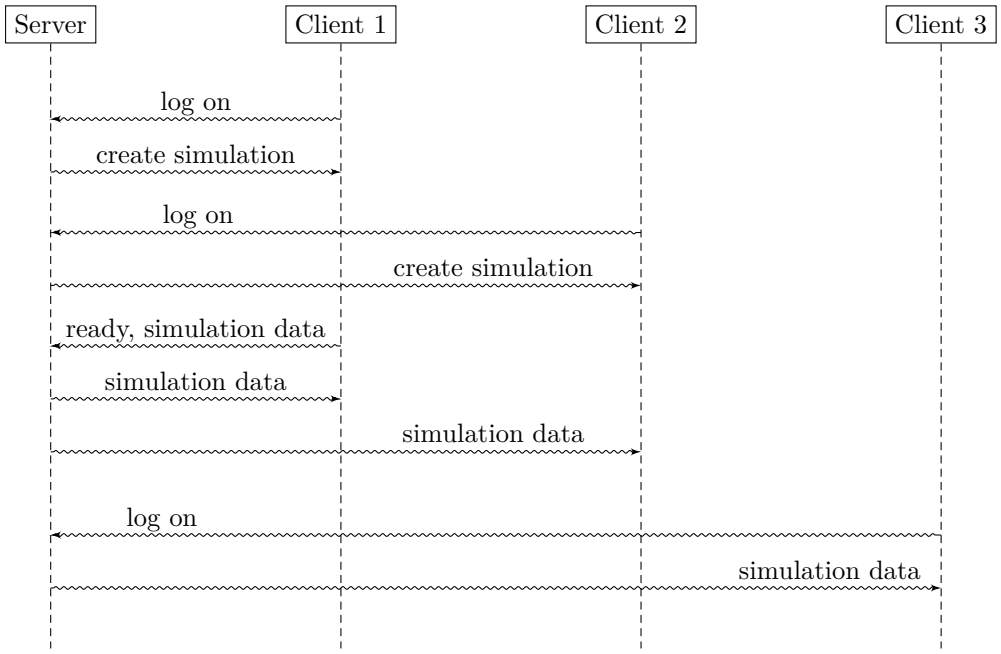

Figure 6.22: Sequence diagram of simulation instantiation and distribution. As soon as the server has got a simulation from a client, it does send it to all other clients.

of the creation code probably happens on the client (see also Chapter 9). In this case the module is already prepared and there would be less changes to the code.

One additional issue with the client-server version is the creation of object IDs. Such identification numbers will be used for synchronization. For that reason *all* the elements to be synchronized must have an unique identifier. Such elements are marked by an interface in TealSim. Classes not defined in TealSim should never be used within the simulation. Those would not implement the interface and, hence, no unique ID could be given to the elements. Java provides the creation of Universally Unique Identifiers (UUIDs) which is used for ID creation.

After the server has initialized the simulation all the references to the server-specific elements have to be removed. Otherwise the serialization process would also pack these elements and send them to the clients. In the first place the simulation object is affected. It is used as an action listener by some simulations, which add some execution code within the listener method. This code often has to be executed only once, i.e. on the server side. Doing otherwise may have strange effects and break the synchronization among the clients. Thus, all elements having the simulation object as action listener

will have to remove this reference. This occurs particularly often with swing objects. On the client side an other action listener can be added instead of the simulation. This allows the client to detect actions to be performed on the server side.

6.4.3 The Control Panel

Most of the user interaction in TealSim happens with the swing user interface. Parameters within a simulation can be changed, the view can be changed (e.g. whether field lines show up or not) or the simulation can be started or stopped using this interface. OW provides functionality as a module called "AppBase" for that purpose. It can render light-weight swing components onto a two-dimensional pane within the virtual world. User interactions are also possible as simply as with swing programming. However there is no synchronizing among clients implemented in the AppBase module. This would break the modularity of OW since synchronization should always be customized with knowledge of the actual application and therefore has to be implemented specifically.

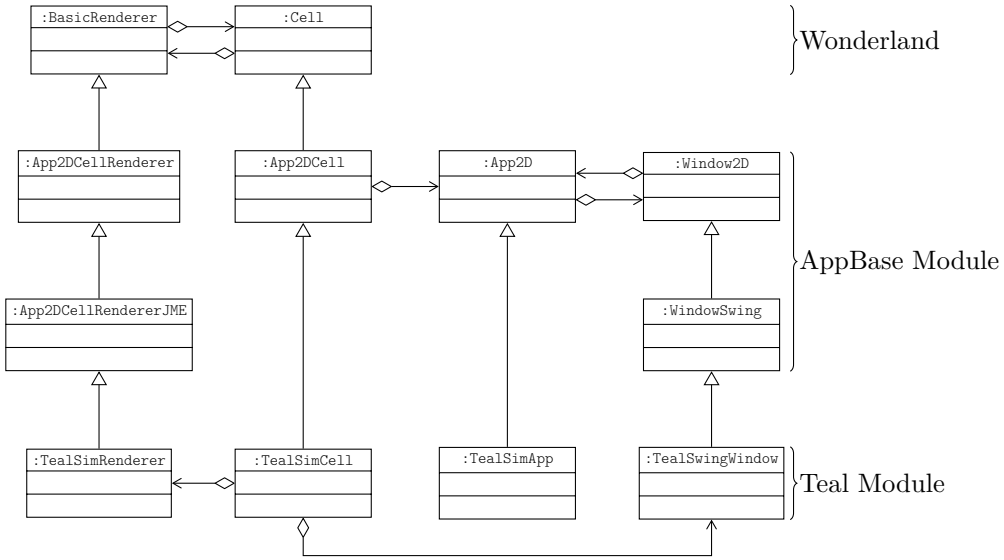

Figure 6.23: Class diagram of module's client-side swing components (simplified)

Figure 6.23 shows the class hierarchy of a cell using the *AppBase* module. In order to define any ordinary cell OW's cell base class has to be extended. The cell represents an object within the virtual world with a reference to a cell renderer. The AppBase module can be placed between the base OW cell

implementation and the TealSim cell, i.e., TealSim's classes are derived from the already extended AppBase classes rather than from OW's base classes. The additional `App2D` class is extended in order to set the pixel scale of the swing panel and to specify the control behavior of the window. The latter is set to allow multiple users to use the panel concurrently. The `WindowSwing` class is extended to specify the actual light-weight swing panel to be shown. The cell maintains a reference to this object and will set TealSim's client simulation player as panel to be displayed. The width and height of the panel are also specified here. Usually, modules using the AppBase module do not need to inherit its renderer. However, the TealSim module does need the renderer to render not only the 2D panel with the swing panel, but also the 3D content of the simulation. Thus, the renderer had to be extended. An additional scene-graph branch is added to show the additional content.

6.4.4 Starting the Simulation and synchronizing Engine States

With the previously explained functionality the cell is capable of showing a simulation with the 3D objects and the swing elements on the OW clients. The next step is to actually run a simulation which requires a lot of synchronizing among the clients and the server.

Firstly, the buttons at the engine control have to be synchronized. They can be either enabled, i.e. clickable, or disabled. When a simulation shows up some buttons are already disabled. The pause button, e.g., cannot be pressed when the simulation is not started yet. However, as soon as the user clicks on "play" the pause button has to be enabled. In order to synchronize the buttons' ability to be clicked this information has to be sent to the server and, subsequently, to all other clients whenever a click event occurs on the engine control's buttons. The server should not need any information regarding enabled or disabled buttons. However, it has to do the communication among the client, and thus, some server-side coding will be necessary. Fortunately, an OW module exists to maintain states among the clients without adding additional server-side code, the *SharedState* module. It encapsulates the server-side code. In order to used it, the building setup needs to be modified as stated previously. The SharedState module is implemented as a so-called "component". OW components can be added to a cell, providing additional functionality. Theoretically, every component can be added to every cell. The user can assign components to cells in-world using OW's user interface. In case of the TealSim cell the shared state component has to be added to the cell in any case. This can be achieved by a simple declaration in the

server-side cell code.

With the SharedState module a map is provided. It maps strings to shared objects. The latter can be booleans, integers or strings. For the engine control's buttons, one entry in this map is used. A shared integer representing the buttons' state as bit mask is the object the chosen key maps to. The SharedState module also provides functionality to add state-change listeners. In the TealSim module such a listener is used by the clients to notice changes to the enabled-state of the buttons on an other client. Newly logged in clients need to obtain the state shortly after login from the map itself.

Clicking a button at the client's engine control user interface must not only affect the button enabled mask, but also the state of the engine. If the user clicks on the "step" button one simulation step should be executed. For that purpose the call is forwarded to the client engine which fires a property change event. The client cell, capable of communicating with the server, is a property change listener to the engine and sends the message about the pressed button to the server. An according message class handleable by PD has to be implemented. For that purpose OW's `CellMessage` base class is inherited. The class can hold any serializable object and will be transferred between clients and the server. In case of the engine message it holds engine state changes as well as all the other communication regarding the engine. On the server side a message receiver to such message classes is defined. For the engine messages the message receiver is a static inner class of the server-side representation of the engine. This is helpful because it is within the same scope as the engine itself and has therefore access to the engine's class members. The receiver is derived from a base class provided by OW. Thus, it can make use of OW's messaging mechanisms, including registering a receiver object. A method of this object will be called whenever an according message is received. To enable the functionality the object has to be registered with the according message class. Whenever an engine message with an engine state changed is received a handler method in the server-side engine is called with the new state as parameter. Since the message receiver will reside in an managed object internally used by OW, it should not contain ordinary references to any objects held by other managed objects including the engine. Because PD serializes managed objects as a whole, an object referenced by two different managed objects would be re-serialized in both managed object, leading to two different instances of the object. To hold references to other managed objects PD provides a so-called "managed reference" pointer. The engine message receiver will use such a reference to maintain a reference to the managed object surrounding the server-side engine.

Whenever the engine is informed of a state change, it will have to react

accordingly. It stops the simulation if it is already running. A PD task can be used to execute a simulation step. PD comes with a scheduler. A task class will have to implement an according interface declaring a call-back method which the scheduler will call on task execution. In the TealSim module an inner class of the engine serves as task class. It is created by the engine itself and will call a `doStep` method on the engine. This will trigger an execution step. If the user clicks on the "step" button, the task will only be scheduled once. PD also provides the functionality of scheduling a task repeatedly. This is used when the user clicks on "run". The same task is now executed until the user clicks on any other button. The repeatedly scheduled task can be stopped with a handle returned by PD when on scheduling.

After every calculation step the clients will have to be informed of the changed simulation objects. As mentioned in Section 6.3.2 this synchronization will be performed by sending the dependent values to each client. These values, held by the simulation objects, can be obtained from the engine which caches the values after each simulation step for performance reasons. In order to transmit the array the engine message class is reused. The dependent values are stored as a single array of type `double`. The order of the data is only defined by the order of the simulation elements in the data vector holding the simulation elements in the engine. In order to guarantee the same order on all clients, the order has to be initially transferred to the clients together with the simulation. With the same order of dependent values, the clients can now update their simulation elements and perform the final doRefresh engine step.

The user can also reset a simulation which reinitializes the simulation objects. This step is implemented within the simulation classes. When the user clicks on the "reset" button, an engine message with the new state will be delivered to the server. As soon as the engine gets the information, the simulation will be stopped and the according method in the simulation class can be called. Subsequently, the dependent values are retrieved and stent to the clients in order to synchronized all participants.

6.4.5 Synchronizing the Swing User Interface

Similarly to the simulation elements some of the swing elements have to be synchronized among the clients as well. This is a rather complicated task since those elements are heterogeneous. Elements not influencing the server engine can be synchronized using the SharedState module. However, some of them are connected with TealSim's "routes" to simulation elements. The server needs to be informed about everything influencing the simulation. On the other hand, some of the swing elements do not have to be synchronized

among clients at all. Those are e.g. check boxes indicating whether field lines are shown or not. Every client can select on its own whether the field lines are shown or not.

Visualization Control

TealSim's visualization control class is responsible for any visualization of fields. It is a swing panel and added to the user interface in most of the electromagnetic simulations. In Figure 6.24 the panel is shown. The simulation

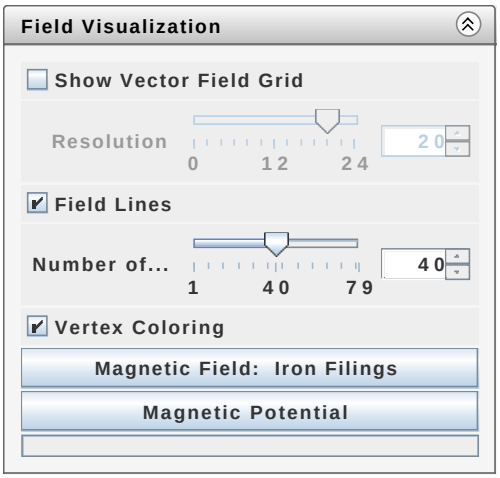

Figure 6.24: The visualization control within TealSim's GUI. It contains options to show client-specific visual data as well as buttons for showing analytical data.

defines which of the elements are shown to the user. Those not needed are set to be invisible. The first three entries concern the field direction grid and the field lines. The user can choose whether those are shown and how many of them are displayed. For the field lines the vertex coloring can be enabled showing different colors for different strengths of the field along a field line. All those elements do not need to be synchronized since every user should be able to set this options individually. The mechanisms to influence the view (e.g. for hiding the field lines when the user disables the check box) on the client side work exactly the same way as with the desktop version of TealSim. Therefore they do not need to be adapted for the OW version at all.

Whenever any of the buttons below is clicked the simulation is paused and the requested representation of the field is calculated. During the calculation the progress bar in the swing window indicates the current progress. When the calculation is completed a 2D panel as shown in Figure 6.25 with

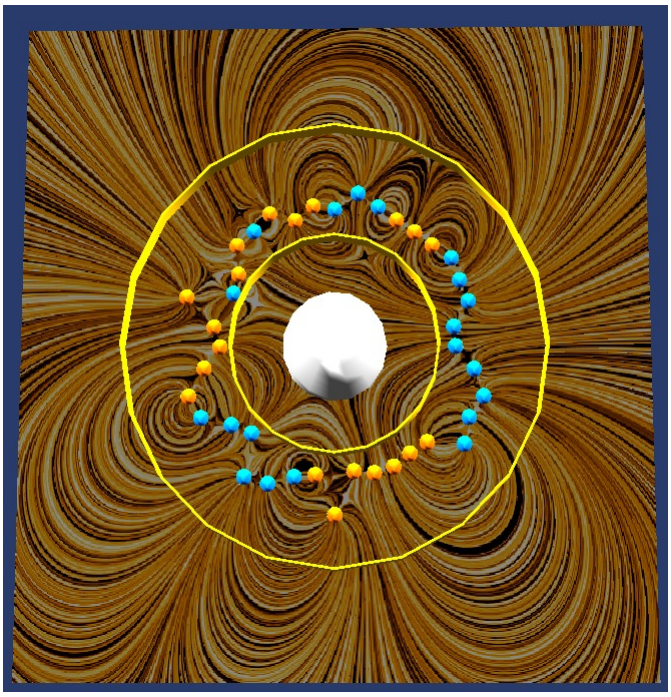

Figure 6.25: Display of electric potential in the “Charge by Induction” simulation

the requested visual representation is shown. Such user interaction requires synchronization among the clients and also with the server. The server is responsible for pausing the running simulation. The other clients need to know they should calculate and display the requested view. With the desktop version the engine's state is changed after a click on one of the buttons. In the OW version the client-side engine object replaces the desktop-version engine. The client-side cell class is registered as a property change listener to the client-side engine listening for a change of the simulation state. With this mechanism the server is informed whenever the engine state should change. This causes the server to stop the simulation when the user clicks on one of the buttons on the visualization control.

In order to inform the other clients that they should show up the panel with the field representation the SharedState component is used. For every visualization control object an entry is put into the shared map. The value is an integer with the flag indicating what button was pressed. The integer keys for the buttons were already used in the swing interface of the desktop version and were thus already defined in TealSim. On every client a listener class is registered to the map entry. This class is also used for every additional functionality regarding button clicks on a visualization control. The user

clicks are emulated on the other clients by calling the visualization control's listener call-back method with the same parameter object as it would be called on the event of a user click. This parameter is an action class defined in TealSim.

The last issue with the synchronization of the visualization control is to get the client side of the module to know whether the user has clicked a button. For that purpose the code in the visualization control class is changed slightly. As stated above an action is triggered on every user click. The visualization control object itself listens to all those actions. Its code is adapted to store a list of all the actions where the visualization control is registered as action listener. Additionally, a new method to add an external listener to all of those actions is introduced. The external listener is declared within the OW module and can now recognize all relevant clicks on the visualization control.

Routes to simulation elements

Routes are a concept for wiring values of simulation elements together. In most cases a swing user interface element is coupled with a value of a physical simulation element. The value of a slider, e.g., can directly change the charge value of a point charge. When defining a simulation such a route can simply be added to any simulation element. A method for doing this is declared in an interface implemented by almost all of TealSim's elements. Routes can also be removed from an object during execution. They are represented by classes implementing the property change listener interface. This way they can be added directly as a listener to any swing component. Whenever an event occurs the routes forward the call to the target object.

With the desktop version of TealSim the according target method of the simulation element can be called directly by the route object. With a route from a slider to a point charge, e.g., this happens whenever the user moves the slider. The changed values of the point charge are recognized by the engine when the next engine step is performed, and thus, influence the simulation even while the simulation is running. With the OW version of TealSim the slider values will have to be transferred to the server because the charges effect the server-side engine calculations. The sliders also need to be synchronized among the clients. This is done by sending the values back to the clients. The server also has to store the new values in order to be able to send them to newly logged on clients. The slider values of the simulation stored on the server side must be kept up to date. After creation of the simulation the server only keeps a copy of the simulation in order to distribute it to new clients.

In order to detect user interactions on swing components a property change listener is introduced to the client side of the OW module. It is an inner class of the client-side player. On initialization of the simulation it is instantiated and registered to all needed swing components. If a processed swing component has subcomponents they are also parsed recursively. Whenever a user clicks on a GUI component or changes any value a call-back method of the property change listener will automatically be called. with the fired event. In this case an event container is constructed and packed into a newly defined event message object which will later be sent to the server. The event container contains the property change event and the ID of the source object which will be a simulation element. The ID will replace the object since it does not make sense to transfer the client-side object to the server. On the server side the ID will be translated to the actual object holding the id. With a property-change call on the server-side representation of the source the client-side event is emulated and triggers the needed events on the server. Subsequently, the message previously received by the server is sent to all the clients. They process the message in a similar way as the server does. The identifier will be resolved to an object and the property change event will be applied to this object.

A problem with this approach is that the network between client and server cannot be synchronized. If several property changes happen with the same object within a short time their order can mix up. This is very likely to happen with the sliders. When a user drags a slider the value changes frequently. This will possibly lead to a feedback loop. In this case one value will be transmitted to the server and then sent back to the clients. In the mean time the client value has changed because the user has dragged the slider further. While the new value is about to reach the server the old one comes back to the client. The old value is applied again causing the slider to jump back to the old value. Every value change triggers a new sending event. This is usually stopped by the server since it does neither process nor forward a message to the client if the property value has not changed. However, this mechanism does not work when the values change quickly since the property value will alternate between two or more values. This causes a slider to continuously jump between different values. The effect is worse if two different clients change the same GUI element. To reduce the effect the frequency of event messages triggered by a single object is limited. The less frequently a client is allowed to send such data to the server the more unlikely are the synchronization problems to occur. However, if the network speed to a user is lower and therefore the latency is higher the problem still occurs. The transmission frequency of the synchronization messages cannot be made arbitrarily long since the user interaction would be delayed too long.

Additional synchronization mechanisms are still to be implemented.

6.4.6 Performance improvements on the server side

The server-side part of the module must be implemented with respect to performance issued. PD requires all server-side objects to be split into logical units, so-called "managed objects". A managed object is serialized and de-serialized as a whole. If it is to heavy the serialization process takes a long time and performance suffers. Thus, the server side of the TealSim module was split into managed object as shown in Figure 6.26. Running a

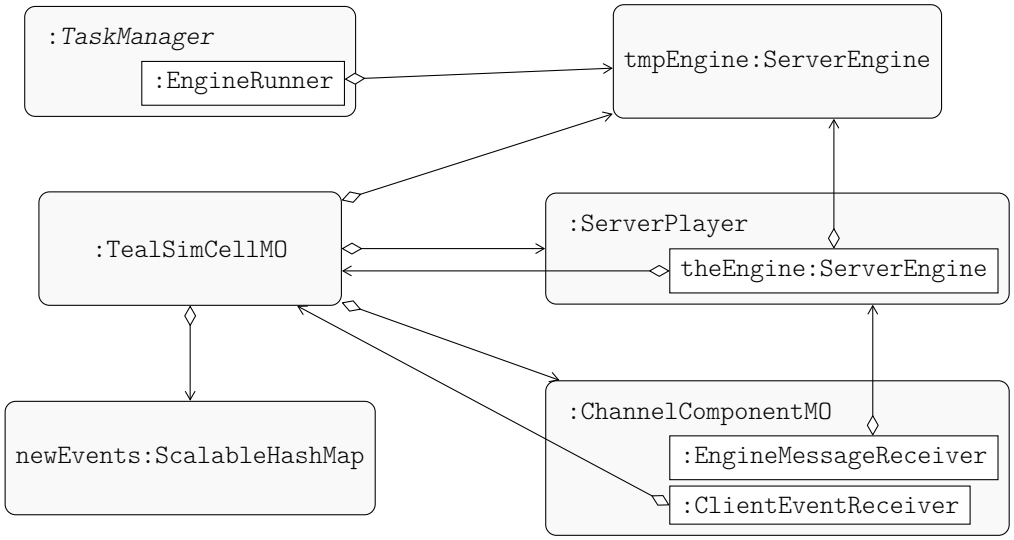

Figure 6.26: The server-side managed objects with references. The arrows indicate managed references. Blocks inside other blocks are referenced as members by the surrounding object.

simulation needs special treatment, since it requires a lot of resources. The simplest implementation would have the server-side player holding all the elements needed for the simulation. This includes the engine with its simulation- and swing elements and the simulation object. Hence, the server-side simulation step would work similarly as with the desktop version. A message from the client would trigger the engine to start the simulation. Whenever a user interacts with the swing user interface the changed value is transmitted to the server where the server-side swing element is updated. This update would trigger changes in the simulation elements through routes automatically. Whenever a new client logs on the server can transmit the current state to it.

However, this simple approach does not work since the player holding all other TealSim elements is much to big to be serialized during a PD task. Additionally, the server-side swing elements should not be updated while the server is highly utilized for the same reason. The implemented solution to this problem uses an additional managed object containing only a temporary engine with its simulation elements. This engine only exists while the simulation is running. It can be created by using PD's serialization behavior; the permanent engine, held by the player, places a reference to itself into another managed object. When this managed object is re-serialized it contains a deep-copy of the permanent engine holding only elements needed for the simulation. An engine runner implements PD's `Task` interface and runs the temporary engine to which it maintains a managed reference. During the simulation user interactions will not update the swing components of the simulation. Instead, only the simulation elements the corresponding routes point to are updated. The changes to the swing elements are stored within a map, enabling the cell to update the GUI when the simulation has stopped. If a single swing element is updated more than once, only the last change needs to be stored. An event receiver listening for such client events is implemented as an inner class of the cell. This allows it to access the event update list, the temporary engine and the player through the cell. When a client logs on during a running simulation the simulation object prior to execution is sent together with all updates. The client can then recover the current state. When the execution is stopped the update list has to be applied to the swing elements. Additionally the permanent engine has to be updated. For that purpose the dependent values of the temporary engine are retrieved and applied to the permanent engine, automatically updating the simulation elements' dependent values. The temporary engine can now be thrown away. When the engine is not running, the element update procedure is switched to the normal one.

6.5 Implementation of a Multi-player Simulation

In order to make use of the virtual world a new simulation was implemented. As stated in Chapter 3 games are a promising way to learn physics. In TealSim several games are already implemented for that reason. One of those was extended to be played by three clients at the same time. The players of the game have to communicate with each other during the game with respect to the physics. As stated in Section 5.3 OW provides several

communication channels including 3D spatial audio.

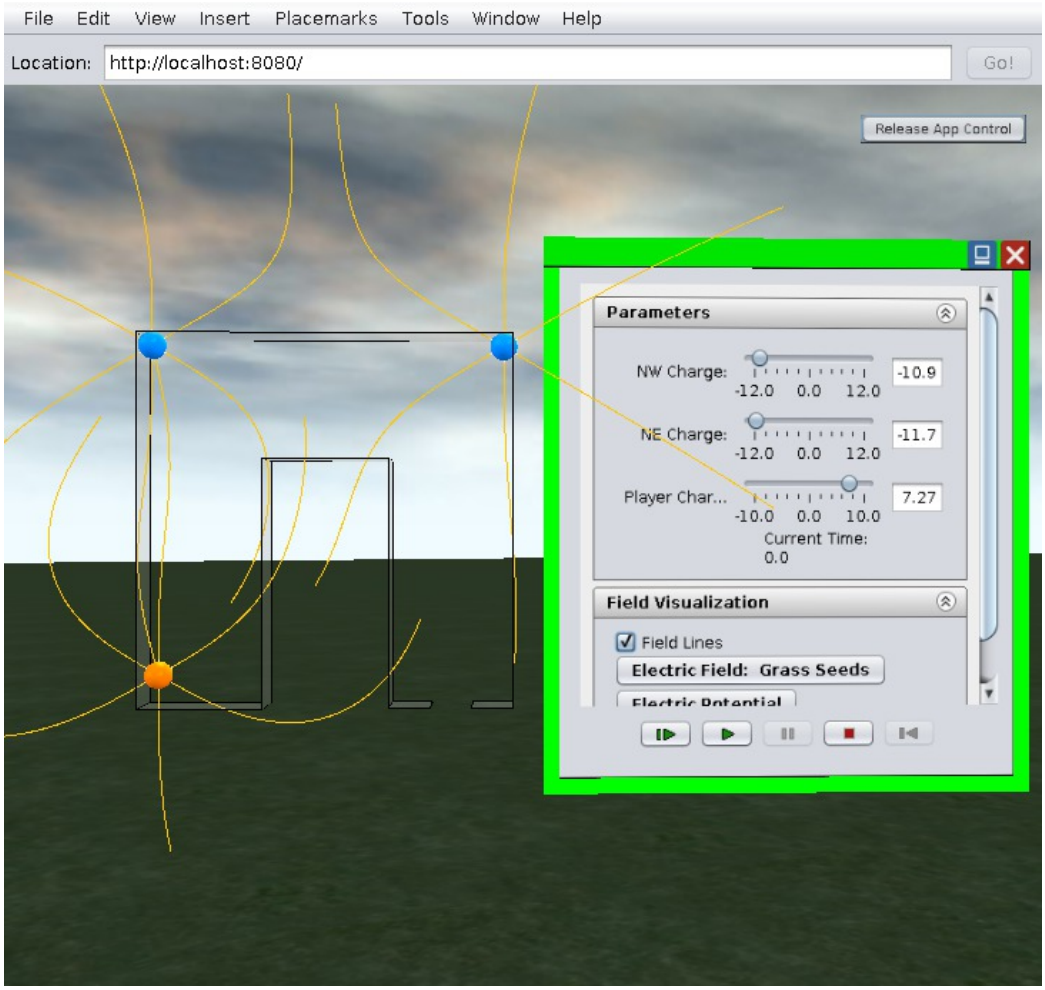

Figure 6.27: Implemented three-player video game in OW

Figure 6.27 shows the implemented game. The simulation shows three charges. Their charge values can be changed using the three sliders in the swing window. According to the charge values the three charges attract or repulse each others. The two blue charges above cannot change their position but the third charge can. The goal of the game is to navigate the movable charge through the horseshoe-shaped body until it reaches the exit on the bottom right hand side. This can only be done by changing the charge values properly while the charge is moving. The colors of the charges change with their charge value. Every user can switch on or switch of the field lines.

The simulation was added to TealSim and can therefore also be used with the desktop version. In that case it is not easy to play though, since

one user will have to change the values of the three sliders during the game. The implementation of the team game was rather simple. An other, already existent game was reused and adapted for three players. In the original game there was only one slider which was needed to change the charge value of the moving charge. The values of the other two charges remained the same. To adapt the simulation to a three player game two sliders were added. Their values were coupled with the charge values of the other charges using routes.

6.6 Summary

This Chapter has explained the major implementation steps towards running TealSim simulation in OW. This was achieved in three steps:

1. Adding JME 3D-graphics support to TealSim. This included implementing all 3D scene-graph elements used by TealSim as well as the support for external modules. With the process of enabling different graphic back-ends much previously Java3D-dependent code was refactored. This lead to a cleaner separation of the graphics back-end in TealSim. The desired back-end can be selected through an introduced factory.

2. Preparation of TealSim for use as client-server architecture. Some considerations about how the synchronization is going to happen were to be made without actually changing TealSim. The engine was split such that it is extendible by the later implemented OW module. The preparation did not just prepare TealSim for OW but also cleaned the engine's threading model. TealSim was *not* made dependent of OW with this step.

3. The OW module was implemented defining all OW-specific elements. Additionally, a mechanism to switch to other simulation was developed. This allows the OW user to put several different simulations in time into the virtual world. Although some performance tweaks needed to be implemented, there was also a focus on clean design.

In order to demonstrate the advantages of using TealSim within a virtual world a multi-player learning game was implemented. Such an implementation can be done by people with little programming skills, such as physics teachers. Theoretically, all of the simulation provided by TealSim can be used within OW. However, many simulations do not follow the implementation guidelines and need to be fixed before being runnable in OW.

The whole implementation lead to improvements concerning object-orient design in TealSim. However, due to a lack of implementation time there is a lot of room for further improvements. Mostly issues with direct influence on the performed implementation were resolved. Additional speed improvements are also possible and necessary in order to have more than a single simulation run at once.

Chapter 7

Installation and Usage of the Module

In this Chapter the implemented module is explained from the user's perspective. This covers the installation on an OW server as well as the usage afterwards. For better understanding several screen shots are added. All the explained details require a running OW server with access to the server administration.

7.1 Installation via OW's web interface

The module will be compiled using the `ant` command. The resulting Java archive file represents the module and has to be uploaded to the OW server. OW comes with a JavaEE Glassfish application server and a corresponding web interface. This interface can be accessed via any arbitrary web browser. The default port is 8080. It is however possible to configure OW's web server to run on arbitrary ports. Apart of some external links to the web page of the Open Wonderland Foundation[1], the shown home page (see Figure 7.1) contains two functional buttons; one to start the OW client as explained in Section 7.2, and one to be redirected to the administration page. This page is shown in Figure 7.2. To the left a menu to select specific pages is shown. The "Manage Modules" page provides an upload form to deploy the module to the server. A list of all installed modules is also provided on this page. The user can select any installed module to be removed. After a change with modules is done the PD server needs to be restarted. This can be done by selecting the "Manage Server" page and clicking the "restart" link beside the PD entry (see Figure 7.2, top).

[1] `http://openwonderland.org`

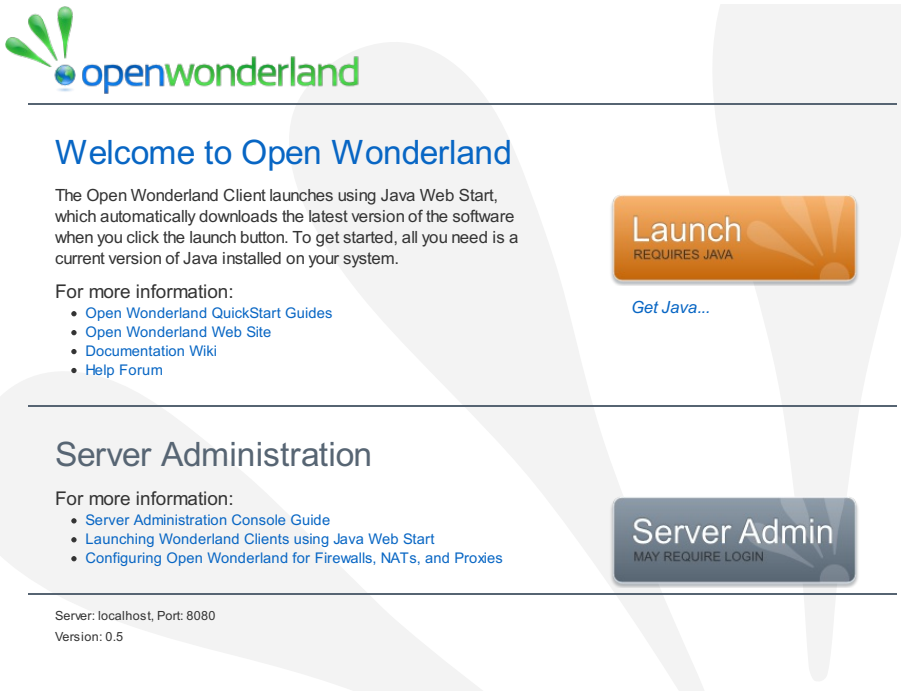

Figure 7.1: Home page of OW's web interface

7.2 Using simulations in-world

After the module is deployed it can be loaded in-world. The OW client is started with a click on the "Launch" button on the home page of the web interface. The login window appears, and a user name can be chosen. A TealSim cell can be loaded by selecting "TealSim Cell" at the dialog shown after clicking the "Insert" and "Object..." item at the task bar. Figure 7.3 shows the cell after it is loaded. The default simulation loaded is the "Capacitor". In OW TealSim looks very similar to its desktop version. On the right hand side a control panel is shown and the simulation can already be started or stopped. The main visual difference is the simulation's 3D window which is constructed directly in-world rather than within a window next to the control panel.

To prevent them from unintended clicks users will have to take control of the panel before it can be accessed. This is done by right-clicking on the panel. The menu shown at the right bottom of Figure 7.3 providing a "Take Control" item appears. This menu can also be used to open the properties of the cell. As shown in Figure 7.4 some "Capabilities" of the selected cell can be changed within the object editor showing up after clicking on "Properties...".

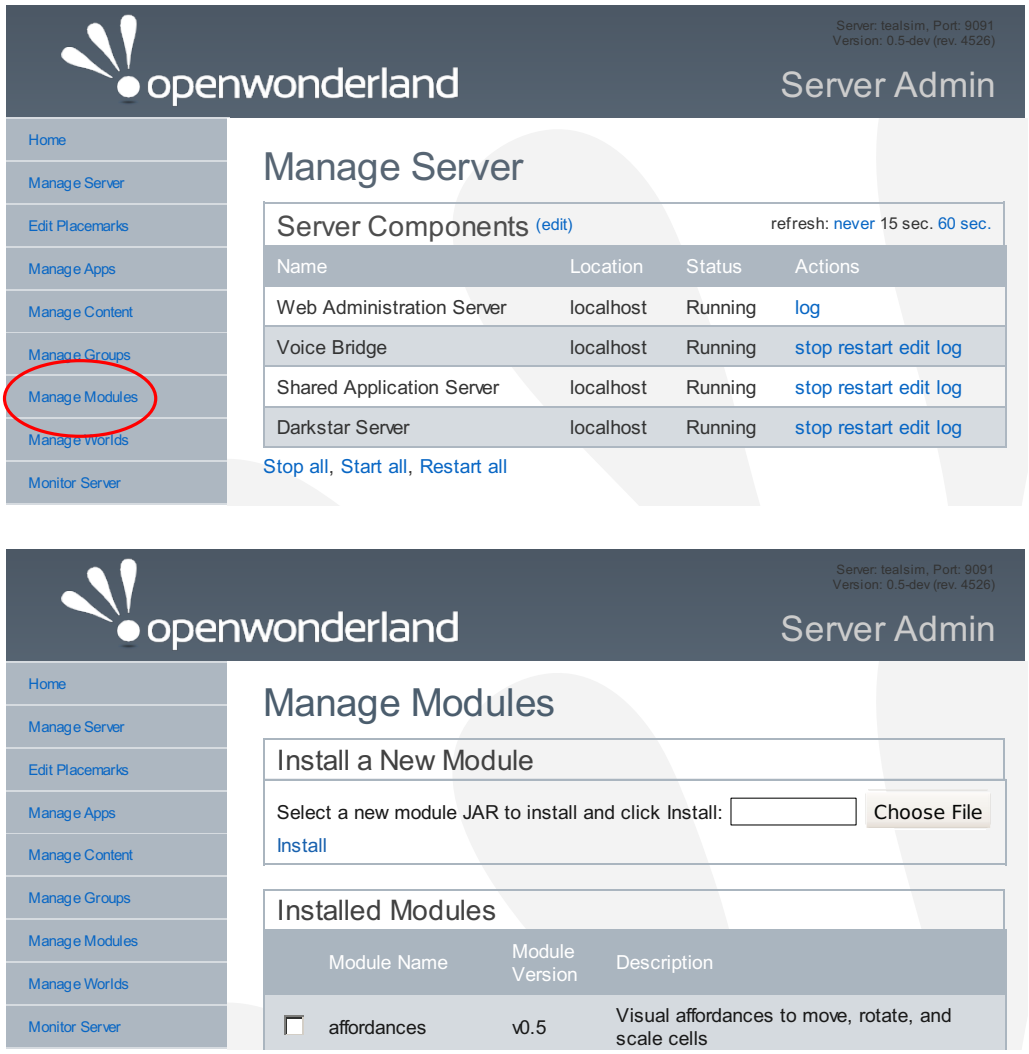

Figure 7.2: The upload process of a module to the server. The top image shows the server administration main page. With a click on the marked link the bottom page shows up showing the upload functionality.

The frame on the right hand side changes according to the selected capability. In order to switch to another simulation the "TealSim Cell" capability has to be selected. The "Properties" frame now shows a drop-down menu with all possible TealSim simulations. The desired simulation can now be changed and confirmed by pressing the "Apply" button.

If more than one simulation should be placed in-world, the user can either load a second cell in the same way, or use OW's cell duplication functionality.

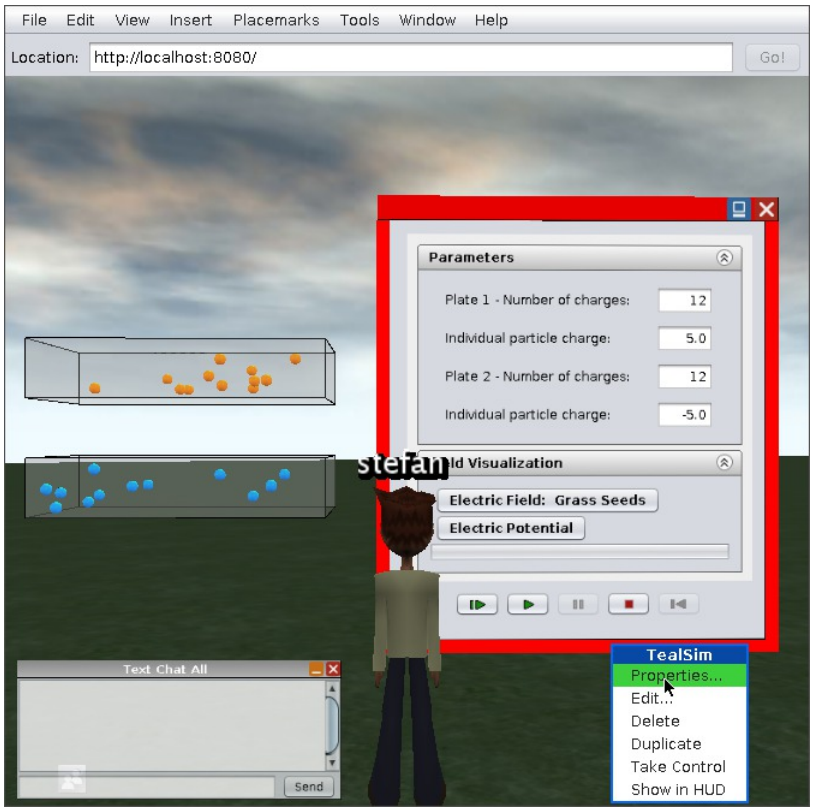

Figure 7.3: Cell after it is loaded

The latter is done by a right-click on the existing cell. The menu should then show a "Duplicate" entry. An arbitrary number of simulations can be added this way. Every single added TealSim cell can be customized with respect to size, position, and simulation type.

7.3 Summary

This Chapter gave instructions how to use the implemented module from the system administrator's and the user's perspective. Since OW provides a web interface it is fairly easy to install the module. Loading a simulation can be achieved by using the menu bar. Simulations can be handled in the same way as most of the other objects within the virtual world; they can be moved, re-sized, or duplicated intuitively. Switching to other simulations is realized with a special menu within the object properties. The tight integration within OW allows faster familiarization of the user with the module. However, since the focus of this work lied mainly on providing functionality, style features

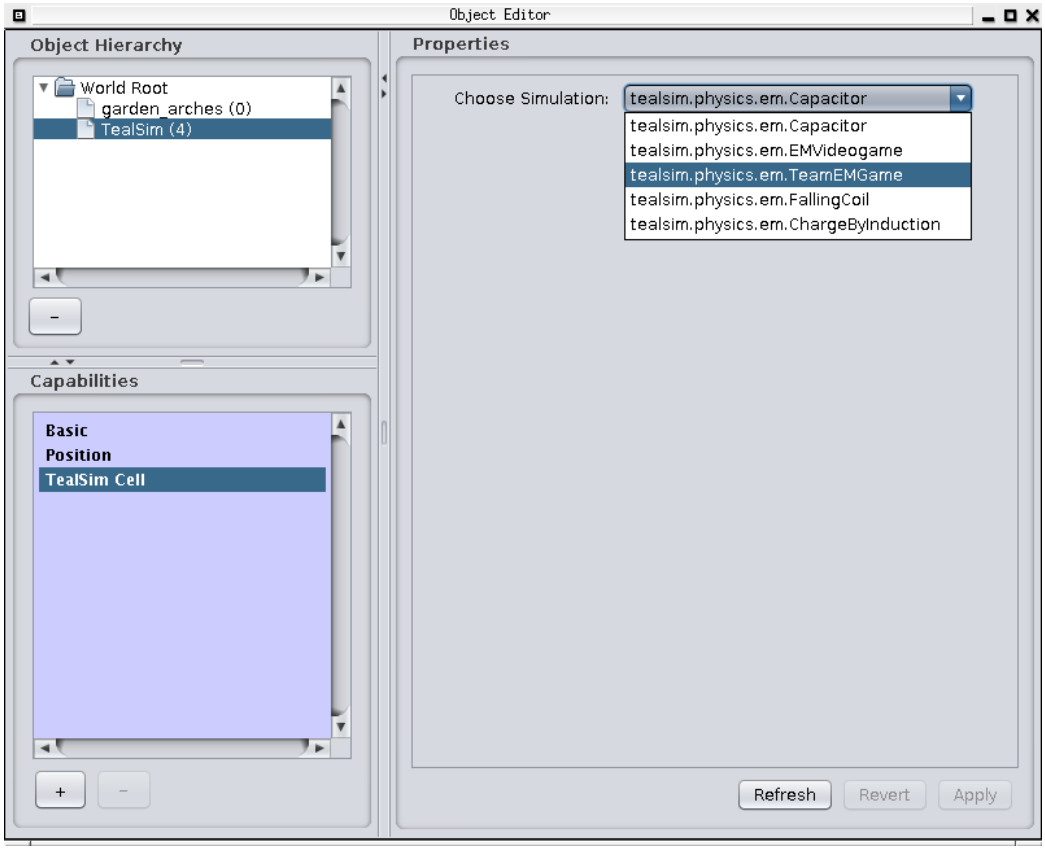

Figure 7.4: Properties window of the TealSim cell

such as fancy graphics were mostly omitted.

Chapter 8

Lessons Learned

Working at a renowned facility such as the Massachusetts Institute of Technology represented a great opportunity to the author. This Chapter summarizes my personal gains in experience during the stay at MIT together with general thoughts about the project.

Although I had previous experience with studying abroad the research at MIT opened some new perspectives. After being welcomed warmly by the CECI stuff I was helped out not only regarding my work, but also regarding every-day life. This included information about local facilities, the registration process and the structures at MIT. A desk with all the equipment I needed, most notably a desktop computer, were provided. Off-topic problems such as defective hardware were always addressed by the CECI stuff, saving me time and effort and enabling me to stay with the topic. Although I previously experienced the problems occurring with distraction, I did not expect such a difference. After all, I owe a good amount of the project's outcome to the well-organized environment.

A similar enhancement compared to previously implemented projects were achieved by the regular presence of people working on the project as well as people intending to use the software. This allows a close cooperation among these participants. Examples are

- Nicole Yankelovich, executive director of OW, who provided feedback at OW user level,
- Jonathan Kaplan, OW developer, who was often capable of giving advice on implementation problems and bugs in the OW software,
- Phil Bailey, former TealSim developer, who had many informations regarding TealSim and often discussed implementation issues with me,

- Mark Bessette who gave feedback concerning 3D-graphic design and brought in ideas regarding that field, and
- John Belcher, professor teaching physics, and thus a potential user of the system.

All these different peers have different views on the project and can therefor bring in many ideas and provide feedback from different points of view. Their regular presence helped a lot during the design and implementation phase and gave me a good idea of the importance of close cooperation.

One of the most challenging parts was getting familiar with the used frameworks. Both, TealSim and OW are fairly unstable. TealSim has been adapted by several different programmers over the past two decades. Since this leads to many inconsistencies, lack of communication among previous developers is presumed. OW is a fairly new project under rapid development. The changes with the current version including the new 3D graphic engine as well as the forced changes regarding the ownership of the project represents a challenge to the project. Getting into the matter was therefore difficult. This, however, allowed me to get a deeper understanding of many interesting technologies including computer graphics, parallelization, client-server architectures and serialization.

Furthermore, the rather short time forced me to manage my time properly. Time management was very critical. This tought me to always remember the use of efficiency mechanisms such as the 80/20 rule (see Koch, 1999). This leads to less frustration with less important issues. Preparation time to get familiar with frameworks is necessary, but a hands-on approach often shortens the familiarization time. This was especially applied to computer graphics. When building scene-graphs in JME I iteratively learned how the underlying concepts behave. It is, however, important not to drift towards a trial-and-error approach. The project was an ideal test field that will help to shape my future career.

Chapter 9

Summary and Outlook

This work presented an approach to provide a collaborative learning environment for physics students. For that purpose a virtual 3D world, namely Open Wonderland (OW), and the TealSim physics simulation framework were combined. The choices for this particular frameworks were mainly made because of their extendability, flexibility and free availability. Within the virtual world not only the simulations can be placed, but also additional learning material for course preparation and for in-world courses. TealSim is well-suited for the use in undergraduate physics lectures. Teachers with only little programming skills are able to define their own simulations or simulation games.

Technically, the approach was realized by adapting TealSim to run in OW. The TealSim framework was adapted to optionally be runnable in client-server mode. This is necessary to fully run most of the predefined simulations within the virtual world. OW prove to be flexible and user-oriented, but also to be in an early stage of development. However, it was well suited for this thesis purpose. In order to point out the advantage of the simulation software within a 3D virtual world compared to the desktop version a multi-player e-learning game was implemented. Tests showed the software's capability to manage several avatars playing at the same time. Since some stableness issues with both environments were detected a lot of effort was taken to overcome such shortcomings. The fact that both used environments are open source software helped decrease this effort slightly.

The implementation does not claim completeness in any respect. The author can suggest some major improvements which are described subsequently. The first issue occurs because the focus lied on providing the functionality. Consequently, little attention was payed to the world around the users. There is a positive social effect when groups of people meet (Callaghan, McCusker, Losada, Harkin, & Wilson, 2009) which can be increased by a virtual world

which is more similar to real-world. There is also a potential in enhancing the usability. This could e.g. be achieved by providing a three-dimensional user interface to control the simulations and games. Compared to the current implementation with the swing windows this approach seems to be closer to real world.

Another concern, which was often neglected because of the focus on functionality implementation, is the usability of the software. The following minor changes regarding this topic are recommended:

- The distance between the 2D swing window and the three dimensional simulation is currently not changeable by the user. Especially when the simulation should be placed within small 3D-spaces as buildings this functionality is essential. For better usability the mechanism to change the distance should be similar to the position change mechanisms built in OW.

- Currently, the 3D elements of the simulations are scaled to a diagonal width of three meters. This size cannot be changed without editing the Java source code. An additional dialog for changing the size of the 3D and the 2D parts separately is needed. Again, the interface for this functionality should be similar to what is already used in OW.

- The dialog for changing the simulation in-world is just a simple drop-down menu. Since well-designed user interfaces increase the acceptance and satisfaction of the users (Rodriguez, Borges, Murillo, Ortiz, & Sands, 2002) this user interface should be refactored.

In order to achieve the functionality for the first two points the cell could be split into two cells, one for the 3D components and one for the swing components. This way OW's mechanism to resize, move and scale cells could be applied to both parts separately.

One major issue with the implementation is performance. The implemented module scales up well for more than a dozen users depending on the simulation. However, some speed improvements can still be made by pulling elements from the server to the client and by optimizing the server-to-client network traffic.

Although many different simulations are supported not all of them work out of the box within the implemented OW module. Many simulations are not fulfilling the specifications. In order to use them with the module they have to be refactored. So far only electromagnetic simulations were tested. With the other simulations in TealSim a broad spectrum of different physics simulations could be used within OW.

Acronyms

OW	Open Wonderland
SL	Second Life
JME	JMonkeyEngine
QA	Quest Atlantis
MUVE	multi-user virtual environment
CSCL	computer-supported collective learning
CECL	computer-enabled collective learning
CLE	collective learning environment
MICA	Meta-Institute for Computational Astrophysics
MUD	multi-user dungeon
MMORPG	massively multi-player online role-playing game
MVC	Model-View-Controller
GUI	graphical user interface
NPC	non-player character
TEAL	Technology-Enabled Active Learning
PD	Project Darkstar
UUID	Universally Unique Identifier

List of Figures

List of Tables

Bibliography

Akilli, G. K. (2007). Games and Simulations: A New Approach in Education? In D. Gibson, C. Aldrich & M. Prensky (Editors), *Games and simulations in online learning: Research and development frameworks* (Pages 1–20). Information Science Publishing.

Barab, S., Pettyjohn, P., Gresalfi, M., Volk, C., & Solomou, M. (2012, January). Game-based curriculum and transformational play: designing to meaningfully positioning person, content, and context. *Computers & Education*, *58*(1), 518–533. doi:10.1016/j.compedu.2011.08.001

Barab, S., Thomas, M., Dodge, T., Carteaux, R., & Tuzun, H. (2005). Making learning fun: quest atlantis, a game without guns. *Educational Technology Research and Development*, *53*, 86–107. doi:10.1007/BF02504859

Bartle, R. A. (2004). *Designing virtual worlds*. New Riders Games Series. New Riders.

Bartle, R. A. (2010). From MUDs to MMORPGs: the history of virtual worlds. In J. Hunsinger, L. Klastrup & M. Allen (Editors), *International handbook of internet research* (Pages 23–39). Springer Netherlands. doi:10.1007/978-1-4020-9789-8_2

Baum, W. (1994, March). *Understanding behaviorism: Science, behavior, and culture*. Behavior analysis and society series. HarperCollins College Publishers.

Belcher, J. (2001). *Studio Physics at MIT*. MIT Physics Annual. Retrieved 22 August 2011, from http://web.mit.edu/physics/news/physicsatmit/physicsatmit_01_teal.pdf

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

Boyd, D. M., & Ellison, N. B. (2007, December). Social network sites: definition, history, and scholarship. *Journal of Computer-Mediated Communication*, *13*(1), 210–230. doi:10.1111/j.1083-6101.2007.00393.x

Burbeck, S. (1992). Applications programming in smalltalk-80(tm): How to use model-view-controller (MVC). Retrieved 13 October 2011, from http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

Callaghan, M., McCusker, K., Losada, J., Harkin, J., & Wilson, S. (2009, December). Teaching engineering education using virtual worlds and virtual learning environments. In *Advances in computing, control, telecommunication technologies, 2009. ACT '09. international conference on* (Pages 295–299). doi:10.1109/ACT.2009.80

Choi, B., & Baek, Y. (2011, December). Exploring factors of media characteristic influencing flow in learning through virtual worlds. *Computers & Education*, *57*(4), 2382 –2394. doi:10.1016/j.compedu.2011.06.019

Crook, C. (1998, April). Children as computer users: the case of collaborative learning. *Computers & Education*, *30*(3-4), 237–247. doi:10.1016/S0360-1315(97)00067-5

Curtis, P., & Nichols, D. (1994, February). MUDs grow up: social virtual reality in the real world. In *Compcon spring '94, digest of papers.* (Pages 193–200). doi:10.1109/CMPCON.1994.282924

Dalgarno, B., & Lee, M. J. W. (2010). What are the learning affordances of 3-d virtual environments? *British Journal of Educational Technology*, *41*(1), 10–32. doi:10.1111/j.1467-8535.2009.01038.x

Damer, B. (2008, July). Meeting in the ether: A brief history of virtual worlds as a medium for user-created events. *Journal of Virtual Worlds Research*, *1*(1). Retrieved 21 October 2011, from http://journals.tdl.org/jvwr/article/download/285/239

Dark, M. J., & Winstead, J. (2005). Using educational theory and moral psychology to inform the teaching of ethics in computing. In *Proceedings of the 2nd annual conference on information security curriculum development* (Pages 27–31). InfoSecCD '05. New York, NY, USA: ACM. doi:10.1145/1107622.1107630

Dempsey, J. V., Lucassen, B. A., Haynes, L. L., & Casey, M. S. (1996, April). Instructional applications of computer games. *Annual meeting of the American Educational Research Association, New York City.*

Dickey, M. D. (2005, April). Three-dimensional virtual worlds and distance learning: two case studies of active worlds as a medium for distance education. *British Journal of Educational Technology*, *36*(3), 439–451. doi:10.1111/j.1467-8535.2005.00477.x

Dieterle, E., & Clarke, J. (2007). Multi-user virtual environments for teaching and learning. In P. M. (Editor). *Encyclopedia of multimedia technology and networking*. Hershey, PA: Idea Group, Inc.

Dillenbourg, P. (1999). *Collaborative learning: cognitive and computational approaches*. Advances in learning and instruction series. Pergamon.

Dillenbourg, P., Baker, M., Blaye, A., & O'Malley, C. (1996). The evolution of research on collaborative learning. In E. Spada & P. Reiman (Editors). *Learning in humans and machine: towards an interdisciplinary learning science* (Pages 189–211). Oxford: Elsevier.

Djorgovski, S. G., Hut, P., McMillan, S., Vesperini, E., Knop, R., Farr, W., & Graham, M. J. (2010). Exploring the use of virtual worlds as a scientific research platform: the meta-institute for computational astrophysics (MICA). In F. Lehmann-Grube, J. Sablatnig, O. Akan, P. Bellavista, J. Cao, F. Dressler, …G. Coulson (Editors), *Facets of virtual environments* (Volume 33, Pages 29–43). Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg. doi:10.1007/978-3-642-11743-5_3

Dori, Y. J., & Belcher, J. (2003). Effect of visualizations and active learning on students' understanding of electromagnetism concepts. *Proceedings of the Annual Meeting of the National Association for Research in Science Teaching (NARST)*.

Dreyfus, H. L. (1991, June). *Socratic and platonic sources of cognitivism* (J. Smith, Editor). Philosophical studies series. Kluwer Academic Publishers.

Durán, M., Gallardo, S., Toral, S., Martínez-Torres, R., & Barrero, F. (2007, January). A learning methodology using matlab/simulink for undergraduate electrical engineering courses attending to learner satisfaction outcomes. *International Journal of Technology and Design Education*, *17*, 55–73. doi:10.1007/s10798-006-9007-z

van Eck, R. (2007). Building artificially intelligent learning games. In D. Gibson, C. Aldrich & M. Prensky (Editors), *Games and simulations in online learning: Research and development frameworks* (Pages 271–307). Information Science Publishing.

Egenfeldt-Nielsen, S. (2006, August). Overview of research on the educational use of video games. *Digital kompetanse*, *3*, 184–213. Retrieved 25 August 2011, from http://www.itu.dk/~sen/papers/game-overview.pdf

Farr, W., Hut, P., Ames, J., & Johnson, A. (2009, May). An experiment in using virtual worlds for scientific visualization of self-gravitating systems. *Journal of Virtual Worlds Research*, *2*(3).

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994, November). *Design patterns: Elements of reusable object-oriented software* (1st edition). Addison-Wesley.

Gokhale, A. A. (1995). Collaborative learning enhances critical thinking. *Journal of Technology Education*, *7*(1).

González-Cruz, J., Rodríguez-Sotres, R., & Rodríguez-Penagos, M. (2003). On the convenience of using a computer simulation to teach enzyme kinetics to undergraduate students with biological chemistry-related curricula. *Biochemistry and Molecular Biology Education*, *31*(2), 93–101. doi:10.1002/bmb.2003.494031020193

Greanier, T. (2000). Discover the secrets of the java serialization api. Retrieved 30 August 2011, from http://java.sun.com/developer/technicalArticles/Programming/serialization/

Gredler, M. M. (1996). Educational games and simulations: a technology in search of a (research) paradigm. In D. H. Jonassen (Editor). *Handbook of research for educational communications and technology: A project of the association for educational communications and technology* (1st edition). Macmillan Library Reference USA.

Gütl, C. (2011). The support of virtual 3d worlds for enhancing collaboration in learning settings. In F. Pozzi & D. Persico (Editors). *Techniques for fostering collaboration in online learning communities: Theoretical and practical perspectives* (Pages 278–299). doi:10.4018/978-1-61692-898-8.ch016

Gütl, C., Scheucher, B., Bailey, P. H., Belcher, J., dos Santos, F. R., & Berger, S. (2012). Towards an immersive virtual environment for physics experiments supporting collaborative settings in higher education. In A. K. M. Azad, M. E. Auer & V. J. Harward (Editors). *Internet accessible remote laboratories: Scalable e-learning tools for engineering and science disciplines* (Pages 543–562). doi:10.4018/978-1-61350-186-3.ch028

Gutwin, C., & Greenberg, S. (1999, September). The effects of workspace awareness support on the usability of real-time distributed groupware.

ACM Transactions on Computer-Human Interaction, *6*, 243–281. doi:10.1145/329693.329696

Hoadley, C. (2010, July). Roles, design, and the nature of CSCL. *Computers in Human Behavior*, *26*(4), 551–555. doi:10.1016/j.chb.2009.08.012

Hodhod, R., Cairns, P., & Kudenko, D. (2011). Innovative integrated architecture for educational games: challenges and merits. In Z. Pan, A. Cheok, W. Müller & X. Yang (Editors), *Transactions on edutainment v* (Volume 6530, Pages 1–34). Lecture Notes in Computer Science. Springer Berlin / Heidelberg. doi:10.1007/978-3-642-18452-9_1

Hut, P. (2007). Virtual laboratories and virtual worlds. *Proceedings of the International Astronomical Union*, *3*(Symposium 246), 10. doi:10.1017/S1743921308016153

Iqbal, A., Kankaanranta, M., & Neittaanmäki, P. (2010). Engaging learners through virtual worlds. *Procedia - Social and Behavioral Sciences*, *2*(2), 3198–3205. doi:10.1016/j.sbspro.2010.03.489

John-Steiner, V., & Mahn, H. (1996). Sociocultural approaches to learning and development: a vygotskian framework. *Educational Psychologist*, *31*(3), 191–206. doi:10.1207/s15326985ep3103&4_4

de Jong, T., & van Joolingen, W. R. (1998). Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, *68*(2), 179–201. doi:10.3102/00346543068002179

Juul, J. (2003). The game, the player, the world: looking for a heart of gameness. In *Conference of the digital games research association*.

Kearsley, G., & Shneiderman, B. (1999). Engagement theory: a framework for technology-based teaching and learning. Retrieved 17 October 2011, from http://home.sprynet.com/~gkearsley/engage.htm

Ketelhut, D. J., Dede, C., Clarke, J., Nelson, B., & Bowman, C. (2007). Studying situated learning in a multi-user virtual environment. *Assessment of problem solving using simulations Mahwah NJ Lawrence Erlbaum Associates*, *54*(0310188), 37–58.

Ketelhut, D. J., Nelson, B. C., Clarke, J., & Dede, C. (2010). A multi-user virtual environment for building and assessing higher order inquiry skills in science. *British Journal of Educational Technology*, *41*(1), 56–68. doi:10.1111/j.1467-8535.2009.01036.x

Koch, R. (1999). *The 80 20 principle: The secret of achieving more with less.* Crown Business.

Li, Z., & Zhao, X. (2008). The design of web-based personal collaborative learning system (WBPCLS) for computer science courses. In F. Li, J. Zhao, T. Shih, R. Lau, Q. Li & D. McLeod (Editors), *Proceedings of the 7th international conference on advances in web based learning* (Volume 5145, Pages 434–445). ICWL '08. Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-85033-5_43

Messinger, P. R., Stroulia, E., Lyons, K., Bone, M., Niu, R. H., Smirnov, K., & Perelgut, S. (2009). Virtual worlds – past, present, and future: new directions in social computing. *Decision Support Systems*, *47*(3), 204–228. Online Communities and Social Network. doi:10.1016/j.dss.2009.02.014

Miller, G. A. (2003, March). The cognitive revolution: a historical perspective. *Trends in Cognitive Sciences*, *7*(3), 141–144. doi:10.1016/S1364-6613(03)00029-9

Molenda, M., & Sullivan, M. (2003). Issues and trends in instructional technology: Treading water. In *Educational media and technology yearbook 2003* (Bd. 28;Bd. 2003, Pages 3–20). Libraries Unlimited.

Mukti, N. A., Razali, D., Ramli, M. F., Zaman, H. B., & Ahmad, A. (2005). Hybrid learning and online collaborative enhance students? performance. *Advanced Learning Technologies, IEEE International Conference on*, 481–483. doi:10.1109/ICALT.2005.166

Narayanasamy, V., Wong, K. W., Fung, C. C., & Rai, S. (2006, April). Distinguishing games and simulation games from simulators. *Computers in Entertainment*, *4*. doi:10.1145/1129006.1129021

Nelson, B., Ketelhut, D. J., Clarke, J., Bowman, C., & Dede, C. (2005). Design-based research strategies for developing a scientific inquiry curriculum in a multi-user virtual environment. *Educational Technology*, *45*(1), 21–27.

Nova, N., Wehrle, T., Goslin, J., Bourquin, Y., & Dillenbourg, P. (2007). Collaboration in a multi-user game: impacts of an awareness tool on mutual modeling. *Multimedia Tools and Applications*, *32*, 161–183. doi:10.1007/s11042-006-0065-8

Paraskeva, F., Mysirlaki, S., & Papagianni, A. (2010). Multiplayer online games as educational tools: facing new challenges in learning. *Computers & Education*, *54*(2), 498–505. doi:10.1016/j.compedu.2009.09.001

Prensky, M. (2001). "simulations": are they games? In *Digital game-based learning* (1st edition). E-Libro. McGraw-Hill.

RedDwarf application tutorial. (2010). Retrieved 1 February 2012, from http://sourceforge.net/apps/trac/reddwarf/raw-attachment/wiki/Documentation/RedDwarf%20ServerAppTutorial.odt

River City Interface. (2007). Retrieved 22 December 2011, from Harvard University: http://muve.gse.harvard.edu/rivercityproject/view/rc_views_interface.htm

Rodriguez, N., Borges, J., Murillo, V., Ortiz, J., & Sands, D. (2002). A study of physicians' interaction with text-based and graphical-based electronic patient record systems. In *Computer-based medical systems, 2002. (cbms 2002). proceedings of the 15th IEEE symposium on* (Pages 357–360). doi:10.1109/CBMS.2002.1011405

Roschelle, J., & Teasley, S. D. (1995). The construction of shared knowledge in collaborative problem solving. In C. O'Malley (Editor), *Computer-supported collaborative learning* (Pages 69–97). Berlin: Springer.

Rutten, N., van Joolingen, W. R., & van der Veen, J. T. (2012, January). The learning effects of computer simulations in science education. *Computers & Education*, *58*(1), 136–153. doi:10.1016/j.compedu.2011.07.017

Scheucher, B., Bailey, P. H., Gütl, C., & Harward, V. J. (2009). Collaborative virtual 3d environment for internet-accessible physics experiments. *International Journal of Online Engineering (iJOE)*, *5*, 65–71.

Slott, J. (2010a). Project wonderland (v0.5): importing 3d models. Retrieved 10 August 2011, from http://code.google.com/p/openwonderland/wiki/OpenWonderland

Slott, J. (2010b). Project wonderland v0.5: working with modules. Retrieved 10 August 2011, from http://code.google.com/p/openwonderland/wiki/OpenWonderland

Smith, P. L., & Ragan, T. J. (1999). *Instructional design* (2nd edition). John Wiley & Sons Inc.

Squires, D., & Preece, J. (1999). Predicting quality in educational software:: evaluating for learning, usability and the synergy between them. *Interacting with Computers*, *11*(5), 467–483. doi:10.1016/S0953-5438(98)00063-0

Stahl, G., Koschmann, T., & Suthers, D. (2006). Computer-supported collaborative learning: an historical perspective. In R. K. Sawyer (Editor). *The cambridge handbook of the learning sciences* (Pages 409–426). Cambridge Handbooks in Psychology. Cambridge University Press.

Stelzer, T., Brookes, D. T., Gladding, G., & Mestre, J. P. (2010). Impact of multimedia learning modules on an introductory course on electricity and magnetism. *Amerian Journal of Physics*, *78*(7), 755–759. doi:10.1119/1.3369920

Stevens, R., Soller, A., Giordani, A., Gerosa, L., Cooper, M., & Cox, C. (2005). Developing a framework for integrating prior problem solving and knowledge sharing histories of a group to predict future group performance. In *Collaborative computing: networking, applications and worksharing, 2005 international conference* (Page 9 pp.). doi:10.1109/COLCOM.2005.1651209

Thagard, P. (1997, June). Collaborative knowledge. *Noûs*, *31*(2), 242–261. doi:10.1111/0029-4624.00044

Thomas, M. K., Barab, S. A., & Tuzun, H. (2009). Developing critical implementations of technology-rich innovations: a cross-case study of the implementation of quest atlantis. *Journal of Educational Computing Research*, *41*, 125–153. doi:10.2190/EC.41.2.a

Twilleager, D. (2008). *MTGame programming guide*. Retrieved 25 January 2012, from http://openwonderland-mtgame.googlecode.com/svn/trunk/doc/MTGameProgGuide.pdf

Vani, V., & Mohan, S. (2010). Interactive 3d class room: a framework for web3d using J3D and JMF. In *Proceedings of the 1st amrita acm-w celebration on women in computing in india* (24:1–24:7). A2CWiC '10. New York, NY, USA: ACM. doi:10.1145/1858378.1858402

Winsberg, E. (2010). *Science in the age of computer simulation*. University of Chicago Press.

Yeo, S., Loss, R., Zadnik, M., Harrison, A., & Treagust, D. (2004). What do students really learn from interactive multimedia? a physics case study. *Amerian Journal of Physics*, *72*(10), 1351–1358. doi:10.1119/1.1748074

Young, M., Schrader, P. G., & Zheng, D. (2006). MMOGs as learning environments: an ecological journey into quest atlantis and the sims online. *Innovate: Journal of Online Education*, *2*(4). Retrieved 23 December 2011, from http://www.innovateonline.info/index.php?view=article&id=66

Young, R. A., & Collin, A. (2004). Introduction: constructivism and social constructionism in the career field. *Journal of Vocational Behavior*, *64*(3), 373–388. Special Issue on Constructivism, Social Constructionism and Career. doi:10.1016/j.jvb.2003.12.005