Patrick Weghofer, BSc

Flexible Science Fiction Prototyping-Environment in Open Wonderland

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Doz. DI Dr.techn. Christian Gütl

IICM, Graz University of Technology, Austria

Co-Supervisor
Dipl.-Ing. BSc Johanna Pirker

Graz, 12. 2014

Patrick Weghofer, BSc

Flexible Science Fiction Prototyping-Umgebung in Open Wonderland

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Informatik

eingereicht an der

Technischen Universität Graz

Betreuer

Univ.-Doz. DI Dr.techn. Christian Gütl

IICM, Graz University of Technology, Austria

Mitbetreuerin
Dipl.-Ing. BSc Johanna Pirker

Graz, 12. 2014

Abstract

Creativity is needed to keep up with fast changes in our society. Creativity is also a driving force behind envisioning how the near future may look like. There are plenty of science fiction stories describing a possible vision of the future. But what if researchers want to explore these possibilities and their implications thoroughly? In order to help understand possible future visions, science fiction prototyping can be applied. This approach builds upon science facts, creates a story around them and analyzes the advantages and disadvantages they may have. Science fiction has always been a great influence for researchers, therefore science fiction prototyping is the next step, combining creativity with the task of building a prototype, which can be analyzed.

Because science fiction prototyping is usually covered in stories, it misses immersion and interactivity. In order to help with those shortcomings, science fiction prototypes can be created with the help of virtual worlds. Virtual worlds offer highly immersive, interactive and collaborative environments. But they are also hard to get into. They offer crude implementations of standard features without usability in mind. Some virtual world platforms even miss key features, like multi-selection of objects. Therefore creating and changing a world may become very tedious. To alleviate these tasks, there are many different methods. Many of them do not need much user input and are mostly automated. But when the users have to do changes by themselves, they mostly have to use the tools offered by the virtual world platform. Also, a generated world may never look as unique as a handcrafted one. Therefore tools are also needed for helping the users make their changes by hand.

The tool described in this work is an editor like program, which is called directly in the client of the virtual world platform Open Wonderland. Its task is to help inexperienced users to create and change a world to their likings. Therefore it is kept as simple as possible and incorporates only basic functionality, which is needed for adding, changing and removing objects. The visualization of the world is also kept to a minimum. The editor features a birds-eye view of the world and objects are represented by rectangles and circles. The specifics of the implementation are explained in further detail in the associated chapter. A usability evaluation was also conducted, which favored the editor over the options Open Wonderland offered. But there is also room for improvements within the editor, which are also explained at the end of this work.

Kurzfassung

Kreativität ist heutzutage ein wichtiger Faktor um auf schnellen Änderungen in unserer Gesellschaft reagieren zu können. Kreativität ist ebenfalls eine treibende Kraft, um herausfinden zu können, wie eine mögliche Zukunft aussehen kann. Es existieren zahlreiche Science Fiction Geschichten, die eine mögliche Vision der Zukunft beschreiben. Aber was passiert, wenn Forscher diese Visionen erforschen wollen, mitsamt ihren Auswirkungen? Um Zukunftsvisionen besser verstehen zu können, kann eine Technik benutzt werden, die sich Science Fiction Prototyping nennt. Dieser Ansatz erfasst wissenschaftliche Gegebenheiten, entwickelt eine Gesichte darum und analysiert mögliche Vor- und Nachteile. Da Science Fiction schon immer ein ausschlaggebender Einfluss auf die Forschung und Entwicklung hatte, kann man Science Fiction Prototyping als den nächsten Schritt ansehen, der Kreativität mit dem Ziel verbindet einen analysierbaren Prototypen zu entwickeln.

Science Fiction Prototyping produziert im Grunde nur eine Geschichte, weshalb das gesamte Konzept relativ statisch ist. Es fehlt an Interaktivität und Immersion. Um diese Nachteile auszugleichen, könnten Science Fiction Prototypen durch Virtuelle Welten realisiert werden. Diese bieten eine sehr immersive, interaktive und auch kollaborative Umgebung. Aber sie sind auch schwer verständlich für Einsteiger. Virtuelle Welten bieten meist nur grobe Implementierungen von Standardfeatures in einer sehr benutzerunfreundlichen Umgebung an. Manche Plattformen unterstützen nicht einmal wichtige Operationen, wie eine Mehrfachauswahl. Aus diesem Grund ist es oft schwer eine Welt zu erschaffen und zu verändern. Natürlich gibt es verschiedenste Methoden, um diese Tätigkeit zu vereinfachen. Viele davon benötigen nur sehr wenige Benutzereingaben und sind größtenteils automatisiert. Aber sollten die Benutzer selbst Änderungen vornehmen wollen, so müssen sie sich wieder mit den Werkzeugen begnügen, die die Plattform der virtuellen Welt anbietet. Außerdem sieht eine automatisch generierte Welt bei weitem nicht so einzigartig aus als eine von Hand erstellte. Darum werden auch Werkzeuge benötigt, die es Benutzern ermöglichen einfache händische Änderungen zu machen.

Das Tool, das in dieser Arbeit beschrieben wird ist ein Editor Prototyp, der direkt aus dem Client der Plattform Open Wonderland gestartet werden kann. Das Ziel dieses Prototyps ist es, unerfahrene Benutzer dabei zu helfen eine Welt erschaffen und verändern zu können und zwar so einfach wie möglich. Deshalb ist der Editor auch möglichst simpel gehalten und beinhaltet hauptsächlich Funktionen, die für das Hinzufügen, Ändern und Löschen von Objekten benötigt werden. Die Visualisierung der Welt ist auch auf ein Minimum beschränkt. Die Ansicht ist aus der Vogelperspektive und Objekte werden entweder als einfache Rechtecke, oder Kreise dargestellt. Die Details der Implementierung werden im dazugehörigen Kapitel genauer erläutert. Eine Evaluierung bezüglich der Benutzerfreundlichkeit wurde ebenfalls abgehalten. Dabei wurde der Editor mit den Funktionen, die Open Wonderland anbietet, verglichen und schnitt dabei um einiges besser ab. Dennoch gibt es Bereiche für Verbesserungen, welche am Ende dieser Arbeit besprochen werden.

Statuary 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.

_____ (date)

_____ (signature)

Note of Thanks

First and foremost I would thank my advisor Christian Gütl, who was always kind enough to guide me when I had questions. His help was much appreciated and without him, I would still have troubles getting started with this work. I would also like to thank him for giving me the chance to work in such an intriguing new field like Science Fiction Prototyping.

I would also like to thank Johanna Pirker, which was also my advisor, giving me great ideas for implementation and helping me with Open Wonderland. Without her help I still would have troubles starting Open Wonderland.

A special thanks goes to all my test users, which were patient enough to endure the long usability test I put them through. Their input helped a lot in finding problems with the editor as well as in understanding usability.

Finally I thank my friends and family, especially my parents, who always helped and supported me during everything in my life and of course during the long time of my study. Without them, this thesis would not have been possible.

Table of Contents

List of Figures

List of Tables

List of Listings

1 Introduction

To react as fast as possible to new trends and technology is becoming more and more important. There even exists a field called future casting, which is used to capture possible trends of the near future. After analyzing these trends the developers try to achieve this vision and build in this direction. (Johnson, 2011) This development process also includes prototyping. (Zheng & Callaghan, 2012) Prototyping has been a very important staple of the industry. (Warfel, 2009) There are a couple of advantages in creating a prototype. The first and most important is to find problems and errors early in the development process. Another one is to introduce new ideas during development. (Hartmann, 2009) But what if the technology is not ready to incorporate the ideas, or visions that came up during the analyzing process? One possible solution is to create a *Science Fiction Prototype* (SFP). (Zheng & Callaghan, 2012) This type of prototype is a vision of the future in which one monitors how a new technology can influence certain fields, like society. These visions are depicted in short stories, comics, or videos. And all these visions are based on a science fact. SFPs are used mainly to promote discussion about the picked science fact and future developments as well as to investigate application on the social and society level. The basis of SFP is of course creativity, (Johnson, 2011) which is one of the most important human skills. (Eow, Ali, Mahmud, & Baki, 2010)

But SFPs do not offer any kind of immersion, interactivity. They also lack collaboration. Therefore an environment is needed, which not only lets someone tell a story, but rather experience one. Virtual worlds can offer such an environment. (Pirker, Gütl, Weghofer, & Feichtner, 2014) They also allow for many different possibilities in other sectors. (de Freitas, 2008) It is possible to conduct experiments in them, which are hard to realize in the real world. (Bainbridge, 2007) They are also a great platform for studies in the social sector. (Fairfield, 2012) The openness and the shared structure of virtual worlds also offer a huge potential for creativity. (Singh, 2012) But virtual worlds can be very limiting, meaning they may offer little possibilities for inexperienced users to create their own worlds and objects. (Burri, 2011) Creating a world can be a hard task, because the users have to make all their changes by hand and on a low abstraction layer. (Smelik, Tutenel, de Kraker, & Bidarra, 2011) This needs attention, because a flexible and easy to use environment is needed in virtual worlds. (Gütl, Chang, & Freudenthaler, 2014) The easiest way is to automate or at least support the building process. (Smelik, Tutenel et al., 2011)

1.1 Background & Motivation

Automatization may help with the creation of a world, but it may not be a substitute for user input. And even if the world is created automatically, the problem of making changes in a difficult to use environment still exists. After the world is created and does not suit the needs of its creators, they need to go back into the world to make their changes. In order to house a SFP in a virtual world, it needs to be easy to use as well as flexible and dynamic. The goal of this work is to make a virtual world more user-friendly for inexperienced users. To achieve this, an editor prototype was created, which helps users in building and changing a world. This prototype uses a birds-eye view of the virtual world, in order to make it easy to interpret. Another important feature is the possibility to make changes in real-time, which enables

collaboration between users. A side-effect of this feature is the possibility to check the world as it is built. Therefore users can go directly into the virtual world to experience their changes.

1.2 Structure

This work is structured in two main parts. The first is the theoretical part, which will discuss SFP in chapter 2 and virtual worlds in chapter 3. The second part will cover the implementation, starting from the concept and design (chapter 4), going over to the implementation details (chapter 5) and concluding with the handbook and the usability evaluation in chapter 6.

Chapter 2 first deals with creativity, how it is defined and how it can be improved. It also shows the importance of creativity in the business sector and shows how it has created its own sector in the industry. This is followed by a brief overview of prototyping and its techniques is given, how to classify a prototype and how the overall process is done. Next, some interesting prototyping techniques will be listed. Then prototyping and creativity are brought together in the following section, which focuses on SFPs. A brief overview on SFPs is given, as well as a description about the process itself. The chapter ends with a quick look on how *Science Fiction* (Sci-Fi) influenced science and development and related work.

Chapter 3 deals with virtual worlds, their history and definition, followed by a couple of important examples. Then an outlook of the possibilities of virtual worlds is given. This section will focus mainly on the research and the creative sector, because they are important for this work to present possibilities for SFPs. The last and longest section of this chapter will conclude with a myriad of tools and techniques to help content creators in building and changing worlds.

Chapter 4 shows the concept and design stage of the prototype. The chapter starts with the idea, why such an editor may be needed in a virtual world and its requirements. This is followed by a simple, conceptual architecture. The design section gives a quick overview of the selected virtual world platform *Open Wonderland* (OWL). It then shows the general structure of the three main parts, the Virtual World Adapter, the Data part and the Editor itself.

Chapter 5 covers the implementation in more detail. It discusses the three main and explains the more complex methods of the implementation. It will not incorporate the complete implementation, because this would go beyond the scope of this work. The chapter starts with an overview of the different parts, as well as their main interfaces. It is followed by a more detailed explanation of the OWL Adapter, the Data part and the Editor, in that order.

Chapter 6 concludes the implementation part with a short handbook, which gives insight on how to operate the editor. The second section includes a usability evaluation of both the editor and OWL featuring the most important findings. It concludes with possible improvements to the editor.

Chapter 7 collects all findings and structures it in three main parts, namely the theoretical part, the implementation and the evaluation. Chapter 8 gives a brief summary of this work and a short outlook. Appendix A contains the questionnaire used for the usability evaluation.

2 Creativity and Design

Innovation is an important part of research. To find new discoveries, two things are vital: The first key part is the urge for knowledge, which can be described as questioning anything known to discover the new and unknown. The second part is creativity, which can be seen as choosing a new direction. (Scales & Snider, 1999) Johnstone (2007) states, that "*information is the raw material of human thought*" (p. 4).

And because of the constant spread of computers, more and more information is created. New technologies provide many new possibilities, like new ways of communication, learning, even thinking and solving problems. These new technologies reflect nothing else but, the creativity of the human mind. (Sarsani, 2011) This chapter provides a short overview of creativity and its importance in the industry, as well as the topic of prototyping. After these two chapters, Science Fiction Prototyping will be discussed, which is more or less a combination of creativity and prototyping.

2.1 Creativity

Creativity is one of the most important skills, a human can possess. Its use however is often underestimated and sparsely encouraged, because the general consensus is that creativity is just another part of the normal human intelligence. There is considerable evidence, that creativity is inherent and not dynamic. But many researches believe the exact opposite. They state that creativity can not only be learned, but also increased. Unfortunately, there is no answer, which can satisfy both the static and the dynamic beliefs. (Eow et al., 2010)

2.1.1 Definition & Models

Creativity defies definition, but this circumstance has not kept researchers from trying. As a matter of fact, there are numerous definitions. Some state, that creativity has a connection to new ideas. Others believe that creativity arises from the recombination of old ideas. But it could be argued that when using old things, only old ideas would appear. (Johnson & Carruthers, 2006) Four main concepts have emerged from the many different approaches to define creativity: the *creative process*, the *creative person*, the *creative product* and the *creative press/environment*. (Warr & O'Neill, 2005; Sarsani, 2011)

The creative process

The creative process is an old definition and sees creativity as a form of internal process. Like the definition of creativity, there are numerous perspectives on the creative process. (Warr & O'Neill, 2005) Usually, the creative process is split into the following phases (Gabora, 2002):

1. *Preparation*: The person considers a problem vigorously and collects information. Some attempts to solve the problems are made, but fail.
2. *Incubation:* The person processes the problem subconsciously, without being concerned with it directly. This phase is sometimes not necessary.

3. *Illumination:* The subconscious ideas manifest themselves in the conscious as a raw idea. This can happen via dreams for instance.
4. *Verification:* The raw idea is refined, verified and can be shared with other people.

There are other definitions, in which these four phases are collected in a *brainstorming phase* and a *focus phase*. These two instances can be associated with the general definition of the *two mode thinking process* (Gabora, 2002):

1. *Associative mode:* Things are associated with one another which do not have an association in real life. This association is relatively weak and only lays the foundation for solving a problem.
2. *Analytic mode:* This mode is more focused and tries to observe cause and effect, thus making the solution for a problem obvious. After this, the solution can be presented to others.

The creative process represents the mechanisms inside the head, but it hardly defines creativity. And with this definition, there is no possible way to measure creativity. (Warr & O'Neill, 2005)

The creative person

This definition states that creativity is a personal trait, which helps a person to be creative. There are humans, who can be more creative than others, thanks to their high developed traits in this area. Contrary to the creative process, a test exists, which tries to measure creativity, but it should not be used as an actual measurement. A description of creative properties is also missing in this definition. (Warr & O'Neill, 2005) Nakakoji, Yamamoto, & Aoki (2002) describe three different types of creative persons:

- *Inspirationalist*: This type generates creativity from unconstrained and unsorted thoughts. Inspirationalists shine at brain storming.
- *Structuralist*: People, who can be described as structuralists are more organized than inspirationalists. They need a certain structure to support their way of thinking. Structuralists often use visual aids, like diagrams.
- *Situationalist*: This type is dependant on the social environment.

The creative product

The creative product is the end product, which is produced by the creative process. For evaluation purposes the novelty of the product can be used. The problem herein lies in the definition of novelty, because there are, as for every other definition in this chapter, too many to list. To separate a creative product from a purely novel product, another differentiation has to be made: The appropriateness. A product is appropriate if the end product displays features, which were defined at an early stage of the creative process. Of course these features depend on the initial problem. Creativity itself can be measured via the end product. It should be noted, that only subjective specialists are able to create a valid measurement. (Warr & O'Neill, 2005)

The creative press/environment

The creative press's mainly concern is the environment, both the material and the social. Creativity is encouraged, or hindered by this environment and the pressure it invokes on the

person. (Johnstone, 2007) Too much pressure and constant rewards can have a bad impact on the creativity. (Friedrich, Stenmark, & Mumford, 2011)

Four P model

All of these four definitions should not be treated as different entities. There are some overlapping parts. Because of that, it is possible to merge all four definitions into one. The result is the *Four P model*, which is shown in Figure 2.1. The Four P model is fairly widespread and the most accepted model for creativity. (Sarsani, 2011)

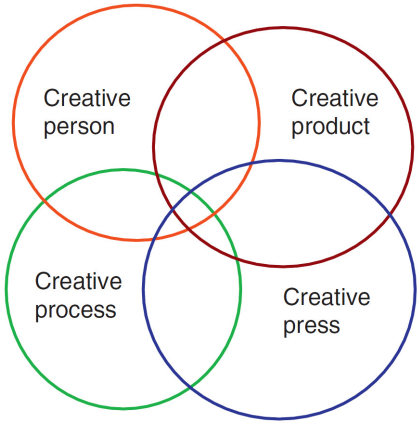


Figure 2.1: The Four P model (Sarsani, 2011)

2.1.2 Creative Environment

As mentioned before, there are opinions, which state, that creativity can not be improved. But there is also a theory, which describes the overall potential of creativity. According to this theory, every human has a certain creative potential with different boundaries. These boundaries can be influenced. The potential depends on different factors, like cognitive components, interpersonal components and others. Only the working environment and how to analyze it will be discussed in this section. (Plucker, Runco, & Hegarty, 2011) The creative process is contrary to the common opinion lengthy and requires a huge amount of motivation. Motivation is the key factor here, because it determines, if a person is more or less eager to participate in the creative process. It is greatly dependant on the work environment. (Friedrich et al., 2011)

To analyze an environment for creative support, the following means can be used (Friedrich et al., 2011):

- *Encouragement for creativity*: The question herein lies: How good is the support for the person undergoing a creative activity? For example, how much appreciation people get from others, like their superiors?
- *Autonomy*: Autonomy is important. The bigger the liberties, the better the creative performance.
- *Resources*: It is important for the person, who participates in a creative activity to know that there are enough resources for the process to finish. This shows the appreciation for the work put into the process. The resources are not limited to physical manifestations, but can also be of the mental kind.

- *Pressure*: Pressure can increase creativity, if the task is challenging enough. But this has to be well balanced, because creativity can be hindered by a task too challenging, or demanding.
- *Organizational Impediments*: These are events, which are beyond a person's power and influence. Such events can affect the creative performance.

When a group undergoes a creative process, there are four additional factors, which should be considered, when classifying the environment for creativity (Friedrich et al., 2011):

- *Vision*: Is a common goal provided for every group member? Does this goal motivate them? Are they trying to reach it?
- *Participative safety*: This is important for group climate. Every group member should be able to bring in their own ideas without having to fear some sort of retribution.
- *Task orientation*: How much is the group is able to focus on the task and how much are they trying to improve it?
- *Support for innovation*: It should be shown that innovation is appreciated by the team and company.

The next possibility for analyzing the creative support is the psychological level. Like the environment, the two main factors are *freedom* and *challenge*. Freedom is determined on how much a person is restricted in the thinking process. The next factors are *acquaintance/openness* and *support*, which were also represented in the team environment. Of course there are other opinions about the properties of an environment, but overall there are 14 areas, which are all sub-groups of the previously mentioned. (Friedrich et al., 2011) It should also be stated, that creativity promoting environments have no influence on the occurrence of creativity whatsoever. It only increases the chance for it, even if the result may not be the desired outcome. (Harrington, 2011)

2.1.3 Creativity in Technology

An interesting focus, when thinking about technology and creativity is the question: Is it possible for computers to be creative? There are two major problems, which arise already at the beginning. The first problem lies in the definition of creativity, which is, as seen in the previous sections, not clear. The second problem is the following question: Is a computer able to form a creative process? This problem comes from another question, whether creativity is a purely human trait or not, which is unknown. It is commonly believed, that computers will never be able to be creative by themselves. This is because they can provide different solutions for a problem, but they do not have any connection to their solutions. Besides, computers have difficulties in finding answers to vague task definitions. (Sarsani, 2011)

But whether computers can be creative or not, they can support creativity in many different ways, which are not only based on sketching and drawing programs. It is even possible for a computer to support the whole creative process. There are four theoretical models on how a computer can be used in relation to the creative process (Johnson & Carruthers, 2006):

- *Computer as nanny*: The computer monitors the whole creative process, analyzes progress and sets deadlines.
- *Computer as pen-pal*: The computer suggests new ideas, examines approaches for solutions and gives feedback.

- *Computer as coach*: The computer has a wide array of knowledge, which is available to users in order to support them.
- *Computer as a colleague*: Computer and person both work as equals together to achieve their goal.

The opinions, whether a computer can support, or hinder creativity are widely divided. Researches who think the latter, state that the logical structure of a computer not only hampers the creative thought, but even deteriorates it. There are also opinions in the middle, which state, that computers do not even have much influence on the creative process. Which opinion is accurate is hard to tell, because there are hardly any studies in this direction. Only the artificial intelligence field is believed to free computer from their rigid, logical patterns. But contrary to common beliefs, this sector is nowhere near as developed as people may think. (Sarsani, 2011)

2.2 Creativity in Industries

As stated in chapter 2.1, a product is created after the creative process. This product can be nearly anything, ranging from merchandise to plain ideas. For many companies, creativity is of major importance to stay competitive. (Seidel, 2009) This means, there is a connection, or better an interface, between creativity and business. This interface integrates new technologies and is often described as *creative industries*, or *culture industries*. The name culture industry is used, because culture often plays an important role in these types of industries. (Wright, 2010)

2.2.1 Overview

According to Flew and Cunningham (2010), "[...] *the concept of creative industries emerged in the late 1990s primarily as a policy discourse*" (p. 1). When speaking of creative industries, the common consent is that there is no difference to culture industries. But the origins of both terms and their weak definitions can cause some irritations. (Hesmondhalgh, 2008; Galloway & Dunlop, 2007)

History

Cultural industries were initially used to differentiate between *commercial art* and *subsidized art*. Commercial art basically consist of music, film, publishing and broadcasting, whereas subsidized art is something that can be seen in museums. (Galloway & Dunlop, 2007) Cultural industries started in the nineteenth century. In the same time period, the capitalism spread, thus cultural industries emerged in countries, where the capitalism dominated. In the following era, the popularity of cultural industries was more or less proportional to the growth of mass culture. The term 'cultural industries' was then introduced, mainly to criticize the increased marketing of art. (Hesmondhalgh & Pratt, 2005)

After 1950 cultural industries continued to rise, which was due to the increase of wealth. In 1980 cultural industries were too big to be ignored, because they started to influence policy making of certain governments. Step by step the label of cultural industries was suppressed by a new upcoming term: Creative industries. The reasons for the change are difficult to find. One important factor is assumed to be of political nature. Some other critics state, that the change was purely made to stimulate economies in the intellectual property area.

(Hesmondhalgh & Pratt, 2005) The main difference of both terms is that cultural industries are more constricted to art and media. Creative industries include more modern technologies, which did not exist when cultural industries were introduced. Basically speaking, both terms are products of their time. (Galloway & Dunlop, 2007)

Definition

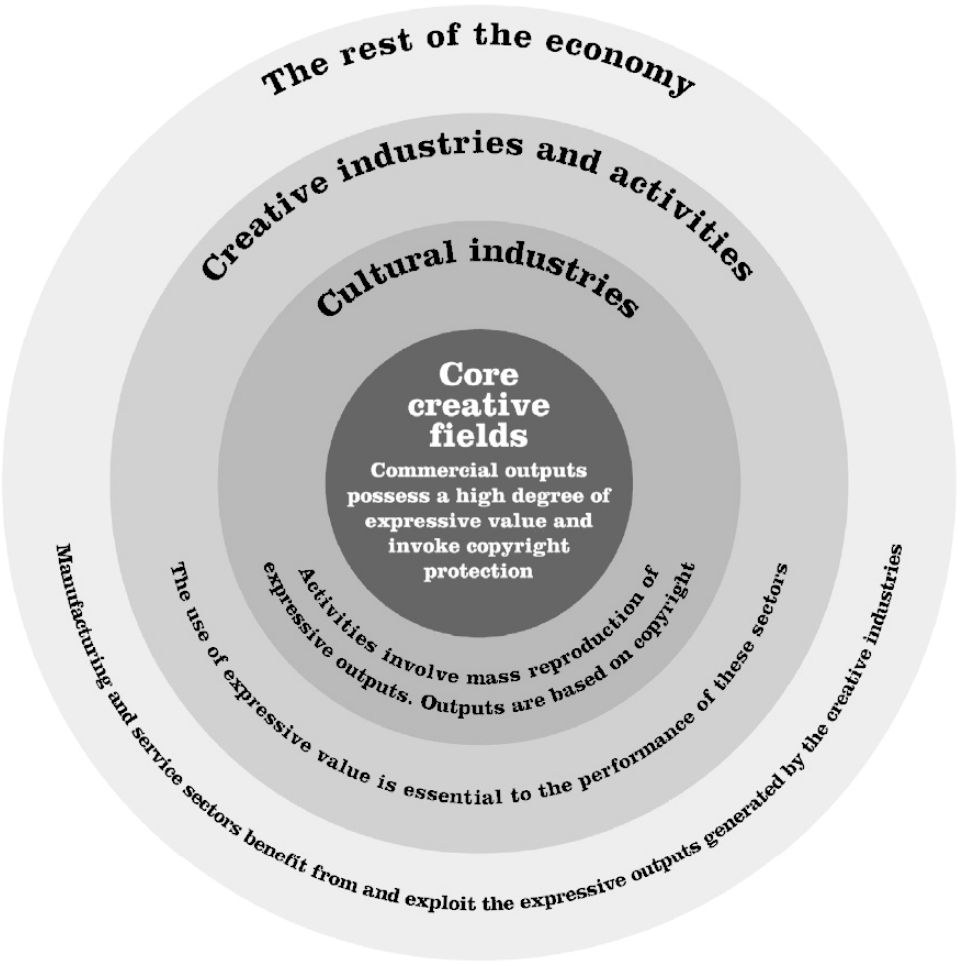


Figure 2.2 Work Foundation model of creative/cultural industries (O'Conner, 2010)

There are a couple of definitions for cultural industries, but most differ in specific ways. Usually some of the following components are used in these definitions (Galloway & Dunlop, 2007):

- *Creativity* plays a key role. But because creativity is a widespread term, it can basically be applied to any industry.
- *Intellectual property* implies that creators have the claim to all of their products. In the United Kingdom (UK), the copyright is the undercoat of defining creative industries. But like creativity, this part of the definition is not very distinctive.
- *Symbolic goods/meanings* are goods, whose money value is equal to their cultural value. Industries falling under this section of the definition have to produce something that can be described as art. This is also the classic definition of cultural industries.
- The *use value* is differentiated by the product's usage, whether it is more about functionality, or more about exchanging ideas. Only products with the primary focus on idea exchange are cultural products. This excludes architecture from cultural industries.

- *Methods of production* are another important factor for the definition, because the product has to be created with industrial means. In conjunction with symbolic meaning, this is the crucial part of the cultural industry definition.

The definitions of creative industries are much wider. One of the more influential definitions was developed by the *Department of Culture, Media and Sport* in the UK. Their definition is built around two principles: Intellectual property and creativity. This is a rather weak definition. With such an open-ended definition, it is difficult to state whether an industry belongs to creative industries or not. (Galloway & Dunlop, 2007) Many different models stem from this definition. They usually try to limit the scope between creative industries, cultural industries and normal art. Figure 2.2shows a model created by *Work Foundation*. It does not use a hierarchy based on art, unlike other models. The use of expressive value, instead of creativity should be noted. (O'Conner, 2010)

2.2.2 Economy

Since there is no uniform and accurate definition of creative industries, many countries use their own. These countries often use definitions, which fit their needs the best. To talk about economy, it is necessary to first know which industries belong to the creative sector. Table 2.1 shows industries, which are classified as creative industries by country-specific definitions. It also shows the similarities between definitions from different countries. (BOP Consulting, 2010)

	UK	**Germany**	**Spain**	**France**
Term used	*Creative industries*	*Culture and creative industries*	*Culture industries*	*Cultural sector*
Architecture	X	X		X
Audio-visual (film, TV , radio)	X	X	X	X
Performing arts	X	X	X	X
Libraries			X	X
Design	X	X		
Art market/Visual arts	X	X	X	X
Publishing	X	X	X	X
Fashion	X			
Software/multimedia	X	X		
Museums/cultural heritage			X	X
Music	X	X	X	
Crafts	X			
Advertising	X	X		

Table 2.1: Creative sectors in different countries (BOP Consulting, 2010)

The economy in the creative field grows rather quickly and counts as one of the leading growing sectors in developed countries. 2.6% of the gross domestic product from the *European Union* (EU) was due to cultural and creative industries in 2010. About 5 million people in the EU work in this sector.The annual growth of the creative sector from 1997 to 2007 was 5% in the UK, whereas other industries only displayed a growth of 3% per year. 2% was the annual growth in employment in this sector. Other industries tended to only grow 1%. Approximately 4.5% of the exported goods were produced by the creative economy. Other countries show similar growth rates. The economy in developing countries is a little different, but a growth in exported creative goods can also be recognized. The export growth increases even faster in developed countries. From 2002 to 2008 the portion of creative goods from

developing countries compared to the rest of the world raised from 37% to 43%. This incline was due to China, which was the leading exporter of creative products in 2008. In the same year, 20% of the overall creative products were produced in China. As shown in Figure 2.3 the global export for creative goods has grown relatively fast, annually about 14%. The export earnings of all creative products, which are not only goods, but also services, were $592 billions in 2008. Their value improved immensely from 2002 to 2008. The value growth was about 11.5% for goods and 17% for services per year. (UNCTAD, 2010)

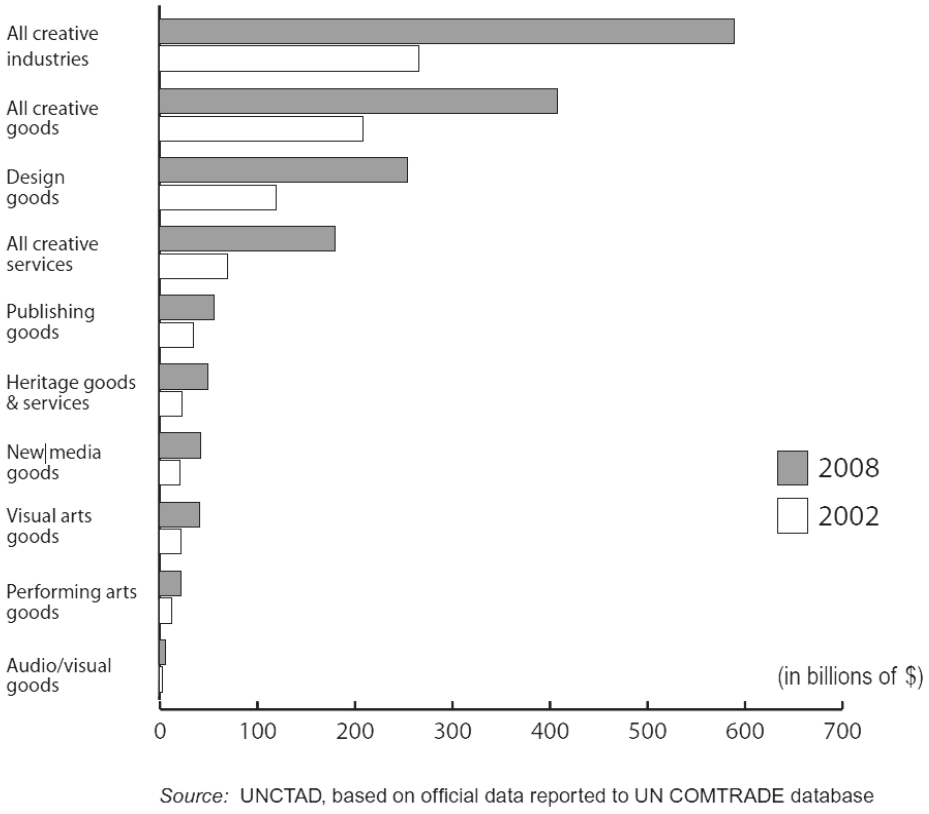


Figure 2.3: Growth of worldwide creative good export (UNCTAD, 2010)

2.3 Prototyping

The previous section gave a short overview of creativity, how to support it and how important it is in the current economy. This chapter is all about getting from the idea to something presentable, a prototype. Prototypes are vital in many areas. It is commonly known, that prototypes are used in industries, like car production. Unfortunately, when producing software, it is often hard to justify the use of a prototype. (Warfel, 2009) But with the aid of prototypes, it is possible to examine an interactive system long before its finalization. It is even possible to incorporate users, which can improve the whole process. (Holmquist, 2005)

But there are other benefits as well. First it is possible to find problems relatively early, which is desirable in software development, because finding errors later on can be costly. The second benefit is the learning by doing effect. People involved in the prototyping process will learn more quickly. The third benefit is that new ideas are gathered, which probably would not emerge by just thinking about the project. And having a representation of the product can often help solving problems. (Hartmann, 2009) This section will discuss prototyping, with a focus on prototyping in the software developing field.

Prototype classification

One way to classify prototypes is to categorize them into two different groups: *mock-ups* and *functional prototypes*. Mock-ups do not have any functionality. They are only visual representations of the design goal. Their purpose is to find errors and problems in the design early on, even before any functionality is built. Functional prototypes are the complete opposite. They may not have any resemblance to the visual goal, but they implement the functionality of the finished product. (Holmquist, 2005)

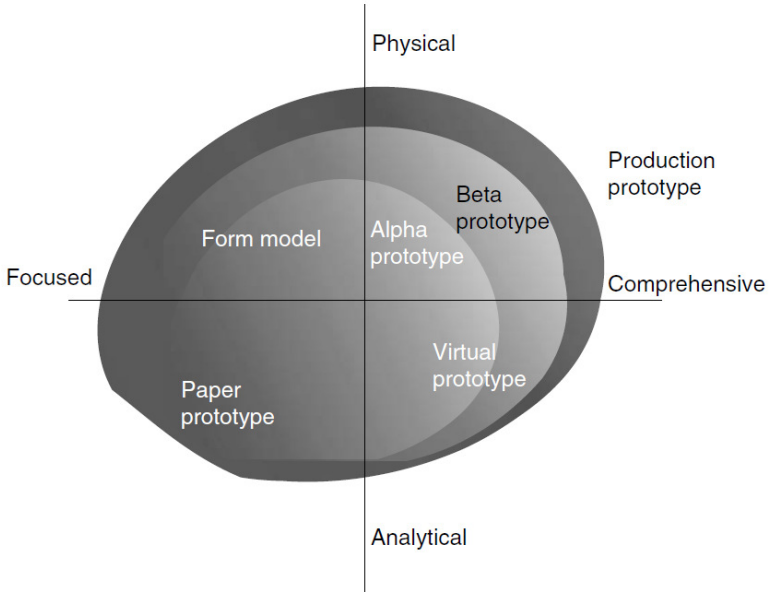


Figure 2.4: Classes of prototypes (Liou, 2008)

Figure 2.4 shows a different classification for prototypes. *Physical prototypes* are real world representations of the finished product. Their counterparts are *analytical prototypes*. They are a theoretical representation, like a mathematic formula. *Comprehensive prototypes* realize the complete desired product. *Focused prototypes* have just one or a few sections of the product built into them. Mock-ups can be categorized as a physical, focused prototype and can also be called *form models*. *Paper prototypes* are drawings on a piece of paper and *virtual prototypes* are virtual simulations. *Alpha and beta prototypes* are prototypes, which exist in real life. Alpha prototypes contain only some functional parts of the product, whereas beta prototypes implement the fully operational product. (Liou, 2008)

2.3.1 The Prototyping Process

There are too many different prototype processes to list here, but there are some guide lines. First, the end product itself should be the focus, not the prototyping process. Second, the prototyping process should be as flexible as possible, without loosing its structure. With a more flexible structure, the process can better respond to adjustments. Third, if the process hinders the workflow, it has to be revised. (Warfel, 2009) Before even beginning with the process, product characteristics have to be defined. These characteristics are then transformed into the prototype's requirements by considering properties, the protoype should have. It should be noted, that the requirements must be as clear as possible, to not hinder the prototype's production. (Liou, 2008)

The prototyping process is basically an iterative process in which one prototype often does not suffice. The problem with this kind of process is to know the amount of iterations and prototypes, which are needed. Basically the process can iterate indefinitely. To prevent this, it

is advised to define fairly grounded requirements and if possible, to split them into primary and secondary goals. The process ends, when all requirements are met. If it is not possible to pursue the process, because of lack of time, or money, the process has failed. (Liou, 2008)

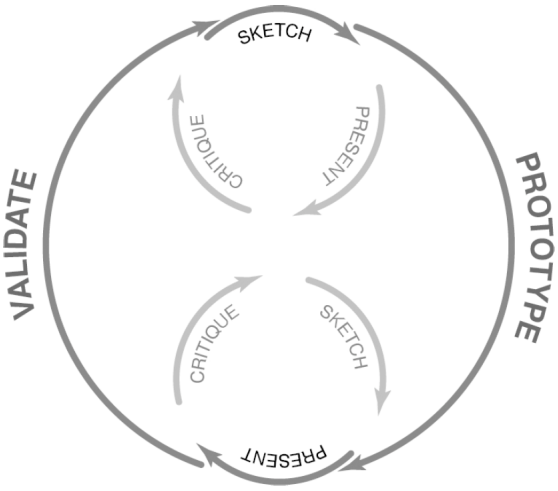


Figure 2.5: Possible prototyping process (Warfel, 2009)

Figure 2.5 shows an exemplary process. The process contains three inner areas, *sketch*, *present* and *critique* and two outer areas, *prototype* and *validate*. The prototyping process starts with a sketch, which should only contain vague ideas. After collecting these ideas, they get presented and critiqued, to eliminate the ones that are not viable. This step is needed for creating the requirements. After they meet everyone's approval, the prototype is then built. When it is finished, it will be presented and critiqued, like the sketch. If the prototype meets the requirements, it is then validated. After this, the cycle can continue, or end. (Warfel, 2009)

Fidelity

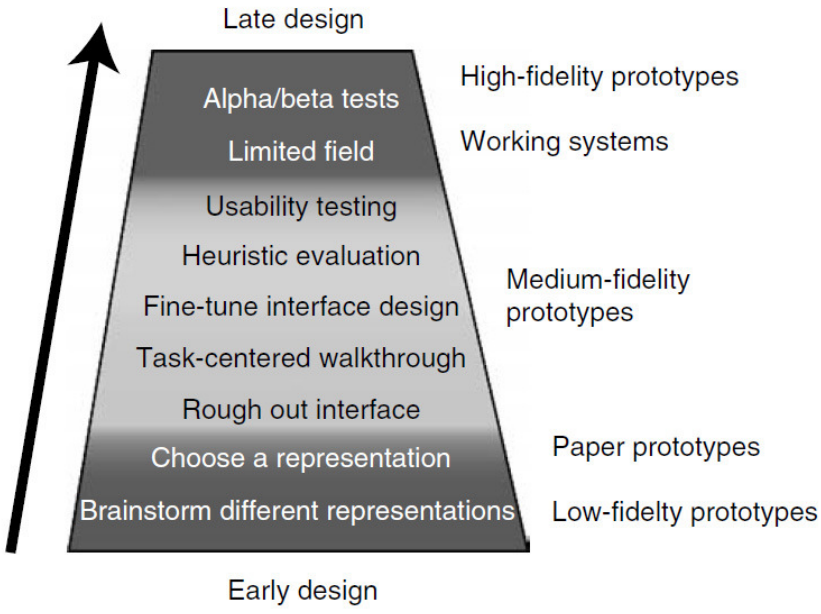


Figure 2.6: Prototype design levels (Liou, 2008)

Fidelity indicates how many features of the finished product the prototype has built into it. It is hard to determine the fidelity when the final features are not all defined in detail due to an early stage of development. When this is the case, fidelity describes the possibility of the

prototype to represent the product. *Low-fidelity* prototypes are rather easy to build and they show a general overview of the design goal. They are limited to rather low functionality and are not suited for problem finding. *High-fidelity* prototypes are almost completely functional, but they are expensive and they do not adapt quickly to new changes. The main purpose of high-fidelity prototypes is to demonstrate the final product, or test it with end-users. *Medium-fidelity* prototypes combine the benefits of both, high-fidelity and low-fidelity. They are quickly to build, are adaptable and can be used for testing. (Liou, 2008) Low-fidelity prototypes are usually used in the early stage of a design process. This stage could be the conceptual phase. High-fidelity prototypes are mostly used in late stages, like testing. Generally it is possible to say, that the further the process goes, the higher the prototype fidelity gets, as shown in Figure 2.6. (Liou, 2008)

Prototype Testing

Usability testing is important and one of the most important objectives in prototyping. (Warfel, 2009) It should not be considered as a single instance in the whole design process. It should be constantly used. One major advantage of continuous testing is that errors in design and functionality will be identified rather early in the design process. The usability testing should proceed as follows: The prototype is given to the users and they are asked to complete certain tasks. The whole user task should be well documented. After all the user feedback is processed, the prototype is revised and redesigned and a new round of testing and revising is commenced. (Liou, 2008)

To find the appropriate users, the following three questions should be clarified before running the tests (Liou, 2008):

- *Representative subjects*: Which background should the test users have? For example, do they already have experience with the tasks ahead? What general education level is appropriate?
- *Physical concerns*: Which physical characteristics should test subjects have? This can range from age limits, or gender to specific physical attributes.
- *Experimental conditions*: These are the test circumstances: Which time to test, or which environment should be used? Should the environment contain distractions?

2.3.2 Prototyping Techniques

"*In recent years, various technologies have been developed to facilitate rapid product development. Among these technologies, virtual prototyping (VP) and virtual manufacturing (VM) may be regarded as important technological advancements.*" (Choi & Cheung, 2010, p. 203) But there are other techniques as well. Because the main focus of this work is on the computer aspect of prototyping, the following techniques incorporate software for prototyping purposes.

Prototyping in Software Development

Prototyping techniques in software development usually aim for one of three objectives. The first is *exploration*, which is mainly used to find certain requirements that are not decided at the beginning of the prototyping process. The second objective is *experimentation*. This objective is usually needed for future decisions, like where to go with a project. The third and last goal is *evolution*. This is a technique, were the prototype itself grows with time and raises

new requirements. The evolution technique is therefore fairly dynamic and adaptive to new circumstances. (Carr & Verner, 1997)

Rapid Prototyping

There are many techniques, which can be classified as rapid prototyping. The rapid prototyping process can be considered as a fast process. Prototypes are built fairly quickly; their types ranging from simple sketches to simulations. (Beaudouin-Lafon & Mackay, 2008) But strictly speaking, rapid prototyping is about the quick creation of a physical prototype through technological aid. It is also called *layered manufacturing*, or *solid freeform fabrication*. (Liou, 2008)

The name layered stems from the resulting model. It basically consists of two-dimensional layers which are used to create the object in 3D. This means in these two dimensions, the prototype is fairly accurate, but in the third dimension it can be rather imprecise. The process to produce such an object goes as follows: First the prototype is modeled in 3D with a graphics tool. Then it is transformed into another format, using triangles for estimation. This transformed model is sliced into 2D pieces. Important in this step is the decision, where the third dimension should be on the model. After slicing, the model is created. This can happen with a multitude of tools, like lasers, or 3D-printers. After the physical prototype is made, some post processing procedures may be necessary. (Pandey, n.d.)

Virtual Prototyping & Virtual Manufacturing

Again, there are many different definitions for virtual prototyping. The differences between definitions mainly exist because of the difficulties to set the right boundaries. For example, it is not clear whether digital mockups should be considered virtual prototypes. In general, virtual prototyping can be described as the creation and testing of a virtual representation of a physical object. (Wang, 2003) To conduct virtual prototyping, two requirements have to be fulfilled. First, a *simulator* is needed for modeling and interaction between objects. Basically it should simulate realistic behavior. The second requirement is an *interface between human and software*, for in- and output. The huge advantage of virtual prototyping is the possibility to use and analyze the prototype without the need of actually building it. Certain hardware requirements are needed in order to simulate the desired graphics. The software itself should not be underestimated, because, besides of simulating the prototype, it also has the task of collecting data. (Liou, 2008)

It should be stated, that there is a difference between virtual prototyping and *virtual environments* (or virtual worlds, which will be described in more detail in the next chapter). Virtual environments are similar to virtual prototyping environments, but with the goal of immersion. Therefore a virtual world is more about experiencing the world. This goal however is not crucial to virtual prototyping. Even if they are not needed, it can be advantageous to incorporate virtual environments into a virtual prototyping process. It can be easier to involve users for evaluation, because of their immersion into the world. This is especially useful when designing esthetics. Using virtual worlds have also the possibility to change objects in real time, which is not always doable in some simulation programs. (Wang, 2003) It is also possible to use *augmented reality* instead of virtual environments. Augmented reality uses environments from the real world and projects computer generated content into them. This can be especially useful for environments too complex to simulate, or when different ways of interactions are needed. Figure 2.7 shows such an augmented reality environment. (Liou, 2008)

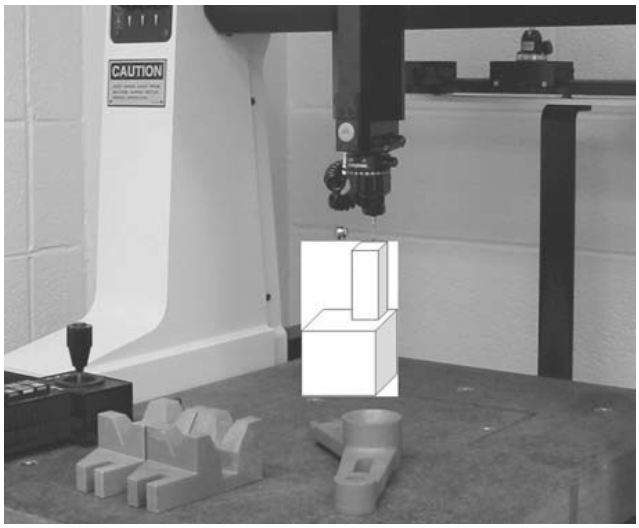


Figure 2.7: Augmented reality environment (Liou, 2008)

Virtual manufacturing can be seen as another form of production virtualization. Virtual manufacturing simulates the manufacturing process of a product. At first glance it may look similar to virtual prototyping, but the focus lies on the production process, instead of the product itself. Its purpose is to estimate difficulties and costs in production. (Wang, 2003)

2.4 Science Fiction Prototyping

Section 2.1 glanced over the fact, that creativity is important for new ideas. But where do engineers and designers get their inspiration from? A great amount is drawn from the Sci-Fi genre. Not only new ideas can emerge from this genre, but also new technologies. In fact, when developing products for the future, Sci-Fi has often played a bigger role in the design process. Future casting is a good example in this case, which tries to analyze trends, or finds needs which may emerge in near future. With this future vision in mind, it is built towards it in order to achieve it. It is not about determining the future, it is more about leading into the direction that was envisioned. To react to changes, like new trends, this process must be fairly adaptive. (Johnson, 2011) Traditional prototypes are used to build on the basis of these future visions. But problems can arise, when the technology is not ready yet, so the envisioned product can not be created. SFPs avoid this problem. (Zheng & Callaghan, 2012) They are not like prototypes someone may think of, they are, simply put, stories about the future. The reaction of people to new technologies can be explored within these stories. It should be noted though, that these stories have to be grounded on a science fact, which builds upon real world science. (Johnson, 2011)

2.4.1 Overview

SFPs can also be called *Creative Fiction Prototypes*. (Graham, 2013) They should be seen more as a tool for developing toward a future, rather than normal Sci-Fi stories. The main goal of SFP is to analyze a science fact, encourage discussions about the particular envisioned future and to realize problems. In SFP the prototype can be seen as fiction. It is not the final product, which is built later, but rather an estimation of it. Generally speaking, a SFP can be a story in *written*, *comic book*, or in *video form*. (Johnson, 2011) SFP in general tries to research upcoming technologies and their cultural impact. (Johnson, 2010) It is stated, that humans are

able to learn better, when confronted with a narrative, instead of only numbers and words. In other words, the usage of SFP encourages creativity. (Schwarz & Liebl, 2013)

History

Sci-Fi and science share a long bond between each another. Scientists are often amazed by the tales they experience in Sci-Fi media. This is the reason why *Hugo Gernsback* in the early 20th century combined scientific articles with Sci-Fi stories in the magazine *The Electric Experimenter*. Until the middle of the 20th century Sci-Fi stories used the Sci-Fi element more as a means to an end. It was never the focus of the stories. That all changed when *Isaac Asimov*, supported by *John W. Campbell*, tried to create something new. They injected their story with more logic and real world science. This lead to Asimovs three laws of robotics, which can be seen as a marriage between science fact and Sci-Fi. At the end of the 20th century many authors wrote Sci-Fi, not only to influence research, but also to show how new technology can be used and which implications it might have on humanity. This shows how implications of technologies can be researched, even before they exist. (Johnson, 2010) SFP was initially indented for helping in the creation of technologies that can be influenced by future developments, especially areas like user needs and society. Therefore engineers crafted Sci-Fi stories to better grasp the future, in which their technology would be working. (Kohno & Johnson, 2011)

Definition

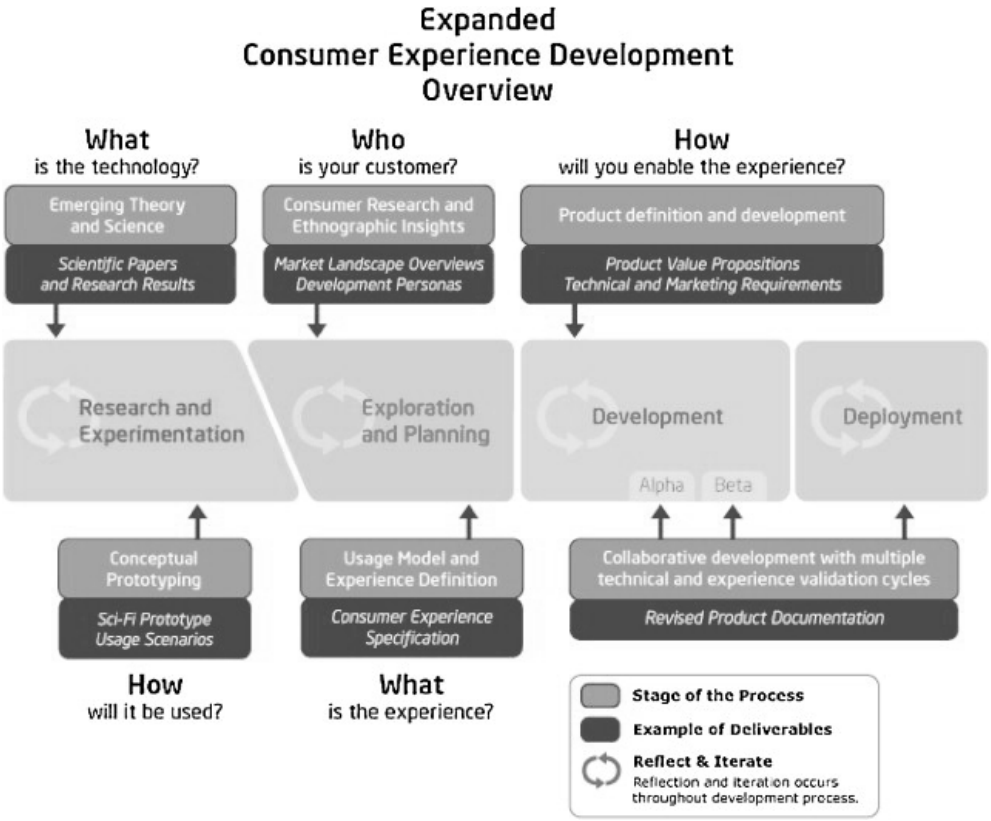


Figure 2.8: SFP used in product development (Johnson, 2010)

As Graham, Greenhill and Callaghan (2013) state, a SFP "[..] *uses storytelling imagery based on science fact as a design tool to explore the social and economic consequences of innovation*" (p. 1). SFPs are created like traditional prototypes. That means the process consists of iterative steps, which lead to a technology that can be produced after the process

has finished. But unlike other prototyping processes, using SFP adds another step into the process. The main focus of this step is the future usage of the technology. Figure 2.8 shows the SFP process used in a standard development field. (Johnson, 2010) For SFP it is important to create an environment in which the new technology can be observed, including its advantages and disadvantages. This can range from the best or the worst outcome to the daily usage of the technology. There are two types of SFPs. The first is limited to the science portion of its name. That is why this type focuses on the technical aspects of the prototype. The second type is more concerned about the economic and cultural impact a new technology could have. Contrary to other future prediction methods, like scenarios, SFPs use a clear interpretation on how the future should be. Furthermore it also encourages discussion about the future vision. The story, which is told by the prototype, is only a vehicle, to transport this vision. (Graham et al., 2013) Figure 2.9 shows that SFPs fulfill different needs than scenarios and models. Scenarios are usually set in the near future, like in five or ten years, whereas SFPs test a vision further away. (Graham, 2013) Scenarios also try to make realistic assumptions of the future, which is not needed in a SFP. (Graham et al., 2013)

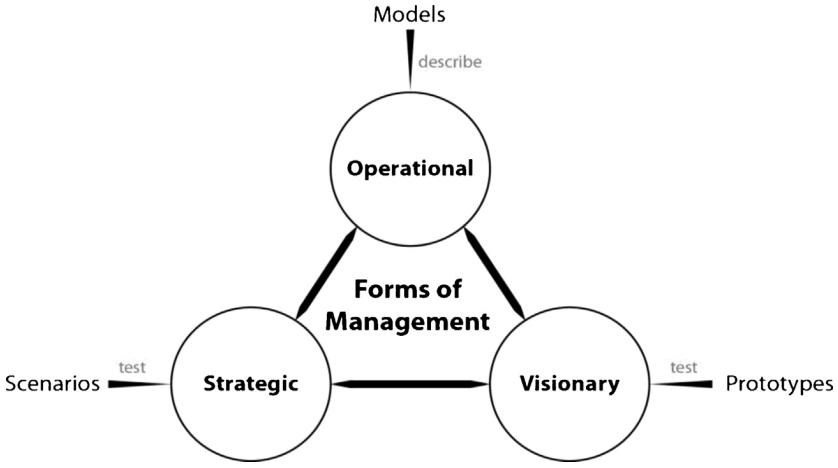


Figure 2.9: Relation between prototypes, models and scenarios (Graham, 2013)

Outline & Creation Process

To construct the story, an outline is necessary. This outline is usually the idea the story will revolve around, not the actual plot. (Johnson, 2011; Kohno & Johnson, 2011) Such stories can be called *idea stories*. At the beginning of the story, there is a question, that needs to be answered, which happens at the end. Idea stories are about the path from the question to the answer. Mystery novels use this kind of story telling: A murder takes place, who is the killer? This is the question. The answer happens usually at the end, when the identity of the killer is revealed. (Card, 2001) But SFP focuses not on writing a story worth a novel. It is about learning something through planning the story, creating characters and their interactions. (Kohno & Johnson, 2011) To create the outline, the following steps can be used as a guideline (Johnson, 2011):

1. *World building*: The object the story will bee built upon has to be selected. This object can range from a certain technology, or an advanced version of an already existing technology to just a problem. With the object in mind, characters and environments can be created. The environment however should be grounded in real science. The technology and its future counterpart should also be well defined.
2. *Scientific infliction point*: In this step, the created world and the selected technology are tied together, which brings in certain implications. The focus should be on the

effects on human beings and the world rather than the scientific facts behind the technology. The technology should have been defined already in the first step.

3. *Technology ramifications on people*: What does the technology bring to the world? How does it change the world and people? These are the questions, which need to be answered. It is even possible to go into some extremes with these questions to gather new ideas.
4. *Human inflection point*: This is the step where the characters defined in the first step react to the technology. These reactions are observed, when the extremes are reached. It is the climax of the story. Usually there are only two different options to resolve the climax: Either the character changes, or the technology changes. These changes should also be grounded in reality.
5. *Lessons learned*: This step reflects on the observations made in the previous steps.

Example

The following short story should give an example on how to create a SFP. It should not be seen as a fully developed prototype. The example takes place in the near future, where flying cars have been around for a couple of years. They are allowed to drive either on the ground level, on still existing streets, or they can fly on a specific height, the high level. When flying on the high level, drivers get a virtual street projected onto their windows, to have some points of references. The projection is called the virtual highway. This is done to make the driving in the sky familiar to people who are used to driving on the ground, but also for safety reasons.

In this world, Billy, a young teenager, who just obtained his license, is driving the car of his parents to pick up his girlfriend Sandra. After they drive on the virtual highway for a while, Billy gets frustrated. He does not want to drive on a virtual street. He wants to experience the real thrill of a flight, so he leaves the virtual road. At the same moment he passes the boundaries, the projection on his window vanishes and he can now see the real world. The car gives some warnings, ordering him to return to the road, but Billy tunes them down and cranks up the music. He just wants to have fun and therefore flies around the residential area he lives in. Therefore he decides to fly closely to the top of the houses there. Everything around him is now a blur and he enjoys the sense of speed. After a while he does not pay attention to his surroundings anymore and talks to his girlfriend. Why should he be alarmed? He is in the air and nothing is in the way. Because of the loud music, he does not hear the warning of the collision detector, which located a chimney, just a little higher than the others. It barely scratches the bottom of the car, but in panic, Billy pulls the flight stick in the wrong direction and crashes into the next building. Next day on the news, the reporter talks about another accident, where a flying car has crashed into a house killing four people, including both the driver and his co-driver. Due to the high number of accidents with flying cars that left the virtual roads, the manufacturers are now updating the vehicle software, to prevent leaving the predefined virtual roads. In order to force people to stay there, the car will shut down and make an emergency landing, if they stray from the virtual road.

1. *World building*
    - The scientific object of this story is the flying car combined with augmented reality/virtual environments.
    - The world is set shortly after the massification of flying cars. The technology still needs some adjustments.

- Virtual highways are created for people to not get confused, when flying through the air and to guide them safely to their destination. They are also a protection for people on the ground, because they only go through empty land.
- Created characters: Billy, a young, cocky teenager; Sandra, his girlfriend, a talkative teenage girl; The news reporter;

2. *Scientific infliction point*
    - Flying cars in combination with virtual highways produce a familiar feeling for people, when driving through the air. It is like driving on the ground.
3. *Ramifications on people*
    - Traveling is a lot faster in the air.
    - Driving in the air is like normal driving, which makes it a lot easier for people, who grew up with normal cars, to adapt.
    - Familiarity may be a good idea, but some people want to experience the joy and freedom of flying, not driving the same way they do on the ground. They get bored with it.
4. *Human inflection point*
    - In this story, the human infliction point is the high number of accidents that occurred due to flying cars leaving the virtual highways. Because it is hard to change the peoples behavior (there will always be someone testing the limits), the technology has to change to keep other people safe. Therefore the flying car just shuts down and lands on the next safe spot, when the driver gets of the road.
    - There are endless possibilities to solve this problem. The variant used in this example is negative, restricting people even further. Another solution may be to change the way the virtual highway is built. Maybe only some reference points should be projected on the windows. Or the virtual highway can be switched on in addition to the reference points. Or creating a three dimensional highway, which will lead to other problems, like the question, where is a car allowed to pass another car?
    - But even with other options, the car shutdown mechanism may be a good idea, when the car drives over residential areas or places where it could endanger people on the ground.
5. *Lessons learned*
    - Changing people's behavior, when it comes to something as mundane as driving may be a difficult problem to solve. Therefore, the technology has to adapt as much as possible to guarantee safety for everybody.
    - On the other hand, helping people to adapt by forcing everyone to drive the same way may also be a risk.

2.4.2 Inspiration and Prototype Usage

Sci-Fi and science influenced each other for a long time. Sci-Fi brings new ideas for researchers and new scientific break-troughs generate ideas for new Sci-Fi stories. (Johnson, 2011) This is especially true, when it comes to interface designs. Probably the best example would be the influence Star Trek, the Original Series had on the mobile phone industry. In

1966, when the show aired, only wired phones and walkie-talkies were available for the general public. Star Trek gave an example, on how communication can be in the future. In 1996 Motorola created the first mobile phone, which could be opened like the communicator in the television show. A comparison of the two can be seen in Figure 2.10. The phone was a wide success, which was partially due to its recognition value. (Shedroff & Noessel, 2012)

Figure 2.10: Comparison between communicator from Star Trek and Motorola StarTAC (Shedroff & Noessel, 2012)

The next example goes a little bit more into the prototyping territory. It depicts the difficulties when designing holograms. Figure 2.11 shows two possible methods for projecting the image of a communication partner and the problems that may occur. The first solution, where both parties are equally large, would be the most desirable one, but it could block the view of the participants. This can be bad, when one of the parties is driving a vehicle. The second solution scales down one communication partner. This would negate the problem with the blocking view, but would also raise new problems. In this case, the person who gets scaled down might have a huge projection on their end, which would not be comfortable, especially if he is the superior of the person he talks to. If the projection is scaled down on both sides, both parties would look down on each other and nobody would look at each others eyes. (Shedroff & Noessel, 2012)

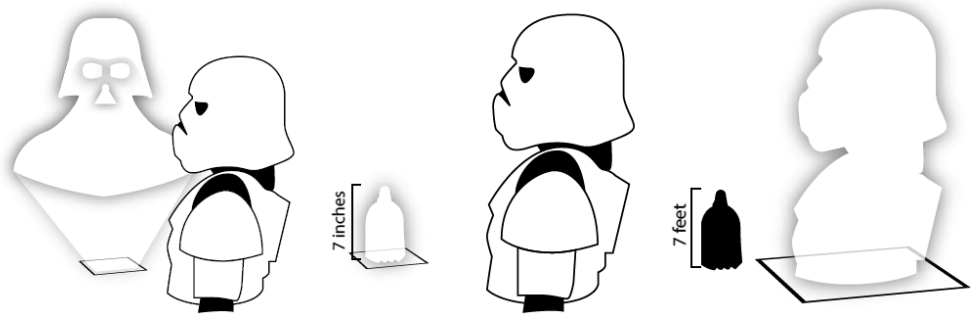


Figure 2.11: Problems with volumetric projections (Shedroff & Noessel, 2012)

But SFP can not only help with designing, but also with learning. At University of Washington a computer security course was supported by SFP. The normal teachings were not substituted by SFP, but rather enhanced. It showed how future users would use the technology and where possible attack points are. Therefore, it helped students to get a better idea on parts where additional security was needed and where not. Furthermore, the students learned more about the interactions between technology and users. SFP broadened the thinking about security, social and contextual issues. And it boosted motivation in most of the students. Some were discouraged because they did not expect creativity being a huge part of

the course. Another downside was that the students were able to examine only one or two key issues in more detail with the help of a SFP. (Shedroff & Noessel, 2012)

2.4.3 Related Work

Wu (2013) describes a SFP, which shows how learning could function in the year of 2050. It uses a combination of virtual environments and AI. The basic idea is that students should have the feeling to sit together in a classroom. The teacher is not a real person, but an AI driven avatar. To make the students feel like they are in the classroom, a desk, called eDesk, exists in the prototype, which lets students fully immerse themselves in a virtual world. But it is not only usable for teaching, but also for participating in films. The eDesk should give a person the feeling to enter another reality. Interestingly enough, a concept and even a prototype has been created in the real world (Figure 2.12). Bostanci & Clark (2011) use a similar approach, but for teaching history. Their prototype describes an augmented reality, which is far more advanced than today. It lets users experience past events, like transporting them back into middle ages, where they can interact with AI driven inhabitants, exploring a medieval city, fight in a battle, or communicate with other users. Sanchez-Lozane (2013) goes even a step further. He describes a future were nanobots stimulate all senses, to blend and replace reality with virtual environments. In his vision, even recreating thoughts is possible.

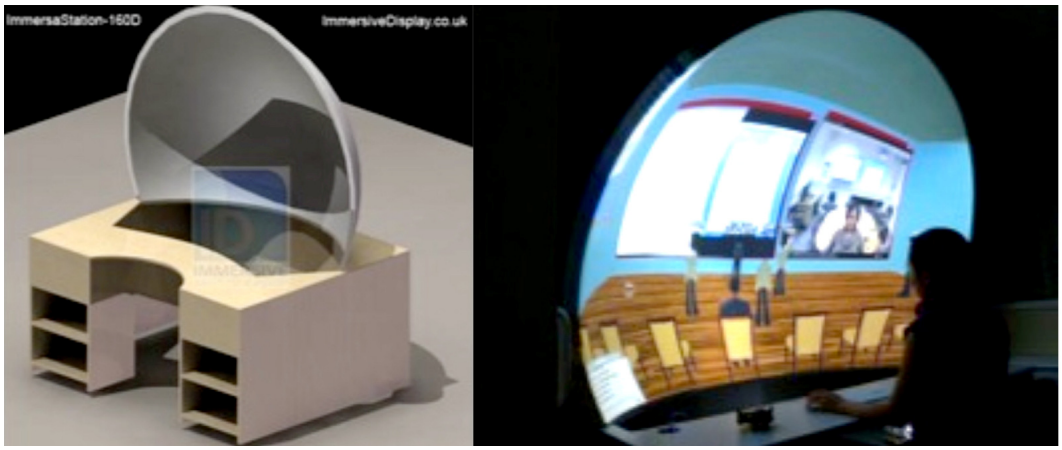


Figure 2.12: eDesk concept left and the prototype right (Wu, 2013)

Another prototype using augmented reality is proposed by Kymäläinen (2013). The prototype depicts a future, where people can model the interior of rooms when buying a new house, utilizing augmented reality in combination with intelligent environment. The intelligent environment is used to create the surroundings and the augmented reality to paint them, like choosing carpets, or wallpapers. The system also features a huge library of pre-made objects and designs, which was done by independent designers. Wu & Callaghan (2011) built a prototype upon three different science facts. The first fact is the usage of intelligent surfaces, which can create interactive screens on walls. The second fact is mixed reality, in which human and AI controlled avatars coexist. And the third is wearable computers, which collect data from their users. The prototype shows a future, where people get gradually addicted to technology. They get over-stimulated by all their intelligent devices, like social networking, which is used all the time. Because of this, they get mental problems. In the prototype a cure is described, which uses the same technology to get people back on track. The described intelligent surface is like paint, which is applied to the room, to create an immersive virtual reality environment. Its usage is fairly similar to the eDesk, allowing people to visit lectures, or other places. Tassini (2011) also describes the dependence to technology in his prototype. In his vision, the daily act of people is guided by a computer. The computer functions like a

personal assistant, recommending them items, or adjust their environment based on their preferences. Even menial tasks, like setting the temperature of the shower, are regulated by the assistant. The people in this prototype are completely absorbed in their technology.

Birtchnell & Urry (2013) created a couple of prototypes revolving around 3D printing, were they become readily available in different stages. These prototypes show an interesting future, with communities and problems not so uncommon in virtual worlds of today (see chapter 3.3). In the first prototype 3D printing is available to the general public, which allows for creating and sharing objects, but also poses the problem of handling intellectual properties. The second prototype envisions a future, where 3D printing is local available in stores, which allows for customization of products. The third prototype outlines the possibilities of co-creation and collaboration in a non-profit community, resulting in workshops. Potstada & Zybury (2014) show a similar future vision, in which nearly every object is created at home. Only large items have to be ordered. Augmented reality is used to browse through virtual stores and to buy and customize items, which are then created directly at home. These items are immediately usable.

2.5 Discussion

This first part of this chapter discussed creativity and its influence on the current industry. The problem with creativity and all its territories is the need for a basic definition. This problem is even more evident, when looking at creative industries, which are predicted to grow even further.

The second part of this chapter showed the application of creativity in prototyping and SFP. But can SFP be seen as part of prototyping? At first glance the process may look different. Prototyping is an iterative process and SFP just creates a story. According to Card (2001), writing a story is usually an iterative process too, because the story is refined over and over and writers never know how far they depart from their original idea. It can also be compared to the prototype fidelity, because the first draft may only be a rough outline of the story. The final story may be completely different from the first draft. In a sense it can be put into the prototype category, because it creates a representation of an idea instead of a product.

SFP may seem a little bit limited, because it uses only three types of media: written short stories, comics and videos. It totally misses the opportunities of two important possibilities to represent a story: video games and virtual worlds. Especially virtual worlds can be put to a good use, in order to analyze user behavior in a given scenario.

3 Virtual Worlds

Solving a problem is usually easier, when working in a group. Group efforts not only solve problems better, but also produce more creativity. That is why teamwork plays a key role in business and why it is important to try to enhance team efforts. Virtual worlds may be one of the possibilities to create a better environment for collaborative groups. (Qui, Tay, & Wu, 2009) This may also be a good environment for a SFP. Virtual worlds not only offer just one advantage. They are also interactive as well as immersive. Users could experience a SFP rather than just talk about it. (Pirker et al., 2014)

It is assumed, that the web of the future will not only consist of rigid documents, videos and alike, but also of three dimensional virtual worlds. On such an Internet structure, links not only lead to documents, but also to positions in virtual worlds. (Thompson, 2011) A trend towards virtual worlds was visible. The Internet users currently use text based content, like e-mail, but more and more people started to meet up in virtual worlds. (Qui et al., 2009) This trend however has stagnated in the last couple of years. (OECD, 2011) This chapter will give a short overview of virtual worlds, their definition and history, as well as their opportunities. The last section is the main focus of this chapter and discusses flexibility, like configurability and adaptivity of virtual worlds.

3.1 Overview

Virtual worlds have become most successful in the entertainment and gaming sector. It is possible to not only use them for gaming and fun, but also for research, learning and other areas. Unfortunately the interest for using virtual worlds has stagnated since 2008, at least in comparison with the time the first big virtual worlds, like *Second Life* (SL) appeared. But it is stated, that the interest is rising again. Some trends point towards this, like the increase of Google searches for virtual worlds, as shown in Figure 3.1. (OECD, 2011)

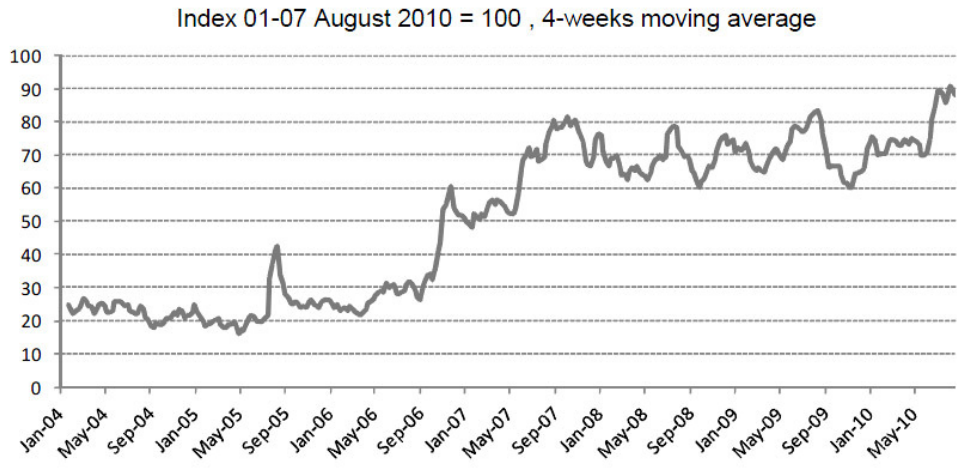


Figure 3.1: Google searches for virtual world from 2004 to 2010 (OECD, 2011)

Virtual worlds or virtual realities (VR) are computer programs, which simulate a world, usually with a social aspect, where people can meet and interact with each other as well as with objects in the world. Today these worlds are usually visited through normal means, like

desktop PCs. (Bainbridge, 2007) Genuinely speaking there is not much difference between the two terms virtual reality and virtual worlds definition-wise, but there are different layers of immersion. The first one is the *desktop level*, where the virtual world is visited through a normal PC. The next layer is the *fish tank level*, in which the desktop level is enhanced via head tracking. The last layer, the *immersive layer*, lets the users immerse themselves totally in a virtual world. (Mazuryk & Gervautz, 1996)

History

In the 1970s the term virtual world was mainly used in conjunction with virtual reality. (OECD, 2011) But the term later wandered into the gaming sector, where virtual worlds, as they are known today, have their roots. (Messinger, Stroulia, & Lyons, 2008) The forefathers of virtual worlds were called *Multi-User Dimensions/Dungeons* (MUDs) and *Multi-Object Oriented MUDs* (MOOs). They came up around 1980 and were only text based, but in many regards similar to modern virtual worlds. In 1985, the first graphical virtual world, called Habitat, was released (see Figure 3.2). It was created by LucasFilm for the Commodore64 and featured a marketplace for trading virtual goods. (de Freitas, 2008) In 1990 LambdaMoo was released, which allowed users to generate their own content. It was not only possible to create new objects, but also to change the landscape. (Marcus, 2007) In the years to come, processing power and Internet connectivity increased, which led to a growth of more complex virtual worlds. (de Freitas, 2008; Marcus, 2007) These virtual worlds quickly became more graphical advanced and went from purely text based to three dimensional graphics. *Massively Multiplayer Online Environments* (MMOs) and *massively multiplayer online role-playing games* (MMORPGs) were created. In 2003 the most noted virtual world platform, SL, started. (Marcus, 2007) From 2003 to 2008 a huge growth in utilizing virtual worlds was recognizable. (de Freitas, 2008) But after 2008, this growth came to a hold. (OECD, 2011)

Figure 3.2: Habitat, created by LucasFilm in 1985 (Lastowka, 2010)

Definition

According to Bainbridge (2007), a virtual world is *"[...] an electronic environment that visually mimics complex physical spaces, where people can interact with each other and with virtual objects, and where people are represented by animated characters.*" (p. 427)

Because the term 'virtual world' is relatively new, finding a concrete definition is not that simple. There are many, but they all differ in some aspects. (Barnes, 2010) There are also

similarities between the individual definitions: A virtual world is a computer generated world, which contains a large amount of users. But this would be too simple, because, when using only this definition, chatrooms would also be considered virtual worlds. It could be argued, that a virtual world would need a sophisticated graphical interface, but two-dimensional graphics, or barebones text would suffice. Another important part of virtual worlds is their persistence. Even after users have logged out, the world would still be there. This excludes multiplayer games, because after a session, the world would no longer exist. Another key part of virtual worlds are avatars. Avatars are representations of their particular user and are controlled by them in real-time. (OECD, 2011) Another definition is fairly similar, but differs in the incorporation of a social aspect. This definition uses five characteristics (Barnes, 2010):

- Virtual worlds use some kind of graphic, either 2D, or 3D.
- They house a large number of people, which engage in social activities.
- They are interactive.
- They provide a common ground for interaction between users.
- They must be persistent.

It should also be noted, that many definitions try to differentiate between virtual worlds and MMORPGs. Usually they state that they look similar, but the later category restricts their users and puts them on a specific path, whereas virtual worlds give users more freedom. (Barnes, 2010) But this separation may blur, when looking at some specific MMORPGs, like Eve Online. This game is constantly evolving. That is not because of the developer creating new content, but because the game world is shaped by the players. (Bainbridge, 2010) To further the problems in the distinction between MMORPGs and virtual worlds, both use a fairly similar architecture and both have a huge user-base, rendered environments, and physics systems. (Thompson, 2011)

Another way to differentiate these two is by the utilization of user generated content, which is mostly seen in virtual worlds. (Farley, 2010) But there are projects, which try to incorporate user generated content into MMO games as well. The biggest challenge herein lies in the connection speed, which is too slow, because user generated content is not embedded into normal program code. To solve this, a second server for such an MMO is needed. (Dalziel, 2010) A better approach to differentiate the two is by looking at their mechanics. MMORPGs focus mostly on a progression based system, like leveling up the player's avatar. Virtual worlds are more about socializing. MMORPGs also need teamwork, to defeat strong opponents, but it all goes back to the mechanics. (Marcus, 2007)

Typology

To distinguish between different virtual worlds, the following five elements can be used (Messinger et al., 2008):

1. *Purpose*: This specifies which content and information is used during interactions in a virtual world.
2. *Place*: The place where interactions take place: Either completely virtual or only partial virtual.
3. *Platform*: Is the communication synchronous, or asynchronous?
4. *Population*: How big is the user-base and which social connections do users have?

5. *Profit model*: Is the virtual world trying to generate money, or not. In other words, is the usage free, or charged. This element can be further distinguished by the type of subscription cost (one time fee, monthly fee, etc.).

Application and Genres

There are many possibilities for utilizing virtual worlds. The following listing shows only a few examples (de Freitas, 2008):

- *Role playing worlds*: These are gaming worlds and are primarily used for entertainment.
- *Social worlds*: Social worlds are about community, social interactions and content sharing.
- *Working worlds*: They are used in the business sector, for meetings, or document sharing. Working worlds are generally utilized for collaboration, especially when the parties involved are far apart.
- *Training worlds*: Training worlds are used for training in professions. They can also be used for training certain situations, which are difficult, dangerous or impossible in real life. Training worlds are mostly used by the military.
- *Mirror worlds*: They resemble the real world, or parts of it, as close as possible. They can be used to visit places, which are otherwise inaccessible to users.

3.2 Current Examples

Currently there exist a plethora of virtual world projects. Some of them target specific age groups, others are for adults only. (Papp, 2011) This section will give a couple of exemplary virtual worlds, which are relevant at the moment.

World of Warcraft

World of Warcraft (WoW) is probably one of the most successful and most known MMORPGs of the last ten years. The game is subscription based and was released in the year 2004 by Blizzard Entertainment. In 2005 and 2006 it was the best selling PC game worldwide. In 2007 it was the second best selling PC game, only topped by its first expansion. In 2008 three of the top five best selling PC games were WoW related and 2009 three of them were under the best six. In 2010 WoW housed over 12 million players worldwide. (Blizzard Entertainment, 2010) But the user base is currently declining. In the first quarter of 2013, the game lost 14% of its users, which leads to an estimated total of 8.3 million. This is still an impressive number. (Kain, 2013) It is said, the current payment model is the cause of this drastic drop. Because the free to play market grows constantly, the competition is getting just too strong. (Kremer, 2013)

Like many MMORPGs, WoW is similar to old *Pen & Paper role-playing games* (like Dungeons & Dragons). The setting of the game is the world of Azeroth, a fantasy realm inspired by Lords of the Rings (Figure 3.3). There are many different races and classes to choose from at the beginning of the game. After the avatar creation, players do quests and kill monsters to get experience points and loot. After they have accumulated a certain amount of experience points, they level up, grow stronger and learn new skills. Like many MMORPGs

collaboration is necessary, especially when the level maximum is reached. The only way players can get stronger in this phase is by getting better items, which can be obtained by defeating strong bosses. These bosses usually require a large group of players to defeat them. To get up to twenty or forty people at once for this venture is difficult. That is why, like many other MMORPGs, WoW uses a guild system. Guilds can be formed by player groups and are an important social element in the game. (Ducheneaut & Yee, 2008)

Figure 3.3: One of the starting areas in World of Warcraft

Second Life

SL was released in 2003 by Linden Lab. In 2007 it was stated to have 8.3 million users. (Messinger et al., 2009) In 2013, in celebration of the 10th anniversary, Linden Lab published some statistics. They stated, that during the ten years SL is running, 36 million accounts were created. This can be a misleading number, because it does not give the total number of users currently, but rather the accounts which were created in ten years. It was also stated, that 3.2 billion US dollar were made during transactions between users. In 2013, the landmass of SL spanned nearly 700 square miles. Roughly 1 million users tended to visit SL per month. (Linden Lab, 2013)

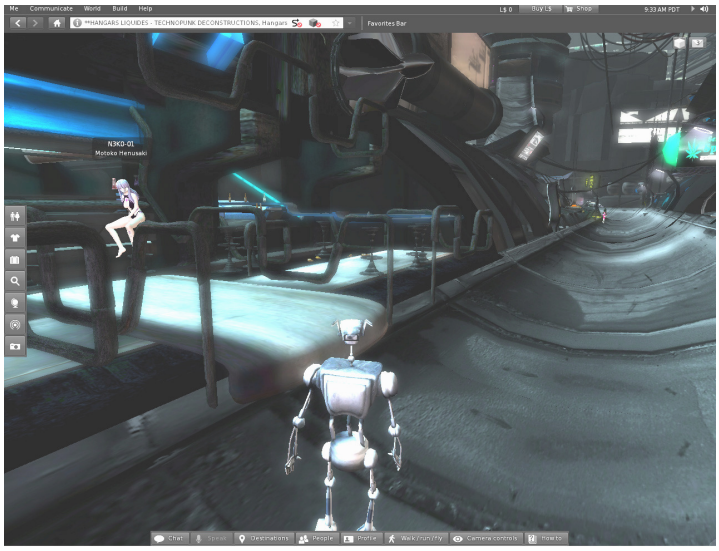


Figure 3.4: A futuristic world in Second Life

The world in SL revolves around user created content. Everyone is able to build objects in the world, even create their own code, to modify the world itself. Figure 3.4 shows an exemplary world created in SL. Another big aspect of SL is the trading aspect. Users can trade their goods and buy territories. To buy these items, SL uses its own currency Linden Dollars, which can be exchanged into real money. This can lead to a real life income through SL. Therefore it is important for SL to support shared content and collaboration. There are also policies, which protect user generated content from duplicating through others. Outside of creating items, SL offers other activities, like games, meetings, chats and dating places. The importance of SL can also be seen through the many real life organizations, which are represented virtually. Some universities for instance have territories for holding their courses online. Even embassies of some countries can be found in SL. (Messinger et al., 2009)

OpenSimulator

The roots of OpenSimulator (OpenSim) go back when the *SL viewer*, which is the client for SL, became open source. A server for this client was built and released in 2009 and was called OpenSim. Today it not only supports the SL client, but others as well. The graphical results may vary from client to client, which is shown in Figure 3.5. This figure shows the same scene, rendered in both the SL and the realXtend client. (Fishwick, 2009) The OpenSim server itself is built with C# and is open source. It runs on Windows with the help of the .NET framework and on Unix via the Mono framework. (OpenSimulator, 2013)

Figure 3.5: Comparison between the SL client on the left and the realXtend client on the right (Fishwick, 2009)

Open Wonderland

OWL was created by Sun Microsystems in 2010 and became totally community supported later on. OWL is completely programmed in Java and is an open source tool for creating virtual worlds. (Open Wonderland, 2013b) Three design goals were in mind, when OWL was created. The first was collaboration. OWL should support all different types of synchronous collaboration. A drag and drop system is implemented to help with sharing documents. This feature allows users to place objects in the virtual world by simply dragging them from anywhere into the OWL client. The object is then immediately represented in the world and can be viewed by every other person in the world (Figure 3.6). The second goal was to create an extensible environment. This is also important for collaboration. OWL only uses some core services to run a virtual world. Everything else is implemented via modules. OWL also supports the use of external data. (Kaplan & Yankelovich, 2011) For instance, it should be possible to import art created with other open source tools. (Open Wonderland, 2013a) The third and last goal was federation. This was a long-term goal that would allow OWL to be like

the World Wide Web, only in 3D. With the help of a browser, users would be able to switch between worlds. (Kaplan & Yankelovich, 2011)

Figure 3.6: Open Wonderland (Open Wonderland, 2012)

realXtend

RealXtend is another open source platform and has its beginnings in 2007. It started with a collaboration of some small companies, which wanted to create a base platform. In 2011 the realXtend foundation was founded to coordinate further developments. The designers had several goals in mind, when they started the project. The first was to build realXtend completely with open source structures. Allowing the incorporation of basically any 3D model, which uses a polygon mesh structure, is one of the key features of realXtend. It also supports the usage of models and scripts in open web libraries. Only the URL is required for this. The second goal was to make the editing of virtual worlds as flexible as possible. Changes are made locally and can be published after they are finished. The last goal was extensibility, which revolved around the addition and removal of the world's functionalities. (Alatalo, 2011) In 2012 Adminotech Ltd launched Meshmoon, a free hosting platform for realXtend virtual worlds. (Nauha, 2012) A screenshot of an exemplary virtual world on Meshmoon is shown in Figure 3.7.

Figure 3.7: Underwater world on Meshmoon

Unity

Unity is a 3D game engine for independent developers (see Figure 3.8). The standard version is free, but limited. A Pro version with all features is also available. The cost for the Pro version is currently at 1500$. (Chu, 2009) Version 1.0.1 of Unity was released back in 2005 by Unity Technologies. (Unity Technologies, 2005) The goal was to create a platform, which allowed everyone to create games on every popular platform. (Unity Technologies, 2013) The unity engine currently supports all common systems, from Windows, Mac and Linux to iOS and Android. Current game consoles, like Xbox 360, Playstation 3 and Wii are also supported. New gaming systems, like Playstation 4 and Xbox One, will also be supported in future releases. In 2013, Unity had over two million registered developers. Approximately a year ago, the amount was around one million. (Unity Technologies, 2013) Adam Frisby, one of the OpenSim founders currently utilizes Unity for creating virtual worlds. His reasons to switch to Unity were its scalability, better graphics and the possibility to run on different systems. He worked on Gojiyo, a virtual world, which was created with Unity. Gojiyo can be compared to SL, but does not feature content creation in the world itself. Other companies, including OpenSim developers, are switching to Unity as well. (Gladstone, 2012)

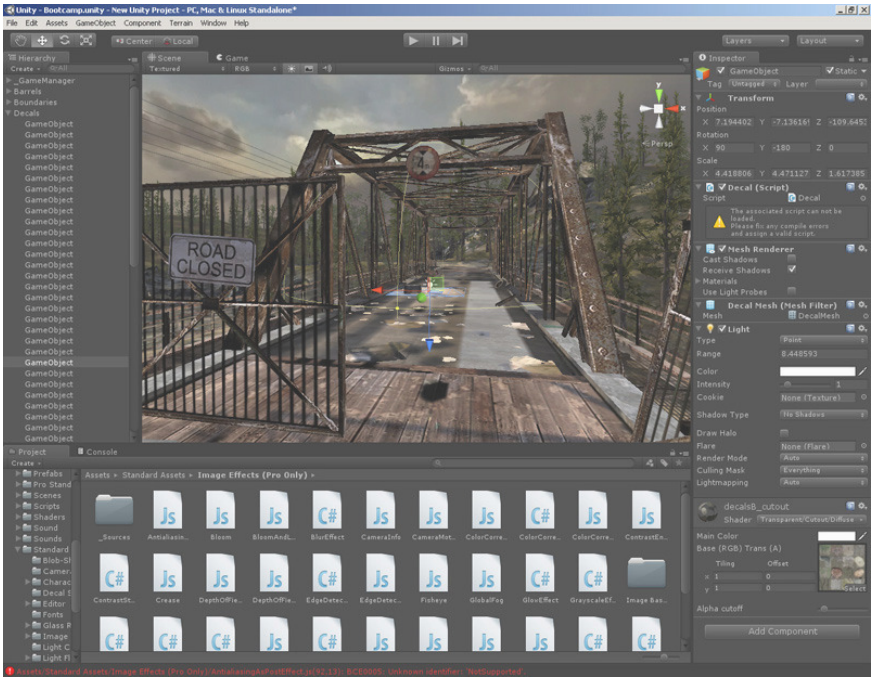


Figure 3.8: Unity editor

3.3 Application Fields of Virtual Worlds

The Internet was a document focused space for the past 20 years, which may change in the near future. The structure will be the same, but 3D links to virtual worlds would be thinkable. (Thompson, 2011) Virtual worlds allow for a myriad of different activities. They have changed the perspective of many people, because they show new ways of learning and collaboration. They also offer high potential in some other areas. (de Freitas, 2008) This section provides a few examples, on how virtual worlds can be used in research and art. It also gives a brief overview of some other possibilities.

3.3.1 Science and Research

Virtual worlds provide the possibilities to create experiments, which are hard to realize in the real world. This can be true for example when a large amount of participants is needed. (Bainbridge, 2007) Furthermore, unlike usual virtual experiments, virtual worlds possess the possibility for collaboration between research groups. (Gütl et al., 2011) The overall utilization of virtual worlds in research seems to be dependant on the research area. (Djorgovski et al., 2010; Fairfield, 2012) Studies are especially popular in the social field. (Fairfield, 2012) Many researches in other areas are currently not aware of the possibilities virtual worlds can hold. There is a steady growth of interest, but it increases only slowly. That is not to say, no research team is trying to find new approaches in their fields. In 2009, the Meta-Institute for Computational Astrophysics (MICA) was established, which only exists in a virtual space. The organization tries to find new ways within virtual worlds to support their research in the astronomy field. (Djorgovski et al., 2010)

Simulation and Data Analyzing

Virtual worlds can be used to simulate physical behavior. One of the key benefits is the possibility to observe the simulation closely. It can also be modified during the experiment. When the simulated aspect is grounded on simple physical laws, the programming effort will be low. But nevertheless, interesting results can still show up. Larger simulations can get problematic, because virtual worlds can not handle such huge amounts of information. The solution to such a problem is, to make the calculations external and import their results. For the visualization of such results, virtual worlds can also be helpful, especially when dealing with numerical output and multidimensional data. Figure 3.9 shows such a representation. It is stated, that data can be represented to a maximum of a dozen dimensions in virtual worlds. Researchers can still interact with it, not only alone, but with the whole team. There are other technologies, which offer 3D data immersion, but their equipment is fairly expensive and allows only one person at a time to use them. In standard virtual worlds, a timed representation can be added, which shows the progression over time. Unfortunately this can be hard to realize, because of technical limitations. There is another problem, which revolves around the maximal amount of objects, which can be represented in virtual worlds. (Djorgovski et al., 2010) Basically it is possible to recreate experiments in virtual worlds. It is also possible to mimic the user interface researchers would have, when they use their usual simulation tools. (Gütl et al., 2011)

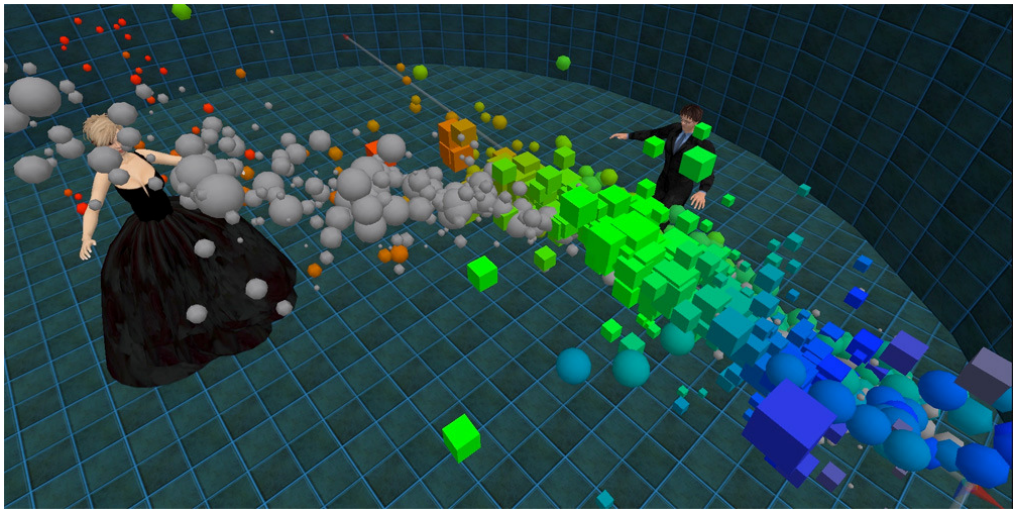


Figure 3.9: Six dimensional (3D-coordinates, size, shape, color) data visualization. (Djorgovski et al., 2010)

Social Studies & Research

Many social studies were already made on the Internet. Virtual worlds however deliver new possibilities for those kinds of studies. A huge amount of people is needed to study and understand complex group behaviors. Virtual worlds can provide researchers with these large groups. (Bainbridge, 2007) There are many studies in social networks and in the gaming area of virtual worlds, but there are few publications regarding other sectors. Comparing 3D virtual worlds and 2 dimensional applications, like forums, there are some differences, which can change the outcome of studies. The biggest difference by far is the user's avatar and its looks. The avatar is not only a representation of the user's identity, but also gives people a sense of space. For example, if an avatar stands in a group of other avatars, the user feels more like being part of the group. (Minocha, Tran, & Reeves, 2010)

Ethic Concerns

An important advantage of virtual world research is that it is not possible to hurt research subjects physically. But it is more likely to do some psychological damage. Publishing usernames, for example, can have the same effect as publishing the real names, at least in the mindset of the research subjects. (Stanton, 2010) Academic researchers usually have to abide the Federal Policy for the Protection of Human Subjects, but other institutes are not bound to this law. They can more or less do whatever they want, when not conflicting with other laws, or rules set by the virtual world platform. (Bainbridge, 2007) To keep the comfort zone and to not violate the privacy of test subjects, it is important for researchers to know which places are private and which ones are public. This differentiation is fairly difficult in virtual worlds. There are other problems, like determining if a research approach is ethical or not: For example, using bots to interview persons and collect information about their behavior during an interview can be seen as acceptable. The reason behind this acceptance is that there is no harm done to the person. But this approach goes against the ethnical principle, to not use people as means to an end. There are much more drastic approaches, like the usage of virtual plagues. They have appeared sometimes in virtual worlds, but are usually errors in the programs. The behaviors in virtual outbreaks do not differ much from real life. Therefore epidemic researchers are considering the usage of virtual worlds for their studies. (Grimes, Fleischman, & Jaeger, 2009)

3.3.2 Creativity and Art

Virtual worlds are novel places to create, collaborate and to exchange ideas. They are often praised for their potential to enhance the creative process. (Burri, 2011) It is true, that the Internet itself also provides such possibilities, but pages like Facebook are far more limited than virtual worlds, when it comes to interactions between people. And the open and shared structure of some virtual world platforms encourages creativity. (Singh, 2012)

New Possibilities for Art

It is assumed that 3D virtual worlds will have a big influence on the art of the 21st century. (Dethridge, 2009) Around 2004 there were some approaches to create art in virtual worlds, which have altered physics (see Figure 3.10). The goal was to create alternate realities in which art can be created and where people can interact with the creation. These virtual worlds were only created with immersive VR in mind. Expensive equipment is required to visit them. (Cavazza et al., 2004) Such concepts do not only work in a VR setup. It is possible to do the

same in virtual worlds, like SL, because most of the real life restrictions do not apply to them. (Ward & Sonneborn, 2009)

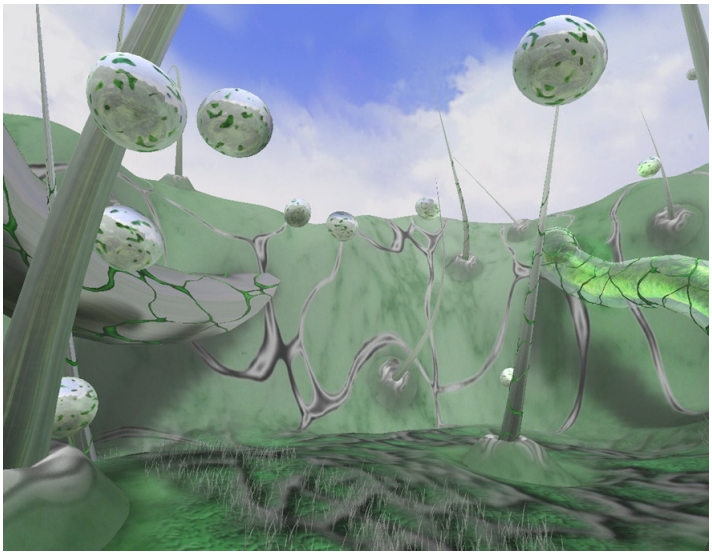


Figure 3.10: Altered physics in a virtual reality world for art purpose (Cavazza et al., 2004)

It should be noted, that there is a big amount of user generated content, which just emulates real life examples. But it is possible for artists to let their imagination go haywire. For instance, avatars usually look like human beings, but there are other possibilities, like robots, or giant dinosaurs. It is also possible for avatars to be only a crystalline structure. Generally art, at least in SL, builds upon real examples, but also on the imagination of its creators. This can also be seen in the environments and objects. There are recreations of art objects, which try to represent their real life example as exact as possible. Other objects are interactive, or avatars can pass through them, to look at their insides. Paintings for example are constraint to two dimensions. In virtual worlds they are not confined by this restriction. There are creations, like in Figure 3.11, which rebuilt paintings in 3D. This allows avatars to wander through these paintings. (Ward & Sonneborn, 2009)

Figure 3.11: 3D recreation in SL of the town depicted in Van Gogh's *Starry Night* (Au, 2007)

Problems and Limitations

Copyright and ownership is a big problem in virtual worlds. Generally speaking user generated content is protected by the copyright law, but it must at least fulfill one requirement: The copyright subject has to be an original product of the author's expression. The copyright only protects the expression of an idea, not the idea itself. Virtual goods have underlying scripts, which have to be seen as literature. Therefore they fall under the copyright law. But the *terms of service* (TOS) or the *end-user license agreement* (EULA) have to be considered as well. These are contracts, which have to be accepted by the users, to use services, like virtual world platforms, or games. In most of the EULAs it is stated that the rights for user created content retain by the platform operators. Therefore the creator does not own the rights, but rather the party which provides the platform. The platform provider can therefore sell and delete content, without the user's consent. The EULA of SL however gives users the rights to their own creations. This can lead to other problems, like copyright infringement. A copyright infringement is simply put a copy of a copyrighted object. It is also an infringement when an object is used and modified without the consent of the creator. In SL it is possible to give other users permission to use, or modify the object. But when users get the rights to modify something, it is unclear which underlying rights they receive. Paired with collaborated creations, ownership can get tricky. When an object is modified several times and through several groups, it is nearly impossible to find out, which user has which rights to the object. SL makes the problem even worse, because only the original author is listed. Co-creators or modifiers are not tracked. (Marcus, 2007)

Another problem can be the limitations by the virtual world. There are some platforms, like SL, which give users much freedom in creating new objects, but there are also many worlds, which restrict their possibilities. WoW, for example, lets users create content, but in a fairly limited way. The world itself was created by the developer and users can only build upon it. Too much freedom, like in SL can also be negative. It can overwhelm users and ruin their experience. It even can destroy their immersion. (Burri, 2011)

3.3.3 Other Applications of Virtual Worlds

This section contains a rundown of other possibilities in virtual worlds. They will be kept short, because they are not the focus of this work.

Business Potential

In the business sector, virtual worlds can be used for communication and meetings. (de Freitas, 2008) A lot of money can be saved when meeting virtually, because traveling is not required. Another interesting possibility is to recruit employees in virtual worlds. People may show skills in virtual worlds, which they might not show in real life. For instance, companies employ people to analyze the social behavior of potential recruits and study their skills, like leadership. (Papp, 2011) Other interesting possibilities lie in co-creation, in which the consumer plays an active part in the creation of a product. It can be seen as collaboration between customer and manufacturer during development. Virtual worlds provide the benefit of interactive collaboration in real-time and they encourage experimentation with their creation tools. Unfortunately, only few users tend to participate in co-creation. Users are generally more focused on their own activities and not interested in taking part of activities from others. (Kohler, Fueller, Matzler, & Stieger, 2011) Virtual Worlds also allow people to acquire and train their business skills. Because these skills are mainly focused on social

activities, like selling, other tools may not be as suitable to improve these abilities. (Pirker & Gütl, 2012)

Education Potential

It is stated, that normal virtual learning environments improve the learning effect, but they are rather limited in the social area. Therefore, they can not ensure the acquisition of deeper skills. Virtual worlds improve communication and collaboration skills. They also provide interactivity with the learning material and prepare students for using their skills in real life. (Wood & Hopkins, 2008) Another big factor is the current change of perspective in education. Some students would like to attend their courses remotely, or they might want to visit lectures at different universities. Therefore a lot more flexibility is needed, which can be provided by virtual worlds. (Gütl, Chang, Kopeinik, & Williams, 2009) Teachers have the possibilities to be creative with the learning material. For example, they could create a microscopic world, which can be visited by students, who can then interact with different kinds of cells. A recreation of a piece of literature is also thinkable, where students can witness it for themselves. It can go even further, like the recreation of a specific time period. Students can then role-play in these worlds. They do not have to watch it from afar, they play an active part in them. (Helmer & Learning Light, 2007) Not only periods can be recreated, but also historical events. Students can take part in a famous battle and see if they can make a difference. (Papp, 2011) This leads to the next possibility: Training in hazardous environments. (Helmer & Learning Light, 2007)

But not everything is positive though. There are some problems when it comes to learning in virtual worlds. Technical difficulties, like program stability and the need of high computational power, play a huge part in this. Furthermore, the interface for virtual environments can get fairly complicated and intimidating. (Helmer & Learning Light, 2007) There are still some problems with virtual worlds as education tool. First of, a learn environment has to be build. The second problem is the provision of tools for the lecture. And the third problem is that the creation and preparation of the lecture is time consuming for the teacher. (Gütl et al., 2009)

Space Flight

Another interesting opportunity for creative utilization of virtual worlds is long duration space flight. With the help of virtual worlds, in-flight countermeasures against strain can be developed, like asynchronous meetings with ground personal or relatives. Meeting people in virtual worlds can have a positive mental effect, because in virtual worlds avatars can interact with each other. On a long flight it might be an effective stimulus to hug a closely related person, even if it is only virtually. A virtual artificial intelligence (AI) is also thinkable, which acts as a counselor. Astronauts can interact with it and get help for social problems. Another possibility is the usage of virtual worlds for relaxation and stress reduction, like a virtual vacation. (Morie, Verhulsdonck, Luria, & Keeton, 2011)

3.4 Flexibility and Dynamics of Virtual Worlds

As shown in the previous sections, virtual worlds have many opportunities and many of them build upon the need of users wanting to create and modify these worlds. Unfortunately, they usually have to create everything by hand in a low level of abstraction. A large project, like a big environment, can get tiring for the user. Modifying a complex landscape can get even

harder. Sometimes the whole project has to be started from scratch. (Smelik, Tutenel et al., 2011) And because of the highly varied use of virtual worlds, environments are needed, which are flexible, easy to create and easy to change. It also should be possible to adjust them to their specific requirements. (Gütl et al., 2014) These methods should all be easy and fast to use. (Smelik, Tutenel et al., 2011) For object creation there exist many different possibilities, like Google SketchUp, or Blender. Some virtual worlds offer their own object creation tools, like SL. Many of these object creation platforms have an object repository, for sharing objects with other users. This chapter however will not focus on simple object creation, because of the huge amount of already existing free tools. (Gütl et al., 2014)

3.4.1 World Generation

An easy way to simplify the creation of a virtual world is to use automatic generation. These *procedural methods* are able to create landscapes, vegetation, rivers and other sceneries. (Smelik, Tutenel et al., 2011) Landscapes are created through *height-maps*. A height-map is a grid, where every cell contains a height-value. There are plenty of different methods to create such height-maps. Seas and rivers can be created during the landscape process, where the height-map is calculated. Rivers are calculated first and with this information, the height-map is created. Inserting rivers in an already existing height-map is also possible. The height-map is analyzed and rivers are created in places where they can flow naturally from mountains downward. Plant creation is done through starting at the roots and building on top of it until branches and leaves are reached. The addition of these parts is done through grammatical rules. Plants are then placed according to rules based on ecosystems. For street generation, there are many different methods, like pattern based generation, or the usage of templates. Like plant growth, it can be seen as a growing process. Urban environment generation uses street networks, which are close together. The algorithm tries to find areas surrounded by streets. These areas are then divided and buildings that fit in the empty spaces are placed. (Smelik, 2011) Procedural methods may relief some of the workload a creator has, but combined with manual user creation they may not fair as good. Users do not have much control over the procedures themselves. Furthermore, it is often not clear, which input parameters lead to which output. This often results in trail and error attempts to get to the outcome they desire. Given the long runtime of these algorithms, it can get frustrating for users. Because of the specific nature of these procedures, one can only be used for their specific purpose. Therefore a tree creation algorithm can not be used to create the landscape. In the end different types of these procedures have to be run and it is up to the users to manually put everything generated together. This is the reason why simple procedural methods are not fit for the usage in virtual world generation. (Smelik, Tutenel et al., 2011)

Declarative Modeling

To make procedural generations work better, they must have more intuitive input as well as better control through users. The method presented in this section tries to encompass the advantages of both, procedural and manual methods and is called declarative modeling. The method is split into two parts: *procedural sketching* and *automated consistency maintenance*. Procedural sketching is done by the users, who draw a rough sketch of the landscape. This is shown in Figure 3.12 on the left. It consists of two modes, which represent the virtual world in a top-down view (Smelik, Tutenel et al., 2011):

- *Landscape mode*: The landscape is segmented into a grid. The users color the grid with ecotypes, which not only represent the height of the grid piece, but also the material. The grid size itself is adjustable.

- *Feature mode*: In this mode, the features of a landscape are defined. These features include forest, streets and rivers, just to name a few. The features are represented by vector lines and polygons.

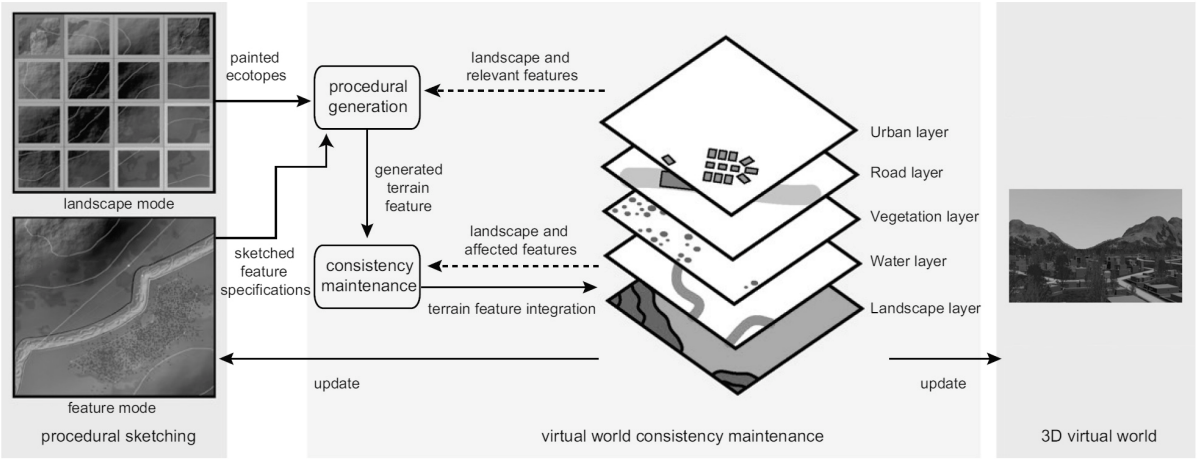


Figure 3.12: Declarative modeling process (Smelik, Tutenel et al., 2011)

After outlining all the features, the environment has to be generated, while checking for consistency. This is important, because all objects and features affect each other. To make everything consistent, a semantic heavy model for features and their relations is necessary. The model consists of five layers, which is shown in the middle of Figure 3.12. The environment features have 3 levels of abstraction (Smelik, Tutenel et al., 2011):

- *Specification Level*: User input and specifications for the given feature.
- *Structural Level*: User outline in the sketch. Basically the area and layout of the feature.
- *Object Level*: The objects that make out the features, for example the trees of a forest.

The generation of every layer can be done in two steps: First the next abstraction level is generated from the current one, beginning with the specification level. The second step is to validate the current abstraction level. A problem can occur, when two features overlap, or the feature does not fit onto the set location because the height does not match with the feature. To resolve this problem, every feature has two request types (Smelik, Tutenel et al., 2011):

- *Claim*: The feature claims the area for itself. There are only two possible outcomes, either the approval or rejection of the claim. A claim can only be done either for the structural level or the object level. This prevents the rejection of a feature, because of tiny objects being in the way.
- *Modification*: The feature can try to change the area, to fit its specifications.

An *interaction* between two features occurs, when both try to claim the same area. There are two possible outcomes to such a situation: A *cooperative solution* or a *conflict solution*. In the cooperative solution, one of the features still claims the area and the other is rejected, but the two features get connected. These connections are abstract, but in the object level a specified object is placed for the connection. For instance, a street and a river result in a bridge object. These connections have to be defined before, but it is unfeasible to try to define every possibility. In the conflict solution, one feature wins and the other is not able to use the area. The conflict area will then be removed from the feature area, if the conflict happens on the structural level. In the object level, the object cannot be placed. There are three different types of priorities to help in finding solutions. They state, whether a feature should try to claim, connect or conflict an overlapping area. These priorities are set by users and are used as

default, but features can contain functions which calculate their priority from context. (Smelik, Tutenel et al., 2011)

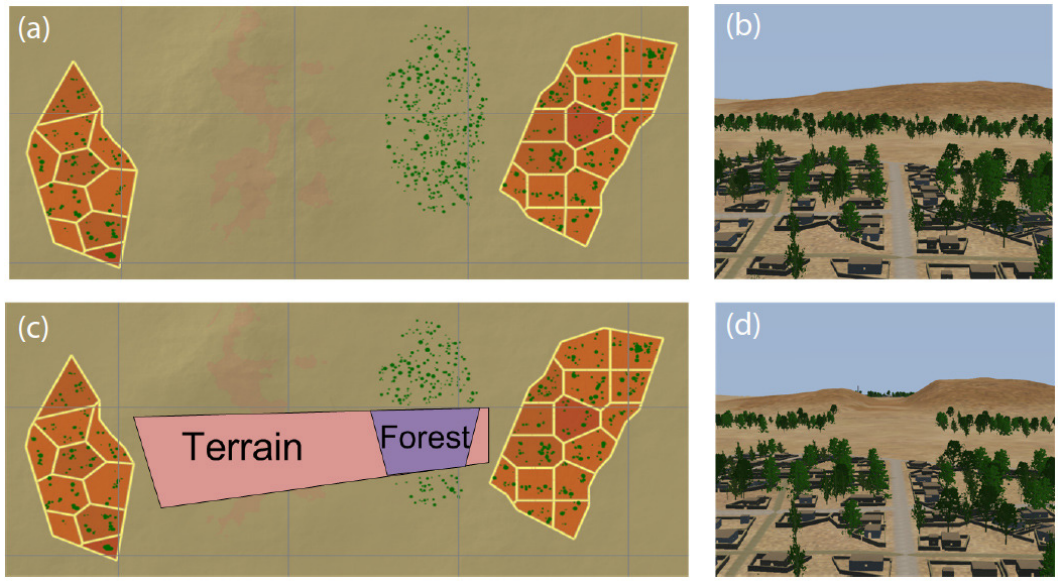


Figure 3.13: Appliance of line of sight constraints in the colored areas (Smelik, Galka, de Kraker, Kuijper, & Bidarra, 2011)

To give the designer more control over the generation process, *semantic constraints* can be defined in an area, which is called *constraint's extent*. Constraints are important in order to make the environment more plausible. *Feature constraints* are sub constraints from a semantic constraint and are used for specific features. For example it is possible to set the maximal height of vegetation. In Figure 3.13 a line of sight constraint is defined. Two cities are separated by a hill and a forest (a, b). The line of sight constraint is applied, which changes the terrain and removes parts of the forest for the two cities to see each other. It also changes the height of the hill between them (c, d). *Associated relationships* are also used for helping the designer. They create a link between a constraint and a feature in a specific context. Figure 3.14 shows an example: A constraint is set to create a road between two points, which are marked by a flag (a). The constraint then uses the existing roads (b) to create the connection. If the city is removed, the constraint is notified and a direct connection is possible (c). To keep track of all the feature constraints, every feature has a stack of constraints. When checking for consistency, the stack is checked in sequence. If a conflict occurs, the constraint parameter is changed, or the constraint is deactivated. Conflict resolution is similar to the feature conflict and done by claim and using priorities. (Smelik, Galka et al., 2011)

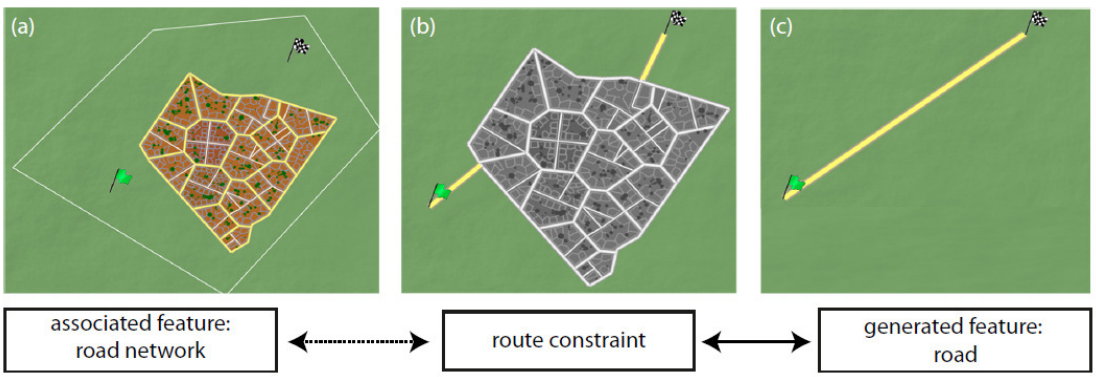


Figure 3.14: Road constraint, which uses existing roads, if present (Smelik, Galka et al., 2011)

Rectangular Dualization

The previous method is generally used for environment creation. (Smelik, Tutenel et al., 2011). The rectangular dualization method is more suited for creating a floor plan out of a planar graph. (Bogdanovych & Drago, 2006) First, an explanation of rectangular dualization is needed: It can be seen as partitioning a planar graph into rectangles, which do not overlap and no more than three rectangles join edges. All these partitioned rectangles also form one big rectangle. Vertices can be seen as neighbors, when two rectangles have common boundaries. The rectangular dual is needed for generating a two dimensional floor plan. But not all graphs have a rectangular dual. They only have one, when there are no *separating triangles*. A graph without separating triangles can be seen like *4-connectiviy* in triangulations: When three random vertices are removed, the graph is still connected. (Bogdanovych & Drago, 2006)

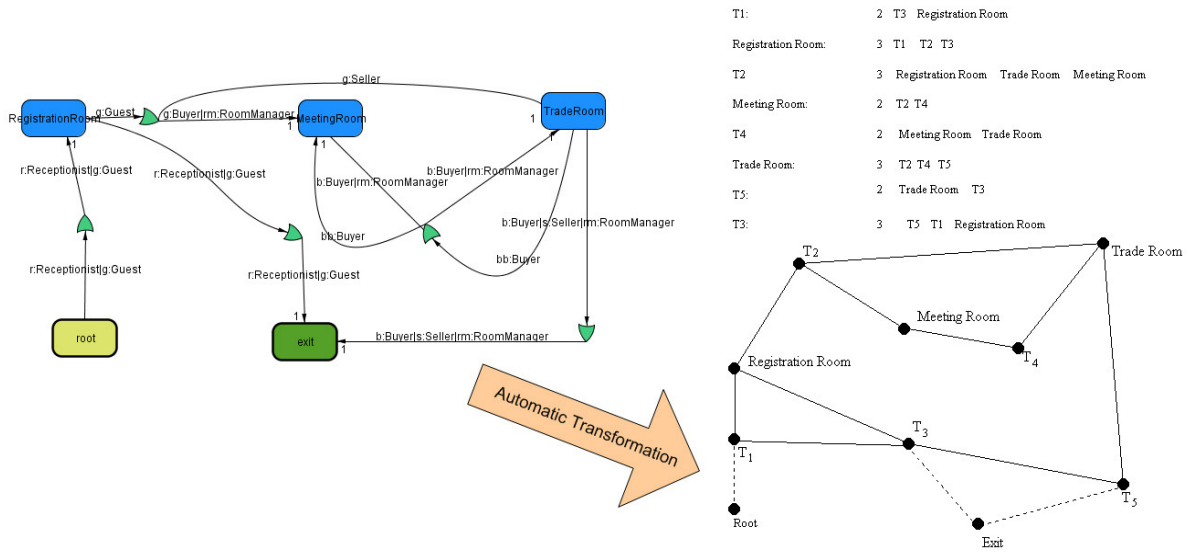


Figure 3.15: Graph transformation into rectangular dual (Bogdanovych & Drago, 2006)

To generate a floor plan, four steps are necessary (Bogdanovych & Drago, 2006):

1. All redundant information is removed and the graph is conversed. To prevent separating triangles, *crossing triangles* have to be added. They are inserted at an arbitrary edge of every separating triangle.
2. From the conversed graph, the rectangular dual is generated (Figure 3.15). Transitions and previously defined places will be created as rooms and connections as doors. Inserted crossing triangles are removed from the graph and the sizes of the adjacent rooms are altered according to the removed triangles. This step produces the two dimensional representation of the floor.
3. With the 2D map of the floor, a three dimensional representation is built, which is a trivial process. The size of the rooms is changed to be big enough to house as many people as previously defined.
4. Visualization of the floor.

Shape Grammar

This method is somewhat similar to the previously discussed rectangular dualization. But rectangular dualization did not allow for much influence by the creator. Shape grammar does allow some freedom in that regard. Shape grammar builds complex items through simple

objects and rules. These rules state, how objects interact with each other. One shape is defined as starting shape. New shapes are created through iteratively applying a rule-set to an already existing shape. This method is called *shape derivation*. An example is given in Figure 3.16. The rule applied to the shapes is always the same and can be seen easily in the changes from the first to the second picture. To accomplish this, a vocabulary is needed, which consists of object definitions and their properties. (Trescak, Esteva, & Rodriguez, 2010)

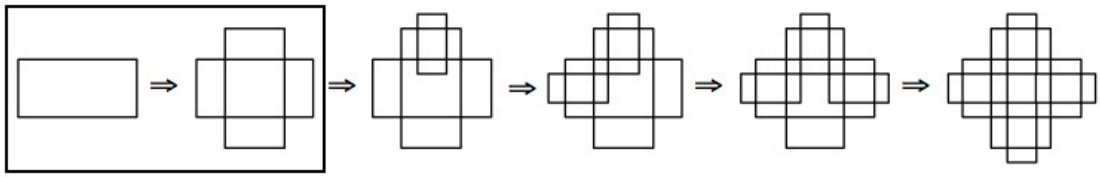


Figure 3.16: Shape derivation (Trescak et al., 2010)

The shape grammar concept has three different components (Trescak et al., 2010):

- *Design wall*: This is the basic component. It is used for building complex shapes. It has different geometrical properties, as well as a type, which is among other things needed for rendering a wall later on.
- *Design space*: This is an area, which represents an object. The object will be created when transforming the 2D floor plan into 3D.
- *Design block*: This is a shape for the shape grammar and can be created with a couple of walls and design spaces.

Basically there are only two types of rules needed for creating a floor plan, or a hand full of buildings. The first is the creation of a room or a building in different places. The second sets rooms according to previous built rooms. To be consistent, a validation has to be made, to remove overlapping objects. Figure 3.17 shows the outcome of this method on the bottom. The top row contains the shapes and their rule set. The first shape is the initial one. (Trescak et al., 2010)

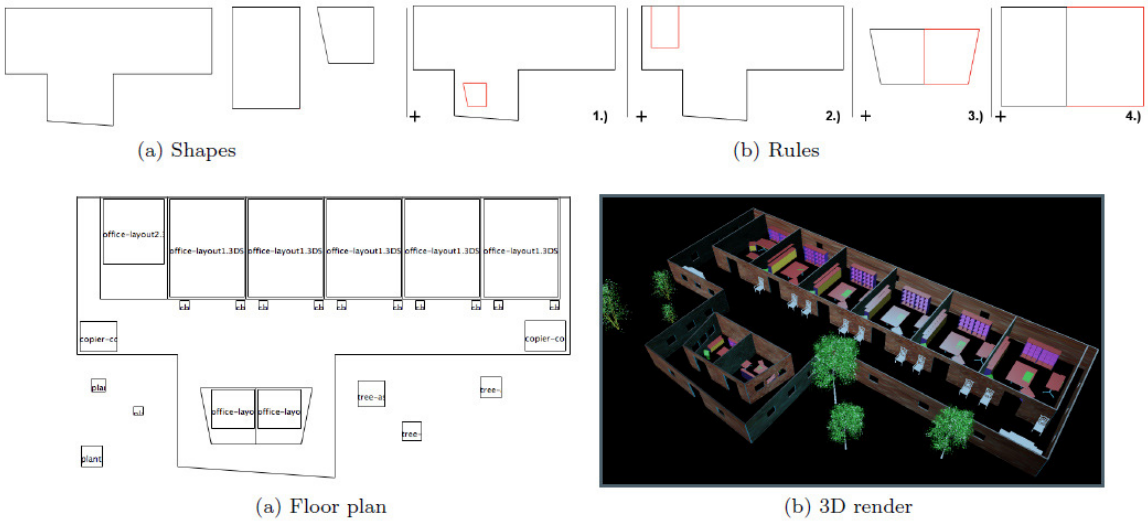
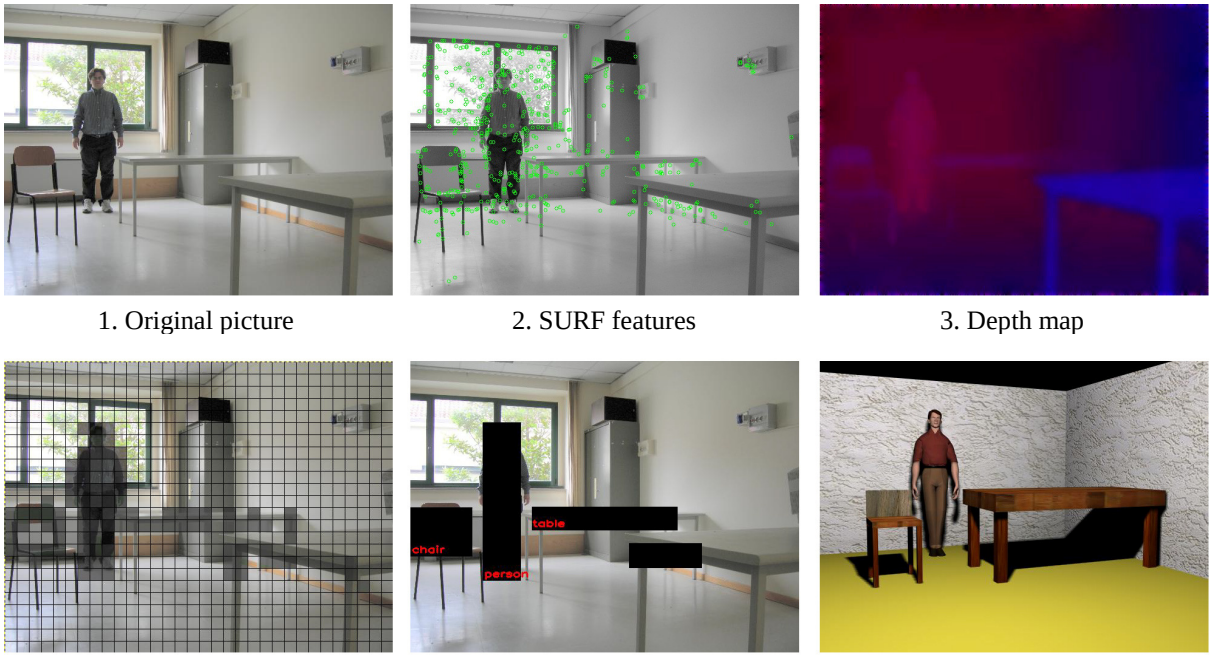


Figure 3.17: Results of shape grammar method (bottom), with shapes and rules on the top (Trescak et al., 2010)

Generation through Real World

With this method, a virtual world is created through a real world picture. No environment is created, only its objects. These objects are picked through *regions of interest* (ROI) detection

and classification. Only edges are used for classification, because it is the easiest way to determine an object. That is because different lighting does not influence the shape and objects of the same group have similar shapes. To create a picture, a stereo camera is necessary. The pictures from the scene are combined and the depth for every pixel is calculated. To find ROI several calculations have to be made. The first step is finding features, which is done by the *Speeded Up Robust Features* (SURF) algorithm. This algorithm does not only find features, but also tries to describe them. The second calculation is the comparison of depths. If some pixels share the same depth value, they feature the same object with high probability. The two calculations are then combined in a basic beliefs function, which results in a three dimensional belief image. The contours are then calculated from the images and declared as ROI. It should be noted, that an additional height control is necessary, because the total height of the object may not be in the ROI. Every ROI is then processed separately. The classification of objects is done via a *pseudo two dimensional hidden markov model*. After every object is classified, a three dimensional *object map* is built. An object map contains shape and position of objects in the environment. Sizes and positions of the objects are estimated from the image. The map can then be used as input for graphical tools. The whole process is shown in Figure 3.18. (Moro, Mumolo, & Nolich, 2010)



1. Original picture 2. SURF features 3. Depth map

4. Areas of interest 5. Object classification 6. Result

Figure 3.18: Object mapping procedure (Moro et al., 2010)

3.4.2 Adaptivity

Adaptivity is important, when it comes to virtual worlds. It not only provides better usability, but can also make the usage of virtual worlds more efficient and less time consuming. An adaptive rout planer, for example, can help users with orientation. The problem with adaptivity in virtual worlds is the three dimensional space. It is hard to find suitable algorithms, because most of them are designed for two dimensional areas. Changing them to 3D can lead to problems. Modifying object code, their behavior, or creating objects in 2D is relatively simple. In 3D however, it has to be ensured that no object overlaps another, which can lead to immersion breaking instances. Furthermore objects have to be visible for the users, when they have to interact with them. (Chittaro & Ranon, 2007) Generally speaking,

there are two levels of adaptivity. The first level is during the setup of world parameters and the second is on runtime. (Gerbaud, Gouranton, & Arnaldi, 2009) The following sections will focus more on runtime adaptivity.

General

Adaptation can be separated into different categories when it comes to in-world objects. Of course, it is possible to combine these categories (de Troyer, Kleinermann, & Ewais, 2010):

- *Graphic*: Changes the graphics of an object. This can range from size, or texture, to visibility and the selection visualization.
- *Behavior*: This is important for dynamic objects. It changes the behavior of objects, or discards behavior in specific situations.
- *User Interaction*: This changes the possibilities users have to interact with objects. Interactions can be prohibited to not flood users with possibilities. On the contrary interactions can also be made available to users as a reward.
- *Avatars*: Changes graphics and behavior of an avatar.

Out of the four types, behavior and interaction adaptivity are the more interesting ones. Usually three different approaches exist to implement adaptivity in a system (de Troyer et al., 2010):

- *Author-driven*: The creator has total control over the adaptation process and models it during the design process. But there is one problem: The author has to predict every possible situation that can ensue. This is unfeasible.
- *Model-driven*: Adaptation is created during runtime, with the help of models and intelligent algorithms. This eliminates the problem of the author-driven approach, but also introduces another problem: It is hard to predict how the adaptation will be.
- *Admin-driven*: This is human-driven, like the author-driven approach, but the human component is used during runtime. A person monitors the adaptation process and controls it. This person is needed for the whole process.

Interaction Adaptivity

Intuitive interactions are inevitable when doing complex work in virtual worlds. Because of the freedom in dealing with objects, it is hard for users, to find the suitable type of interaction. To improve usability, the following framework is proposed for three dimensional interfaces. First of all, it should be adaptive, personalizable and it should automatically find the appropriate interaction for an object to a given situation. Furthermore, it should be attuned with user characteristics and preferences, as well as previously actions executed by each user. To create a knowledgebase for the adaptation process, interaction information has to be collected. Also, a *user model* is needed, which contains repeating patterns. The user model has to be constructed during the monitoring. In the beginning only a general user model and a group user model is used. Combining knowledge and user model, the adaptation process can present the most likely interaction possibilities to users. The adapting process consists of three steps. The first is *mapping*, which is a context-sensitive selection of interaction possibilities. The second is *predicting*, which selects interaction possibilities through the previously made actions on a similar object. The third is *adapting*, which either adapts the interaction, or enhances it through feedback. (Octavia, Raymaekers, & Coninx, 2009) Celentano, Nodari, & Pittarello (2004) propose a similar approach. User interactions are monitored and analyzed.

Then a pattern search is done, which results in an adaptive interaction. The biggest difference in this approach is the possibility to change the parameters of the adaptivity. It can not only be chosen, whether or not the adaptivity occurs, but also when and in which scope.

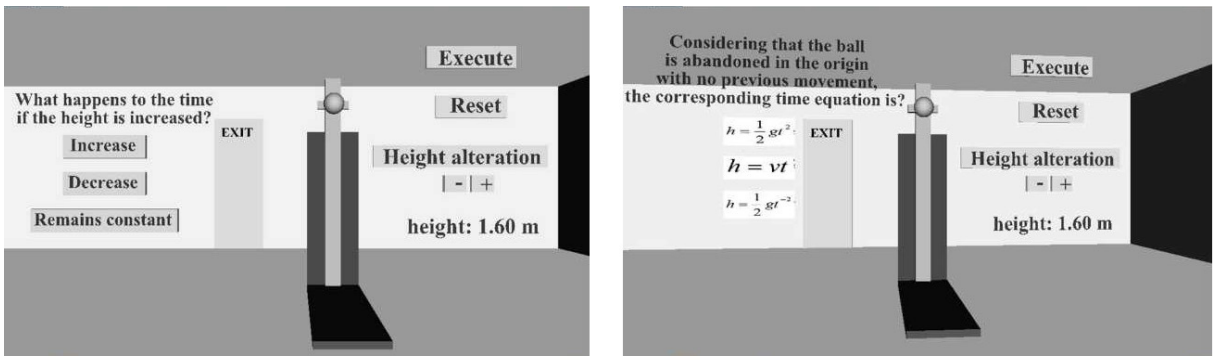


Figure 3.19: A test for inexperienced users left and for experienced ones right (de Aquino & de Souza, 2012)

The next approach does not focus on interaction itself, but rather on the environment. It uses real time adaptivity, which is achieved with the help of a multi-agent system. This system is responsible for creating and maintaining the environment as well as changing the world based on the knowledge it has of the current user. Like in the previous approaches, a *user model* is necessary, but this time it contains user characteristics and behavior. To keep track of world changes, an *environment model* is also needed. The changes are registered by sensors, which detect user input. Four agents are needed for this approach: The *manager agent* surveys users and world. It also controls the other agents and can assign tasks to them. The *personal agent* is responsible for the user model and updates it, if necessary. The *environment agent* knows the current world state and updates the environment model when changes occur. The *updating agent* updates the world itself. Information from the other agents is used to learn the structure of all changes done by the user. This allows the agent to transform the world according to user behavior. The world can also be changed according to the knowledge users have, which is saved in the user model. This leads to other possibilities, like locking certain areas for users, until they are ready for them. Figure 3.19 shows a test based on the user knowledge. The left test is presented to users not familiar with the task. The right test is for users with a higher education level in this particular field. (de Aquino & de Souza, 2012)

Another kind of interaction is group interaction, or better collaboration. Collaboration can also be enhanced by context-based adaptivity. The group context in the proposed approach is based on the context of every individual user in the group. Inner factors, like the role a user has in the collaboration process, as well as outer factors, like the used devices, all play a part in this context. The user model is therefore built for every single user. The model then tries to find roles for the user through estimations on how long the user needs for specific tasks in these roles. Afterwards estimations for a combination of roles are calculated. The combination with the lowest approximated time is then selected for the user. (Beznosyka et al., 2012)

Behavior Adaptivity

The focus of this section is character behavior or AI and *AI agents*. Agents tend to be static and contain preprogrammed behavior. Because everything has to be programmed in advance, creating them can be costly. Static agents do not seem to be believable, because when they make a mistake, they will make the same one in the future. Therefore an AI is needed, which learns from experience. The following are problems, which need to be addressed, when creating behavior in a virtual environment (Mehta & Ram, 2009):

- *Real-time*: Because everything runs in real-time, calculations have to be done in a short amount of time.

- *Human component*: Users are able to influence the environment and change its state. The AI has to react quickly to such changes.
- *Large decision space*: The decision space is fairly complex, which excludes the usage of search based AI.
- *Behavior authoring effort*: The effort for programming an agent is fairly complex, because most of it has to be author-driven.
- *Unexpected scenarios*: Prediction of all possibilities is hard, when designing the agent.
- *Support tools*: Because the agent is created by humans, errors may be programmed into it. Therefore tools are needed to prevent those errors.
- *Behavior replayability and variability*: When agents are static, they always behave the same way.

One method uses the agent's knowledge of its own state. The agent is able to change their behavior through evaluating its state. Every behavior it makes is monitored by itself. When an error occurs, it uses the world as feedback and searches for the point in the behavior it did something wrong. Then it tries to improve this point. The whole process consists of three parts. The first is *trace recording*, which saves every important event in a trace. This part also contains the own behavior state and the world's state. Failure detection then checks the trace for erroneous behavior. When a wrong behavior is found, the *behavior revision* tries to change the behavior. This happens with the aid of *behavior modification routines*, which are a combination of basic operators. They modify behavior, add new elements into the equation, or resort the already existing ones. Behavior itself consists of three layers. The *behavior representation layer* has a behavior library among other things, which is a set of behaviors to fulfill predefined tasks. The *behavior execution layer* selects and executes behavior sets in real-time. The behavior sets are selected through the world's state. These two layers are combined into one big layer. The last layer is the *reasoning layer*, which contains the execution trace. It finds and improves erroneous behavior. Furthermore it monitors every behavior which is done by the agent. Because of the complexity of the world, the trace can be long, which can result in long search times. To prevent this, failure patterns are used to find errors. Failure patterns are predefined patterns, which can identify erroneous behavior. (Mehta & Ram, 2009)

Buche (2012) proposes the following three behavior types for agents. First an agent has to learn by doing, either supervised or not. Supervised learning is done with an expert, which controls the learning process. Learning without supervision has only influences through the world and needs adaptation in real-time. Adaptation can be done through a simulated world in the mind of the agent. An agent has the possibility to start an inner simulation, which simulates itself and the environment. Humans can be a guide for this type of adaptation. They are able to control the agent and show behavior it can later accumulate. The learning process is different and fairly dependant of the environment. Furthermore, for believable behavior same agents should have individual behaviors.

Buche (2012) also describes several different projects. The first tries to create believable agents for video games. Learning through imitation is the best way for adaptivity in such a scenario. This means, agents study behavior of human players and try to imitate it. The project grounds on *Le Hy's model*, in which an agent has a set of internal and external sensors, as well as motors. The motors describe movement, like jumping or rotating. The current decision is based on the previous decision and the sensory input. It is then calculated with a probability distribution. The calculated value is used to select a motor. But the model of Hy was not very believable and needed to be changed and expanded. This resulted in better

performance. Letting the agent learn and not helping it by entering decisions by hand does produce wrong associations, which can destroy the illusion of playing with another human player.

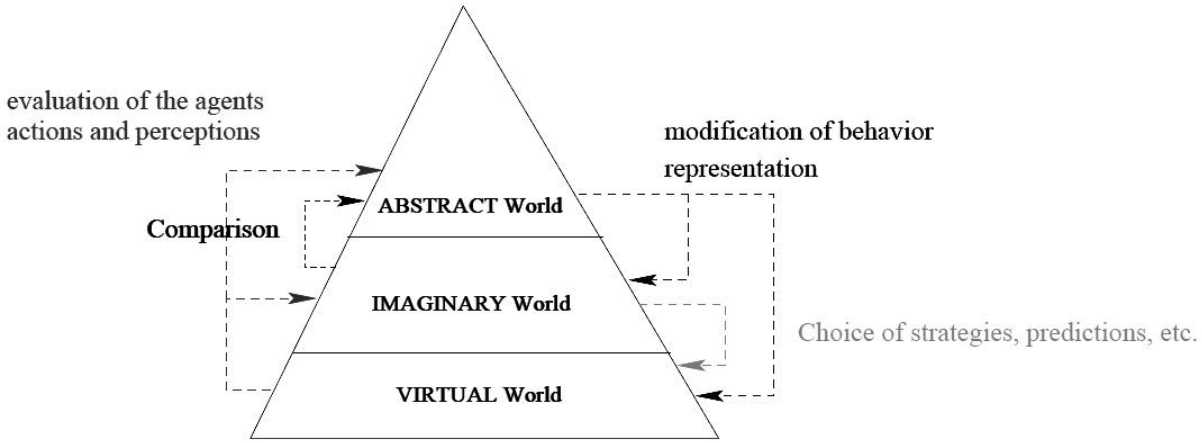


Figure 3.20: Multiple agent dimensions (Buche, 2012)

The next project is a decision making agent in a virtual world. In decision making, anticipation plays a huge role. Anticipated behavior uses past, present and an estimated future to create behavior. The presented framework uses an internal simulation with included anticipation. The agent simulates behavior in its *imaginary world*, which is parallel and asynchronous to the virtual world it represents. As shown in Figure 3.20 a third item is used, called *abstract world*. This world is used for learning and adapting behavior. This framework can be used for movement. A juggler was given as an example. The program the juggler is based on predicts the area the balls might fall and moves the hands according to this prediction. It does not calculate the exact path of the balls. (Buche, 2012)

The third project incorporates *fuzzy cognitive maps* (FCM). FCMs are oriented influence graphs in fuzzy mode. Figure 3.21a shows an example. The fear of an agent is dependant on the distance of an enemy. If the enemy is far away, the agent has less fear. If the enemy is closer the fear rises. When its fear is higher, the agent has a higher probability to try to escape. This example shows only external influence, but for believable behavior internal influences are needed as well. Figure 3.21b adds two new variables to the FCM. The first variable, γ, is used to make the agent feel more fear, when its fear is high. The variable λ is used to make the agent believe the enemy is closer than in reality, according to its current fear level. The learning process is done via learning through imitation, and adaptation is created only by changing the weights on the transactions in the FCM. (Buche, 2012)

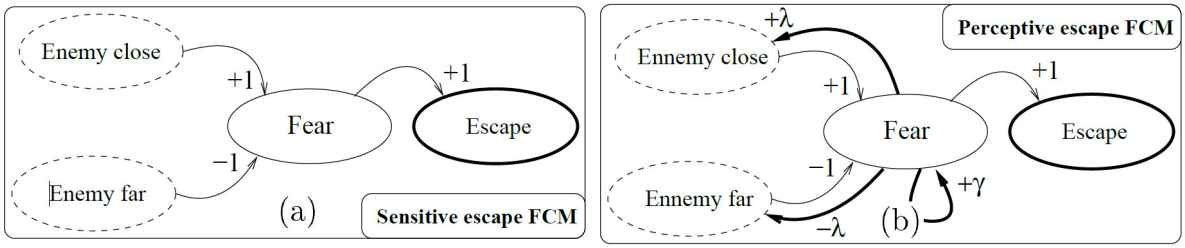


Figure 3.21: Fuzzy logic applied to a situation (Buche, 2012)

The last project describes an *intelligent tutoring system* (ITS). Such systems already exist but are more specific and dependent on the environment. ITS tries to be more generic. Therefore concepts of the ITS should be easy to add, remove and to modify. The approach uses the 4 ITS models: The *domain model* contains the environment, semantics for an internal representation of the environment as well as information about the task. The *pedagogical model* is used to create own knowledge on the basis of already existing knowledge and is used

as context for decision making. Then there is the *learner model*, which is responsible for the learner and the *interface model*. Two new models are added to the four existing: The *error model*, which is used for finding errors and the *instructor model*. With the instructor model it is possible to set the knowledge for the current lecture. For every model, an agent is used and all agents communicate with each other. The following steps are used by the ITS to help the learner. First, the agents observe the learner. Then they try to find errors in the learner's behavior, through analyzing the actions that were made. This is done in the learner model. The result is compared with the target, which happens in the domain model. When an error occurs, the error model tries to identify it. With the help of the domain and learner model a set of appropriate aids for the learner is generated, even if there is no error. Finally one of the assistances is selected and offered. (Buche, 2012)

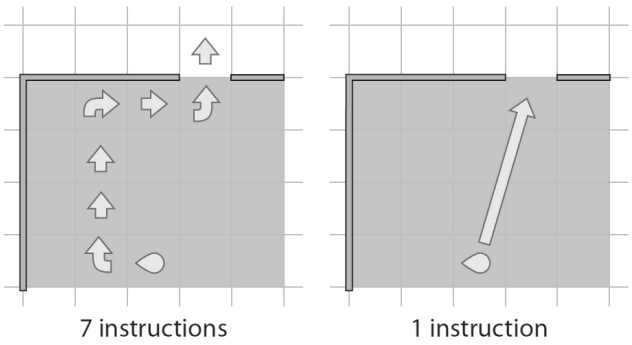


Figure 3.22: Comparison between step-by-step left and higher level instructions right (Dionne, Puente, Leon, Hervas, & Gervas, 2009)

Dionne et al. (2009) describe a virtual guide that provides instructions adapted to the user's progress. First, sets of instructions have to be constructed, which are needed to finish a task. These instructions have to be executed in order. There are a couple of algorithms, which can translate these sets into natural language. Usually they are step-by-step instructions, which can tire a human user. An example is given in Figure 3.22, where the step-by-step instructions are made for every step the users have to take, which results in a couple of instructions. Therefore it is better to use the environment to create higher level instructions, which are not detailed, but much more human. Instead of presenting every step to leave the room, the guide can just tell the users to leave it. For communication a multi-level tree is used. The leaves represent the lowest possible level of instructions. Higher nodes couple the lower nodes together. Lower level instructions are not just there to consume empty space, but will be selected, if users need a step-by-step instruction. Every instruction in the tree has pre and post conditions. The selection of the current instruction is done as follows: Through the world state the current progress of the users can be determined. For selecting the instruction level, pre and post conditions are used. If the post condition of an instruction is achieved, it can be removed from the tree, because it already has been solved. The next instruction with satisfied pre conditions is then selected for showing to the users. When the pre and post conditions are not met, the abstraction goes one level deeper. The selection process is done with the help of agents. There is another part, which has to be considered. When users make an action, which can result in a dangerous outcome, they have to be warned. In such a situation, the instructions are not important anymore. The agents monitor users if they are doing something dangerous. Whenever the possibility for a dangerous outcome exceeds a certain threshold, the agents warn the users. There are four types of agents: The *information agent*, which tries to find certain things in an area, like hotspots, the *status agent*, which monitors the users, the *area agent*, which monitors special areas, like hazardous zones and the *alarm agent*, which watches the user for dangerous actions.

Input	Extracted Information	Knowledge Base
Do you have any red couches?	#hasMainColour(#Sofa,#Reds)	{f:Sofa_Alatea f:hasMainColour f:Reds} {f:Sofa_Franky f:hasMainColour f:Reds} {...}
I would like a leather one.	#hasMaterial(#Sofa,#Leather)	{f:Sofa_Alatea f:hasMaterial f:Leather} {f:Sofa_Alatea f:hasMaterial f:Leather} {...}
Who is the boyfriend of Madonna?	#hasBoyfriend(#Madonna,?)	{f:Madonna f:hasBoyfriend f:GuyRichie} {f:Madonna f:hasBoyfriend f:VanillaIce} {...}

Table 3.1: Examples for extracting semantic information (Klüwer, Adolphs, Xu, Uszkoreit, & Cheng, 2010)

Klüwer et al. (2010) tried to create *non playable characters* (NPCs) that talk, instead of giving instructions. They modeled the knowledge needed in the *resource description format* (RDF). RDF statements consist of subject, predicate and object, where the predicate links the other two items. Every object the NPC should be able to talk about has to be modeled closely in RDF. An ontology is built out of these statements. The NPC can use multiple ontologies at the same time. A barkeeper for example can use one ontology for serving drinks and another for talking gossip. The dialog system is built around three parts. The *input analyzer* analyzes sentences from users. This analysis is transferred into RDF and passed to the *dialog manager*. The dialog manager tries to interpret its input through context and its knowledge based on the ontologies. An action is selected, which is performed by the *output generator*. The action can not only be verbal answer, but also gestures. For task based conversations, like selling goods, a finite state model is used. Objects and properties, which are talked about, are saved in a *form*. The form holds only entries which match the concepts in the ontologies. During conversation, the form will be enhanced step by step, because users will voice new features and desires. The dialog will last until only a few objects remain. As a consequence, the dialog is defined by the form and the knowledge base. The dialog manager either searches for facts from its ontologies, or tries to carry on the dialog by giving users different choices. Table 3.1 shows some examples, where the input is a user statement, the extracted information is an RDF triple and the results found in the knowledge base.

Adaptivity helping Creation

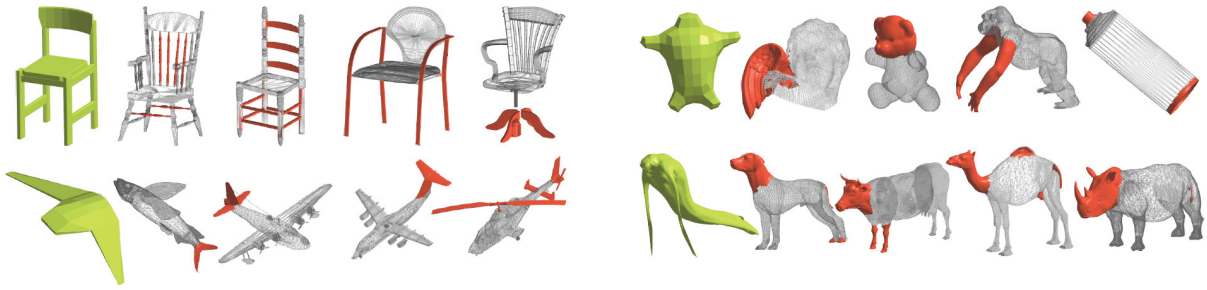


Figure 3.23: Used shape, green and proposed parts, red (Chaudhuri & Koltun, 2010)

To help users with modeling, a data-driven approach can be used. This approach suggests parts based on the model's shape. The current shape is used for database queries and does not need input from users. This can lead to some unexpected suggestions, which may not only make the modeling process easier and faster, but also encourages creativity. The comparison between shapes is complex though. That is why statistical signatures are calculated for the models instead. These signatures contain information about the shape of an object. Therefore they are easier comparable with one another. Matching shapes only have to be estimated,

because exact matches may not be found. When a matching shape is found, parts of it are suggested which may fit the current model the best. To make the suggestions more coherent, the models in the database are already split into parts. Figure 3.23 shows the top suggested parts in red based on the green shape. (Chaudhuri & Koltun, 2010)

Adaptive Content Management

Vatjus-Antilla, Hickey, & Koskela (2013) address an interesting problem, which occurs due to the fact that content distribution of virtual worlds is static and does not adapt to the device the client is running on. This often results in highly complex objects some devices can not handle. And for mobile devices a generic solution is not always possible, because they are just too different. Therefore a system is required, which adapts three dimensional objects, as well as images to the device the client uses. The adaptation process is done during the delivery phase. The system decides which format is best suited and transforms it accordingly. 3D-objects can be adjusted through reducing the polygon count. Images can be transformed into special formats, which are compatible with the device. Graphical effects can also be reduced. To get the best results, it is necessary for the device to identify itself at the beginning. The proposed framework uses a proxy between client and asset server, which makes all the changes described above. The biggest problem with this approach is the high latency because of the proxy. If transformations have to be made it will take even longer. Figure 3.24 shows an example, where only the images were adapted. In (a), the original world is shown, whereas in (b) it is the same situation, but the textures were transformed into another format. (c) shows the same format used on a tablet and (d) shows the mobile version. The biggest differences can be seen on the grass textures.

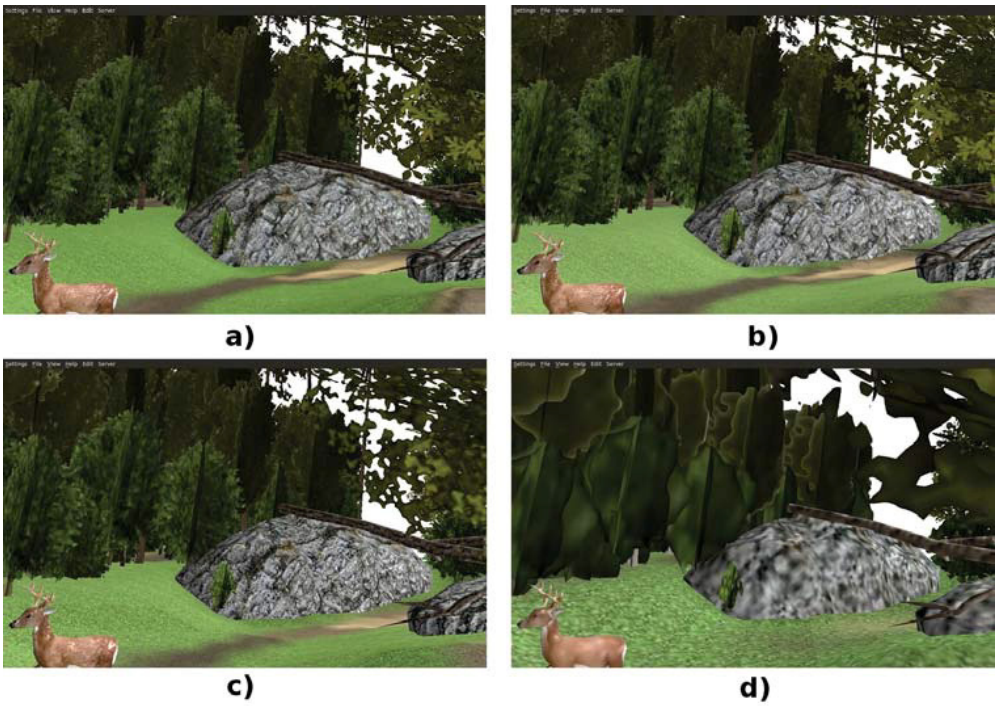


Figure 3.24: The same scene adapted for different devices (Vatjus-Antilla et al., 2013)

3.4.3 Configurability & Reusability

The need for sharing and reusing objects in virtual worlds rises constantly. User-created content is a big part of virtual worlds, which is the reason they usually offer in-world tools

and scripting languages for object creation. The only problem is that mainly experienced users are able to use them effectively. Additional constraints, like different user roles and rights, will make the whole creation process even harder. Many of the offered tools are fairly limited. There are approaches trying to avoid these problems, but this situation shows the need for easy to use tools and scripting. (Gütl, Chang, & Freudenthaler, 2010) Dafli, Vegoudakis, Pappas, & Bamidis (2009) show that the reuse of existing virtual worlds can save a lot of time, but also that repurposing the world can get complex, when done only manually. According to Cai, Sun, Farh and Ye (2008) virtual worlds need to be both more flexible and individually adjustable for the needs of its users. There is also a lack of standardization and support of reusability.

Generation through Configuration

With the help of XML configuration files, it is possible to create an environment. The presented approach uses SL and the *Learning Activity Management System* (LAMS). LAMS is a tool for building and managing learning environments and activities. (Apostolidis, Kyropoulou, & Chaldogerides, 2011; LAMS, 2012) The learning activity is created in LAMS. The output of the process is an XML file, which is produced by LAMS. The XML configuration file contains everything about the activities and the participants. Activities are described through objects in SL. Unfortunately it is hard for the framework to create the whole learning environment from LAMS because SL does not offer enough collaborative objects. This means not everything can be represented through SL objects. But generally speaking, every XML file can be used as input. The approach is not bound to LAMS. Reusing and sharing XML files is easy, so that other users can also generate the learn environment. (Apostolidis et al., 2011)

Modeling and Model-Reusability

Reusability of worlds can be important, because the created world can be needed in another form. To customize certain parts of the virtual worlds, procedural filters are proposed. With the help of these filters, the users describe how the scene should be changed. Figure 3.25 shows the same building with different filters applied. From left to right: The original object, a cracked window filter, a graffiti filter and a garbage filter. Filters are represented in a dataflow diagram, which contains the following instructions (Tutenel, van der Linden, Kraus, Bollen, & Bidarra, 2011):

- *Basic operations*: These are simple operations, like mathematical calculations and the creation of objects.
- *Object transformations*: Transforms objects already existing in the world, like changing position, height or rotation. These are all little modifications which do not change the world entirely.
- *Material alterations*: Changes the materials, textures and colors of objects. Furthermore it can add or remove shader, or change their parameters.
- *Changing or loading assets*: Loads new textures for the previous instruction, as well as replaces objects. This can be used to change a tree with all its leaves into one without any leaves for example.
- *Semantic queries*: Finds attributes of object classes. When semantics are added to an object, it usually connects the object to a class. Attributes can be linked with other instructions and parameter settings. The semantics are high-level.

- *Automatic content generation*: Contains procedural methods.

Figure 3.25: Different filters applied to the same house (Tutenel et al., 2011)

When a filter is created, other filters can use it as well. They can also add additional parameters. Filters are very customizable. It is possible to make a global effect on certain material classes, or only apply the filter to one specific object group. The reusability of a filter is dependant on the way it is programmed. If only used for a specific object group, it will not work when the object group is not in the scene. Figure 3.26 shows a clean office on the left and a party filter applied to it on the right. The addition of objects like balloons and cans is done in a generic fashion. If, for example, the cans are only added to office desks, the filter will not work in other areas, like a living room. The editor for these filters tries to make the creation easy through building a graph of predefined blocks. A filter has one ore more inputs, which can also be parameters specified by users. Scene objects or object groups can also be used as input, which leads to the changed object as an output. (Tutenel et al., 2011)

Figure 3.26: Normal office on the left and applied party filter on the right (Tutenel et al., 2011)

Behavior Modeling

Behavior is the hardest and slowest part to implement when creating an object, but dynamic content is becoming more and more important. Creating dynamic behavior is done through scripts, which can be difficult to understand. Even with scripting simplified, only experienced users can handle it. Besides, scripts have to be done manually, which costs additional time. One way to solve the problem is modeling behavior with the help of design patterns. These patterns should be customizable and combinable with other patterns. Additional, it should be easy to define new patterns. To help inexperienced users as much as possible, the overall behavior modeling is done in a graphical language with diagrams. The model approach is *action-based*. This means the model revolves around the actions an object can have. Relationships between actions are modeled. The state of the object is not important in this approach. The behavior is independent from the object, which furthers reusability. Using only

graphical representations of the behavior does make the modeling process easier, but only with patterns, an easy configurable system can be achieved. There are three different pattern types. The first is the *behavior pattern*, which is related to an action the object can perform. An example can be the movement of the object on a predefined path. The *interaction pattern* is used for describing user interactions. The *structural pattern* is one abstraction level above the other two and is a collection of patterns to implement more complex behavior. For every pattern, there are different configurations that can be customized, as shown in Figure 3.27. (Pellens, de Troyer, & Kleinermann, 2008)

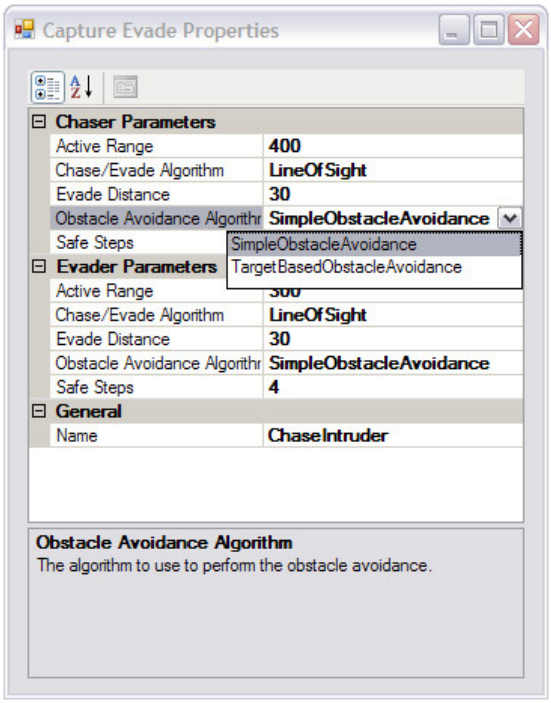


Figure 3.27: Customizable configuration of a behavior pattern (Pellens et al., 2008)

Interoperability

The creation of 3D objects, scripts and behavior may take a while, but when wanting to use them in other virtual worlds it is often impossible. Interoperability between objects, especially across platforms would give certain benefits. It would make sharing worlds a lot easier. When mixing different formats it is also possible to use their specific advantages. And only one general client would be needed for accessing different platforms. (Berthelot, Duval, Royan, & Arnaldi, 2011) Creating a virtual world would be faster, if different assets from other platforms could be used. (Pape, Anstey, Dolinsky, & Dambik, 2003)

A framework, called *Ygdrasil*, tries to makes reuse and combining parts of virtual worlds possible, but only within their own platform. It uses a *distributed scene graph* for representing objects. A scene graph is a graph, with nodes containing data of specific objects. This structure makes swapping nodes fairly easy. Furthermore, the graph is distributed, as shown in Figure 3.28. Different sub-graphs are on different hosts, there is no central server. All objects the hosts have stored are their own and only they are able to make changes to their nodes. If objects from other hosts are needed, they are loaded as proxy versions. The nodes usually contain both the graphical part to render the object and its behavior. The scripting language used for Ygdrasil is a text based representation of the scene graph. The compiled versions of the nodes are objects that can be loaded individually, which makes them easy to add and change. (Pape et al. 2003)

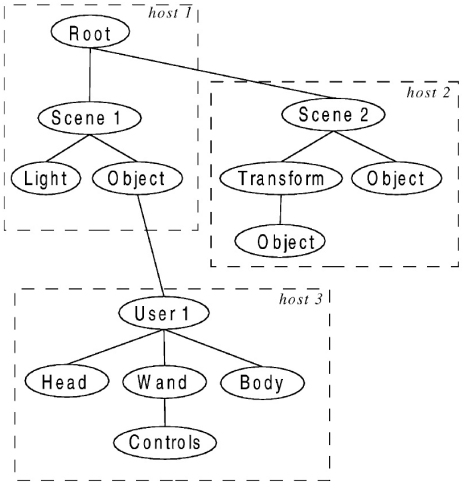


Figure 3.28: Distributed scene graph (Pape et al., 2003)

Another approach for interoperability between virtual world platforms proposes a *scene graph adapter* (SGA), which does the communication between the 3D objects and the graphical representation. For the formats and renderer a graph structure is used. The SGA consists of four components: The *renderer adapter API*, which access and changes the renderer graph, the *format adapter API*, which does the same for formats, the *node indexer*, which makes connections between format graph nodes and renderer graph nodes and the *kernel*. For data exchange, wrappers are also needed between the SGA and the format graphs as well as the renderer. Figure 3.29 shows an example on how the SGA is working. In (1) an external file is found during parsing. This results in a kernel call in (2). The matching format wrapper is loaded during (3-4). In (5) the format decoder is called and in (6-7) the file is loaded, which is then given to the renderer (8-10). (Berthelot et al., 2011)

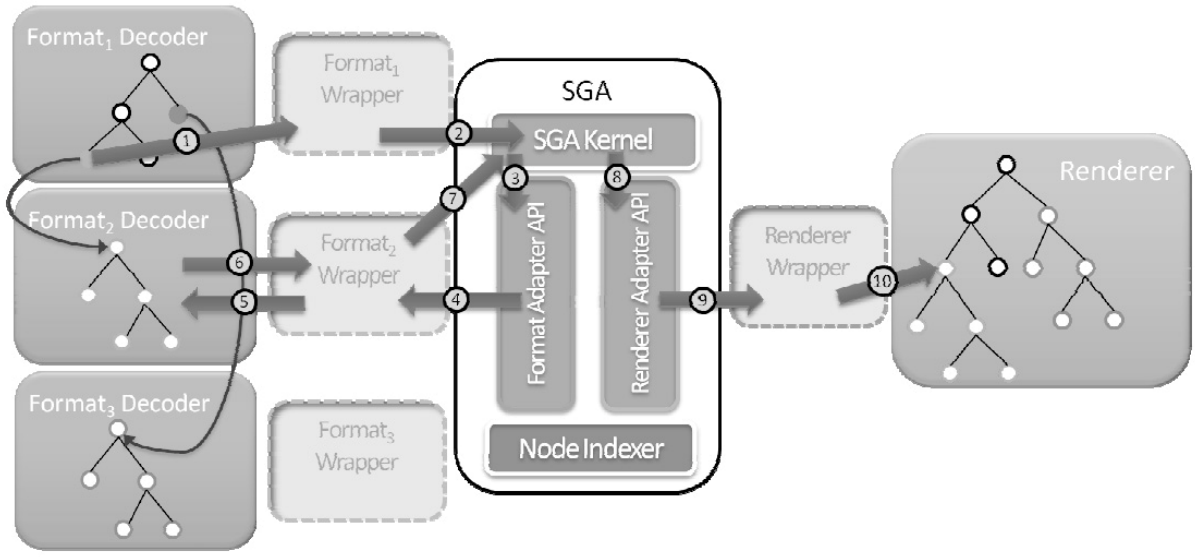


Figure 3.29: Scene graph adapter usage (Berthelot et al., 2011)

Like sharing objects, sharing behavior scripts has the same problems, especially within different platforms. It is even harder, because scripting in those systems was not created with reusability in mind. Object behaviors are usually hard-coded and platform dependent. Vague separations between generic and platform-specific functions and the missing possibility for sharing created behavior are further reasons why reusability is hard to achieve. There is one approach, which tries to make behavior reuse possible through replacing system-specific content with reusable modules. When creating a new module, a separation between generic

and platform-specific functions is made. The generic functions have interfaces for other modules to use them. The virtual world platform needs some changes though. A *broker server* is necessary to forward interfaces to the modules requesting them. (Liu, Bowman, Hunt, & Duffy, 2012)

Interoperability between virtual worlds and external tools is also a big problem. There is one approach, which tries to create this sort of interoperability. It implements a special bus, which is called *WAFFLE bus*, which enables users to add and remove tools. Web service interfaces are used to organize these tools. (Booth & Clark, 2009) There is a need for interoperability between virtual worlds and external sources, especially for l*earning management systems* (LMS). Theoretically a communication dispatcher can be used, which remains between virtual world and the LMS. Such dispatcher tools already exist, like *Sloodle*, which can be used to integrate the LMS *Moodle* with SL. (Farley, 2009; Leidl & Rößling, 2007)

The complete opposite of the previously mentioned methods is the *Web3D* based approach. Instead of trying to change virtual worlds from the outside, Web3D tries to enrich normal web pages with 3D content. This makes accessing the content a lot easier for users. A database for objects was planned for reusability purposes. (Albion, 2009) Unfortunately the Web3D approach was abandoned due to the lack of interoperability of 3D objects and the early stage of the *X3D* standard, which was used with the Web3D approach. (Albion & McKeown, 2010) X3D is an open and extensible standard for describing three dimensional scenes. It is a rather complex standard. But it is possible to simplify it and make it more accessible. The result of this simplification was the *DWeb3D* tool kit. (Quintella, Soares, & Raposo, 2010)

Scalability

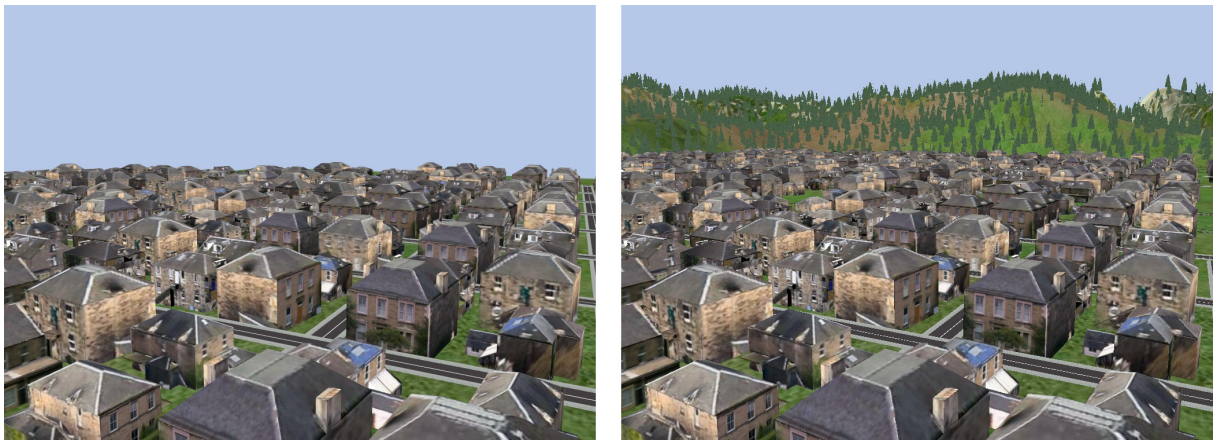


Figure 3.30: Distance approach left and solid angle approach right (Cheslack-Postava et al., 2012)

Scalability is a big issue when it comes to virtual world platforms. (Cheslack-Postava et al., 2012) There are three dimensions of scalability. The first revolves around how many users can be present at the same time. The second dimension is about the complexity of the virtual world itself, like the amount of objects that can be placed in the world. Object behaviors also belong to this complexity. The third is the user's interactivity. This includes the level of interactivity and how broad their possibilities are. In general, a distributed scene graph may offer the best solution for performance issues. (Dionisio, Burns, & Gilbert, 2013) On normal platforms, like SL, visibility and the range of objects that can be manipulated are limited. This is done for scalability reasons. Users can only manipulate objects near to them. Objects far away can not be used, even if they are visible. This can ruin the immersion. Usually virtual world platforms like SL use distances for visibility checks, but there is another method, which is used on the Sirikata server. This approach is called *solid angle* and calculates the amount of pixels an object uses on the screen. The advantage of this approach is that it only returns

objects that are visible by the current user, which minimizes the traffic. When objects are too small, only simplified versions of them are returned. Figure 3.30 shows the difference between a normal distance approach and solid angle. Both of them return 3.000 objects, but the solid angle approach does have a much higher visibility. (Cheslack-Postava et al., 2012)

3.4.4 Related Work

As shown in previous sections, tools for world creation are needed, which are simple and easy to use and adaptable to new changes. (Gütl, Haas, & Chang, 2013; Gütl et al., 2014) One of these tools was an editor like prototype called the *second life room manager* (see Figure 3.31). It is an external management tool for learning settings and provides a graphical editor for room creation. Some of its requirements were to support as many learning situations as possible and to give users the possibility to reuse them. The application consists of two parts, a web application and a build-in SL part. The web application is the interface for users. It saves everything into an SQL database, which contains all information about SL objects and areas. The database communicates with SL through a PHP backend interface. (Freudenthaler, 2011) During development, certain problems occurred. Because of the separation into two parts, the web application needed to be browser independent, which caused some problems. Positioning objects in SL was also fairly difficult, because every object needed a script in order to be placed. Exact positioning of certain object was also problematic. Rights management caused further problems. But the prototype showed promising results for creating and reusing virtual environments. (Gütl et al., 2014)

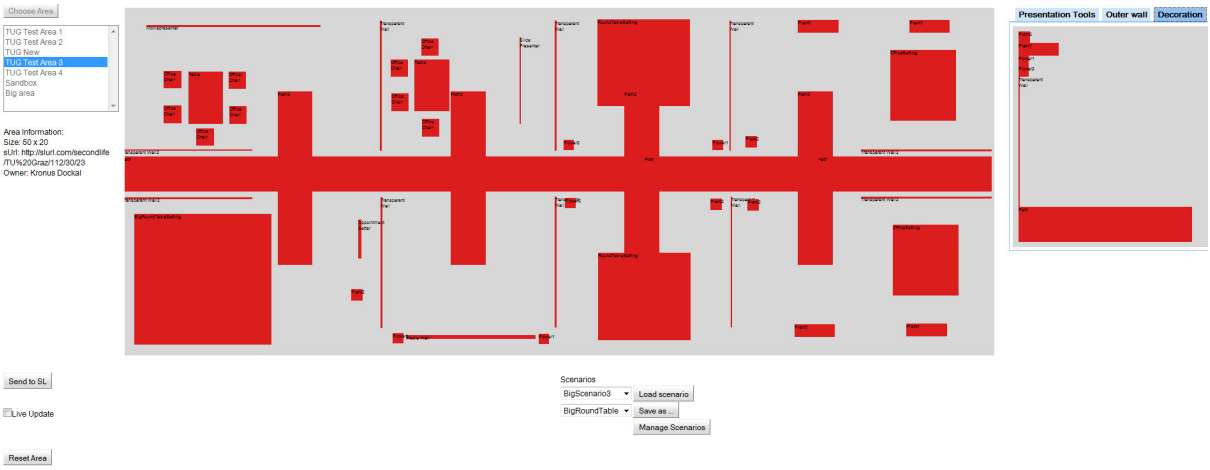


Figure 3.31: Second life room manager (Freudenthaler, 2011)

Another editor prototype was created for OpenSim learning environments. A screenshot of the tool is shown in Figure 3.32. Like the SL room manager it is an external tool. Some of the main goals of this project were the simultaneous management of different virtual worlds, building an inventory of simple objects and learning tools, as well as extensible templates. The management tool consists of three components. The *room management server* is the bulk of the application. It has world and room configurations, as well as other settings, like room permissions, stored in a database. It provides APIs to give information to the other two parts. The API for the client is realized through REST. The *room manager client* is a web application which implements the graphical user interface. The *OpenSim region module* is the last component and builds the configuration, stored on the room management server, in the virtual world. (Haas, 2012) This prototype was built upon the premise to support multiple virtual world platforms. The problem concerning OpenSim was the limitation of the region size. To make bigger regions, a module would be necessary to group regions together. (Gütl et al., 2013)

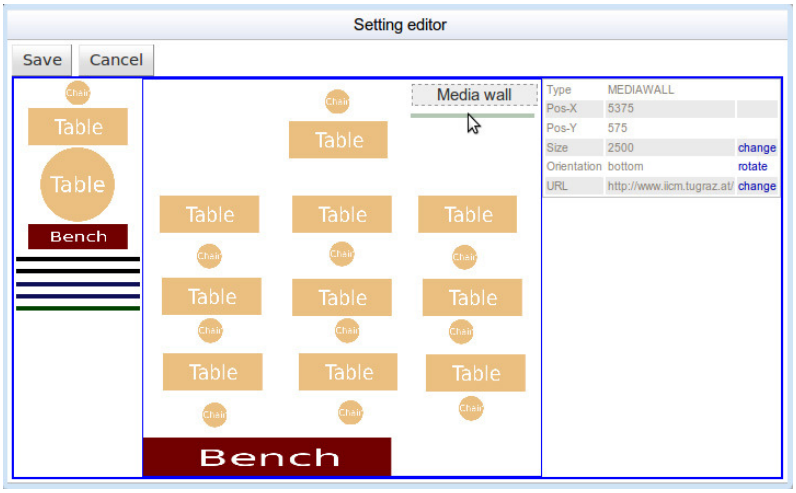


Figure 3.32: OpenSim room manager (Haas, 2012)

3.5 Discussion

Virtual worlds give their users a huge amount of possibilities, especially in the creative sector. But not everything is as good as it may look at first glance. First, the definitions are rather vague. A better term for virtual worlds would be useful, because as it stands now, it is confusing which platforms belong to the definition of virtual worlds. Are virtual reality worlds also part of this definition? They are virtual, but they tend to be a single user experience. It would be better to use genres to identify virtual worlds, like social worlds, to avoid misunderstandings. MMORPG worlds can be regarded as virtual worlds for the most part. They fit some definitions as well as other virtual worlds.

Collaboration seems to be of huge importance for virtual worlds, because the community builds upon collaborative efforts in many areas, like creation. Interestingly enough, there are few to no build-in tools in virtual worlds to support collaborative creation. Heck, there is even little support for simple creation. Therefore creating and programming objects is hard for inexperienced users, because the build-in tools lack simplicity, reusability and configurability. There are many approaches, which try to deal with these problems or support users as good as possible, but because of the lack of standards, it makes the whole situation difficult. Interoperability between different platforms is also a big issue, which has to be yet resolved in a satisfying way.

4 Concept & Design

Virtual worlds can be an interesting new playground for SFP, but they all lack user-friendly interfaces. As shown in the previous chapter, there are a lot of tools which alleviate the task of creating a world. But most of them use automated routines for this task. This could be a problem when users want to do changes by hand. If that is the case they have to edit directly into the virtual world. Therefore a tool is needed, which helps users making their changes. The goal of this work is to create an easy-to-use editor for building, modifying and sharing worlds in OWL. The world should be represented in a 2-dimensional bird's eye view, which should be familiar and easy to understand for inexperienced users. The first section goes over the initial concept as well as the first idea of the prototype's structure. The next section will go over the structure in more detail.

4.1 Idea and Concept

The problem with virtual worlds is they offer plenty of possibilities but lack simple tools to create and maintain a virtual world. For instance, OWL only lets users move one object at the time, with the help of the object properties or an in-world tool. This is tedious, when users need to move several objects. The same goes for various operations, like rotating and scaling. Copying objects is also difficult, because it is only possible to create one copy at a time, which spawns near the original object. An editor like tool may offer all these commodities. Another problem is that the in-world tools do not let users undo their mistakes, which is implemented in nearly every piece of software that allows for creating content. Therefore a tool is needed, which offers all these operations. (Pirker et al., 2014)

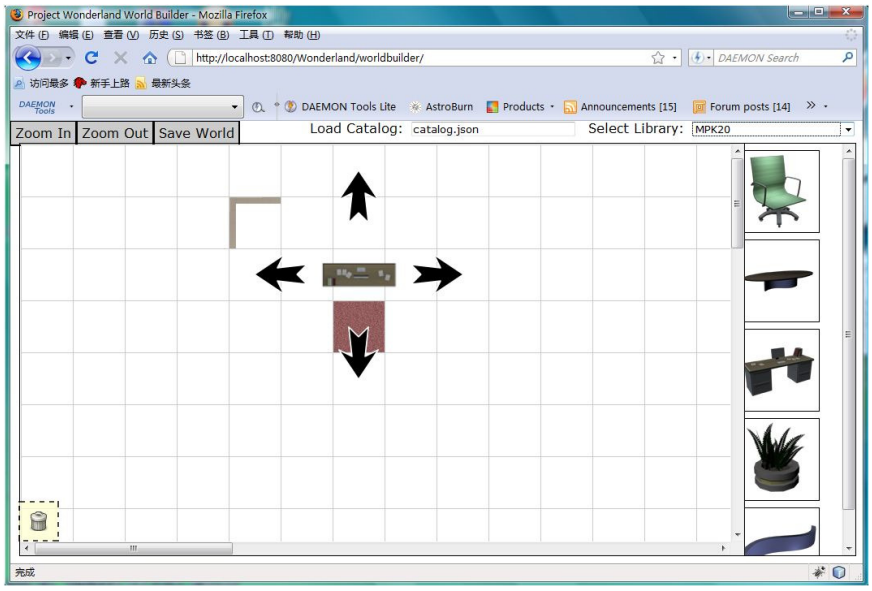


Figure 4.1: Original Wonderland World Builder (Li, n.d.)

The general idea behind this work is to support users during the creation of virtual worlds. As stated before, virtual worlds are in strong need for tools which help inexperienced users. OWL is no exception to this. A two dimensional editor, which allows users to import, modify

and remove items in an easy fashion should be especially useful. With such help users can easily determine the rough looks of a room, or a city they have created, without the need of viewing the actual 3D world. Of course seeing and testing the newly created environment as fast as possible is another important factor, which should not be underestimated.

Requirement	Description
Java only	To keep the intentions of OWL, the editor should be implemented only in Java.
Top down view of the world	Simple two-dimensional representation of the world's objects from a birds-eye perspective.
Usable through client	The editor should be called through the client, to see changes done in the editor instantly.
Object import	Users should be able to import objects like in the normal client program.
Representation image	Users should be able to import representational images of objects, to easily distinguish objects from another.
Implementation of standard operations	Easy selection of one or multiple objects; drag and drop to move objects; operations for simple rotation, scaling and removal of one or multiple objects; copy and paste of one or multiple objects, allowing users to position them beforehand.
Undo/Redo operations	Operations for undoing mistakes.
Load/Save functionality	For sharing worlds, a possibility to save and load worlds is needed.
Rights	Users should be able to set rights for objects in the world.

Table 4.1: Prototype requirements

Such an editor already existed in the early versions of OWL. This editor was an external tool, as shown in Figure 4.1. The idea is to create something similar, which can be accessed directly through the OWL client. An advantage with such an approach is that changes can be seen directly in-world, because users are already present in it. With an external tool this might not be possible, because the tool might not be able to create a real-time connection and changes have to be sent to the server every time. This can become annoying for users, when they have to send their changes, wait until the server has made them and then start the client to see them. Table 4.1 shows the initial requirements for the editor. The main goals were to create a program in Java only, to meet the intentions of OWL, as well as to create a top down view of the virtual world. In addition there were several other requirements, which all revolve around usability and offering certain operations which are not or crudely implemented in OWL.

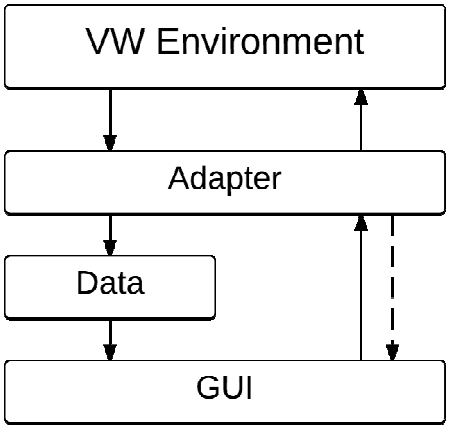


Figure 4.2: Conceptual prototype components and their communication

The conceptual architecture is built on several conceptual components, as depicted in Figure 4.2. The virtual world environment is the OWL server/client structure which provides object and world data. It also receives changes made in the editor prototype. The adapter is an integral part of the prototype, because it will allow the editor to be used with other virtual world platforms, therefore reusing the program becomes a better option. All important data provided by the server will be stored in the data component, for quick access and to relieve the server from too much requests. The biggest portion of the prototype will be the *Graphical User Interface* (GUI), which will handle the representation of the world in 2D as well as the user input. So the GUI is the part of the editor the users will see and work with. Changes made in the editor will be forwarded to the adapter, which signals the virtual world environment. Changes made in the environment will be forwarded to the adapter, then to the data component and finally to the GUI (Pirker et al., 2014)

4.2 Design

This section takes the concept from and builds upon it. The resulting design is shown in this section. But to make it better understandable, a quick overview of the OWL structure is given first. This is only brief and shows the important parts for the implementation. The editor design and its components are discussed afterwards.

4.2.1 Open Wonderland Architecture

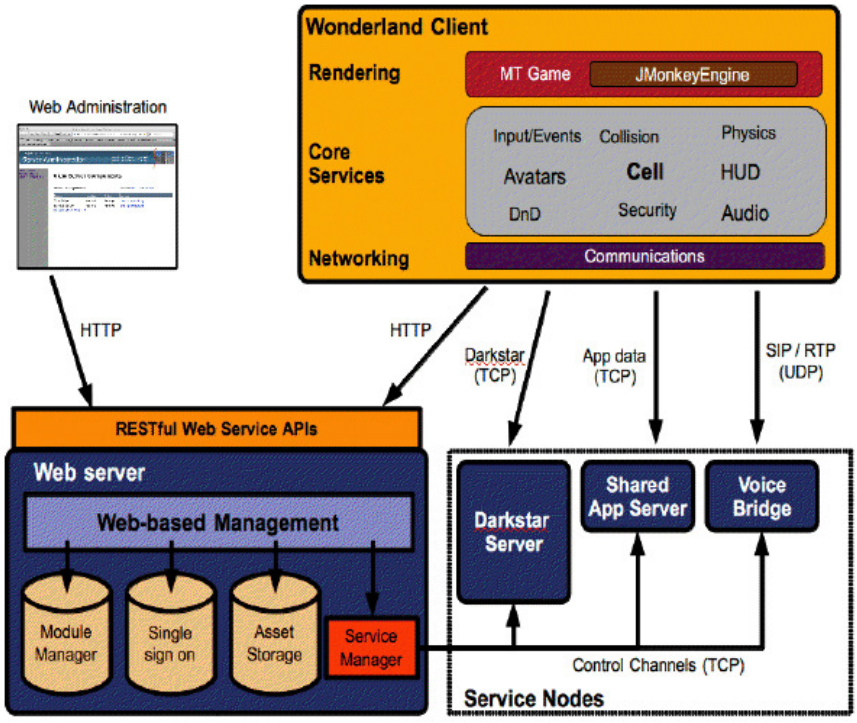


Figure 4.3: Open Wonderland Structure (Kaplan & Yankelovich, 2011)

OWL is a virtual world platform, which is free, open source and completely coded in Java. One of the primary design goals was extensibility. (Open Wonderland, 2013a) Therefore a modular architecture was chosen. The program itself implements only a couple of core services to make the platform work. Every other extension is done via modules. The extensibility is used at many different levels, from creating new functionality, or menus to

using external data. As shown in Figure 4.3, OWLs structure is based on a sever-client model. The server is split into four different parts. The *Web Administration Server* is responsible for coordinating all services. It is also used for user authentication and content management. It is based on a Glassfish Java EE Application Server. The second part is the *DarkStar Server*, which was another project of Sun. It was created for online games and is used to capture states, like the position of an object. The third part is the *JVoiceBridge* and is used for audio mixing. The last server is the *Shared Application Server* and like its name suggests is used for application sharing on the server side. (Kaplan & Yankelovich, 2011)

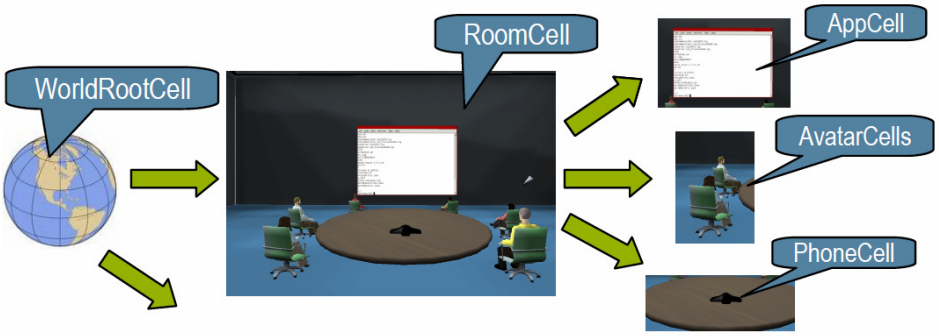


Figure 4.4 Example of world tree structure (Haberl, Proctor, Blackman, Kaplan, & Kotzen, 2008)

The client provides a browser, which primarily renders the 3D world and communicates with the server. The specific client details are not of interest, because the editor prototype will not communicate much with the actual client. Only the *Core Service Layer* should be mentioned. Its task is to offer features for modules, like object positions. The communication between server and client happens through network protocols, which are all tailor-made for specific data types. Because the protocols themselves are not important for the implementation, they will not be explained here. (Kaplan & Yankelovich, 2011) An Open Wonderland world is divided into cells. These cells follow a tree structure (Figure 4.4). Cells can be described as objects in the world. The client side uses a list that specifies which cells are visible to the user at the moment. (Haberl et al., 2008) Each cell can have a server and a client behavior. To give cells even more functionality, capabilities are used. They can be added dynamically to a cell. Each instance of a capability is related to a specific cell. (Kaplan & Yankelovich, 2011)

4.2.2 Prototype Structure

This section contains an overview of the prototype structure. The first section goes over the general structure of the editor, whereas the next sections will talk about the different parts in more details. The editor itself is written in Java and the GUI is created through Java Swing.

General Structure

The architecture is based on the conceptual components from chapter 4.1. Figure 4.5 shows a more refined structure. The first part is the virtual world platform, which is currently OWL. The OWL client and server parts are combined in the figure for convenience reasons, but the adapter only communicates with the server part. The main task of the OWL server is to update object data and to assure that changes made by one client will affect all other clients. The OWL client also updates data, but only its local data. Aside from this task, both client and server have to forward updates and changed objects to each other. The client is connected to the adapter, which is needed for decoupling the editor from OWL. It changes object data into readable information for the editor and forwards it to the data component. Furthermore it also

updates the server when changes are done by the user via the GUI. For this task it also converts information from the editor to messages the server understands. (Pirker et al., 2014)

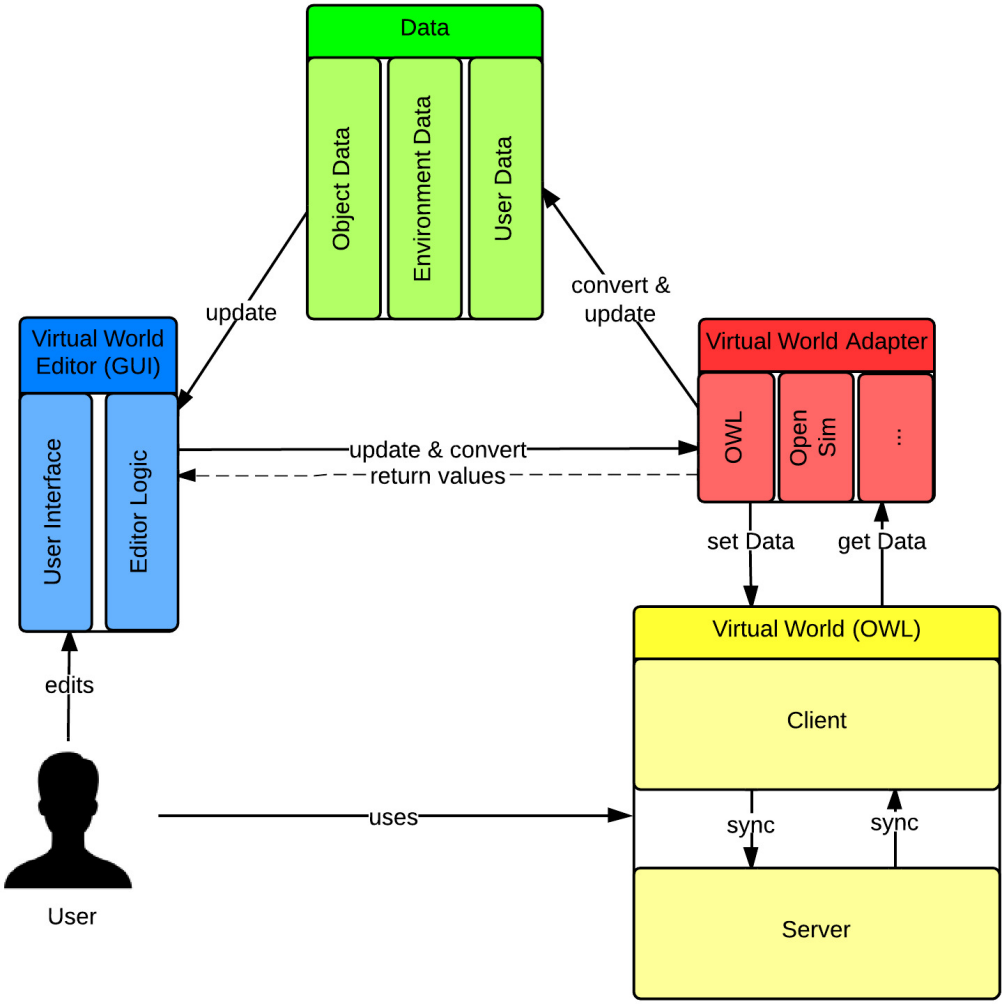


Figure 4.5: The general architecture (after Pirker et al., 2014)

The second component is the data component, which stores every data relevant to the editor. It also forwards changes to the GUI. The last component is the editor itself, the GUI where users do all their changes. The server is used for synchronization purposes. Therefore the editor does not have to synchronize data itself. Every time an object is changed, the server will send an update message to the adapter. This message will be sent, whether or not the changes were made through the editor, or through another user. The adapter will inform the data component, which will forward the change to the editor. (Pirker et al., 2014) The three main components will be discussed in further detail in the next sections. Because the requirements were well defined at the beginning of the development and most of the functionality needed for the prototype was already implemented in other OWL modules, the adapter part is a more straight forward design. Only the GUI is more complex, because it has to deal with two dimensional graphics as well as transformations on those. Because of its unexpected nature, an iterative development was used for the editor, which let it growing according to arising needs.

OWL Adapter

Figure 4.6 shows the general design of the OWL adapter component and its different packages. The main purpose of the adapter is to convert data and coordinates from the server to data the editor understands. The transformed data is then forwarded to the data component.

It also has to do the same the other way around, when changes are made in the editor. All these functionality is in the *Wonderland Adapter* package. The *Snapshot* package is needed to save and load worlds. In the *Cell Components* package the cell capabilities are implemented which are used directly by the adapter (the package is named only Components in the prototype but has added the Cell part for better understanding in the figure). *OWEditor Common* contains the server and client state of these capabilities. *OWEditor Server* implements the server part of the cell capabilities. The server side of the capabilities is either updated by the adapter, or other client applications, which forwards a change message to the common part of the component, resulting in an update of the components itself. The adapter then gets these updates through change listeners.

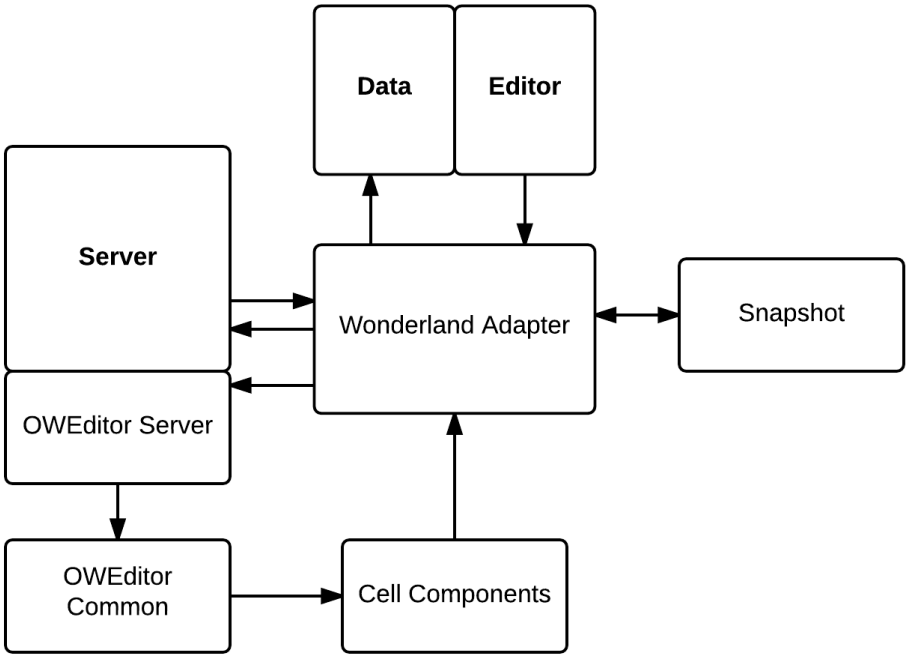


Figure 4.6: Design of the OWL Adapter Architecture and its communication

Data

The data portion of the editor prototype stores every relevant detail of the virtual world. The most important is the object data. It contains all information about the objects currently in the virtual world. The objects in this component are only a slim representation of the real objects, containing only information needed for the editor, like coordinates, scale et cetera. The environment data is only used to store the current size of the world, as well as other information, like server lists. The user data contains data specifically for users, like representational images of objects.

GUI

The GUI is probably the most complex part of the editor prototype and consists of four layers as depicted in Figure 4.7. The top layer is only used for communication between the data component and the adapter. It uses commands and stores it, to make undo and redo actions available. The second layer consists of *Input* and *Window*. *Input* handles mouse input, as well as some keyboard input. *Window* is mainly used for creating the third layer and forwarding messages between the first and the third layer, as well as from the *Input* package. The *Menu* package is used for creating menus. The *Graphics* package contains all necessary classes to

create and transform 2D objects. It also paints them in the mainframe, which is the frame users work in. *Frames* creates all frames for the editor, like the mainframe. For the most part the third layer packages only communicate with *Window*, to keep the methods simple, because usually they have to call different packages. The bottom layer consists of the *Shapes* and *Toolbar* packages. *Shapes* is used by the graphics package and contains all different shapes, like standard rectangles and circles. The *Toolbar* package creates the toolbars, which is used by the mainframe.

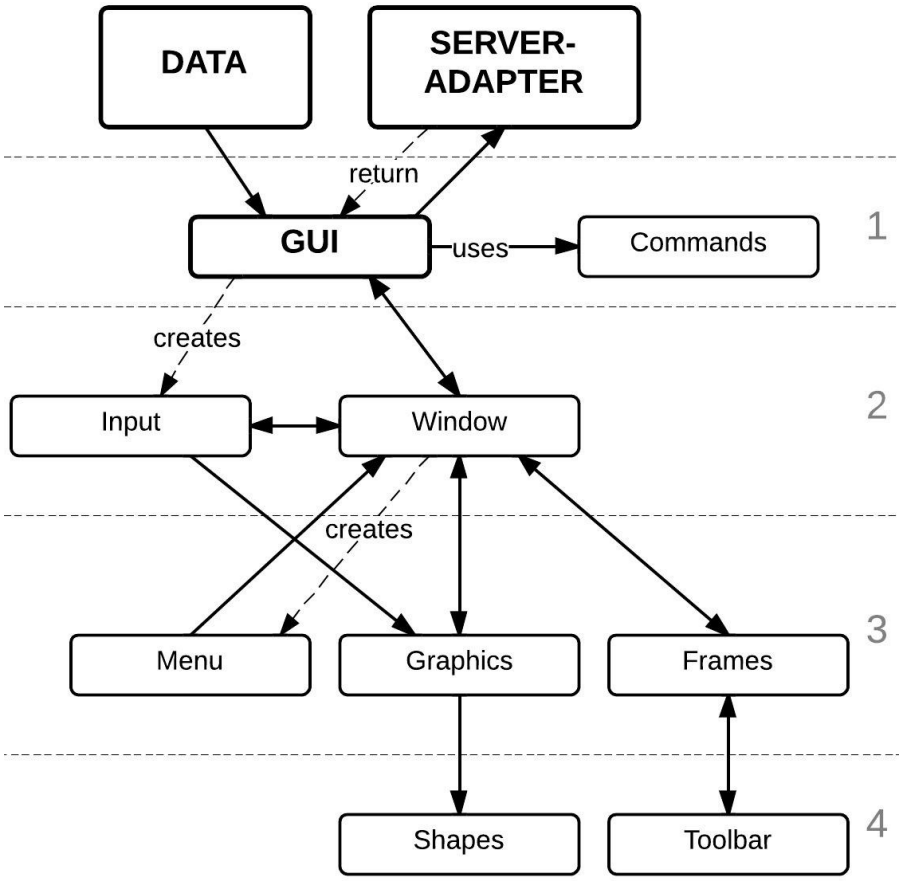


Figure 4.7: The GUI architecture

4.3 Discussion

As shown in previous chapters virtual worlds need tools to improve usability. This also applies to OWL, which offers a bunch of in-world tools for building worlds, but lacks easy to use operations common in other creative programs. OWL uses a tree structure for its objects and calls them cells. These cells are extendable through cell capabilities, which can add new features and behaviors to cells. In order to create an editor tool, to help users with the creation of a virtual world in OWL, it has to be easy to understand and easy to work with, as well as familiar to operate. A top-down view of the world is a good representation of the world, because many users are familiar with it . Furthermore, such a tool should offer all standard operations other creational tools should offer, including easy to use copy and paste, undo and redo, as well as save and load. In order to stay compatible with OWL, the tool has to be programmed in Java. Building upon these requirements three components emerged. The first component was an adapter which should be able to transform data from a given virtual world platform into readable data for the editor and vice versa. It also has to notify the editor or the

platform in case a change is made in one of these two. It may also be useful, when trying to run the prototype with other platforms, because the only part that needs changing is the adapter. The second component was the data component, which stores transformed data given by the adapter. It is required to minimize the traffic between editor and virtual world platform. The last component is the GUI, where users work. It receives all necessary information from the data component and notifies the adapter upon changes. The GUI is divided into four different layers, where only the top layer is allowed to communicate with the other two main components. The second layer is used for high level instructions, such as user input and communication between layers. The third layer is comprised of the main GUI components, such as frames and graphics. The bottom layer consists of individual graphic objects, like circles and rectangles, as well as different toolbars.

5 Implementation

The previous chapter presented three main components in the design stage. Keeping these components during implementation was the main goal. But other components had to be created in order to start the editor. In this chapter a short overview of the packages and their communication with each other is given. After this, the three main components will be explained in further detail. Please note that there will only be descriptions of the more complex or important parts of the components. Further on, interfaces will not be shown in the figures, except they are important, or there is more than one implementation of them.

5.1 Program Structure Overview

As indicated by the previous chapter, the general architecture is fairly simple. Figure 5.1 depictures the building and communication process of the prototype. Some additional components had to be created in order to start the program. The program starts in the OW editor component, which is simply used to create a menu entry in the OWL client. It is further possible to start the program without OWL, which is also done in OW editor. Through this package, the main controller is created, which has one purpose: instantiating the other components. If the main controller is started from the OWL menu entry, it will create the OWL adapter component; if it is started without OWL, it will create a dummy adapter.

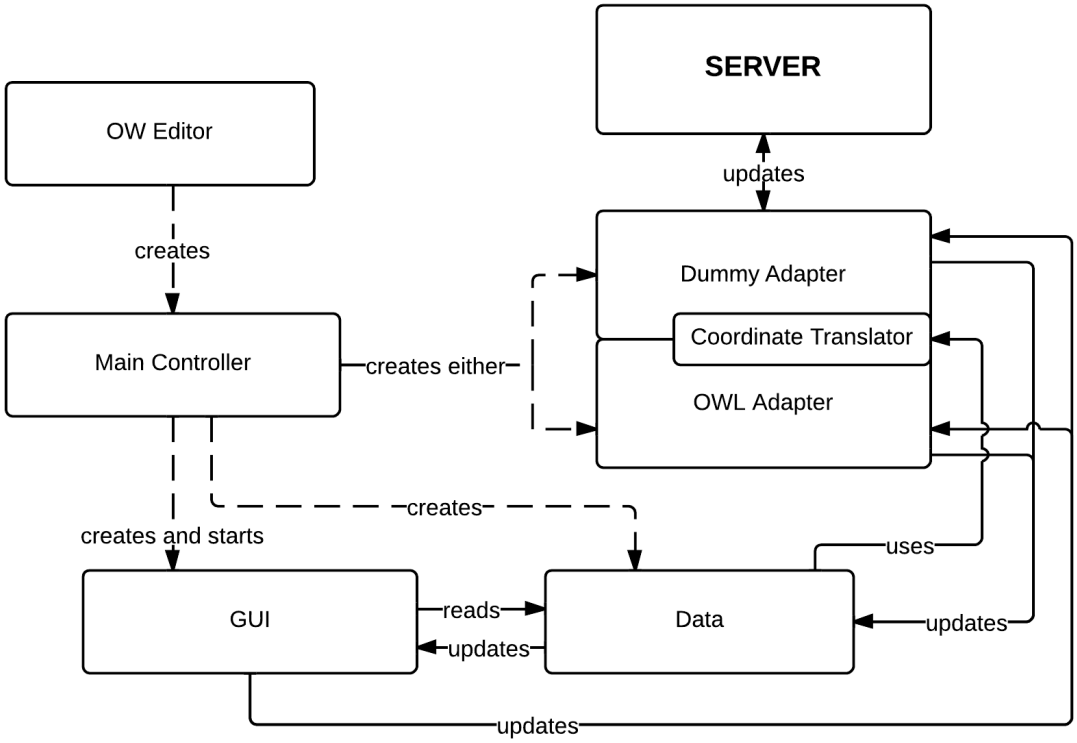


Figure 5.1: Building and communication between the main components

After everything is created, the main controller starts the GUI and signals the adapter to read-in the current world. Communication is done mostly in one way. The adapter gets messages from the server, forwards it to data component, which stores it and informs observers from the

GUI. After user changes are made, the GUI sends update messages to the adapter, which informs the server. There are three exceptions to this one-way of information. The first are possible return values from the adapter to the GUI, which are only used when importing KMZ files. The second exception is the coordinate translator. It is needed in the data component, to transform stored OWL coordinates into editor coordinates, when sending the update messages to the GUI. The last exception is the biggest, where the GUI has to read information from the data component, which will happen, if the GUI needs further information, or untransformed coordinates. Additional to the components listed in Figure 5.1, there are several packages, which contain the interfaces for these main parts:

- **Adapterinterfaces**
  - *AdapterMainInterface*: Is used by the main controller to build the adapter, and to register other interfaces. It is also used to start reading in the current world.
  - *CoordinateTranslatorInterface*: The interface used by the data component to gain access to the coordinate translator.
  - *GUIObserverInterface*: Interface for an observer, which is registered to the GUI and notifies changes made in the GUI. The implemented observer is found in the adapter package.
- **Controllerinterfaces**
  - *MainControllerPluginInterface*: The interface in which the main controller is started. It also allows changing the visibility of the editor.
- **Datainterfaces**
  - *IAdapterObserver*: Interface for an observer, which is registered to the adapter and forwards changes to the data component. The implemented observer is found in the data component.
  - *IDataObject*: An interface for a `Data Object`, which is used by both, the adapter and the GUI, when accessing information of an object stored in the data component. It is the editor representation of an in-world object.
  - *IDataToGUI*: The interface used between the data component and the GUI, for accessing stored data.
  - *IDataToMainController*: Is used by the main controller to build the data component and to register other interfaces.
  - *IImage*: An interface for the `Image` class, which is used by the GUI.
  - *IRights*: An interface for the `Right` class, which is used by the GUI.
  - *ITransformedObject*: This interface is used for `Transformed Objects`, which will be sent to the GUI after changes are made through the adapter and the adapter observer.
- **GUIinterfaces**
  - *IDataObjectObserver*: Interface for an observer, which is registered to the data component and forwards changes to the GUI package.
  - *IEnvironmentObserver*: Is the same as the IDataObjectObserver, but observes environment data instead of object data.
  - *IGUIController*: Is used by the main controller to build the GUI, and to register other interfaces.

The structure and functionality of the main controller, as well as the OW editor component will not be explained further on. They are only needed for the startup. The dummy adapter will also not be explained, because it is only for testing purposes when implementing new features to the GUI. The remaining three components, the OWL adapter, the data component and the GUI, will be discussed in further detail. Every package itself has a controller class, which stores most of the important instances and interfaces from other packages. This lets internal classes access them.

5.2 OWL Adapter

The adapter mainly consists of several server listener and classes which communicate with the server and transform data (Figure 5.2). To be easily readable the `Adapter Controller` (AC) is not depicted in the figure, as well as other more unimportant components. For instance, there are managers, which communicate directly with the server, like the `File Manager` (FM), which are also not depicted in the figure to make it easy to read. Some of the communication is also excluded.

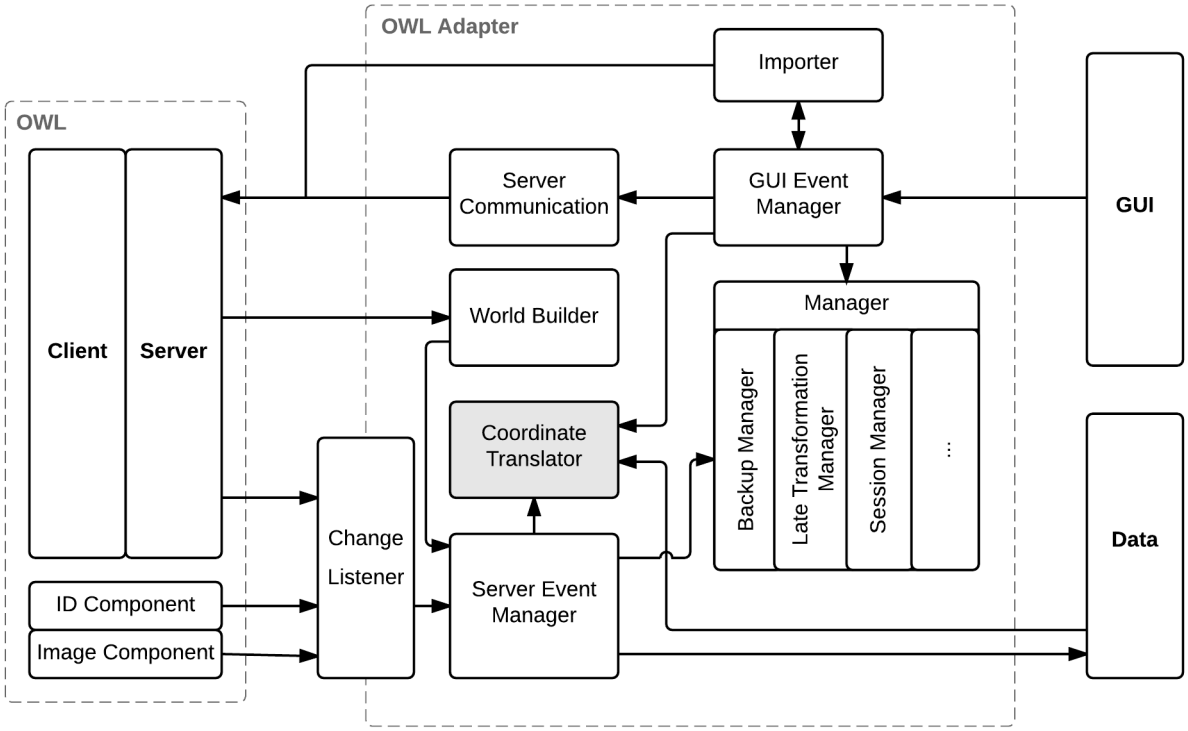


Figure 5.2: Simplified OWL adapter structure

The adapter starts in the `AC`, which main purpose is to create other parts of the adapter and to start the process of reading in the current world. It also stores every instance of the classes for easy access. The most important parts of the adapter are the two event manager. The `Server Event Manager` (SEM) is called when changes are made on the server. The `GUI Event Manager` (GEM) is responsible for forwarding changes from the GUI to the server. The second purpose of the `GEM` is to call the `Importer`, if there are 3D models to import. Another integral part of the architecture is the `Coordinate Translator` (CT), which main purpose is to translate coordinates back and forward between server and GUI. Some of the `CT`'s functions have to be made available for the data component, to transform the stored OWL coordinates and sizes into their GUI counterpart. The `World Builder` is only used at the startup phase

and collects all currently existing objects. It then calls the SEM to forward the data of the collected objects to the data component.

To make the adapter work properly, a set of managers are necessary. Only the two most important managers will be discussed here, the detailed description of every manager will follow in section 5.2.2. First, there is the Backup Manager (BM) which stores deleted cell objects. These objects could be needed during an undo action, or even when copying an object. The Late Transformation Manager (LTM) stores a specific ID of a cell and its destined transformation, which is applied when the cell is created. In addition to the adapter itself, two cell components are necessary to make the whole prototype work. The first is the *ID Component*, which stores the cells original ID. This is necessary for undo- and redo-operations. The second component is the *Image Component*, which stores the name and user-ID of an image linked to a cell component. The less important parts not seen on the figure are the Adapter Settings, which only store some initial settings, like the global scale and avatar sizes and the Cell Info Reader, which is used for reading cell information. The Image Monitor is only important for the representational image of a cell. This class maps cell IDs to image IDs. This is important when an image is overwritten on the server to signal the editor to change the overwritten image of all the shapes using it. Another package, which is not depicted on the figure, is the snapshot package, which is used for loading and saving a world. This package contains primarily code from the OWL module SubSnapshots and is slightly altered to work with the prototype.

Coordinate Translator

Because the GUI is implemented in Java Swing, the items in the GUI use integer coordinates, contrary to the float coordinates of OWL. Furthermore the object coordinates from OWL are in the center of the object, but the coordinates of Swing objects are at the top left corner, as shown in Figure 5.3. Therefore new coordinates have to be calculated with the sizes of the object. Furthermore OWL uses the y-coordinate for the object's height, therefore the z-coordinate has to be used for the Swing y-coordinate.

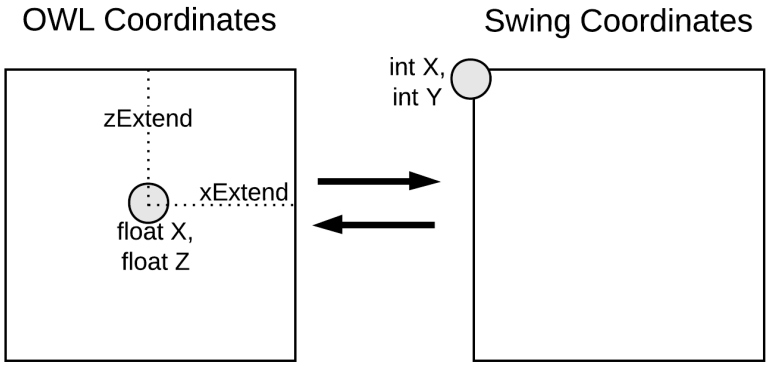


Figure 5.3: Coordinate transformation of a rectangle

Another aspect of the CT is the global scale, which has to be applied to the coordinates as well as the object sizes. Getting the object's dimensions is also not that simple, because OWL uses Bounding Volume to describe them. Bounding Volume can either be a Bounding Box, or a Bounding Sphere. The Bounding Box has three dimensions, which is not the actual size, but the extend from the center to the outline of the box (also in Figure 5.3). Bounding Sphere has only a radius from the center to the outline. As stated before, Swing uses integer and if a global scale of 1 is used, the objects will be too small. There will also be a problem with the translation. The integer coordinates would be too imprecise when not scaled properly.

Currently an initial scale of 50 is used, which produces a suitable match between size and precision. There are also transformations not correlated to the object size and coordinates. The rotation in the x-axis needs to be multiplied with (-1). The scale stays the same, but is included in case another virtual world platform uses other depictions of scale.

World Builder

The `World Builder`'s only real purpose is to read in the current world on startup and sending cell creation events to the `SEM` for every cell found. Because the cells are organized in a tree structure, the `World Builder` has to iteratively go through the whole structure. The only other task it has is to read in the server list, which is used during importing KMZ models in the GUI.

Server Communication

`Server Communication` (SC) is the part of the adapter, which does most of the outgoing communication with the server. Parts of it are altered code snippets from the cell editor module. It contains three `Cell Component Factory SPI` for the three cell capabilities *ID*, *Image* and *Security*, which will be discussed in section 5.2.1. The methods contained in the *SC* are used primarily for cell transformations, adding and changing capabilities and getting the cell server state. If something goes wrong during any of these processes, it will throw a `Server Comm Exception`. More complex operations, like loading worlds, or storing files are not implemented directly in the SC.

Listener

Currently there are only three listener classes in the adapter. Other listeners do exist, which are all `Component Change Listener`, but these are implemented directly in the `SEM`. All listeners are used to observe the server. The first listener class is the `Cell Status Listener`. It will be notified when a new cell is created, or a cell is deleted. The next listener is the `Transform Listener`, which will be called on transformation changes, like translation, rotation and scaling. The last one is the `Change Listener`, which observes the image capability and registers image changes, as well as name changes.

KMZ Importer

The `KMZ Importer` is used for reading .kmz files and transforming it to a cell. The code used for importing the model is taken from the `art importer` module. The two important methods in this class are `importKMZ(String url)` and `importToServer(String module_name, String name)`. During importing, it uses the `KMZTransformProcessorComponent`, which is also from the `art importer` module and unaltered. `ImportKMZ(String url)` imports a file. The `url` parameter of this method is the path of the file. The model itself will only be imported locally on the client. This is important to obtain the dimensions of the object as a return value for the editor. `ImportToServer(...)` builds a module out of the imported model and uploads it to the server. The `KMZ Importer` class also contains the method `checkName(String moduleName, String serverName)`, where the `moduleName` is checked on the server, specified with `serverName`, for a name conflict.

5.2.1 Capabilities

Capabilities, also known as cell components, had to be implemented for the editor to work properly and to include all its features. Capabilities are extensions of cells, which were described in chapter 4.2.1, but not in detail. Table 5.1 shows the different parts a capability needs in OWL. There are currently only two capabilities implemented in the adapter. The first is the *Image Component*, which is needed for the representational image of a cell and the second is the *ID Component*, which stores the original ID of a cell.

Package	Name	Description
Client	CellComponent	The cell capability used by the client side of the adapter.
	CellComponentFactory	Creates the `CellComponentServerState`, which is used to create the capability with `CellServerComponentMessage.newAddMessage`.
	CellComponentProperties	This is a form for the cell editor module. Without properties code, OWL could crash if the cell editor looks up an item, that does not have the properties.
Common	CellComponentClientState	The client state of the capability.
	CellComponentServerState	The server state of the capability.
Server	CellComponentMO	This server-side capability implementation.

Table 5.1: Necessary parts for a capability

Image Component

This component is used to store the URL of a representational image. The image will be shown in the editor, if it is possible to retrieve it. The URL consists of the image name and the name of the user. A new directory in the user directory, called `img`, will be created if it does not exist. All uploaded images by the current user will be stored there. The adapter has to register a `Cell Change Listener` to the `Image Cell Component` class to get updates on image changes. The listener also informs about name changes, therefore the image capability should not be removed under any circumstances from any cell.

ID Component

One problem occurred during developing the undo/redo functionality. Deleted objects can not be restored that easily. Therefore the `BM` is used to save cells, which are deleted by the current user. But unfortunately these cells can not be restored with their original ID. Therefore a component is needed, which contains the original ID of the cell. Figure 5.4 shows how this works. The cell ID is 6, therefore the ID in the editor will also be 6. After deleting it and restoring it later, the ID in the editor is still 6, because it would be a horrendous effort to change everything correlated to this ID in the data component and the GUI. Not only that, but all clients would have to receive an update notification on the new ID, which is probably not even possible. Therefore the `BM` has a map for current IDs and original IDs. Whenever a cell is created with this capability, the new ID is inserted into this map. When the ID is forwarded to the data component, or from the GUI, the `BM` transforms the IDs into the correct ID. Otherwise the object does not have an original ID and therefore was never deleted by any of the users. Because of this implementation, there has to be a check, if an original ID exists or not during every update from and to the server. The `BM` has to be called to receive the confirmation of an original ID or not.

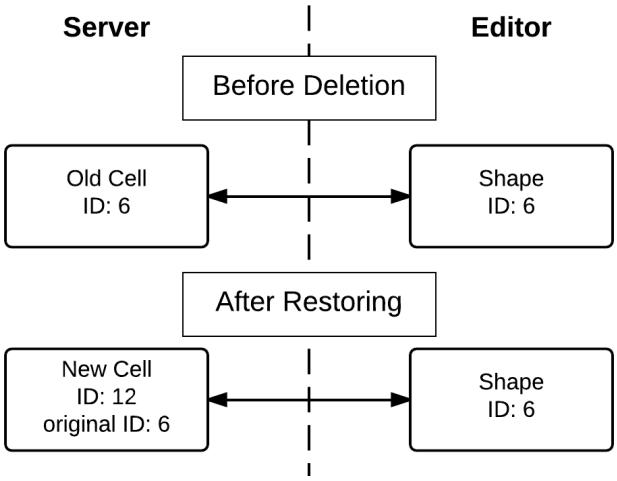


Figure 5.4: IDs before deletion and after restoring a cell

5.2.2 Managers

This section contains a description of all managers implemented in the adapter. It is structured from the most important to the least important.

Late Transformation Manager

The `LTM` is used for transformations and adding images after a cell creation event. This is done when copying or importing cells. OWL does not allow specifying transformations during copy. And most of the times the cell is not ready after the creation message is sent back to the editor. Therefore those operations are only possible later on. OWL provides these features for importing, which is also faulty and for the most part will not work correctly. Therefore the `LTM` had to be created. It stores transformations, like translation, rotation and scaling, before the import or copy message is sent to the server. The server creates the new cell, which the `SEM` will then receive. The `SEM` sends the ID of this cell to the `LTM`, which transforms it according to its stored data. The `LTM` consists of four array lists, for translation, rotation, scaling and adding images. Most of the methods are used for adding new items to the lists, removing them, or getting the transformation of specific IDs. The two most important methods are `invokeLateTransform(ServerCommunication server, long id)` and `invokeLateImage(ServerCommunication server, long id, String dir)`. These two methods start a late transformation, or add an image.

Server Event Manager

The `SEM` is usually called by all the server listeners. It implements a `Component Change Listener`, which looks for certain capabilities to be created or deleted. It also contains a list of observers. Currently there is only one observer registered to the `SEM` and that is the observer from the data component. The list was created for extensibility in mind, if further extensions to the prototype are developed. The methods from the `SEM` are generally simple in structure. They read the incoming data from the server and transform it into data the editor understands. Then it notifies the observers. Figure 5.5 shows the general structure of `SEM` methods. After an event occurred on the server and the `SEM` method is called, the first task is to search for an original ID and change it accordingly. Then the data is transformed into editor data. Please note, that the coordinates are not transformed in this step.

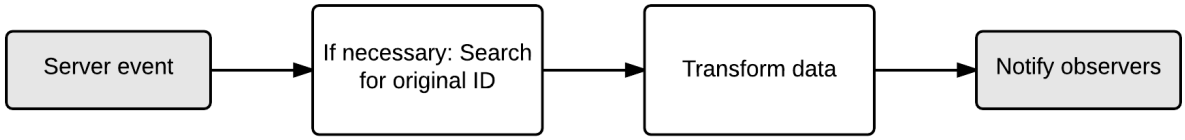


Figure 5.5: General structure of SEM methods

The only complex method is the `creationEvent(...)` when a new cell is created on the server. Figure 5.6 shows the whole process of a cell creation event in the SEM in order. First, a check is done, if the cell has a `Movable Component`. The `Movable Component` is the OWL capability for moving objects. Without it, it is not possible to move the cell. Usually it is generated automatically, but sometimes it does not get created, therefore this check. Then the LTM is used to make transformations and add the representational image. If the server is too slow in creating the `Movable Component`, the LTM is not able to do its transformations. Therefore it will not remove the cell from its list. The `Component Change Listener` implemented in the SEM will be notified, when the `Movable Component` is created. This results in a call of the LTM, so the transformation can be done later. After the LTM has done its transformations, the representational image for the newly created cell is retrieved. Then a check for the original ID is done which is entered in the BM if there is one. Then the position, size, rotation and scale of the cell are retrieved. After all relevant data is gathered, the adapter makes a call to the data component, which returns an empty `Data Object`. This object is filled with the retrieved data. Afterwards it is decided if the cell is an avatar or not, and if not, which shape it has (circle or rectangle). This information is put into the `Data Object` as well. The final step is to send the object back to the data component via observer.

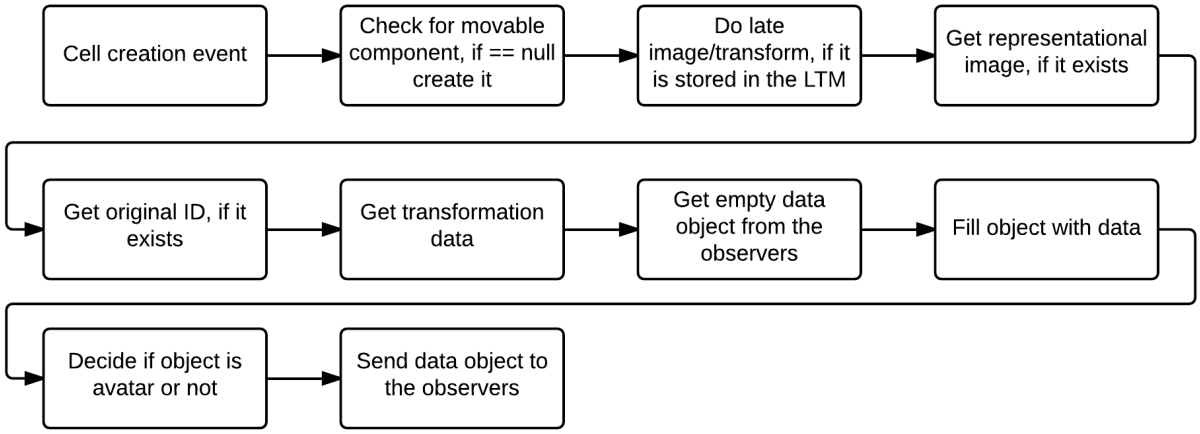


Figure 5.6: Events during a cell creation event

GUI Event Manager

This manager catches events from the GUI, transforms them into data the server understands and forwards it to the SC. Figure 5.7 shows the structure of a typical GEM method. First, the ID has to be transformed back in the actual ID the cell is currently using. Then the data is transformed. Unlike in the SEM, coordinates are transformed back into server coordinates. The rest of the code is not very complex and will therefore not be explained.

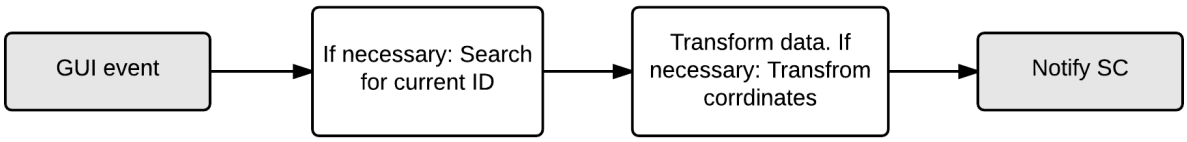


Figure 5.7: Method structure in the GEM

File Manager

The `FM` is used to store and retrieve files from the server. There are two methods that can be used:

- `uploadFile(File file, String dir)` is used for uploading a file. `File` should be a file to upload and `dir` is the directory where the file should be stored. Every file uploaded will be saved in the directory of the current user. The method returns `fileInfo`, which contains the file name and the user directory on the server. It should be noted that the `dir` parameter should only be the name of the top directory, not a string of nested directories, like "dir/dir1/dir43".
- `downloadFile(String fileName, String dir, String userDir)` downloads a file, where `filename` and `dir` are equivalent to `uploadFile`. `userDir` is the users directory. The method returns an input stream, which can be used for getting the file from the server.

The same two methods also exist for image upload and download. The images are used as representation in the editor for a cell. Figure 5.8 shows where images are stored on the server. All images are stored in the directory `img`. But they only share the same directory if they were uploaded by the same user. Usually every user can access the image files from every other user. There is no need for a shared image directory on the server, which would result in more complexity.

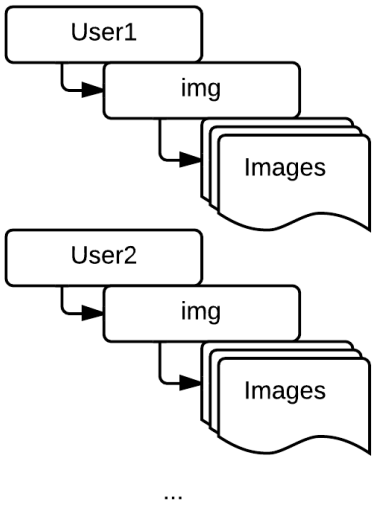


Figure 5.8: Structure of representational images on the server

Security Manager

The `Security Manager` is used for the *Security Component*. This means it is used for setting and getting user rights on different cells. The following two methods should be used for changing the rights:

- `getSecurity(Cell cell)` is used to retrieve the rights from the passed cell. It returns a `LinkedHashMap<String, Right>`. It is empty, if there is no *Security Component* within the cell. The string is the name of the user and the right class is an internal class, which is described later on.
- `changeRight(Cell cell, String oldName, String type, String name, boolean owner …)` changes a right entry for a given cell. The parameter `oldName` is used for replacing a user name, which could have been altered in the editor. `type` is the

usertype and `name` the new user name. Then there are only booleans which concur with the `Right` class.

Table 5.2 shows all attributes the `Right` class contains. Most of them are booleans, which represents the user's permissions.

Attribute	Function
`String name`	This is the name of the right.
`String type`	This is the type of the right. It can be either "User", "Group", or "Everybody".
`boolean owner`	If true, the current user is the owner of the cell.
`boolean permitSubObjects`	Permits the user to create sub objects.
`boolean permitAbilityChange`	Permits the user to change cell abilities.
`boolean permitMove`	Permits the user to move the cell.
`boolean permitView`	Permitts the user to see the cell.
`boolean isEditable`	This states, whether the user can edit this entry or not in the editor. This is currently not done in the adapter, but it is implemented, if the need for it arises with new implementations.
`boolean isEverybody`	If it is true, then this is the entry for everybody.

Table 5.2: The attributes of the Right class

Backup Manager

This class is used for backing up objects, which are later needed for undo/redo, or copy actions. Because OWL does not store deleted objects, this has to be done in the adapter. It also stores original IDs, which are needed for recovery, when an object is deleted and recreated with undo or redo. Table 5.3 shows the different backup lists and gives an explanation for each of them.

List Name	List Type	Function
`backup`	`HashMap<Long, Cell>`	This list backs up every removed cell from the current session. This is needed for undo and redo, as well as copy. The cell is only backed up, if it was deleted in the editor.
`originalIDs`	`HashMap<Long, Long>`	This maps the active ID, which is currently used by a cell to its original ID, which is used by the editor. When a new cell with the original ID is created, the old ID is removed and the new one is put into this map.
`lastActiveID`	`HashMap<Long, ActiveCell>`	This contains the last known cell mapped to the original ID. The *ActiveCell* class has a Boolean named `active`, which means the cell exists currently in the virtual world, if it is true. If the value is false, it was removed. This is used for checks to be sure, that there is only one active cell with the original ID.
`whiteList`	`ArrayList<String>`	This list contains names of cells, which are created by copying other cells. It is used as a safety mechanism, because the ID capability will be copied too. Whenever a cell is on the `whiteList` and there is already an active cell, the ID capability can be removed. If the cell is not on the `whiteList`, it is a duplicate and can be removed.

Table 5.3: Lists in the backup manager

Copy Name Manager

The `Copy Name Manager` is only used to create the names for copied objects. It just counts the number of copied objects and builds a name with this count and the specified copy phrase, like "Copy_of".

Session Manager

This class stores the current session and can be used to get instances linked to the session, like the `Cell Cache`. With the help of the `Cell Cache`, all cells in the current world can be retrieved. It is also used to retrieve the name of the current user.

5.3 Data Component

The data component is the second of the three main packages. It is stores the data the editor needs and also updates the GUI when changes are made. Figure 5.9 shows the overall structure. If the adapter notices changes, the `Adapter Observer` is called, which then notifies one of the three storage classes. It only forwards calls to the necessary destination and therefore will not be further explained. The three managers store data and notify the GUI on changes. The `Data Controller` forwards calls from the GUI to the managers and returns the requested data. The `Data Controller` also creates instances of the three managers, as well as the adapter observer. It will also not be explained further on. The `Image` class is a class which stores one representational image as well as its name and user directory, therefore no need for further explanation. `Data Object` is the editor's representation of a cell. It can also have several `Right` instances, which describes the permissions of the object. The `Right` class has the same attributes as the internal class in the adapter, as depicted in Table 5.2 on page 74. The `Transformed Object` is used for updating the GUI. It is created with the help of the `CT` from the adapter and contains all transformed coordinates.

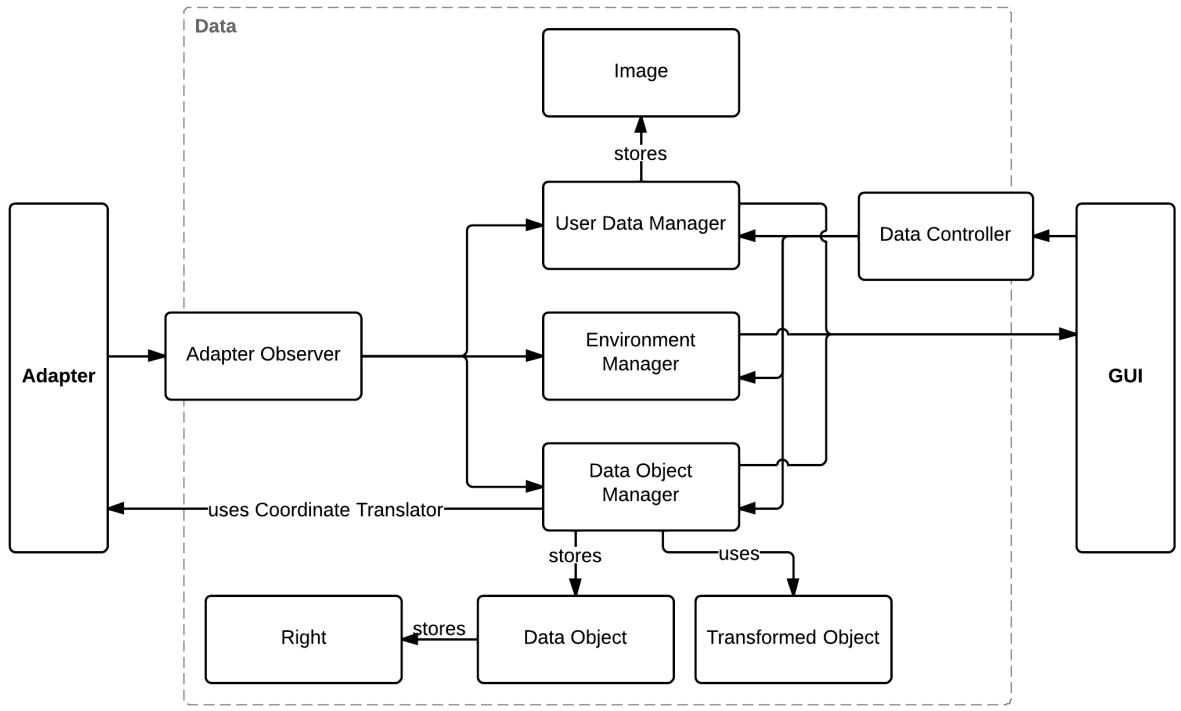


Figure 5.9: Simplified data component structure

User Data Manager

The `User Data Manager` (UDM) is used to manage data concerning users. It stores the user directory from the current user. Currently the `UDM`'s only use is to store the representation image library. The image library is split into two parts, the own library and the foreign library. The own library is called `imgLib` and consists of pictures from the current user, whereas the foreign library, called `foreignImgLib`, stores images from other users, that were collected during the session. The `UDM` has the typical getter and setter methods for the user directory and the image libraries. It should be noted that the call to add a picture and to retrieve one is done in both libraries and the destination is dependant on the user directory given as a parameter to the call.

Environment Manager

The `Environment Manager` is used to manage data concerning the virtual world itself. Currently only the world's bounds are stored here. It calculates the size of the world from the objects created in the virtual world. If a new object is created, or an object is transformed, this class calculates the new bounds of the world. It should be noted that the same is not done, if an object is removed. Therefore the world bounds grow bigger if necessary, but do not shrink down.

Data Object Manager

The `Data Object Manager` is used to manage data about cell objects. It stores data in form of a `Data Object`. It contains the typical methods to create, remove, get and update a whole object, or specific values of one. It also houses an instance of the `CT` that is used to create `Transformed Objects` out of normal `Data Objects`, which are needed to update the GUI. Figure 5.10 shows this process. This is only done, if a new cell was created or a transformation to a cell happened. On other circumstances `Transformed Object` are not needed. To create such an object, the necessary data is read from the `Data Object` and the coordinates are transformed into integer values with the help of the `CT`. The `Transformed Object` is filled with the data and the transformed coordinates and then sent to the GUI.

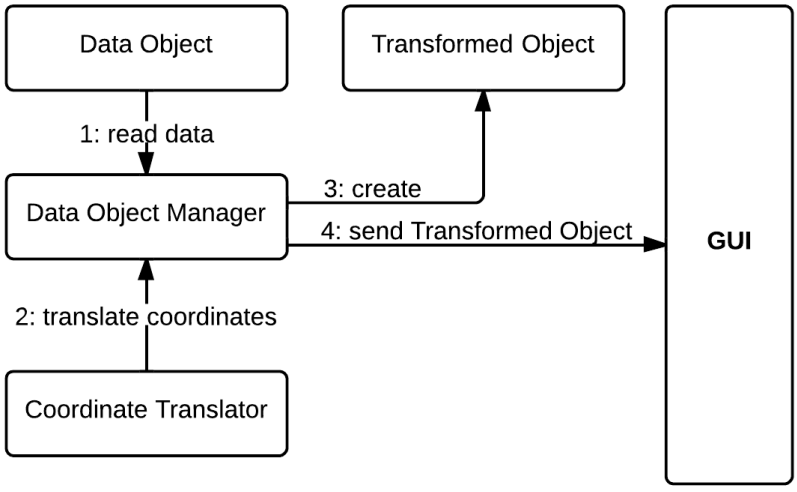


Figure 5.10: Process of updating the GUI during a creation or transform event

Data Object and Transformed Object

The `Data Object` is a representation of an OWL cell. It contains all relevant data of the cell. It also stores its permissions in a list called `rights`. The `Transformed Object` is only a temporary object, which contains the transformed integer coordinates, scale and rotation. It does not contain all information about the `Data Object`. Figure 5.11 shows an overview of the attributes of both classes. It should be noted, that `Data Object` contains rotation in every axis, whereas the `Transformed Object` only contains one. The methods of the two classes are simple getter and setter methods; therefore they are not explained here. The `type` states the form of the cell. It can be a rectangle, an ellipse, or an avatar. The type is a static definition in the `Transformed Object` interface. The interface for the `Data Object` is an extension of the `Transformed Object` which grants it all these definitions.

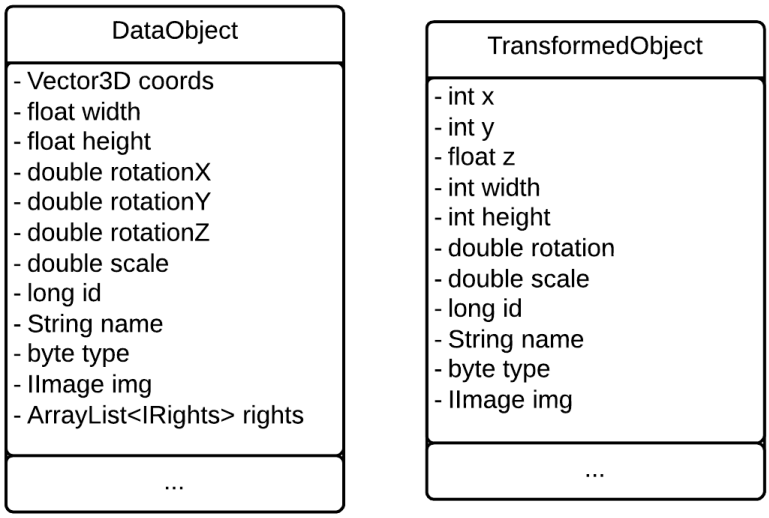


Figure 5.11: Comparison between `Data Object` and `Transformed Object`

5.4 GUI Component

The GUI component consists of everything concerned the editor's representation. This includes the window of the editor as well as the two-dimensional graphics that are shown in it. Figure 5.12 shows the communication of the GUI interfaces. Most of the descriptions are simplified, to give an overview on how the communication is working. If there is no description to an arrow, this means there are simply too much different options for this communication path to simplify it. Furthermore only the main purpose of the connection is listed in the figure, in order to keep it readable.

The *GUI* package is the only package that communicates with the other two components, like the data component and the adapter. The packages will be explained separately in the upcoming sections. Generally all interfaces follow a specific naming convention. First of all, ever interface name starts with an "I". Interfaces that are mostly used to create the package only have the name of the package. Other interfaces have the name of the package, then "To", followed by the package it is registered to. For instance, if the window package has an interface that is registered and used by the graphic package, the name is "IWindowToGraphic".

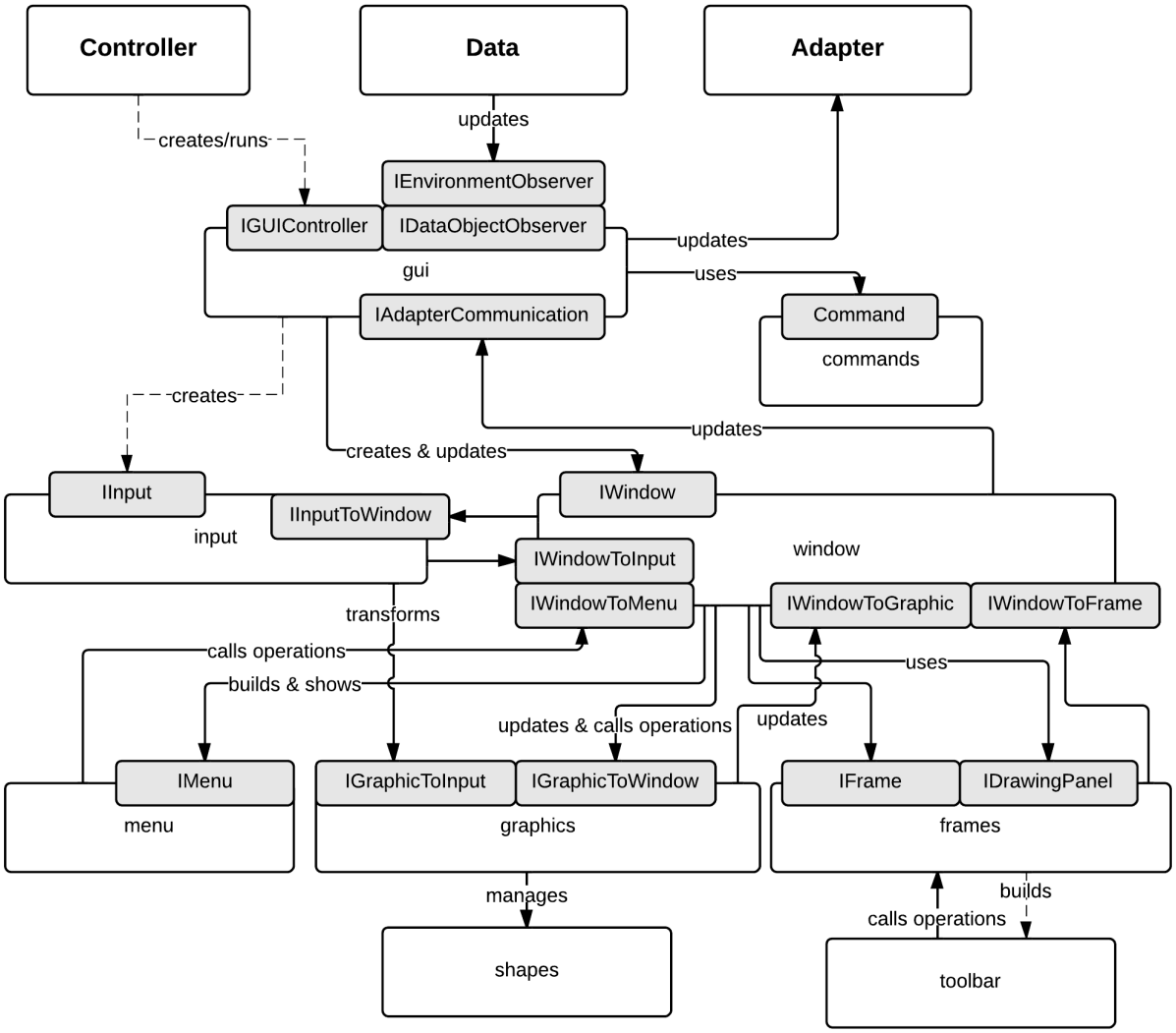


Figure 5.12: Interface communication in the GUI

5.4.1 GUI Package

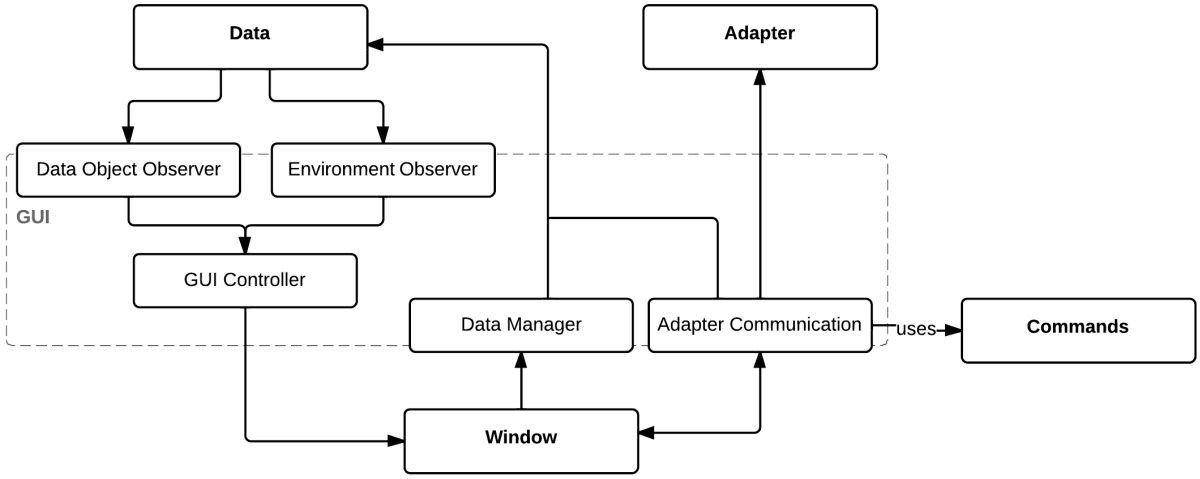


Figure 5.13: Simplified *GUI* package structure

The *GUI* package has the highest level of all packages, as previously shown in chapter 4.2.2. Therefore it is the only package that is allowed to communicate with the data component and the adapter directly. The structure is fairly simple, as shown in Figure 5.13. Incoming update

messages are handled by two observers, outgoing messages are done by the `Adapter Communication` (ACom). It uses commands from the *Commands* package to perform these actions. The `Data Manager` is used as a bridge between the actual data component and the *Window* package and it fetches information from the data component and returns it to *Window*. The `GUI Controller` is used to instantiate other classes and registers interfaces. A class not shown in the figure is the `GUI Settings` class. It houses several static attributes, which are used for setting up the editor, like object color, zoom speed and other important options.

Adapter Communication
The `ACom` is the only class in the GUI component that is allowed to communicate with the adapter. It uses commands, which actually do the communication with the adapter. To retrieve old data, like current position, or scale, the `ACom` has to retrieve this information from the data component, to be sure, the newest information is used. The `ACom` is also the basis for undo and redo actions, which is done with the help of the *Commands* package. If a message to the adapter is required, the `ACom` first builds the appropriate command, executes it and then stores it in the `undoList`, which is a simple array-list and should be handled like a FILO stack. This means, the last command put on top of the list, is the first that gets chosen for the next undo action. After an undo is done, the command is put onto the `redoList`, which functions similar to the `undoList` and puts every redo action again onto the `undoList` after its execution (see Figure 5.14).

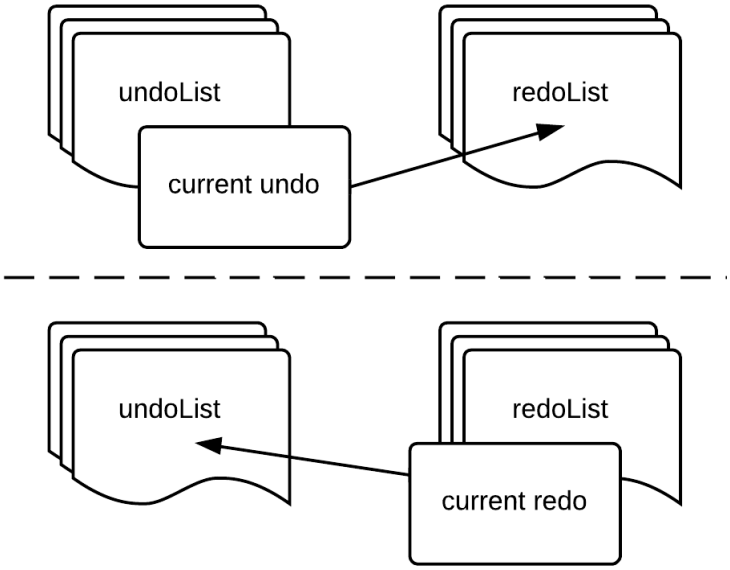


Figure 5.14: Functionality of the undo and redo lists

5.4.2 Commands Package

The *Commands* package includes all commands that are necessary to carry out all actions used in the GUI. They are needed for undoing and redoing actions. Figure 5.15 shows the structure of this package. Every command implements the `Command` interface, which is used by the `ACom`. It is also possible to combine certain commands together. The `Rotate Translate` and `Scale Translate` commands combine two different transformation commands into one. Listing 5.1 shows an example code of a more complex command. The commands used in the complex one, can either be parameters in the constructor, or created

within the complex command itself. `Set Properties` for instance uses many commands, therefore the commands will be given as parameters.

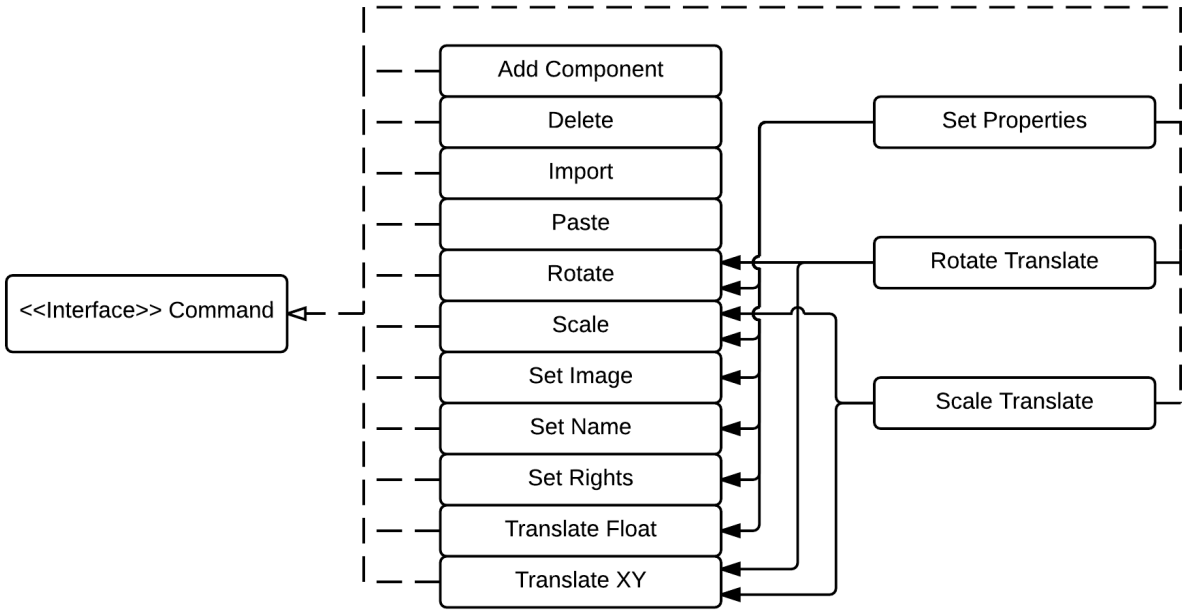


Figure 5.15: Simple structure of the *Commands* package

```
public class ComplexCommand implements Command{
    private ArrayList<Command> commands;

    public ComplexCommand(){
        commands = new ArrayList<Command>();
        commands.add(new SimpleCommand1());
        commands.add(new SimpleCommand2());
    }

    @Override
    public void execute(GUIObserverInterface goi) throws Exception {
        for(Command command : commands)
            command.execute(goi);
    }

    @Override
    public void undo(GUIObserverInterface goi) throws Exception {
        for(Command command : commands)
            command.undo(goi);
    }

    @Override
    public void redo(GUIObserverInterface goi) throws Exception {
        for(Command command : commands)
            command.redo(goi);
    }
}
```

Listing 5.1: Sample code of a complex command

It should be noted, that `Set Properties` has also an array list as parameter that allows for adding an undefined amount of commands. This is for extensibility purposes, to allow new commands, if new panels are added to the properties frame. Currently the only command in this list is the `Set Rights` command. The other commands are separate, because they are needed for the properties frame and therefore could be forgotten easily. To combine commands, the simpler commands have to be stored in the complex command. The stored commands can be called by their methods, provided by the `Command` interface. It provides the

following three methods, which use all the `GUI Observer Interface` from the adapter as parameter to communicate with it:

- `execute (GUIObserverInterface goi)` is used to execute the command for the first time.
- `undo (GUIObserverInterface goi)` is the undo action of the command and undoes its `execute` action.
- `redo (GUIObserverInterface goi)` is the redo action of the command and could be compared to the `execute` method. Most of the commands just call `execute` in this method, but some commands require a special treatment in their redo actions, therefore this method exists.

5.4.3 Input Package

The *Input* package is only used for key and mouse actions. Figure 5.16 shows a simplified structure of the package. The `Input` class is used to create the `Input Controller` and to register the interfaces from *Window* and *Graphic*. The `Input Controller` creates instances of the rest of the classes and forwards communication from the `InputToWindow` class to the `Mouse and Key Listener` (MKL). All these mentioned classes, except the `MKL` are simple structured. Additionally, there are strategies, which are used by the `MKL`. They have four methods that can be accessed by the `MKL` and are used for the following actions: mouse pressed, mouse released, mouse moved and mouse dragged. Actually not all these methods are used by every strategy, some are just empty. There are plenty of strategies, which are swapped by the `MKL` on different occasions.

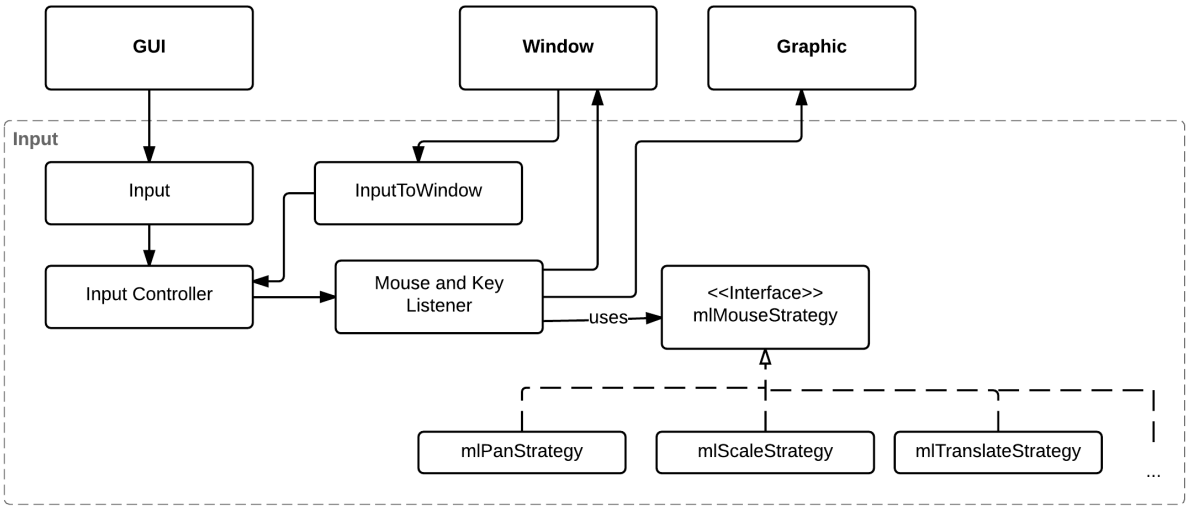


Figure 5.16: Simplified Input package structure

Mouse and Key Listener

The `MKL` is the most complex class in the *Input* package. It handles every input from the mouse and some from the keyboard. Shortcuts are usually done via Swing components. Because of the usage of strategies, most of the mouse events are easy implemented. The `MKL` only needs to call the current strategy. The problem however is to set up the right strategy. Usually a strategy is only created, when a mouse button is pressed, except the `mlPasteStrategy`, which needs to be created previously and the `mlTranslateStrategy`,

which is used only on one special occasion, when importing a .kmz-model and picking a location.

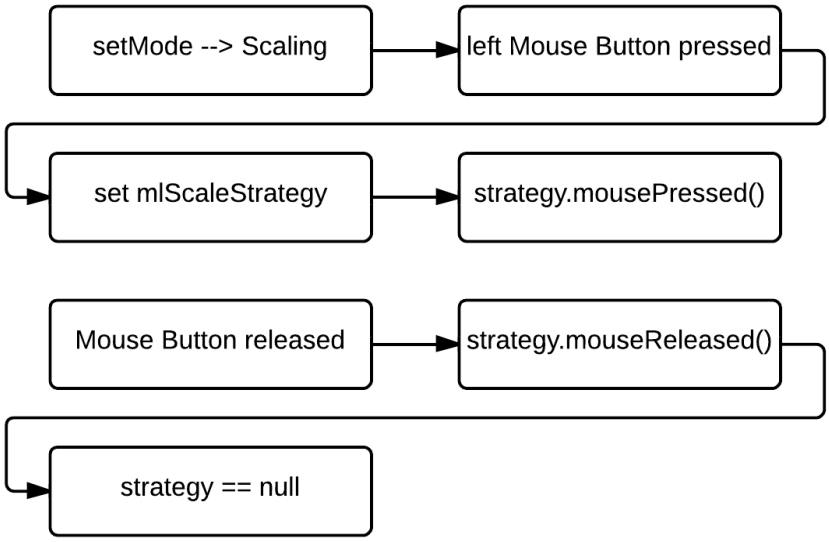


Figure 5.17: Simple example of mode and strategy workings

When a special operation, like rotation, or scaling is used, there is always a mode set in the `MKL`. With the aid of this mode, the strategy can be determined when pressing the left mouse button. It could be possible to set the strategy instead of a mode, but there is a problem, where, on some occasions, a different strategy is needed. The rotation of an object for example uses two different strategies, one for rotating the selected objects and one for translating the rotation center. These decisions can only be made, when the mouse button is pressed, because only then the right mouse position can be obtained. Figure 5.17 shows the workings of modes and strategies. First the mode is set in the `MKL`. Now when the left mouse button is pressed, the `MKL` creates a new strategy according to the mode and other influences that can occur. After that, `strategy.mousePressed()` is called. Now that the strategy is set, when other mouse actions are done, it is a simple function call. Strategies are removed, when the mouse button is released. Because it is possible to zoom in the editor, most of the strategies require the unscaled and untransformed editor coordinates and not the OWL coordinates.

5.4.4 Window Package

The *Window* package is used as central coordination for other packages. Its main purpose is to forward messages between them. Figure 5.18 shows its structure. Even the *Input* package has only clearance to access the *Graphics* package directly, because there are too many methods to cover. Every other package communication is done through *Window*. This structure was created to minimize the interfaces to a manageable amount. Because creating interfaces for every package would result in a large amount of them, each containing only few methods. The main hub is the `Window Controller`, which holds instances to all of the interfaces for the layers above and below. Therefore most of the classes have to access the `Window Controller`. Because all interface implementations only forward calls to other packages, there is no need for further explanation. `Mouse Coordinates` is a simple class, which is called from *Input*. It transforms the given coordinates back, with the help of the *GUI* package and then forwards the result to *Frames*, which also forwards it to *Toolbar*, where they will be displayed. Another interface class is `stateInput`, which is currently only implemented by `stateImport`. This is needed for a translation during the import of a KMZ model. It is needed

to set the coordinates, after the translation has finished. It is created by `WindowToFrame` and stored in `WindowToInput`.

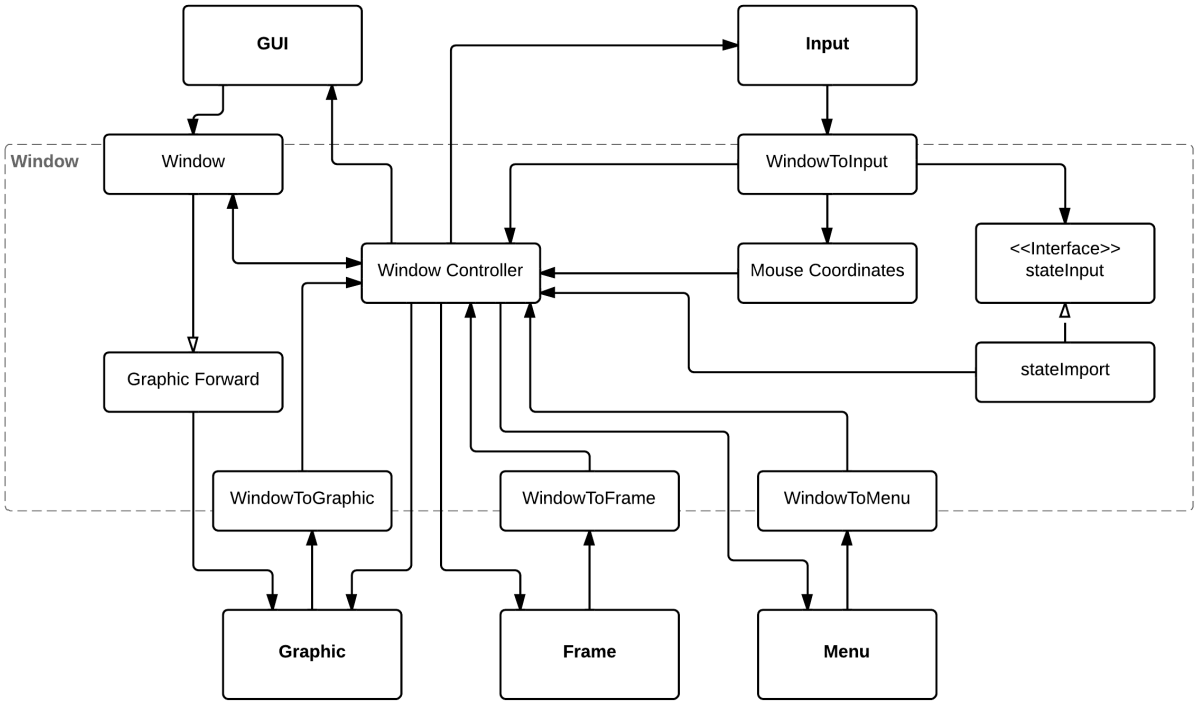


Figure 5.18: Simplified structure of the *Window* package

5.4.5 Menu Package

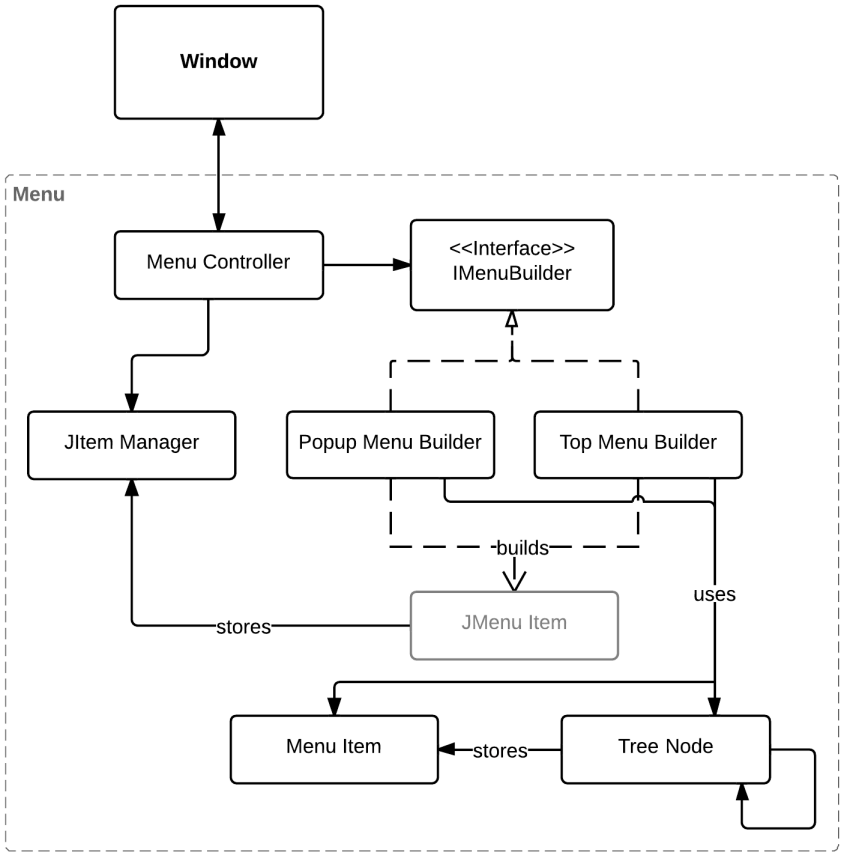


Figure 5.19: Simplified structure of the *Menu* package

The *Menu* package may only house a few classes, but the code in these classes is fairly complex. Figure 5.19 shows the basic structure. Here the `Menu Controller` implements the interface to the *Window* package and therefore is the only class allowed to communicate with it. It uses a builder interface, which is creates the menus. Currently there are two builders implemented. The `Popup Menu Builder` builds the popup menu that shows up, when the right mouse button is pressed and the `Top Menu Builder`, which creates the main menu on the top of the `Main Frame`. These two builders use a tree structure to help with the item creation. They also use `Menu Item`, which is stored in the `Tree Nodes`. After the tree is created, they use the structure and the `Menu Items` stored in them to create the Swing `JMenu Items`. These items are then stored in the `JItem Manager` to be accessed later on.

Building a Menu

The initialization of menu building is done in the `Menu Controller`. There, every entry for the menu is created in the method `createMenu()`. To create a menu entry, a callable function has to bee defined, as shown in Listing 5.2. In this example the method is called `function`. Callables need a return value in Java, but there is currently no use for it in the implementation. Then the item has to be added to a builder, with `addItem(...)`. The first parameter of this method is used for the submenu's name, where the item should be put. If the name does not exist, the entry is created at the root. The second parameter is the items own name, the third is the callable. If this parameter is set to null, a submenu entry is created. Then, every time the name of the submenu is used as first parameter, the entry will go into the submenu. It is possible to create submenus within submenus. After that a key binding can also be included as parameter, but it might be null as well, if not needed. The last parameter is a Boolean, which will create a separator before the component if the parameter is true. The order the items will be added is dependant on the order the items are added to the builder. Figure 5.20 shows the menu, which is created by the code in the listing.

```
    //create the functions
    final Callable<Void> function = new Callable<Void>() {
        public Void call(){
            window.doSomething();
            return null;
        }
    };
    //add an actual menu entry to builder
    menuBuilder.addItem("Menu Name", "Entry 1",
            function, KeyStroke.getKeyStroke(
            KeyEvent.VK_N, ActionEvent.CTRL_MASK), false);

    //add a submenu
    menuBuilder.addItem("Menu Name", "Sub menu Name",
            null, KeyStroke.getKeyStroke(
            KeyEvent.VK_N, ActionEvent.CTRL_MASK), true);
    //add a menu entry to the submenu
    menuBuilder.addItem("Sub menu Name", "Entry 2",
            function, KeyStroke.getKeyStroke(
            KeyEvent.VK_N, ActionEvent.CTRL_MASK), false);
    ...
    //add menu to JItemManager
    itemManager.setMenuItems(menuBuilder.getMenuItems());
```

Listing 5.2: Example code for adding menu entries

Figure 5.20: The menu created by the code in Listing 5.2.

Before building the menu though, the items are stored in a tree structure. Each node on this tree holds the information provided by the `addItem(...)` method. The structure was used for an easy build process. The tree is stored in the corresponding builder. After every entry is added, the `buildMenu()` call creates and returns the menu. Figure 5.21 shows the process. The builder stores all created menu items separately in a `HashMap<String, JMenuItem>`. This map should be given to the `JItem Manager`. The string which is used to access them in the `JItem Manager` is the name of the menu item. The name should be predefined in the bundles to prevent misspelling. The `JItem Manager` is currently only needed for activating and deactivating menu entries.

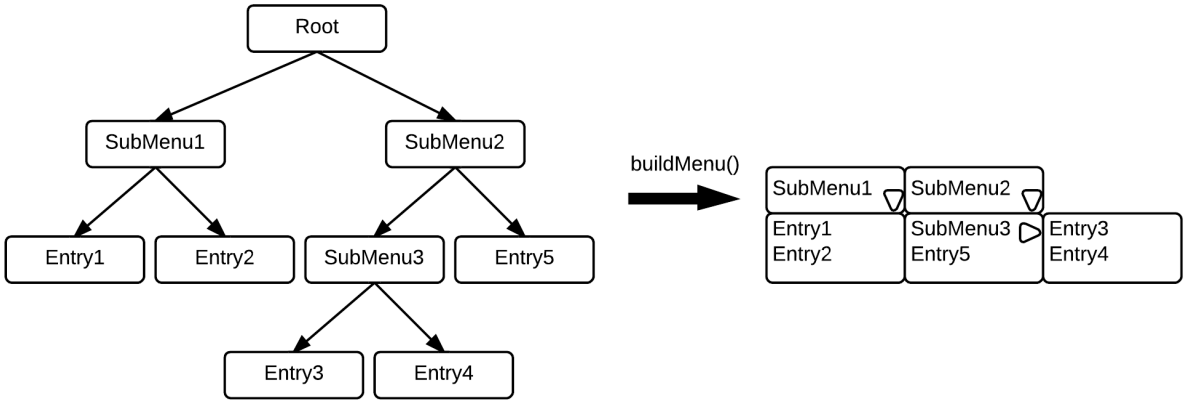


Figure 5.21: Building the menu

5.4.6 Graphics Package

Figure 5.22 shows the structure of the *Graphic* package. Nearly every class in this package uses classes from *Shapes*. This is not shown in the figure. The `Graphic Controller` functions as interface implementation to the *Window* package, the `GraphicToInputFacade` to *Input*, therefore these two have to access most of the other classes in this package. Because much internal communication happens, a mediator pattern was used, called `Internal Mediator`. Because of the amount of internal communication, the mediator has a lot of methods, but in return it makes the package structure much clearer. The `Graphic Controller` and the `GraphicToInputFacade` classes could also have used the mediator, but it was decided against it, because this would blow up the proportions of the mediator class even further. The `Shape Manager` is used to store shapes from the *Shapes* package. It uses the `Shape Factory` to create shapes. The `Copy Manager` is used for the copy operation and stores the ID of copied shapes. For identifying the current selection, the `Selection Manager` is used. The `Server Transformation Manager` is used for transforming shapes when a server update happens. The `Editor Transformation Manager` (ETM) is used for transforming dragging shapes and can use two different sorts of strategies, which determine how to handle shape collisions. Dragging shapes are shapes, which are used during operations like moving and rotating and differ from normal shapes. They will be discussed in chapter 5.4.6.1.

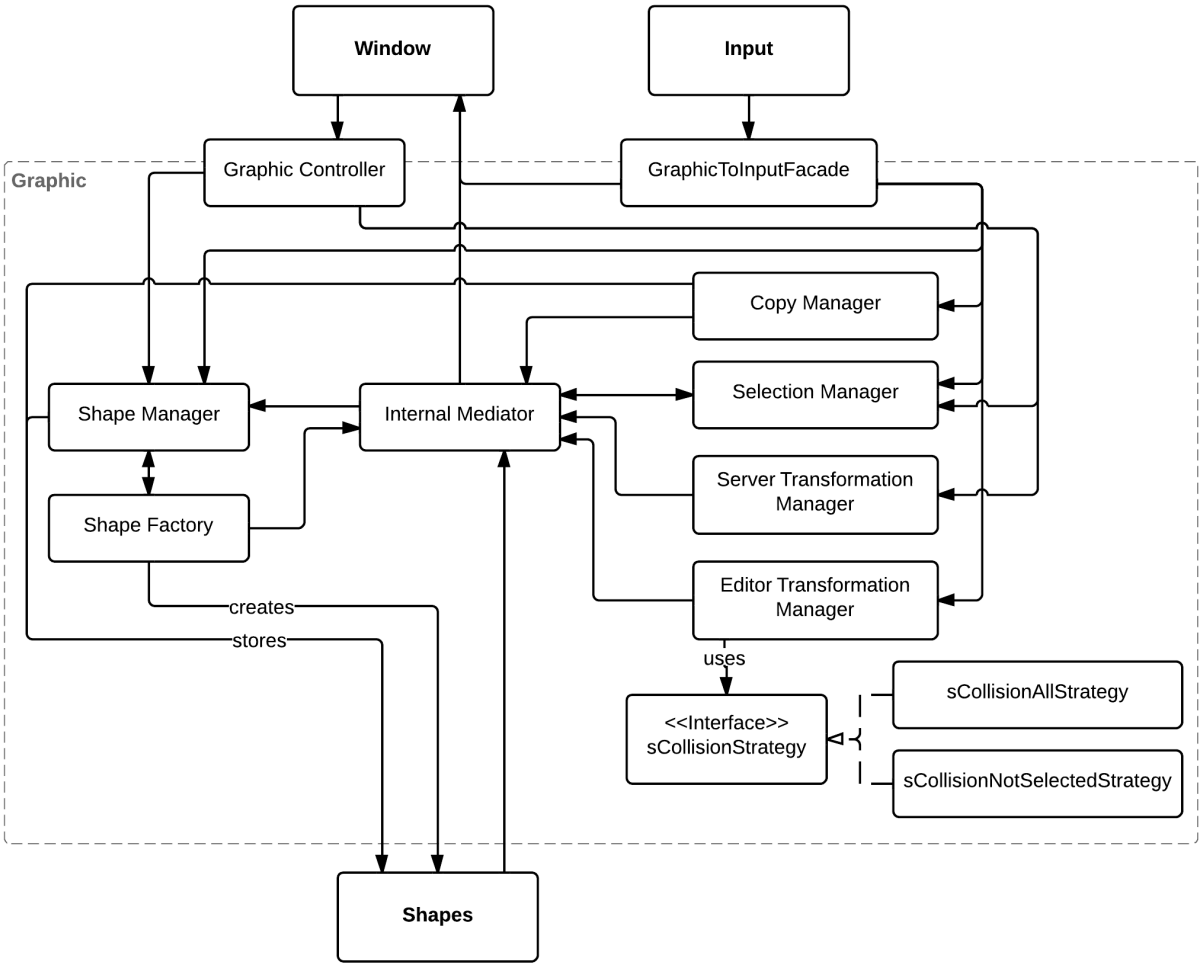


Figure 5.22: Simplified structure of the *Graphic* package

Shape Manager and Shape Factory

The `Shape Manager` stores every shape used in the editor. It has several lists, `shapes`, which stores every shape in the foreground and `background`, which stores shapes in the background. Background shapes are shapes that can not be modified and do not throw a collision when normal shapes are translated. The next list is `avatarShapes` and contains the avatars. `DraggingShapes` stores all shapes which are currently dragged. This type of shape is explained later in section 5.4.6.1. `SelectionRectangle` stores the current selection rectangle and `border` is used, when a border is painted around the selection during a rotation or scale operation. The `Shape Manager` implements all functions to manipulate its stored shapes. It also paints them in a specific order. Figure 5.23 shows the order, which is important, because otherwise shapes will cover each other. All shapes in the background are painted first. Then the normal non selected shapes will be drawn and after them the selected ones. The selected shapes need to be painted later for normal shapes not to cover their border. Then the avatars are painted, in order to see them when they are standing on an object. The last shapes that need to be painted are the dragging shapes, the selection rectangle and the transformation border. All three showing up at the same time is not possible due to restrictions in the implementation. Only dragging shapes and transformation border can exist at the same time.

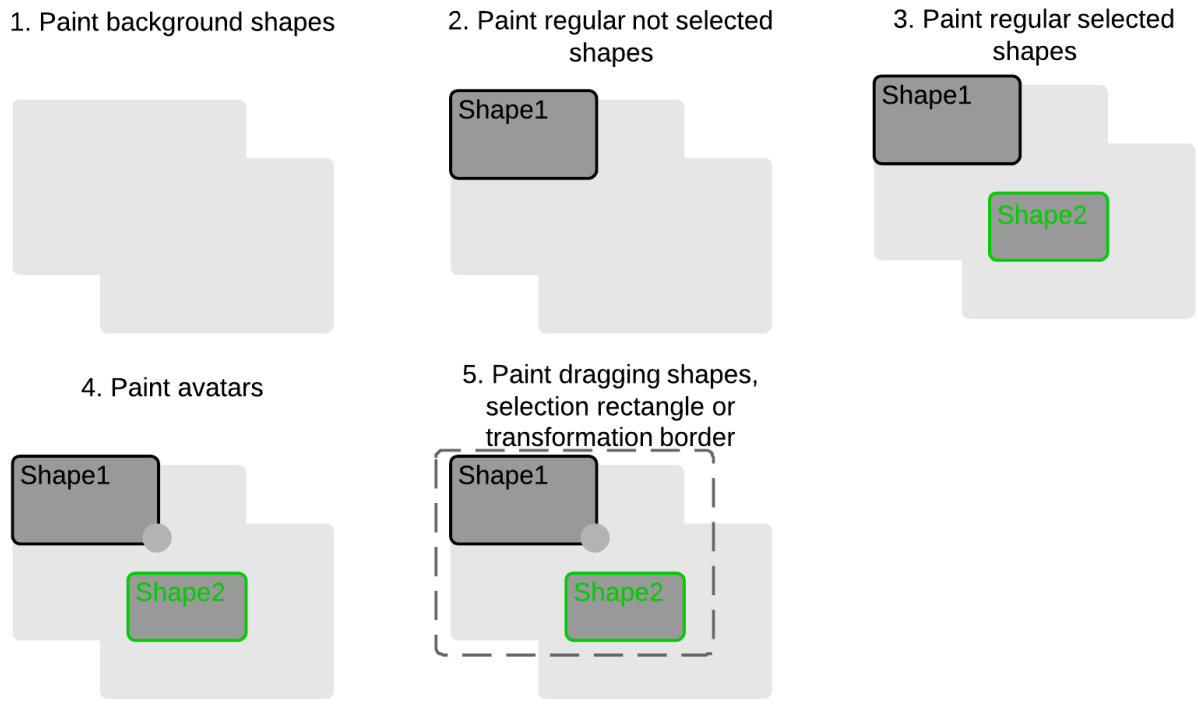


Figure 5.23: Order in which shapes are drawn

The `Shape Manager` creates shapes through the `Shape Factory`. It should be noted, that the `Shape Factory` has three static variables that determine the outcome of the building process and should be used as types. `RECTANGLE` and a `CIRCLE` determine if the shape will be a rectangle, or circle, whereas using `AVATAR` will result in an avatar shape. The following methods can be used for creating specific shapes:

- `createShapeObject` creates a standard shape. All three static variables can be used with this method.
- `createSimpleShapeObject` only creates the selection rectangle, therefore only `RECTANGLE` can be used.
- `createDraggingShapeObject` creates a dragging shape. Only `RECTANGLE` and `CIRCLE` can be used with this method.
- `createTransformBorder` builds a transformation border. This method does not have a type field, therefore none of the predefined types can be used.
- `createToolTip` creates a tooltip shape. This method is similar to `createTransformBorder` and does not use a type.

Copy Manager and Selection Manager

The only purpose of the `Copy Manager` is to store IDs of shapes that were copied with the appropriate command. When the current user does a copy action, the `Copy Manager` saves the IDs of all currently selected shapes. When a paste action is done, the shapes are retrieved with the help of the stored IDs. The `Selection Manager` is used for everything regarding the selection of shapes. It stores the IDs of selected shapes in a list for quick access. The most important part of this class is to switch the selection of shapes. It can also calculate the center of the current selection, which is used when doing a copy action. Another function of this class is to transform the selection rectangle and to scale it to the mouse position. It also finds shapes in the selection rectangle.

Transformation Managers and Collision Strategies

The `Server Transformation Manager` is used for translation, rotation and scaling of normal shapes. The `ETM` is used for the same operations but with dragging shapes. The difference between transforming these two types of shapes lies in their methods. Translations for normal shapes originate from the server and therefore use normal coordinates. Translations of dragging shapes stem from the editor and do not use coordinates, but rather a starting and an end point. This is important, because dragging shapes may be moved in groups, whereas normal shapes are only moved one at a time. And passing one distance is much simpler than passing a list of new coordinates. Therefore the actual coordinates for the dragging shapes are calculated in the shapes themselves with the distance at hand. Rotation and scaling bring even more difficulties to the `ETM`. Then an additional shape, the transformation border, has to be transformed as well.

Another function of the `ETM` is to check for collisions, whether a dragging shape overlaps with a normal shape or not. The collision check itself is done in a strategy, which has to be an attribute in the transformation call. If the strategy in the method call is null, the `ETM` will use the strategy from the last collision check. There are currently two different strategies. The first is `sCollisionNotSelectedStrategy`, which is used for normal transformations, like translation, rotation and scaling. This strategy finds collisions with all normal shapes, except the currently selected ones. Because the selected shapes are transformed at the moment, they do not need to be checked. The second strategy is `sCollisionAllStrategy`, which finds collisions with all shapes and is used for copy and import operations.

5.4.6.1. Shapes Package

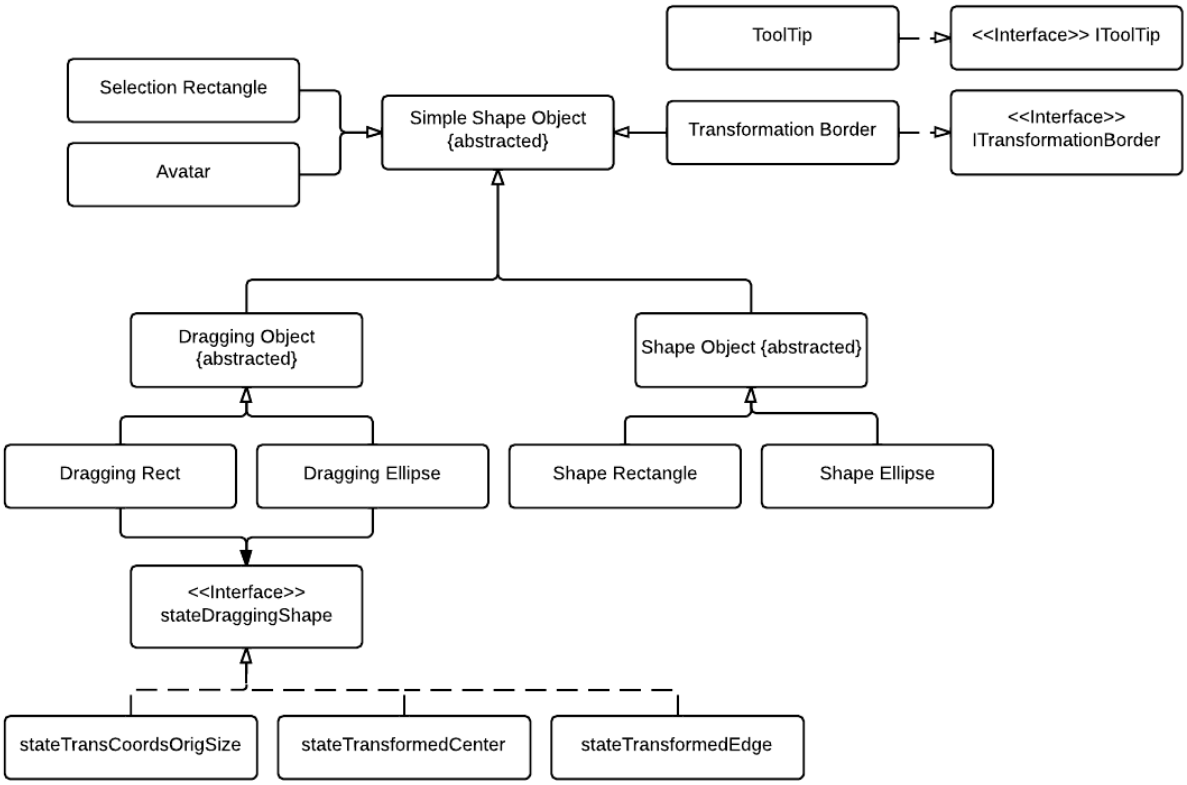


Figure 5.24: Simplified structure of the *Shape* package

Figure 5.24 shows the structure of the *Shapes* package. Every class, except `ToolTip` is a derivation from the abstract class `Simple Shape Object`. The most important classes here are the implementations of `Shape Object` and `Dragging Object`. There are two of each,

which either implement a rectangle or an ellipse. The normal `Shape Objects` are the ones representing items in the virtual world. A `Dragging Object` is a shape, which is seen, when doing some kind of operation, like translation, rotation, or scaling. They represent a normal `Shape Object` during the transformation to help users to see where the normal `Shape Objects` will be after finishing the process. `Dragging Objects` are slim copies of `Shape Objects`. The usage of both `Shape Object` and `Dragging Object` is done for synchronization purposes, which will be explained later on. The `Avatar` class and the `Selection Rectangle` class are self-explanatory. `Transformation Border` is a delicate class which is used, when a border for transformations is needed. This border is used during rotation and scaling operations. It is a simple border encompassing all currently selected shapes. Additional it has four edges which are used for drag and drop. The border also has two different modes, which either have a center in the middle of it or not.

Figure 5.25 shows all different versions of shapes. The first is the regular shape, either an ellipse or a rectangle, with a black (or red, if selected) border and name. It is filled with a predefined color. The second is the dragging shape, which is the implementation of the `Dragging Object`. This is either a rectangle or an ellipse with a grey border and no name or fill. The selection rectangle is a rectangle with a grey, dotted border and is used for selecting objects. The transformation border is a rectangle, with grey border and four tiny rectangles at its corners. It may have another rectangle in the center. This shape is used for rotation and scaling objects. Only during rotation does the border have the center rectangle. The avatar is a grey circle of predefined dimensions and represents a user in the virtual world. The tooltip shape is used to present an object name which is too long to fit in the object itself. It could be used for other occasions, but currently it is only used for showing object names.

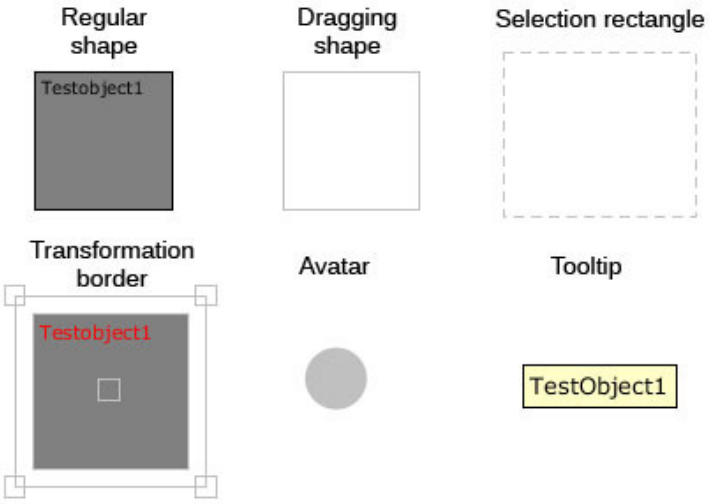


Figure 5.25: All different versions of shapes

Figure 5.26 shows the process of transforming a shape, when it is printed on screen. The initial shape has the transformed coordinates from the data component. The shape then is transformed with its own settings. These transformations are rotation and scaling and are individual for every shape. Each shape has rotation and scale values stored, which will be altered, when the shape is transformed. After the local settings have changed the shape, it is then transformed again with a global setting. These global setting contains the global zoom factor, as well as translations to keep objects centered. This setting is used for every shape. If the printed shape is a `Shape Object`, the representational image, if it exists, will be drawn after the shape, to lay over it. Then the name is printed.

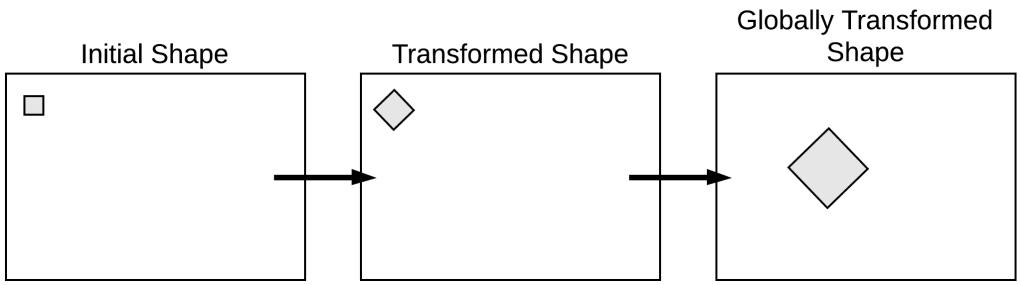


Figure 5.26: Transformation of a shape for onscreen printing.

Dragging Object

The `Dragging Object` classes were created for transforming a shape without the need to change the `Shape Objects` themselves. For example, if a user drags a normal `Shape Object` and another user changes it too, there would be a conflict and it would be hard to resolve. If the action of the second user would change the object, it would disturb the dragging action of the current user. Ignoring the changes of the second user would result in the editor being out of sync with the server. In order to prevent this problem, every time a transformation has to be made, a `Dragging Object` is created. The `Dragging Object` is not an object in the virtual world. It is only a hollow copy of a `Shape Object`. Figure 5.27 shows the functionality in detail when moving an object. When a user wants to move a `Shape Object`, a `Dragging Object` is created, which shows the user the position the `Shape Object` would have, when the translation is done. After finishing the operation, the server is signaled and the `Dragging Object` is deleted. The server then will send a change message to the editor, which will result in the position change of the actual `Shape Object`.

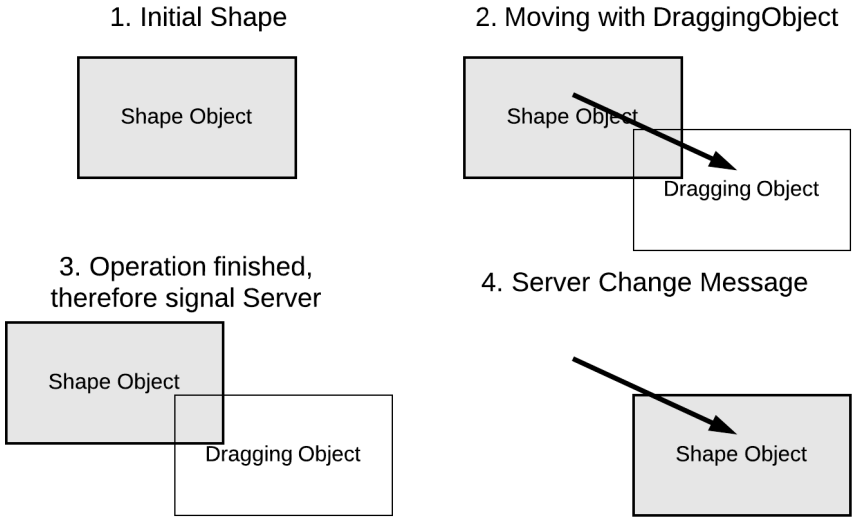


Figure 5.27: Functionality of `Dragging Object`

There are several different states that are used in conjunction with `Dragging Objects`. These states are used for retrieving coordinates. When no state is set, the methods `getX()` and `getY()` return the coordinates of the initial shape. When `stateTransformedEdge` is set as state, the methods return the coordinates of the transformed shape. The method `getBounds()` is used in this state, which returns the coordinates of Swing objects. The second state is `stateTransCoordsOrigSize` and is needed after a rotation. As Figure 5.28 shows, using `getBounds()`on rotated shapes will clearly produce wrong coordinates. The bounds are much bigger than the actual size of the shape, but the coordinates for a not rotated shape are needed. Therefore the coordinates are calculated differently in this state. The center is used as origin

and the coordinates are calculated with the help of the original width and height of the shape. The results are the appropriate values for X and Y. The reason behind not using just the initial shape is that the initial shape has never been rotated. Therefore, the transformed shape, which has been rotated, has to be used.

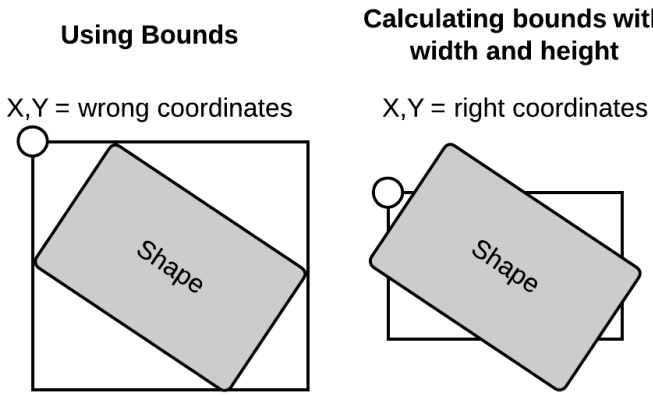


Figure 5.28: Difference between using Swing bounds and calculating the coordinates

Transformation Border

The `Transformation Border` is used during rotations and scaling operations. It encloses all currently selected shapes. Its interface has several different static variables. The first set determines whether or not the `Transformation Border` shows its center. These variables have to be used in the attribute `mode` in the constructer. When using `MODEONECENTER`, the border will show its center, when using `MODENOCENTER` the border will not display it. The next set of variables is used to determine where a point is on the border. This is for checking a mouse position. The method which is used for this check is called `checkShapes(Point p)`. If `INNOTHING` is returned, the point is neither in the center nor in the edge shapes. `INROTATIONCENTER` points out, that the point is in the center and `INEDGES` shows that the point is in one of the edges. If this is the case it may be helpful to know in which of the edges. Therefore another set of variables exists, that determines that. They are return via the method `getCurrentClicked()`.

5.4.7 Frames Package

The frames package holds all frames that can be displayed in the editor. All frames extend the `JFrame` class from Java Swing. Figure 5.29 shows the simplified structure of this package. The `Frame Controller` is used as communication center for different classes and the *Window* package. The rest of the contents are frames that are shown during runtime. The `Main Frame` is the prime frame, where most of the user input is done. It creates the *Toolbar* items. Furthermore it uses the `Drawing Panel`, which paints the actual shapes and is wrapped in a `JScroll Pane`. The `Import Frame` is used for importing models. The `Properties Frame` is used to set all kinds of properties. It also has the `Properties Rights Pane`, which is used for setting up rights. Because both, the `Properties Frame` and the `Properties Rights Pane` can be seen as forms for user input, they will not be explained further on. The `IPermissionTableEntry` interface is used in the *GUI* package to determine changed settings in the `Properties Rights Pane`. The last frame is the `Image Selection Frame`, which is used to select the representational image. It will also not be discussed further on, because it holds only images, the current user can select from. The difference between the `Image Button` and the `Image Toggle Button` should be noted. The `Image Button` extends a

normal `JButton` with an image. The `Image Toggle Button` extends the `JToggleButton` the same way. The only difference between these two buttons is that the `JToggleButton` stays pressed until it is pressed again. The `Waiting Dialog` is a dialog created for waiting. This is used when something is loaded or saved (like when importing a new object). It blocks user interaction within the current frame.

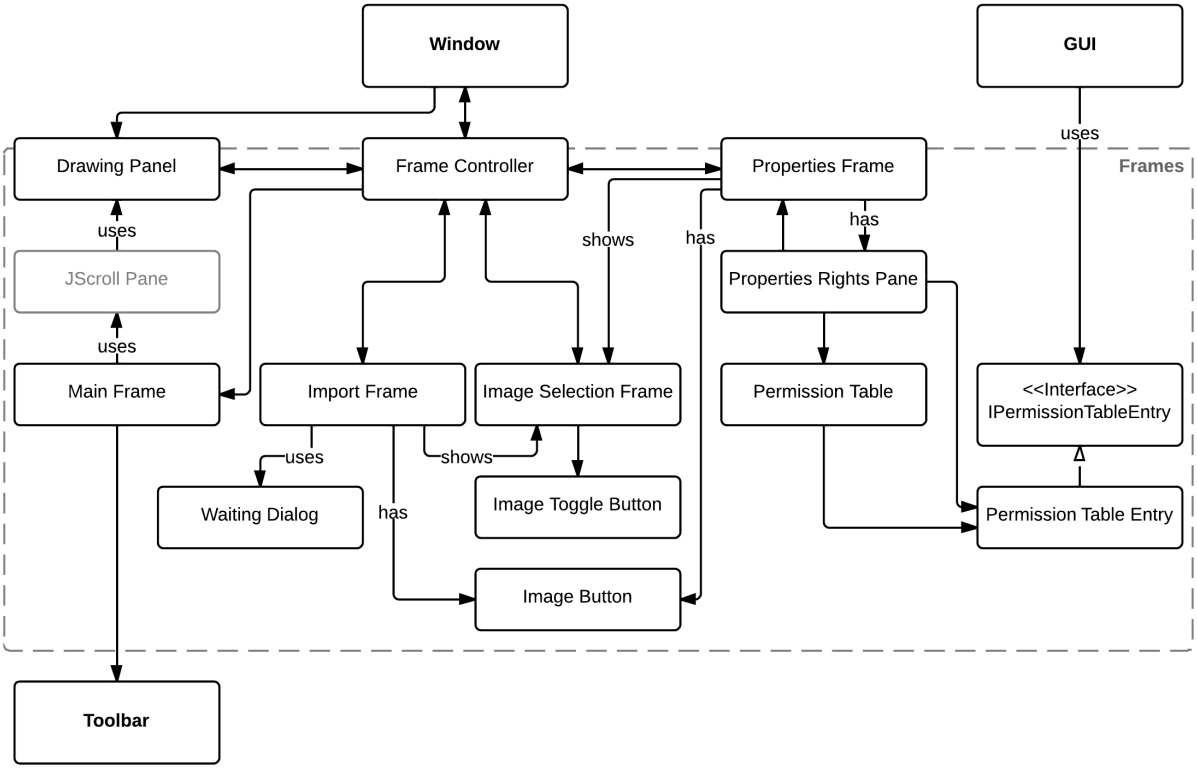


Figure 5.29: Simplified structure of the *Frames* package

Drawing Panel

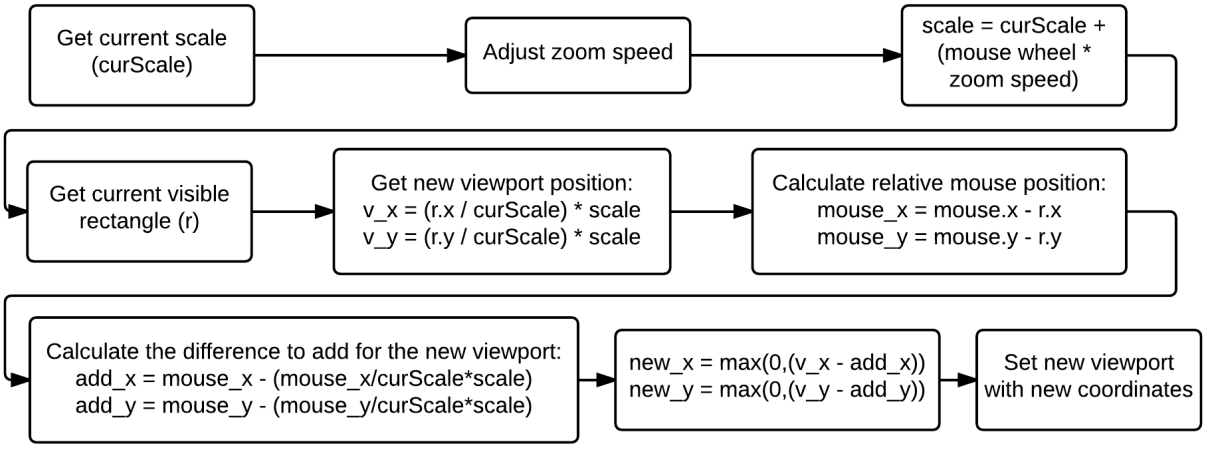


Figure 5.30: Zooming and setting up a new viewport

The `Drawing Panel` represents the drawing area, where the users are editing the shapes created in the *Graphics* package. It is also the origin of the repaint call for the *Graphic* package. Because it is the main drawing area, its methods implement everything needed for zooming, resizing and changing the viewport. The only noteworthy part is the zooming implementation. Figure 5.30 shows how it is implemented. First the current scale has to be backed up. Then the zoom speed is adjusted. This is necessary if the zoom level is 1, because

the next zoom will reduce it to 0. So every time this could happen, the zoom speed is divided by 10. In return, when the zoom level hits a maximum it will be multiplied by 10. After this, a new scale will be calculated, which takes the mouse wheel input and multiplies it with the adjusted zoom speed. Then the whole panel is redrawn with the new scale. Unfortunately, the viewport does not change according to the scale, so it has to be repositioned. First of all, the current viewport has to be fetched. Then its new position in the new scale has to be calculated. This calculation takes the current viewport and transforms its coordinates to the new scale, which are called `v_x` and `v_y`. Then it retrieves the current mouse positions and calculates `mouse.x` and `mouse.y`. These values are also transformed into the new scale. The difference between old mouse coordinates and new mouse coordinates is then added to the new viewport coordinates.

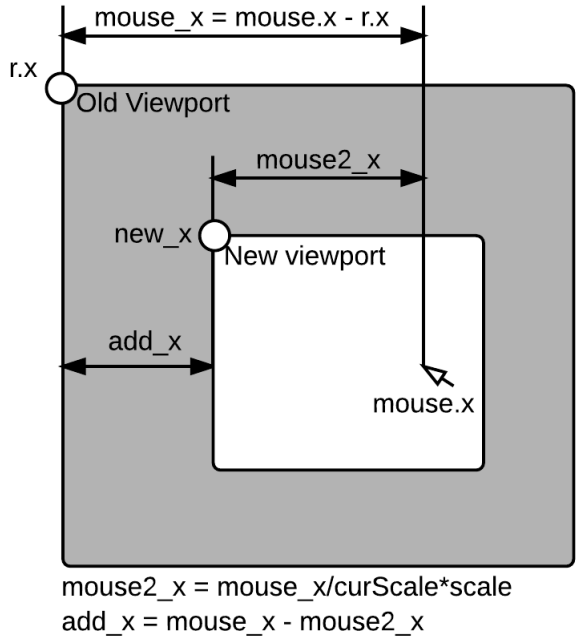


Figure 5.31: Calculating the new viewport (y-coordinates are equivalent)

In order to make the calculation more clearly, Figure 5.31 shows how the new viewport is set up. The variable `r.x` is the x-coordinate of the old viewport, `curScale` the current scale and `scale` the new scale. Transforming only the coordinates into the new scale will not work, because it would be the same coordinates, only in the new scale. So the viewport, which like all Swing components has its coordinates on the top left, would wind up zooming to the top left corner. Therefore new coordinates have to be calculated. For the zooming implementation, the mouse has to stay in relative position to the new viewport. So first the distance between current mouse position (`mouse.x`) and old coordinates is calculated, which is called `mouse_x` and `mouse_y`. These new distances have to be transformed to the new scale. Therefore they are divided by the old scale and multiplied with the new one. The results are called `mouse2_x` and `mouse2_y` in Figure 5.31. To calculate the distance, which should be added to the old viewport coordinates, `mouse2_x` has to be subtracted from `mouse_x`. These values, called `add_x` and `add_y`, can then be substracted from the old viewport coordinates after they are transformed with the new scale. The `mouse2_x` and `mouse2_y` calculation is done directly in the `add_x` and `add_y` part of the implementation, as shown in Figure 5.30. `Add_x` and `add_y` can then be subtracted from the new viewport coordinates called `v_x` and `v_y`. In order to prevent negativity in the viewport coordinates, it is necessary to either take a non-negative value of the calculation, or 0.

Frames and Waiting Dialog

The `Import Frame` and the `Main Frame` use the `Waiting Dialog` for importing models. It should be noted, when using this dialog, a new thread has to be created, because the `Waiting Dialog` blocks the current thread. The first idea was to make the dialog modal, but it is not displayed in OWL. So, to show the dialog, it has to block the thread that is used to start it. This means it also blocks the thread of the frame used to create it and prevents further work in this frame. Therefore, a new thread has to be constructed before the `Waiting Dialog` is shown which should contain the work that needs to be done during showing the dialog. The text can be changed while the dialog is present.

5.4.7.1. Toolbar Package

The *Toolbar* package only consists of three classes (as shown in Figure 5.32), which are all instantiated in the `Main Frame` of the *Frames* package. The `Bottom Toolbar` is more or less a naked toolbar, containing only the coordinates of the current mouse position. It has only a method to set new coordinates. The `Transformation Bar` is only shown when the user is doing either a rotation or a scaling operation. It has two buttons which either cancel the transformation or applies it to the shapes. The `Undo Bar` also has two buttons, which are used for undoing and redoing actions the user has made.

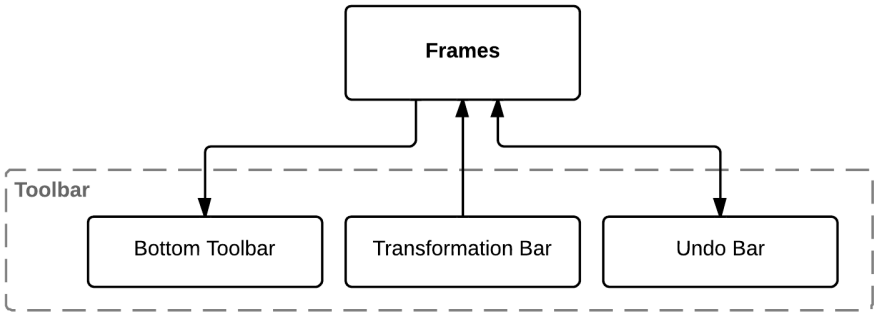


Figure 5.32: Simplified structure of the *Toolbar* package

5.5 Discussion

The implementation uses Java, like OWL itself. There are certain problems which occurred during development, because OWL handles some things differently, or does not offer important information. Therefore some parts of the adapter implementation needed a couple of tricks, like the `Late Transformation Manager`, to get all operations work properly. While the adapter side had difficulties with getting the prototype work together with OWL, the editor part was more a conceptual challenge. Because it generated much code only to do minor operations, like moving an object, the design had to grow with every new addition. Another huge problem was the implementation of the Swing shapes in a zoomable view. Because Swing does not offer any kind of zooming mechanism, the constant change of scale was a inconvenient to implement. Shapes also have a scale value which makes the implementation needlessly complex.

Furthermore the question was raised, in which state the shapes should be transformed, because some shapes, like the `Selection Rectangle` do not have to be transformed. But

doing so would lead to the problem of finding shapes in the rectangle. Therefore and for the sake of being uniform, every shape is created with the initial unscaled and untransformed coordinates. Only when painting them they are transformed, first locally, then globally. The next challenge was to transform the shapes, without disturbing the server during the operation. As stated before, using the original shape would either result in the user getting disturbed during the action, or the editor being out of sync with the server. Therefore the dragging shapes were used, to prevent these two problems. The implementation can be difficult, but the usage of the editor should be as simple as possible.

6 Usage and Evaluation

The aim of this chapter is to show how the editor can be used and how it compares to the build-in tools available in OWL. Section 6.1 explains the usage of the editor prototype and all of its functions in detail. Section 6.2 shows a study, which discusses the usability of the editor compared to the tools OWL offers.

6.1 Usage

This section describes the editor from the usage point of view. First a quick overview of all the contents of the editor will be given. Then it is explained, how objects can be imported, followed by moving and transforming objects. Last, the properties frame will be discussed.

Overview

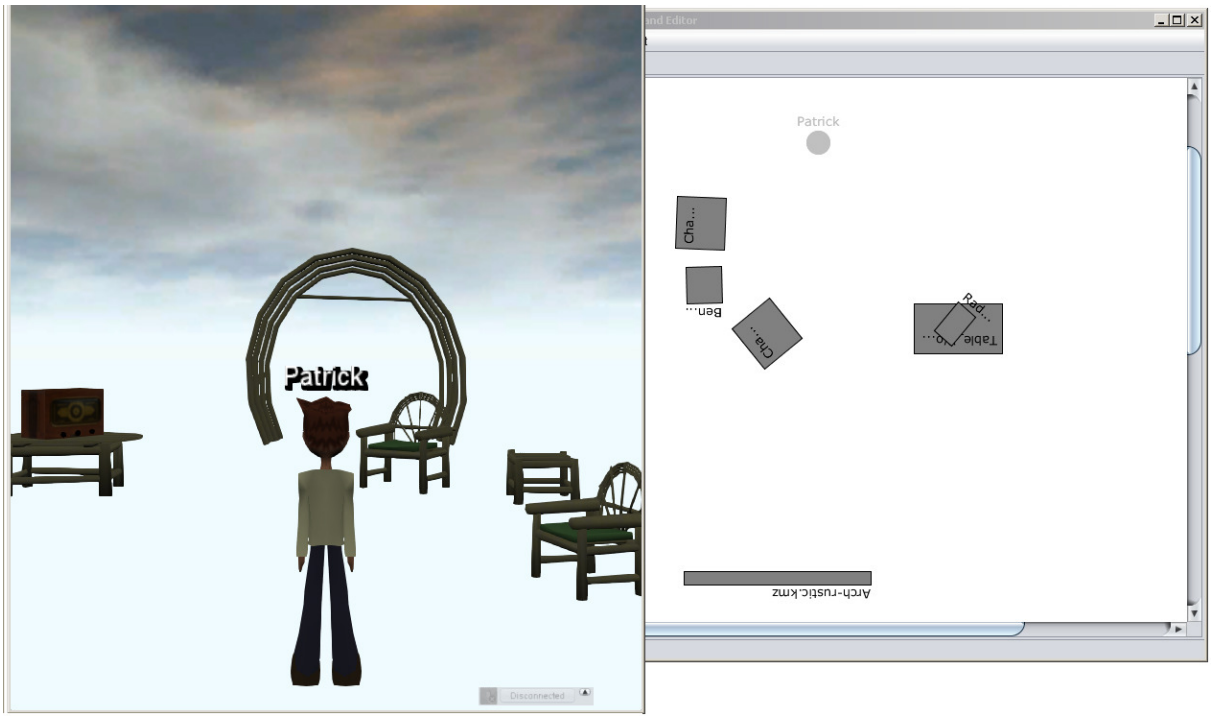


Figure 6.1: The editor (to the right) loads the OWL world automatically and shows modifications in real-time.

As described in chapter 4 the prototype works in sync with OWL. So, the current world, the user is in, can be automatically seen in the editor as shown in Figure 6.1. In other words, the users do not have to import the world or start an import process. Furthermore, all changes made in OWL can be seen directly in the editor in real time. To start the editor it has to be selected under the *Tools* register in the client. The entry name for the prototype is *OWL Editor*. Figure 6.2 shows the different elements of the editor. (1) is the main menu, which includes three entries. *File* is used to load and save worlds, *Edit* is used for all kinds of transformations and *Import* is used for importing objects. (2) displays the top toolbar. This bar contains the undo bar, where users can undo or redo their actions. It also contains the

transformation bar, which is only shown, when a rotation or scale operation is done. (3) is the work area, where the users can make their changes to the world. (4) displays an actual object in OWL, whereas the object in (5) is currently selected. (6) shows the representation of an avatar in the editor. All users currently in the world will be represented in such a way. (7) is the drop-down menu. This menu contains most of the functions in the *Edit* entry of the main menu. The drop-down menu will be displayed when the right mouse button is clicked. (8) is the bottom toolbar which contains the coordinates of the current mouse position. It is also used to show important messages to the user.

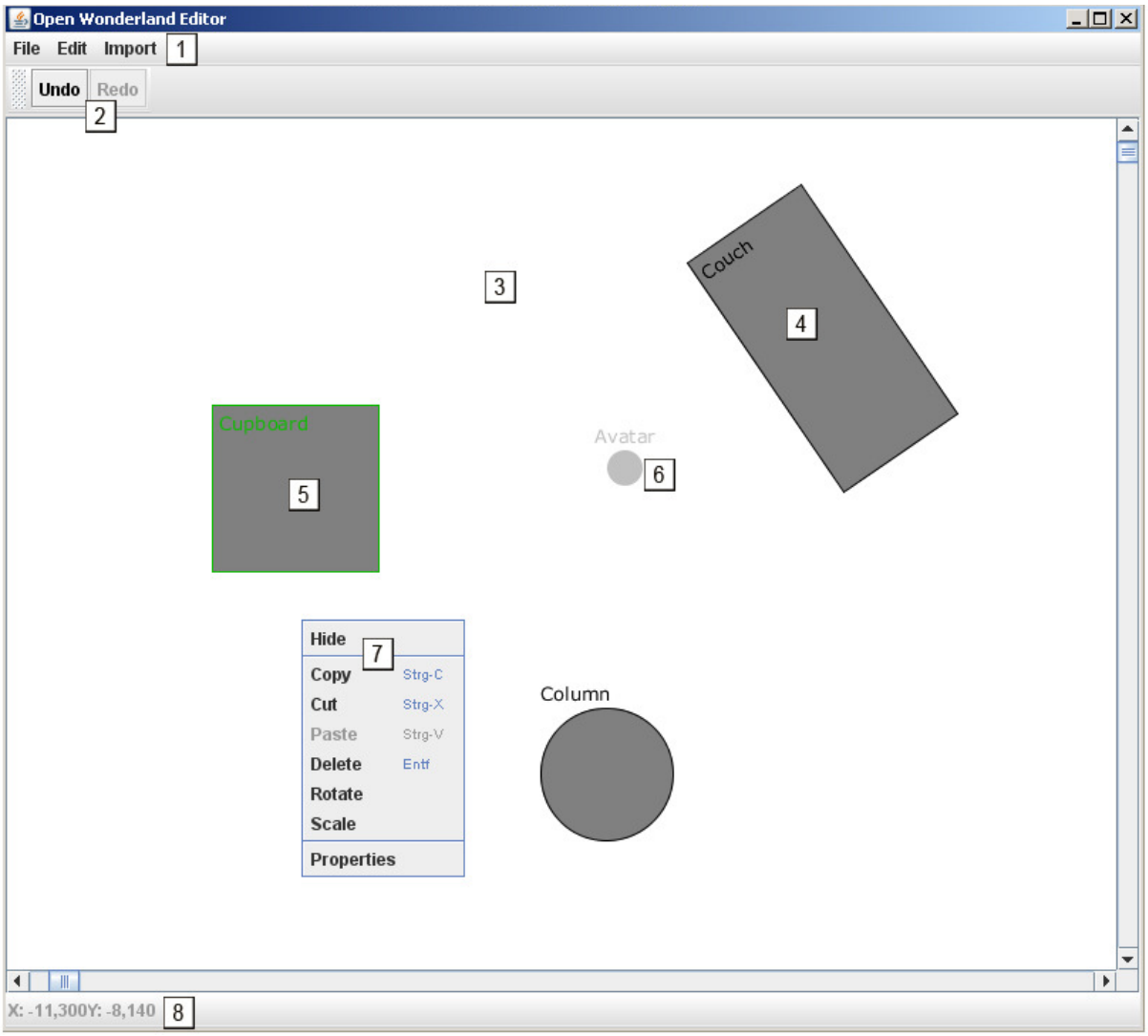


Figure 6.2: Editor elements: 1: Main menu, 2: Top toolbar, 3: Work area, 4: Object, 5: Selected object, 6: User avatar, 7: Drop-down menu, 8: Bottom toolbar;

Importing Objects

Figure 6.3 shows the import frame, which can be accessed through the *Import* entry in the main menu and then selecting *KMZ*. The import process can also be started by dragging the .kmz-file directly into the editor's drawing panel. The important part during import is to choose a module name, which is not already taken. It is also possible to select different servers. Position, rotation and scale can be entered manually. If users want to directly place the object in the world before importing the model, *Choose Location* will allow them to do so. This will put the users back into the work area, where they can select a suitable place for the object (Figure 6.4). It should be noted, that all z-coordinates are calculated with the height of

the object in mind, so that users only have to pick the same z-coordinate to align several objects at the same height.

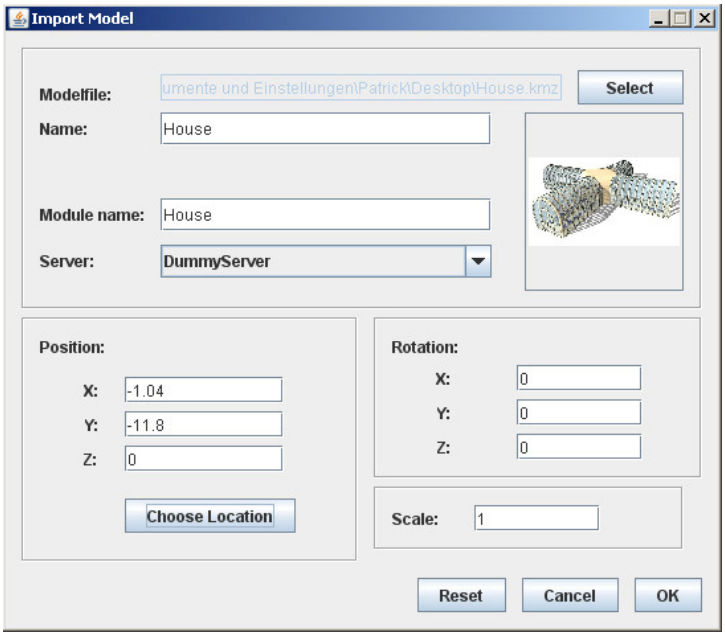


Figure 6.3: The import frame

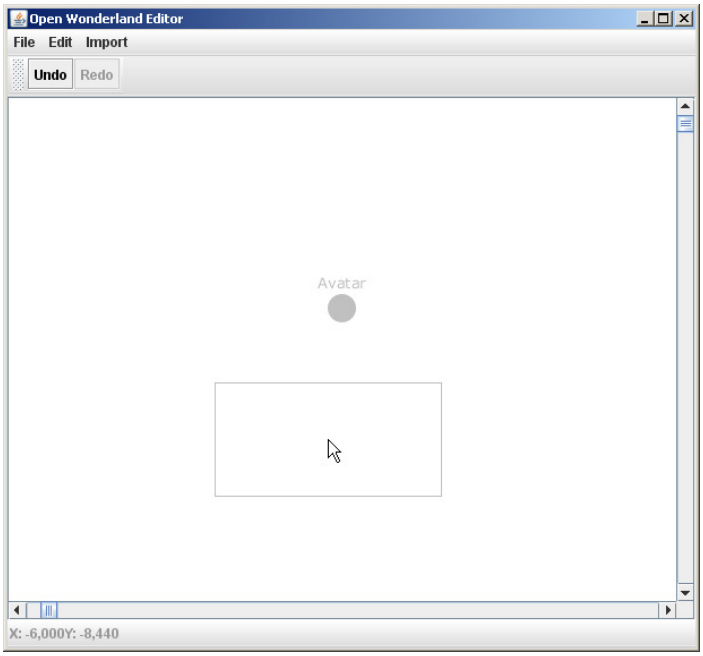


Figure 6.4: After pressing the *Choose Location*-button, a position for the new object can be chosen directly in the work area.

Selecting and Moving Objects

In this regard the prototype is controlled like every other editing program. Left click selects items whereas a right click will prompt a popup menu. Other staples are also included, like a selection rectangle. Selected items have a green border and green text instead of the normal black ones. Moving objects is realized by holding down the left mouse button on a selected item and dragging the mouse over the screen. All selected objects will then be moved. Moving objects is only prohibited, when at least one of the objects overlaps with another object. If that is the case, the shapes of the dragged objects turn red, signaling it is not

possible to move it to the new location. The left part of Figure 6.5 shows the house blocking the new position of the car. If an objects needs overlap with another one, it is possible to hide the blocking object. This can be done by the *Hide* entry in the *Edit* menu, or the drop-down menu. When an object is hidden, it can no longer block other objects, but it can not be edited either until it is shown again. This is done by the *Show* entry. Figure 6.5 on the right side depicts an object that is currently hidden. The car can now be placed on top of it, without any problem.

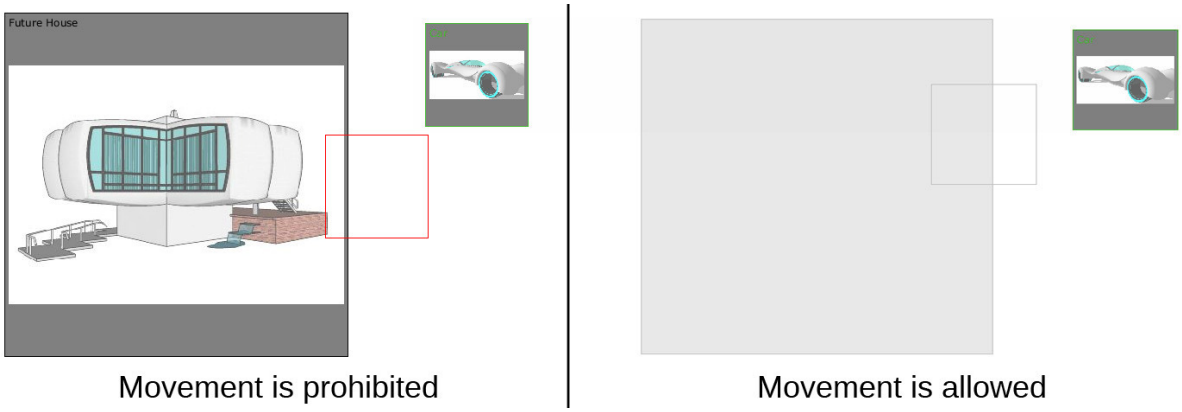


Figure 6.5: Difference between normal objects (left) and hidden objects (right)

Rotation and Scaling

Rotating and scaling is implemented almost in the same way as moving objects. The only difference is the border surrounding the selected objects. By left clicking on the four edges of this border, the objects can be manipulated. Unlike scaling, rotating allows to change the rotation center. At the center of the border there is a rectangle, equal to the border's edges. This is the rotation center, which can be moved. The rotation will be done around this point. Figure 6.6 shows an example rotation where the rotation center has been moved out of its position, to rotate two items to a specific spot. A new toolbar will appear on the top toolbar, which only contains *OK* and *Cancel*. *OK* will finish the operation and *Cancel* will abort it. It is also possible to finish the operation with a right mouse click, or by hitting the return key. To cancel the operation, the escape key can be used.

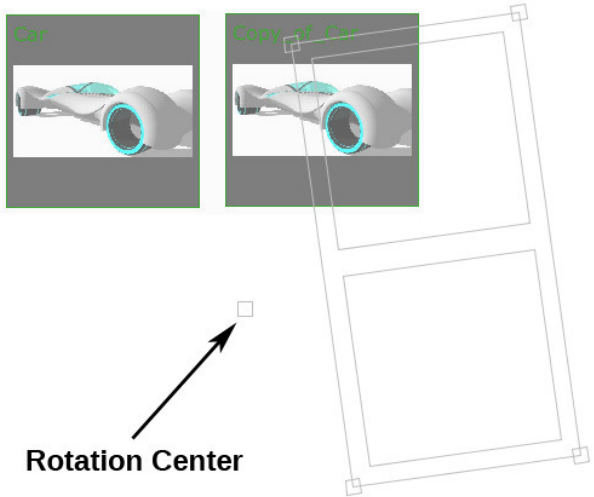


Figure 6.6: Rotating items around the rotation center

Copy and Paste

Copying is realized by selecting the target objects and then using *Copy* either in the *Edit* menu, or in the drop down menu. It should be noted that no new object will be created by using *Copy*. This operation only stores the current selection in the background. When using *Paste*, the users have to first position the copies. Contrary to moving objects, inserting them does not require the left mouse button to be dragged, because the paste operation finishes when the left mouse button is hit. Figure 6.7 shows the copy and paste operations in action. On the left four selected objects are copied. On the right, after hitting Paste, four silhouettes appear. With their help, users can determine the position of the four copies. Because *Copy* did store the objects, it is possible to create multiple copies of the same selection only by using *Paste*. *Copy* should therefore be used to make a new selection for the paste operation. There also exists a *Cut* entry in both the Edit menu and the drop down menu. This operation functions similar to *Copy*, but will delete the selected objects.

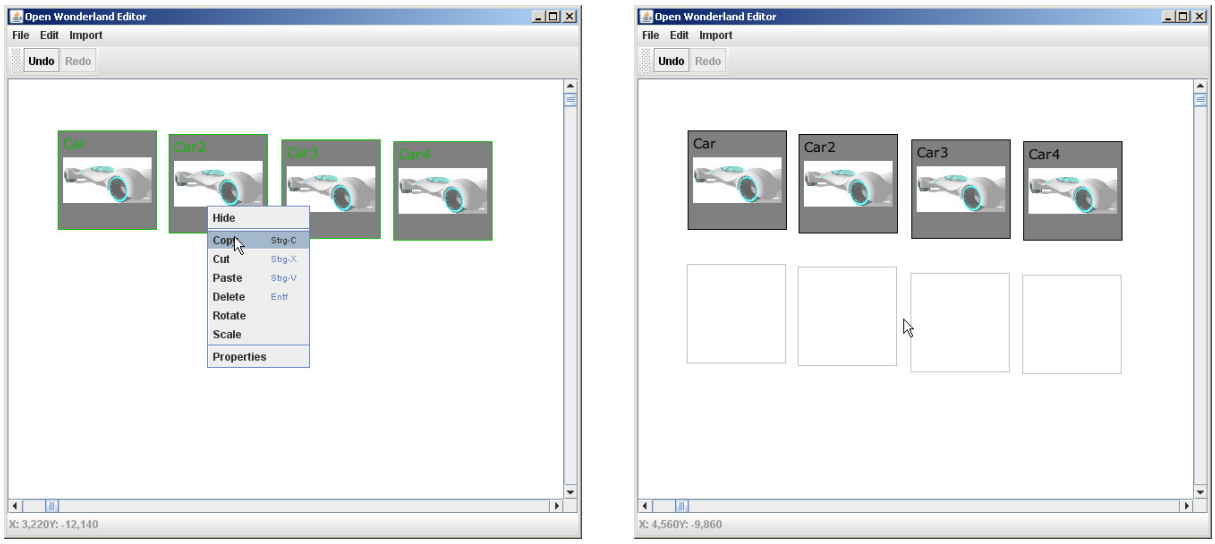


Figure 6.7: First *Copy* has to be used to store the objects (left), then after using *Paste*, users are able to select the position for the new objects (right).

The Properties Frame and the Image Selection Frame

The properties of an object can be separated into two categories: *General* and *Rights*. *General* contains name, position, transformation as well as representational image (Figure 6.8). The coordinates and the rotation axis are color coded, which is similar to OWL when doing a rotation, or translation in the world with the arrows (as shown in Figure 6.18 in chapter 6.2.3). If multiple objects are selected, it is possible to change only certain variables. Name and position will be deactivated. If a value, which can be modified, is different in-between the items, the editor shows this by displaying "different". Changing it will set all selected items to this value. A representational image can be selected with the help of the button right to the name. It usually shows the current image used, but if no image is selected it will state that there is no image selected. After pressing the button, a new frame will open, the *Image Selection* frame, as shown in Figure 6.9. A simple click on the desired picture and then pressing the *OK* button will change the representational image of the object. If no image is wanted for the item, a click on the selected image has to be done to deselect it. To upload new images the corresponding button can be used, or dragging the images into the frame.

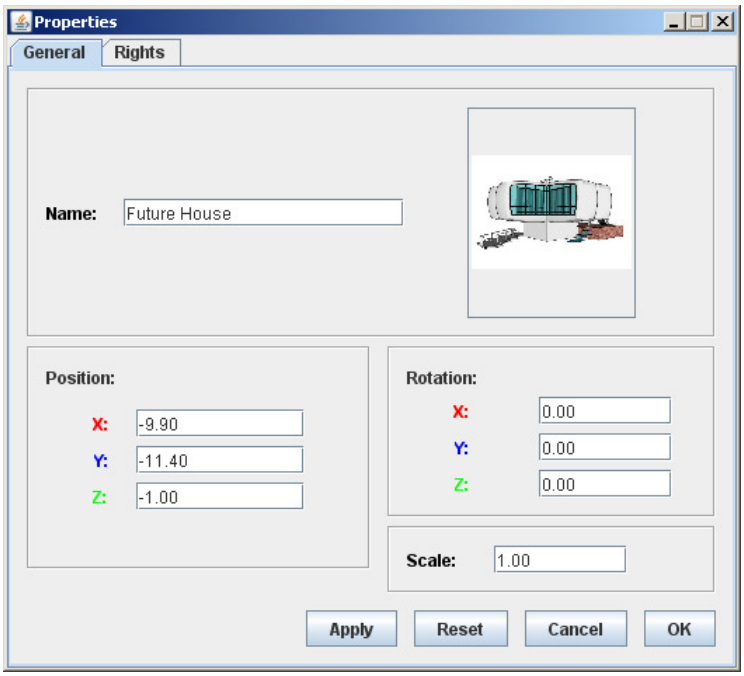


Figure 6.8: The *General* section of the *Properties* Frame

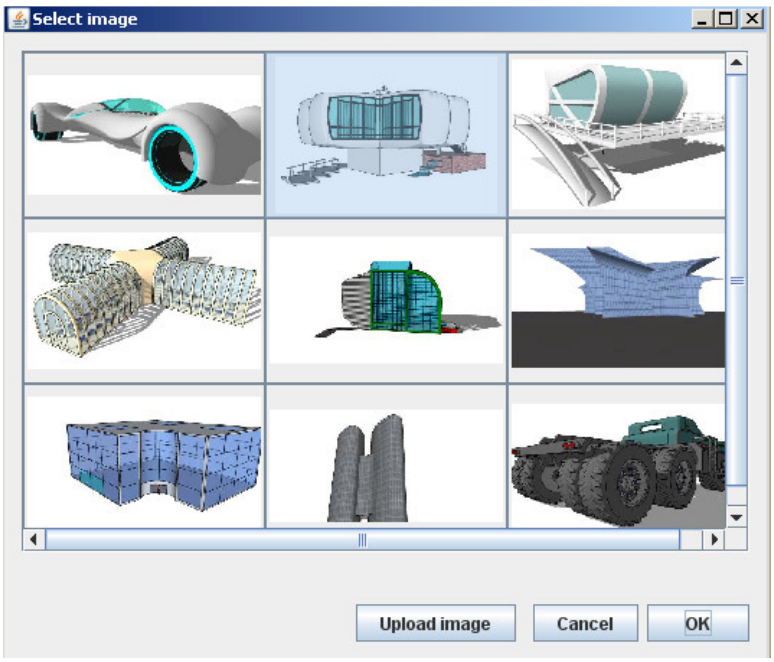


Figure 6.9: The *Image Selection* frame

The second category of properties is the *Rights* section (Figure 6.10), which can be used to manage the rights of objects. If an object does not have the *Rights Component*, the component can be created by clicking on *Add Component*. If an object actually has the component, the *Add Component* will be disabled and the rights can be set. A rights entry consists of several parts. First of all, the three user types have to be distinguished. User and usergroup are self explanatory. Everybody means every other user except the users and groups entered in the field. The second column contains the name of the user or group. The name of the everybody entry can not be changed. Then there are several columns with checkboxes, which represent the rights they have. The first is the owner, which has rights to everything, therefore other columns can not be changed, when owner is selected. The other rights will not be explained, because they are self explanatory. Clicking *Add* will create a new row. To remove an entry, the entry has to be selected and *Remove* has to be pressed.

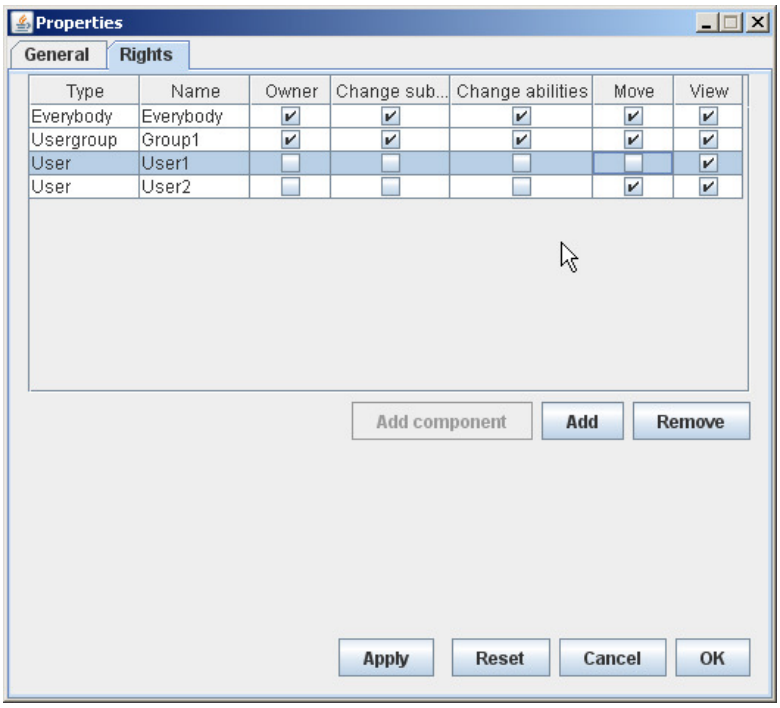


Figure 6.10: The *Rights* section of the *Properties* Frame

6.2 User Study

This section contains the user study, which was conducted after finishing the prototype. First of all, the research questions are listed. After this the setup of the study will be explained in detail, including a general overview of the test subjects. The section concludes with the findings and future work.

6.2.1 Research Questions

Because creating a virtual world should be easy for professional and novice users alike, the main goal of the study was to learn, if the editor was simple enough for both user groups. Figure 6.1 shows a quick overview of all questions used in the study. The first research question was created to learn, if it the editor is easy to pick up for inexperienced users. This is because SFP might not be done by experienced virtual world users. Neglecting inexperienced users would lead to beginners needing help from technical personal in order to create an SFP. But excluding experienced users would also be not an appropriate approach. A program may be usable for beginners, but would miss out on functionality for experienced users. This is why the second question exists. The third was the main focus of the study. It revolves around comparing the editor with already existing tools in OWL. This should show, which one is easier to use for both types of users and how much the editor can improve the process of building and maintaining a virtual world. The fourth questions deals with world creation. It asks how much the editor can help with the world creation process and its maintenance. The last question investigates the possibility, whether the editor can fully substitute OWL during world creation or not.

Nr	Question
1	Is the editor prototype easy to use for people without much computer knowledge?
2	Is the editor prototype easy to use for people with high knowledge revolving around computers?
3	**Which is easier to use: The functions OWL offers, or the prototype?**
4	Can the editor aid with the creation and maintenance of a virtual world?
5	Can the editor substitute OWL completely during world creation?

Table 6.1: The research questions asked during the study, where the third question was the main focus.

6.2.2 Setup

The study was done in a thinking aloud setting, where test users will get a set of predefined tasks to execute in OWL and the editor. During the test, the users were encouraged to speak out every thought they have. The users were filmed during the test sessions and the screen was captured. The users had to complete four different tasks in a certain amount of time as show in Table 6.2. Task 3 is listed twice, because in order to see which tool is easier to use, the subjects had to use both, the editor and OWL.

Figure 6.11: The test environment for the first three tasks.

The first test group was only allowed to use OWL for task 2 and 3 and then they switched to the editor. The second test group used the editor first and then switched to OWL. The first task, as shown in Table 6.2, was created to make the subjects familiar to the controls and workings of both OWL and the editor. The point of the second task was to examine the subjects durring moving and rotating objects in the world. The third task was to make them learn to copy objects and to set rights. The last task should teach them how to import objects, load and save a world and incorporate the actions they learned in previous tasks. The test environment for task 1-3 was a school room, which contained only one a small amount of objects (Figure 6.11). For the last task an empty room was provided for the subjects to import objects. The objects they had to import were hand picked before the study started, so that every user had the same objects to work with. Figure 6.12 shows the whole test environment in the editor. Because the building is also an object in the world, it was also displayed as such an entity in the editor. And because the editor has sorted objects in ascending z-axis before

the study, the building blocked the view to every other object. This can be seen in Figure 6.12on the left side. In order to see the rest of the objects, the building had to be hidden. This decision was made deliberately in order to see, if the test users would grasp the concept of hiding objects.

Nr	Task	Time
1	Look around Open Wonderland and the editor for a few minutes.	8 min
2	The world is filled with a couple of class room objects. There is a desk, some chairs and other objects. Please clean up the class room.	4 min
3a	Now the room is clean, but a classroom needs more than one desk and 2 chairs. Please make more of them. Also make it so that only you are allowed to move the desks.	5 min
TOOL CHANGE		
3b	Now the room is clean, but a classroom needs more than one desk and 2 chairs. Please make more of them. Also make it so that only you are allowed to move the desks.	5 min
4	Build your own future living room with objects provided to you, then save and load it.	10 min

Table 6.2: The tasks for the test users.

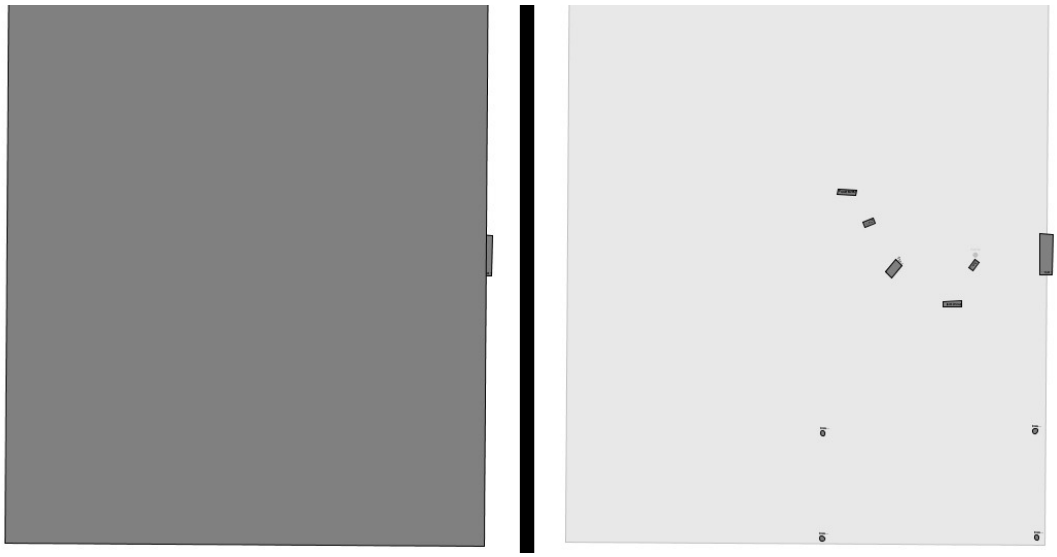


Figure 6.12: The test environment in the editor at the start (left) and after hiding the building (right).

The study consisted of 10 test users, which were recruited through asking through friends and acquaintances from different areas of life, like fellow students. Before and after the tests the users had to answer a questionnaire, which can be found in Appendix A. As shown in Table 6.3, the age of the testers ranged from 21 to 30, with an average of 25,5. The study consisted of 70% males and 30% females. 30% of the subjects were employed, the remaining 70% had studied at universities. To keep the ratio between experienced and inexperienced testers balanced, only 50% of the test users came from the computer science field. The other 50% came from different fields of study, or employment. A detailed depiction of all fields is shown in Figure 6.13. The main objective for the inexperienced group was to gather as much different fields of expertise, without much in-depth computer knowledge. Many of these fields include computer usage, but the understanding of it does not reach deep enough to call them experts on this sector.

Average age	Lowest Age	Highest Age	Male/Female %-ratio	Employee/Student %-ratio
25,5	21	30	70/30	30/70

Table 6.3: General test user information

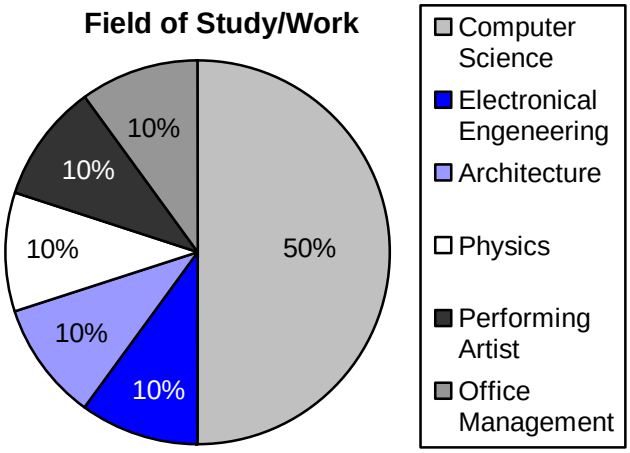


Figure 6.13: Field of study, or working field of the test users.

As shown in Figure 6.14, the average computer usage of the subjects per week was between 10 to 20 hours, with an estimated internet usage of 6 to 10 hours. Most of this time was spent for emails and instant messaging. Only a few of them played video games more than 1 to 5 hours a week. Even less were playing MMOs. Only one of the users was spending time with virtual worlds outside work. Most of the subjects had a spare awareness of virtual worlds. They did not know what to make of them and they especially did not know what to use them for. A great number of them did not see many opportunities outside of gaming. Before showing the subjects the editor, they were asked, what they would expect from such a tool. Most of the subjects stated, they would like to have a clear and easy to use interface and it should offer a good overview of the virtual world.

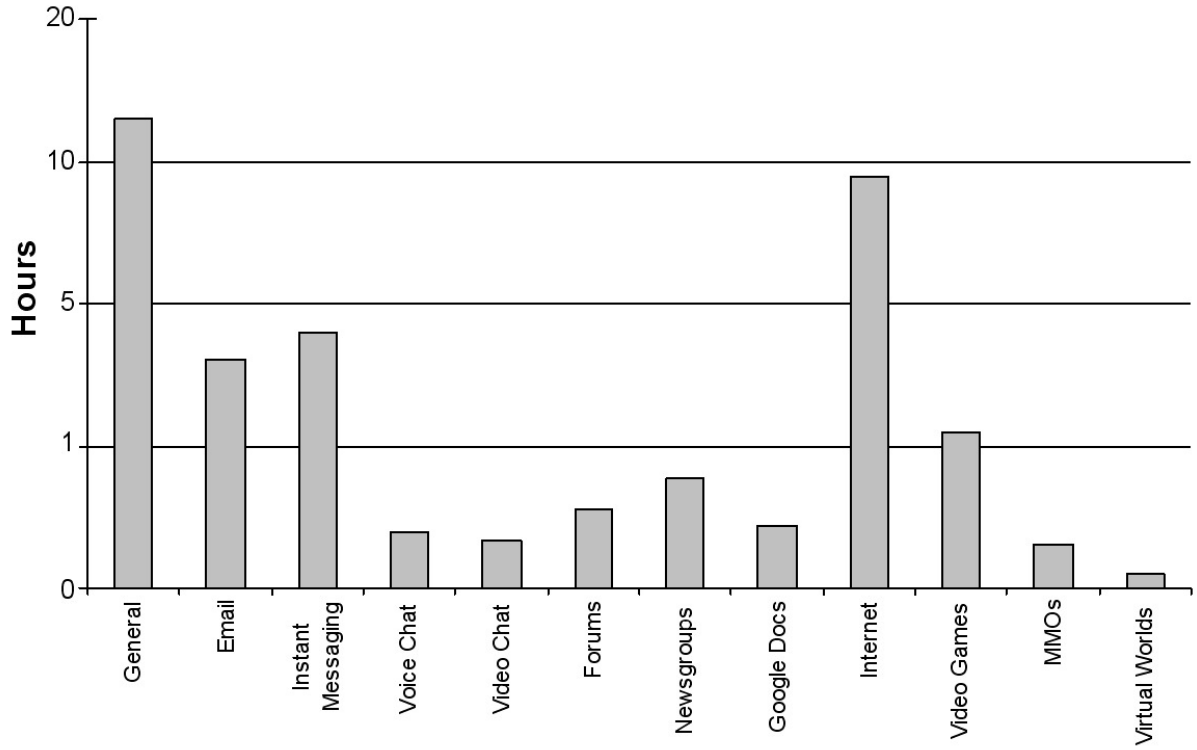


Figure 6.14: Average computer usage of the test subjects per week

6.2.3 Findings

A majority of subjects thought the editor was easier to use, as shown in Figure 6.15. Also many of them would prefer the editor to build a virtual world instead of using OWL. Some of them even stated they would use both. One test user stated that she would likely use both, because both had their advantages over the other. Figure 6.16 shows a more detailed feedback. Many users strongly agreed that the editor was much more user-friendly than OWL. The results also show that the editor was perceived as not very time consuming by them. The features of the editor were also more self-explanatory and more logical than OWL. The users felt the editor was more familiar in its functions. The differences in the category "Usage without any knowledge" are not as far apart as in the other categories. This means both the editor and OWL would need someone to teach the users all their functionalities.

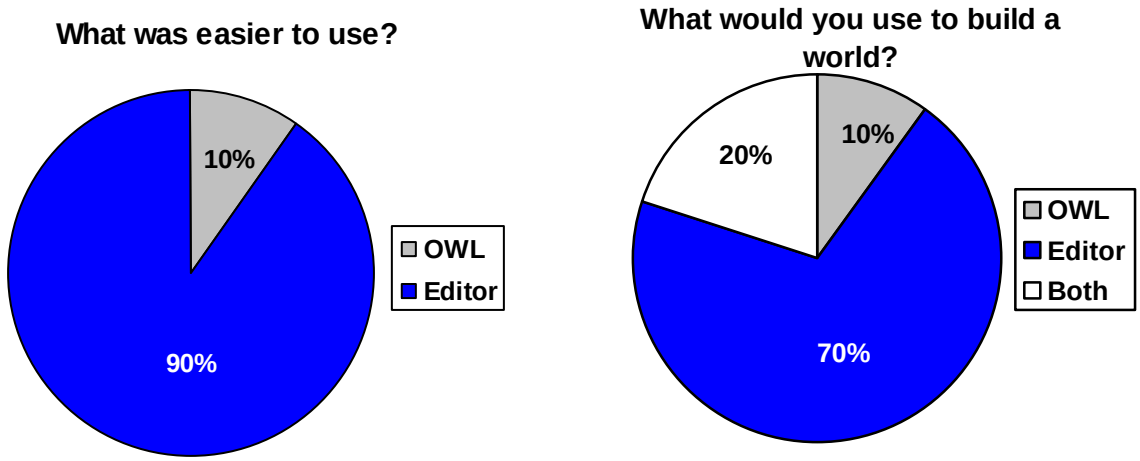


Figure 6.15: Feedback for OWL and the Editor

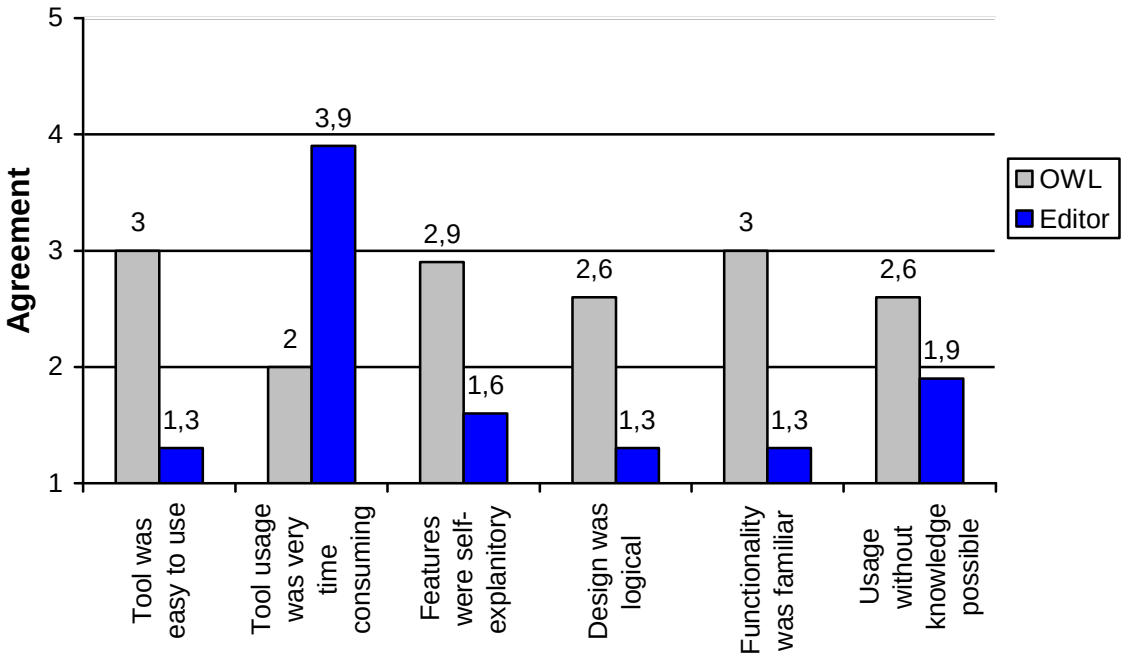


Figure 6.16: Detailed feedback for OWL and the Editor (1 is strongly agree, 5 is strongly disagree)

Figure 6.17 shows a more detailed view on the difficulty the users had with certain operations. OWL has an average difficulty for moving, copying and importing objects.

Setting the rights, saving and loading was very difficult according to the test users. All these operations were described as very easy in editor prototype by contrast. It is also worth mentioning that the group that worked with the editor first did not like it as much as the group that came from OWL to the editor. Another interesting observation is that users with computer science background had a lot more trouble working with both the editor and OWL than other subjects.

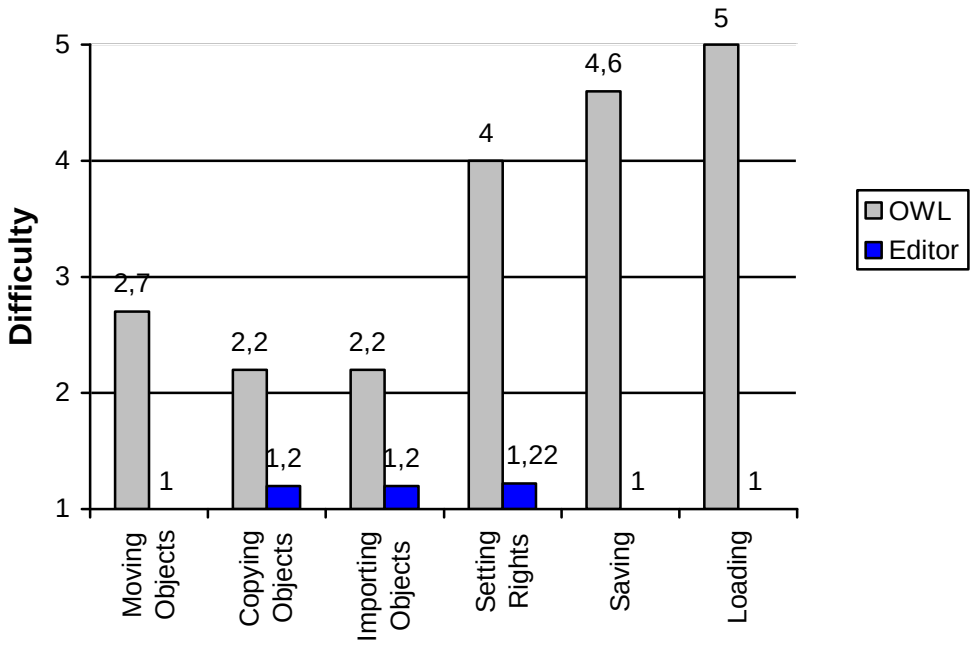


Figure 6.17: Difficulty the test subjects experienced for certain operations (1 is easy, 5 is hard)

There are a number of observations that came up during the tasks. Usually the duration for the simpler tasks was much longer in OWL, especially when looking at the best times, which are shown in Table 6.4. For task 4 the times were pretty much all near the 10 minute limit. The subjects needed a lot more assistance in OWL for the more complex operations, like setting the rights, or exporting the world. Without any tips the subjects would have only stumbled upon the solution with sheer luck, which did also happen. Table 6.5 shows the best and the worst times the users needed for completing the tasks. The reason why both the editor and OWL have the same worst time is that at least one test per task had to be aborted, due to reaching the time limit. But comparing the best times makes it clear that the first two tasks can be solved in a much faster way with the help of the editor. Only task 4 had similar times. The success rate, which is the amount of tests that was not aborted due to the time limit, was much higher in the editor.

Task	Max Time	Average Time OWL	Standard Deviation OWL	Average Time Editor	Standard Deviation Editor
2	4:00	3:54	0:12	3:04	1:06
3	5:00	4:59	0:01	3:08	1:29
4	10:00	9:53	0:14	9:36	0:25

Table 6.4: Averaged completion times for tasks 2 to 4

Task	Best Time OWL	Worst Time OWL	Best Time Editor	Worst Time Editor	Success Rate Owl	Success Rate Editor
2	3:32	4:00 (aborted)	1:01	4:00 (aborted)	20%	60%
3	4:58	5:00 (aborted)	0:59	5:00 (aborted)	11,11 %	80%
4	9:25	10:00 (aborted)	9:03	10:00 (aborted)	20%	50 %

Table 6.5: Best and worst completion times for tasks 2 to 4

The biggest problem for the subjects in OWL was the perspective. Moving one object is simple in OWL, but if the avatar stands in the direction of one of the axis, the object movement will slow down significantly. Moving the chair in Figure 6.18 on the red axis would result in a very slow position change. Therefore the subjects had to move the avatar constantly around. Due to the perspective, they also had to move, because they often could not see whether the object was placed in the right position. So they had to move closer or shift the perspective to inspect their work. While moving they suffered from another disadvantage which is the camera. Sometimes the camera went through walls, so subjects were not able to see their avatar. Other times it would zoom too close to the avatar, so they could not select anything, because of the avatar's hitbox or a wall blocking their actions. Another problem, a couple of subjects had, was to get rid of the transformation axis that would show up during editing an object (the arrows in Figure 6.18). And sometimes the boundaries of these axes would be huge in their proportions. In those cases, the subjects had to leave the house to see the rotation axis. The menu bar for selecting the edit operation (move, rotate and scale) was also not clearly visible for a couple of subjects. Some of them also tried to select multiple objects. Such an operation is not possible in the world itself, which they really did not like.

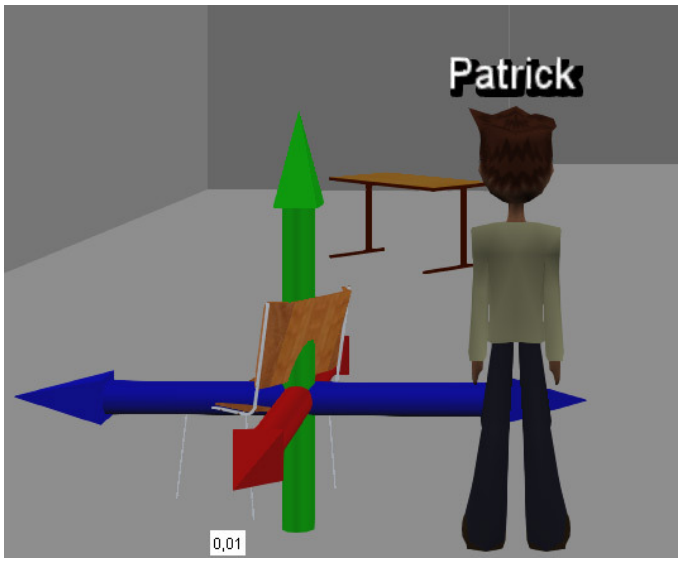


Figure 6.18: Moving the object in the direction of the red arrow is slow, when using this perspective.

Minor confusion arose with the duplicate function. This function creates a duplicate in front of the original object. Many subjects stated that this is annoying because they had to move the object to its location after duplicating it. Another problem was making more than one duplicate from the same original. In that case all duplicates share the same position. The subjects did not figure out that there was more than one duplicate. They only found out when they tried to move one duplicated object and saw the other ones. It was also frustrating for some users to only be able to duplicate one object at a time. The grouping mechanic in the object editor was another feature the subjects did not know of until explained. Some of them stumbled upon it and did not know what to make of it. General speaking, many users stated that OWL was not intuitive. Few also stated that it was cumbersome to use and that many operations were difficult to find. One subject stated that the menu structure of OWL is too unclear and has too much overlap. Therefore it needs an overhaul.

But the editor also has some room for improvement. The first task the subjects had to do within the editor usually tended to take much more time, because the subjects were confused by the lack of objects, because the building was blocking the view to the other objects. As stated in chapter 6.2.2, this was intended to see if people would understand the hide mechanic in the editor. Because they only saw the house, most of the subjects thought, they had to import the objects somehow. The editor was improved to show objects overlapped by other

objects. Another problem was that the house was only one block, because objects are not displayed as renders of the original object. It was hard for subjects to imagine the room they were working in, because of the missing walls. Using the editor for every operation was also hard. The barebones graphics made it hard for subjects to see the alignment of objects, resulting in rotating them in wrong directions. Another huge problem was the z-axis, which was difficult for the users to work with, when only using the editor. This was obvious during Task 4, where the subjects seemed to need more time to build a room. Furnishing it was looking easy in the editor, but when the subjects had to go back to OWL to see their creations many objects were floating in the air or were aligned in the wrong direction. So they had to switch constantly between editor and OWL to see their changes in the z-axis. Another problem occurred during importing. Many subjects did overlook the *Choose Location* button, where they would have been able to pick the position of the imported object (see Figure 6.19). Another instance where subjects would overlook a button was during the rotation of an object. Many subjects had troubles in finding the confirmation bar at the top tool bar, or the message in the bottom tool bar. Some of them did try out the Return-key instinctively, which was already implemented as a way to confirm the operation. Others tried to click with the mouse. This was also implemented after the evaluation, where a right mouse click applies the rotation. The zooming implementation was also critiqued. At the time of the evaluation the mechanism only zoomed to the center of the frame, not to the mouse position. Many subjects were used to zoom to the mouse, so the implementation was changed afterwards.

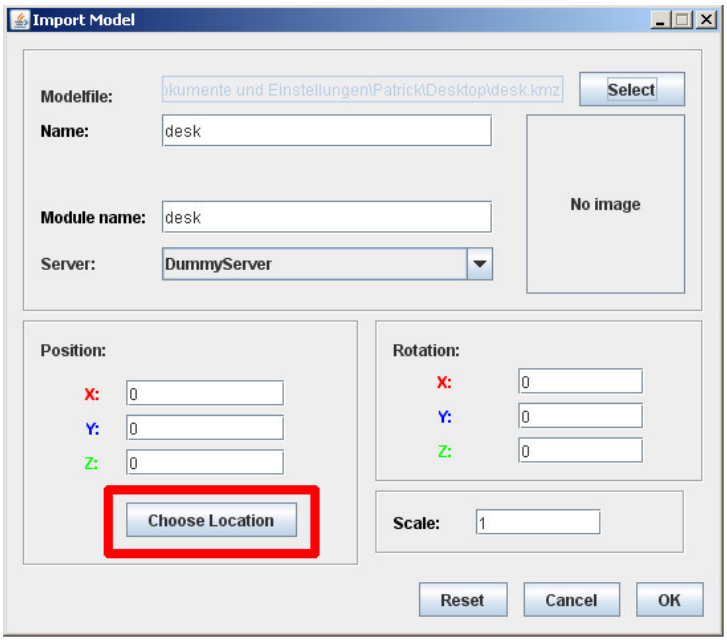


Figure 6.19: *Choose Location* was missed by nearly everyone when importing an object for the first time.

The evaluation also included the *Standard Usability Scale* (SUS), which the subjects had to fill out twice, once for OWL and once for the editor. The results show that the editor has a much higher usability score (see Table 6.6), which was also expressed by many users. Many stated that the editor was easier and faster to use. Some features were hard to find in OWL. But they also liked the fact that changes can be seen directly in OWL during object manipulations. The subjects were able to see the room they were building, they did not have to imagine it, contrary to the editor. But as stated before, both systems have their limits. Even if the editor is easier to pick up and to use for inexperience users, it is no substitute for the tools OWL offers. Creating a world in the editor alone may be difficult and cumbersome, because of the missing z-axis in the two dimensional view. But one subject stated that it is also hard in OWL, because he did not know when the object was standing on the ground. Sometimes it was floating just above it and other times it went too far into it. All in all, many

subjects suggested a combination of both: Using the editor for constructing a rough outline of the world and OWL for fine tuning. The editor alone would not suffice to create a virtual world. It should be noted, that the subject of the pilot test used a combination of both, the editor and OWL, for task 4 and achieved a time of 2:58 minutes, which trumps every time accomplished by other subjects. The high times in the editor are due to the z-axis problem. Subjects working with OWL did build their room faster, but had troubles in finding the mechanism to export and re-import their creation.

Question	Average OWL	Average Editor
I think that I would like to use this system frequently	2,7	3,4
I found the system unnecessarily complex	3,4	1,3
I thought the system was easy to use	2,7	4,5
I think that I would need the support of a technical person to be able to use this system	2,2	1,4
I found the various functions in this system were well integrated	2,8	4,4
I thought there was too much inconsistency in this system	2,4	1,2
I would imagine that most people would learn to use this system very quickly	3,4	4,6
I found the system very cumbersome to use	4	2,1
I felt very confident using the system	3,2	4,1
I needed to learn a lot of things before I could get going with this system	2,1	1,4
SUS Score	**51,75**	**84**

Table 6.6: SUS Results (1 is strongly disagree, 5 is strongly agree)

As for the five research questions, the first two can be answered with a yes. Both groups, inexperienced as well as experienced users were mostly able to complete the tasks, or came close. The third question, if the editor is easier to use than OWL, can also be answered with a clear yes. Both the users themselves and the SUS show that they prefer the editor when it comes to usability. The editor can also help with the creation and maintenance of a virtual world, because many of its functions are a lot easier than their OWL counterparts. Only the last question, whether the editor can fully substitute OWL must be answered with no. Because both offer different perspectives on the world, both have their advantages and their disadvantages. In order to fully substitute OWL, the editor needs further work.

6.2.4 Future Work

One drawback of the editor is its current representation. Take the example of a house, which will always be a grey block, because it is a whole object in OWL. Even if the house has different rooms, they will not be shown in the editor. Also object alignments are hard to tell. So, instead of using grey shapes, a two dimensional rendering of the objects would be much easier to work with. This was also expressed by many subjects. The second troublesome point was the z-axis. Some subjects stated that a mechanic that would align objects directly to objects underneath would be desirable. But the problem with such a mechanic lies in the difficulties to know whether the big object is something like a room, or a solid matter like a brick. In a room or a house it is possible to place objects directly in it. Placing an object in solid matter, like a brick, should not be possible. So therefore a distinction has to be made, whether objects can be placed in, or only on top of items. This may be realized within the editor, where users can set a variable which switches between hollow and solid. This variable

should also be created as a cell capability in OWL. One subject stated that multiple views would also provide a better overview, especially when it comes to the z-axis. Some subjects also wanted to be able to create a layer structure. This could be very useful. For example, one layer could be used for buildings, a second layer for large objects in them and a third layer for tiny objects standing on top of the large ones. It should be noted that using a layer structure would also need them to be linkable. For instance, if the objects layer is linked with the house and the house is moved, the objects should also be moved. This can be connected with the grouping mechanic in the OWL Object Editor.

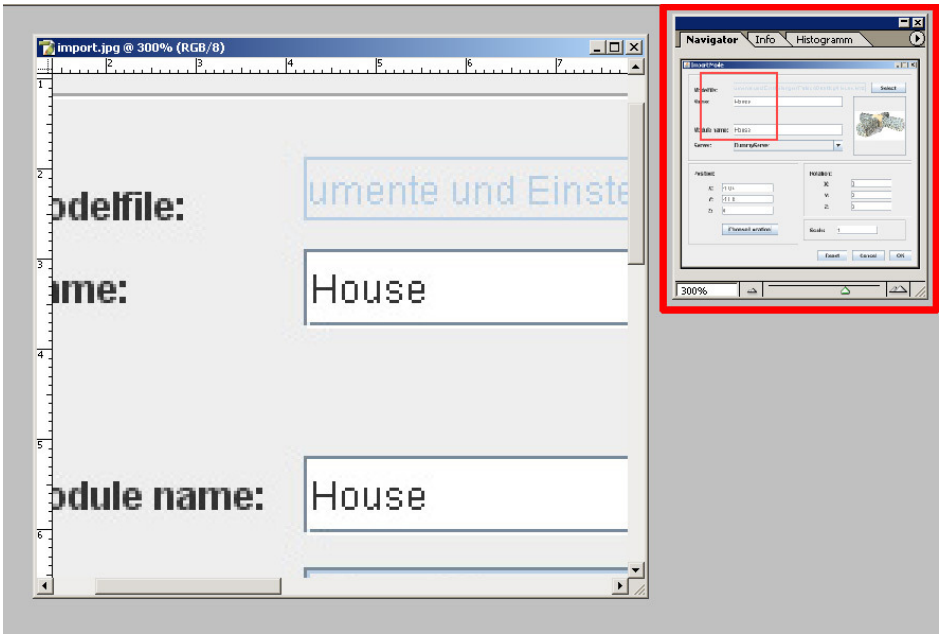


Figure 6.20: A navigator like the one in Photoshop (top, right) could improve the world's overview.

One user stated that the current editor would be good for building something small, like a room or a house, but for a whole world, he misses some kind of overview. For large worlds, he would not be able to see quickly where he currently makes his changes. Some kind of navigator would be thinkable, like in Photoshop (see Figure 6.20 for an example). This navigator would show the whole world and the current frame position users have in the word. Also the "Choose Location" button in the import frame was overlooked by many subjects. There needs to be work done in that direction too, removing the whole position part and just let users pick the position after they clicked the *OK* button. Another user stated it would be good if there was a button in both the editor and OWL to switch between them. This idea could be expanded to an option, where the user clicks on a space in the editor and the avatar will be teleported right there. A raster or an option to snap objects to other objects was also desired from a couple of subjects.

6.3 Discussion

In order to make the creation of a virtual world easy, the editor was kept as simple as possible. There are only a couple of functions the editor supports. These are importing, moving, transforming, copying and deleting objects. It is also possible to set rights of objects and load and save the current world in a file. A usability study was conducted in order to find out if the editor was easier to use than the tools integrated in OWL. The study showed promising results. Many subjects felt the editor was much easier to understand and to use. But there was also room for improvement. Some minor issues were found and corrected after the evaluation.

As it stands now, both OWL and the editor have their strengths and their weaknesses. OWL is less intuitive and it takes time to do most of the simple operations, but in contrast it is easier for users to see their changes and imagine their world they are building. The editor is much easier to use, but because of the two dimensional perspective it misses out on the z-axis, which is impossible to mimic in this kind of view. So objects placed in the editor may fly in the air and users are only aware of this, after they inspect the object directly in the virtual world. This problem could only be avoided by implementing another perspective in the editor. But that should not be necessary, because it would be easier to just go into OWL and edit the object there.

At this stage, the editor is not a substitute for the OWL tools, but it can accelerate certain tasks, especially copying objects and setting their rights. A combination of both can lead to the best results. The editor should be used for big changes, whereas OWL could be used for quick and basic changes. The next big step in the development of the editor would be to replace the current object representation with a 2D render of the object. This would alleviate the problem of users not knowing which orientation the object currently has. It would also be easier for users to imagine the world they are currently building.

7 Lessons Learned

This chapter provides the knowledge that has been learned during research, implementation and testing. The theory part will focus on the possibility of SFP in virtual worlds, while the implementation part will discuss some of the difficulties of the prototype development. The evaluation part will close with a short abbreviation of the findings learned in the user study.

Theory

Creativity is one of the most important driving factors of human kind. How powerful creativity can be, is seen in the influence it has on development of new products and ideas. Many of these ideas stem from Sci-Fi works. This is evident in the new approach of SFP, which is, in short, a Sci-Fi work revolving around a single science fact. Currently SFP is mostly used within non-interactive media, like stories or films. Virtual worlds could provide another possibility to house a SFP. There are plenty of possibilities for creativity and research in virtual worlds. Combining both research and creativity possibilities could be the basis of interesting SFPs. Behavior of people could be observed rather than imagined, which can be a great advantage when developing technology for the near future. But there is a reason why virtual worlds are not commonly used. Virtual worlds are hard to get into. There is a lot to learn and the tools, virtual worlds offer, are not user-friendly for the most part. Building a world can become quite cumbersome without the help of additional tools. As shown, there are procedural methods which can help with the creation of a world. The problem with these tools is that they are automated. If users want to make changes by hand, they again have to resort to using the tools the virtual world platform offers. To help alleviate this task, tools are also needed for helping users making their changes directly. An editor like program which shows users the world from a birds-eye view would be de most desirable tool, because it would be familiar enough to be easy to use. Therefore an editor prototype for OWL was developed.

Implementation

The most important part of developing a prototype is the proper selection of the structure. The three separate entities, adapter, data and graphics, made it easier to react to changes during development. Decoupling the editor part from the adapter was a great idea. It had the advantage of creating a dummy adapter, which simulated a server with a couple of predefined objects. This dummy adapter was a huge time saver because changes to the editor's GUI could be made quickly and without starting OWL. The time to start OWL, upload a new version of the prototype and then start the client may not be that long, but when you have to do this constantly it will add up. Another advantage of the dummy adapter was debugging. Because OWL does not feature a sys-out, only an error debug output, it was not possible in OWL to make fast tests of new features. So without OWL, working on the editor was a lot faster.

The adapter part for OWL is relatively easy in structure, but has probably the most complex methods. While most of the easier functions could be reused from OWL's Object Editor, with slight alterations, there were a couple of operations that could not be implemented that easily. Like the representational image for example. The first idea of storing the image of a cell in a capability was not possible due to the limits OWL set on capabilities. So the next idea was to store the image on the server and store its ID in a new capability. Unfortunately OWL did not

allow for a shared folder using its default storage mechanism. There were two options. The first was to implement a much more complex save and load function. The second was to store the images in the user directory of the current user, which is open to everyone. The second option was chosen to keep the program as simple as it needs to be. One of the main difficulties within the adapter part was to transform all the data from OWL in such a way the editor needs them. The biggest obstacle in this situation was the different coordinate systems. A coordinate translator had to be created to easily transform coordinates into editor coordinates and back.

The design of the data component was really straight forward. It only stores all the necessary information for the editor, so the server has not to be called every time something is needed. In fact, the data component is mainly used for properties and images, which in retrospect could have been implemented in the shapes themselves. But this would have blown up the proportion of the shape classes even further.

While the adapter and the data component are straight forward structure-wise, the structure of the GUI did change several times. Packages grew too much in size, so they had to be split in-two. The resulting structure, while not perfect, does allow for quick fixes and changes. The problem that arose was the complexity. The structure is not easy to understand, but affords were made to name the packages in a way everyone would know what they would find in each of them. The biggest problem of the presentation was the missing Java libraries for zooming in a `JPanel`. Therefore this function had to be implemented. The viewport changes during zooming can become confusing, when implementing the scale. The zooming is probably one of the more complex operations implemented in the GUI part, which is due to the consideration of many variables and two different scales (the current scale and the scale after the zooming). Implementing the 2D objects was also a challenge, especially because the objects had a scale on their own. So not only the global scale, but also the object scale had to be considered when drawing objects. Fortunately Java does offer simple methods to transform a shape. Unfortunately those methods did not always work as needed. The scaling of a shape did always change the coordinates, which was not desirable. Therefore some methods had to be implemented, which did offer the wanted behavior.

Using the OWL server for synchronization was a good idea, because the editor will always be in sync with the virtual world. The implementation of the dragging shapes was a lucky find. With their help, the editor could stay in sync, while users can make their changes in the editor. So the synchronization with OWL was never a problem to begin with. With this structure, multiple users can build a world simultaneously.

Evaluation

The evaluation was made with the focus on usability. This was done to understand the difficulties people would have when being confronted with the task of building a virtual world. It was necessary to find out, how easy it would be for inexperienced users to create a SFP in a virtual world. The evaluation featured a SUS for both, OWL and the editor. While the editor was significantly better in the SUS, there were also some things it was not able to substitute. The usage of the z-axis in the editor was cumbersome for the subjects, because they had to switch between editor and OWL constantly to see their z-axis changes. This behavior is not desirable. Adding a different view in the editor could solve this problem, but as many subjects stated, it would be more comfortable to just use OWL for z-axis operations. Another problem the subjects pointed out was that OWL allowed them to better understand how the room looks. They did not have to imagine it. The editor may never have this advantage.

Another disadvantage is the current visualization. The usage of blocks to represent items in the world may be easier to implement, but users are not able to see the orientation of objects. This definitely requires further work. To make it adequate, objects should be represented by a 2D rendering of them. Many subjects voiced their opinion in favor of this suggestion. OWL did receive even more negative feedback. Creating a world, while running around with the avatar, is hard because of the perspective. The subjects had to change their position constantly to make changes. Many of the options provided by OWL were not user-friendly and even hard to find. A couple of subjects stated their dislikes for many operations in OWL. The missing multi-selection was one of the features nearly every subject wanted to have. But they also stated they would create their own world, if they had the time. Many of them would like to use a virtual world for their own enjoyment, but are discouraged by the complexity. They think they need a lot to learn before being able to create a world properly. The editor may help them, which was also expressed by many of them, but it would not be a suitable substitute for OWL. Of course this is only logical. A two dimensional representation of a three dimensional world will always miss out on one dimension. And it will never be the same as walking around in the world. Therefore a combination as both, the virtual world as well as the editor would be the best solution for now, for quick and efficient world building. This was also shown by the subject of the pilot test which had the freedom to use both systems at once for the last task and achieved the best result, which was approximately a quarter of the time the others needed.

8 Summary and Outlook

Creativity is an integral part of our society. It not only helps people in their daily life, but it can also create powerful visions of the future. SFP can then be used to harness the ideas created in these visions. Building a SFP should be an easy task to achieve, given a high creative potential of all participants. Creating a story around a science fact and measuring its implications on society sounds easy, but it needs a good amount of imagination and creativity. Seeing how people react to something that is not possible yet can be a powerful help for development. Virtual worlds are also powerful tools, offering interactivity, collaboration and immersion. As the user study showed, few people have knowledge about virtual worlds and they do not know what to make of them. In the study, the subjects did not to see much use for virtual worlds outside of gaming. Many of them stated that they could not imagine using them for collaboration or communication. One reason for these uninformed claims may be the complexity of current virtual worlds. There is a lack of usability in building and changing a world. Several approaches already exist which automate world building. This can be good for creating a rough outline of a world, but changes by hand have to be done by tools the virtual world platform offers.

Making virtual worlds easier to jump in would help draw in new users, which are currently daunted by the idea of creating something in a virtual world. Many subjects of the study were in favor of creating their own world just for fun, if the process would not have been so complicated. The automated world creation solutions are a good start, but there should also be tools that simplify the manual tasks when doing quick changes in the world. Good usability is important for virtual worlds to be possible instruments for SFP. Because SFP is missing an interactive part, they would benefit from using virtual worlds. As for now, SFPs are only told in a linear fashion and behavior of people has to be estimated. Therefore SFP is very static. Putting a SFP into a virtual world would mean that people can interact with it and researches can observe their behavior. Therefore, behavior of people does not have to be approximated by the SFP's author anymore. This approach would be especially useful for SFPs revolving around society. Because virtual worlds are not bound by any laws of physics, ideas like flying cars can be realized. Researchers could also collaborate, in order to build a SFP. Therefore discussion about the SFP is promoted from the beginning. This is contrary to the normal approach, where discussion is only intended for the end of the process.

In order to create a more user-friendly environment for virtual worlds, an editor prototype was created for OWL. The goal was to create a tool which is easy to pick up and does not need much prior knowledge. Therefore its functions were also kept to a minimum. Generally the editor was received positively by most of the subjects. They commended the editor for its easy interface and fast functionality. Especially the option to select multiple objects was praised, a feature which is not available in OWL.

But there is also room for improvement. The graphical design needs to be changed. A 2D-render of the objects would help people immensely, because they would see the objects alignment as well as rooms in buildings. It would also be easier for users to imagine the world. The current representation of objects is too simple for this. Another important addition would be some kind of layering, where objects have an assigned layer and users can only work on one layer at a time. Linking layers would also be important. The third addition would be a navigator, which would help users navigate huge worlds. The editor can also not be used for changing behavior of objects, which needs to be done by creating capabilities. But

capabilities are created directly in programming language, which means inexperienced users may not be able to do this. Therefore another editor like prototype could be created which covers behaviors and lets users change them with simple commands.

9 Bibliography

Alatalo, T. (2011). An Entity-Component Model for Extensible Virtual Worlds. *IEEE Internet Computing, 15*(5), 30-37. http://dx.doi.org/10.1109/MIC.2011.82

Albion, P. R. (2009). Mere Mortals Creating Worlds: Low Threshold 3D Virtual Environments for Learning. *International Journal of Learning, 16*(6), 663-678.

Albion, P. R., & McKeown, L. (2010). *The seamless integration of Web3D technologies with university curricula to engage the changing student cohort* (CG7-488). Sydney, AU: Australian Learning and Teaching Council. Retrieved August 20, 2013 from http://eprints.usq.edu.au/7153/1/Albion_McKeown_PV.pdf

Apostolidis, H., Kyropoulou, K., & Chaldogerides, A. (2011). Exploiting XML-RPC: A framework for adaptive creation of Virtual Worlds in Second Life. In *Proceedings of the 14th International Conference Interactive Collaborative Learning* (pp. 202-208). New York, NY, USA: IEEE. http://dx.doi.org/10.1109/ICL.2011.6059576

Au, W. J. (2007, July 18). REMAKE THE STARS. *New World Notes*. Retrieved July 22, 2013 from http://nwn.blogs.com/nwn/2007/07/remake-the-star.html

Bainbridge, W.S. (2007). The Scientific Research Potential of Virtual Worlds. *Science, 317,* 472-476. New York, NY, USA: AAAS. http://dx.doi.org/10.1126/science.1146930

Bainbridge, W.S. (2010). Virtual Worlds as Cultural Models. *ACM Transactions on Intelligent Systems and Technology, 1*(1), Article No. 3. http://dx.doi.org/10.1145/1858948.1858951

Barnes, S. (2010). Virtual Worlds Come Of Age. *Journal of Virtual Worlds Research, 3*(3). Retrieved April 8, 2013 from http://journals.tdl.org/jvwr/index.php/jvwr/article/view/885

Beaudouin-Lafon, M., & Mackay, W. E. (2008). Prototyping Tools and Techniques. In J.A. Jacko & A. Sears (Eds), *Handbook of Human-Computer Interaction* (2nd Ed., 1017-1041). New York, NY, USA: Lawrence Erlbaum Associates.

Berthelot, R. B., Duval, T., Royan, J., & Arnaldi, B. (2011). Improving Reusability of Assets for Virtual Worlds while Preserving 3D Formats Features. *Journal of Virtual Worlds Research, 4*(3). Retrieved August 5, 2013 from http://journals.tdl.org/jvwr/index.php/jvwr/article/view/6123/5781

Beznosyka, A., Renny, J., Hariandjaa, O., Coninxa, K., Quaxa, P., & Lamottea, W. (2012). Providing Context-Based Adaptation in Collaborative Virtual Environments and Video Games. *International Journal for Infonomics, 5*(3/4), 603-611.

Birtchnell, T., & Urry, T. (2013). 3D, SF and the future. *Futures, 50*, 25-34. http://dx.doi.org/10.1016/j.futures.2013.03.005

Blizzard Entertainment (2010, October 7). *WORLD OF WARCRAFT® SUBSCRIBER BASE REACHES 12 MILLION WORLDWIDE*. Retrieved July 11, 2013 from http://eu.blizzard.com/en-gb/company/press/pressreleases.html?id=10007508

Bogdanovych, A., & Drago, S. (2006). Euclidean Representation of 3D Electronic Institutions: Automatic Generation. In *Proceedings of the working conference on*

Advanced visual interfaces (pp. 449-452). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1133265.1133356

Booth, A. G., & Clark, B. P. (2009). A service-oriented virtual learning environment. *On the Horizon, 17*(3), 232-244. http://dx.doi.org/10.1108/10748120910993268

BOP Consulting. (2010). *Creative and Cultural Economy series / 2: Mapping the Creative Industries: A Toolkit*. London, UK: British Council.

Bostanci, E., & Clark, A. F. (2011). Living the Past in the Future. In *Workshop Proceedings of the 7th International Conference on Intelligent Environments* (pp. 167-172). Amsterdam, NL: IOS Press. http://dx.doi.org/10.3233/978-1-60750-795-6-167

Buche, C. (2012). *Adaptive behaviors for virtual entities in participatory virtual environments.* (Habilitation Thesis). Retrieved August 7, 2013 from hal.archives-ouvertes.fr/docs/00/67/25/18/PDF/HDR_BUCHE.pdf

Burri, M. (2011). MISUNDERSTANDING CREATIVITY: USER CREATED CONTENT IN VIRTUAL WORLDS AND ITS CONSTRAINTS BY CODE AND LAW. *International Journal of Communications Law and Policy, 14*, Retrieved April 8, 2013 from http://ijclp.net/ojs/index.php/ijclp/article/view/16/6

Cai, H., Sun, B., Farh, P., & Ye, M. (2008). Virtual Learning Services over 3D Internet: Patterns and Case Studies. In *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 2* (pp. 213-219). New York, NY, USA: IEEE. http://dx.doi.org/10.1109/SCC.2008.81

Card, O. S. (2001). *HOW TO WRITE Science Fiction & Fantasy*. Cincinnati, Ohio, USA: F+W Publications.

Carr, M., & Verner, J. (1997). *Prototyping and Software Development Approaches.* Retrieved March 28, 2013 from http://rguerrero334.blogspot.es/img/Prototyping_and_Software_Development_Approaches.pdf

Cavazza, M., Lugrin, J.-L., Hartley, S., Libardi, P., Barnes, M. J., Le Bras, M., . . . Nandi, A. (2004). New Ways of Worldmaking: the Alterne Platform for VR Art. In *Proceedings of the 12th annual ACM international conference on Multimedia* (pp. 80-87). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1027527.1027542

Celentano, A., Nodari, M., & Pittarello, F. (2004). Adaptive Interaction in Web3D Virtual Worlds. In *Proceedings of the ninth international conference on 3D Web technology* (pp. 41-50). New York, NY, USA: ACM. http://dx.doi.org/10.1145/985040.985047

Chaudhuri, S., & Koltun, V. (2010). Data-Driven Suggestions for Creativity Support in 3D Modeling. *ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH Asia 2010, 29*(6), Article No. 183. http://dx.doi.org/10.1145/1882261.1866205

Cheslack-Postava, E., Azim, T., Mistree, B. F. T., Horn, D. R., Terrace, J., Levis., P., & Freedman, M. J. (2012). A Scalable Server for 3D Metaverses. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association.

Chittaro, L., & Ranon, R. (2007). Adaptive Hypermedia Techniques for 3D Educational Virtual Environments. In *IEEE Intelligent Systems, 22*(4). 31-37. http://dx.doi.org/10.1109/MIS.2007.63

Choi, S.H., & Cheung, H.H. (2012). Virtual Prototyping for Rapid Product Development, Modeling and Simulation in Engineering, In. A. Catalin (Ed.), *Modeling and*

Simulation in Engineering (pp. 203-224). Rijeka, Croatia: InTech Europe. http://dx.doi.org/10.5772/1415

Chu, P. (2009). *Game Development with Unity*. Retrieved July 16, 2013 from http://forrest.technicat.com/games/unity.pdf

Dafli E. L., Vegoudakis K. I., Pappas C., & Bamidis, P. D. (2009). Re-use and exchange of an OpenSim platform based learning environment among different medical specialties for clinical scenarios. In *Proceedings of the 9th International Conference on Information Technology and Applications in Biomedicine*. Washington, DC, USA: IEEE Computer Society. http://dx.doi.org/10.1109/ITAB.2009.5394369

Dalziel, S. (2010, Aug 19). MMO games can do user generated content. *the Inquierer*. Retrieved July 10, 2013 from http://www.theinquirer.net/inquirer/feature/1727926/mmo-games-user-generated-content

de Aquino, M. S., de Souza, F. d. F. (2012). Adaptive Virtual Environments: The Role of Intelligent Agents. In H. Xu (Ed), *Practical Applications of Agent-Based Technology* (pp. 87-110). Rijeka, Croatia: InTech Europe. http://dx.doi.org/10.5772/37877

de Freitas, S. (2008). *Serious Virtual Worlds: A scoping study*. Bristol, UK: Jisc. Retrieved April 8, 2013 from www.jisc.ac.uk/media/documents/publications/seriousvirtualworldsv1.pdf

de Troyer, O., & Kleinermann, F., & Ewais, A. (2010). Enhancing Virtual Reality Learning Environments with Adaptivity: Lessons Learned. In *Proceedings of the 6th international conference on HCI in work and learning, life and leisure: workgroup human-computer interaction and usability engineering* (pp. 244-265). Berlin, Germany: Springer.

Dethridge, L. (2009). The Magic Realism of a Virtual Second Life. *Literature & Aesthetics, 19*(2), 262-278.

Dionisio, J. D. N., Burns, W. G., & Gilbert R. (2013). 3D Virtual Worlds and the Metaverse: Current Status and Future Possibilities. *ACM Computing Surveys, 45*(3), Article No. 34. http://dx.doi.org/10.1145/2480741.2480751

Dionne, D., Puente, S. d. l., Leon, C., Hervas, R., & Gervas, P. (2009). A Model for Human Readable Instruction Generation Using Level-Based Discourse Planning and Dynamic Inference of Attributes Disambiguation. In *Proceedings of the 12th European Workshop on Natural Language Generation* (pp. 66-73). Stroudsburg, PA, USA: Association for Computational Linguistics. http://*dx.doi.org/10.3115/1610195.1610205*

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 ScientificResearch Platform: The Meta-Institute for Computational Astrophysics (MICA). In F. Lehmann-Grube & Sablatnig J. (Eds.), *Facets of Virtual Environments* (pp. 29-43). Berlin, Germany: Springer.

Ducheneaut, N., & Yee, N. (2008). Collective Solitude and Social Networks in World of Warcraft. In C.T. Romm, C. Romm-Livermore & K. Setzekorn (Eds.), *Networking Communities and E-dating Services: Concepts and Implications* (pp. 81-103). Hershey, PA, USA: Information Science Reference.

Eow, Y. L., Ali, W. Z.b.W., Mahmud, R.b, & Baki, R.. (2010). Computer games development and appreciative learning approach in enhancing students' creative perception.

Computers & Education, 54(1), 146-161. http://dx.doi.org/10.1016/j.compedu.2009.07.019

Fairfield, J. A. T. (2012). Avatar Experimentation: Human Subjects Research in Virtual Worlds (Washington & Lee Legal Studies Paper No. 2012-26). Washington and Lee University, VA, USA: School of Law. Retrieved April 8, 2013 from http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2066659

Farley, H. (2009). Reusable Learning Designs and Second Life: Issues and Strategies. In G. Siemens & C. Fulford (Eds.), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2009* (pp. 4047-4052). Chesapeake, VA, USA: AACE.

Farley, M. (2010). Making Virtual Copyright Work. *Golden Gate University Law Review, 41*(1). Retrieved July 10, 2013 from http://digitalcommons.law.ggu.edu/ggulrev/vol41/iss1/4/

Fishwick, P. A. (2009). AN INTRODUCTION TO OPENSIMULATOR AND VIRTUAL ENVIRONMENT AGENT-BASED M&S APPLICATIONS. In *Proceedings of the 2009 Winter Simulation Conference* (pp. 177-183). Washington, DC, USA: IEEE Computer Society. http://dx.doi.org/10.1109/WSC.2009.5429324

Flew, T., &. Cunningham, S. (2010). Creative Industries After the First Decade of Debate. *The Information Society - Creative Industries and Urban Development, 26*(2), 113-123. http://dx.doi.org/10.1080/01972240903562753

Freudenthaler, S. (2011). *Flexible Learning Settings in Second Life* (Unpublished master thesis). Graz University of Technology, Graz.

Friedrich, T. L., Stenmark, C. K., & Mumford, M. D. (2011). Climate for Creativity. In S. R. Pritzker & M. A. Runco (Eds.), *Encyclopedia of Creativity* (2nd ed., vol. 1, pp. 208-213). London, UK: Elsevier Inc.

Gabora, L. (2002). Cognitive Mechanisms Underlying the Creative Process. In *Proceedings of the 4th conference on Creativity & cognition* (pp. 126-133). New York, NY, USA: ACM.

Galloway, S., & Dunlop, S. (2007). A CRITIQUE OF DEFINITIONS OF THE CULTURAL AND CREATIVE INDUSTRIES IN PUBLIC POLICY. *International Journal of Cultural Policy, 13*(1), 17-31. http://dx.doi.org/10.1080/10286630701201657

Gerbaud, S., Gouranton, V., & Arnaldi, B. (2009). Adaptation in Collaborative Virtual Environments for Training. In *Proceedings of the 4th International Conference on E-Learning and Games: Learning by Playing. Game-based Education System Design and Development* (pp. 316-327). Berlin, Germany: Springer. http://dx.doi.org/10.1007/978-3-642-03364-3_40

Gladstone, A. (2012, February 7). OpenSim founder goes for Unity. *Hypergrid Business*. Retrieved July 17, 2013 from http://www.hypergridbusiness.com/2012/02/opensim-founder-goes-for-unity/

Graham, G. (2013). Exploring imaginative futures writing through the fictional prototype 'crime-sourcing'. *Futures, 50*, 94-100. http://dx.doi.org/10.1016/j.futures.2013.04.002

Graham, G., Greenhill, A., & Callaghan V. (2013). Exploring business visions using creative fictional prototypes. *Futures, 50*, 1-4. http://dx.doi.org/10.1016/j.futures.2013.04.001

Grimes, J. M., Fleischman, K. R., & Jaeger, P. T. (2009). Virtual Guinea Pigs: Ethical implications of Human Subjects Research in Virtual Worlds. *International Journal of Internet Research Ethics, 2*(1), 38-56.

Gütl, C., Chang, V., & Freudenthaler, S. (2010). How to Support More Flexible Learning Settings in Second Life. In *Proceedings of the International Conference on Interactive Computer Aided Learning* (pp. 129-141). Retrieved August 5, 2013 from http://www.icl-conference.org/dl/proceedings/2010/contributions/Contribution127.pdf

Gütl, C., Chang, V., & Freudenthaler, S. (2014). Supporting Diverse Needs of Learning Groups: Towards Highly Flexible Learning Settings in Collaborative 3D Virtual Environments. In Hebbel-Seeger, A., Reiners, T., & Schäffer, D. (Eds.), *Synthetic Worlds. Emerging Technologies in Education and Economics.* (pp. 355-378). New York, NY, USA: Springer. http://dx.doi.org/10.1007/978-1-4614-6286-6_14

Gütl, C., Chang, V., Kopeinik, S., & Williams, R. (2009). 3D Virtual Worlds as a Tool for Collaborative Learning Settings in Geographically Dispersed Environments. In *Proceedings of the 12th International Conference on Interactive Computer Aided Learning* (pp. 310-323). Retrieved March 09, 2014 from http://espace.library.curtin.edu.au/webclient/DeliveryManager?pid=132345&custom_att_2=direct

Gütl, C., Haas, K., & Chang, V. (2013). Configuarable and Flexible Immersive Learning Environment: An Enhanced Solution for the OpenSim Plattform to Support Endusers. In *2013 International Conference on Interactive Collaborative Learning* (pp. 232–237). New York, NY, USA: IEEE. http://dx.doi.org/10.1109/ICL.2013.6644576

Gütl, C., Scheucher, T., Bailey, P. H., Belcher, J., dos Santos, F. R., & Berger, S. (2011). Towards an Immersive Virtual Environment for Physics Experiments Supporting Collaborative Settings in Higher Education. In Azad, A. K. M., Auer, M. E., & Harward, V. J. (Eds), *Internet Accessible Remote Laboratories: Scalable E-Learning Tools for Engineering and Science Disciplines* (pp. 543-562). Hershey, PA, USA: IGI Global.

Haas, K. (2012). *Flexible Learning Environment in Virtual 3D Worlds* (Unpublished master thesis). Graz University of Technology, Graz.

Haberl, K., Proctor, S., Blackman, T., Kaplan, J., & Kotzen, J. (2008). *Big Project: Project Darkstar* [PDF docuent]. Retrieved March 28, 2013 from http://docs.huihoo.com/darkstar/darkstar-2008.pdf

Harrington, D. M. (2011). Creative Environments, Conditions, and Settings. In S. R. Pritzker & M. A. Runco (Eds.), *Encyclopedia of Creativity* (2nd ed., vol. 1, pp. 208-213). London, UK: Elsevier Inc.

Hartmann, B. (2009). *GAINING DESIGN INSIGHT THROUGH INTERACTION PROTOTYPING TOOLS*. (Unpublished doctoral dissertation). Retrieved June 16, 2013 from http://hci.stanford.edu/publications/2009/hartmann-diss.pdf

Helmer, J., & Learning Light. (2007). Second Life and virtual worlds. *Learning Light*, Retrieved April 8, 2013 from http://www.norfolkelearningforum.co.uk/wp-content/uploads/2009/04/virtual-worlds_ll_oct_2007.pdf

Hesmondhalgh, D. (2008). Cultural and Creative Industries. In T. Bennett & J. Frow (Eds.), *The SAGE Handbook of Cultural Analysis* (pp. 552-569). London, UK Sage Publications Ltd.

Hesmondhalgh, D., & Pratt, A. C. (2005). Cultural industries and cultural policy. *International Journal of Cultural Policy, 11*(1), 1–13. http://dx.doi.org/10.1080/10286630500067598

Holmquist, L. E. (2005). Prototyping: generating ideas or cargo cult designs? *interactions – Robots!, 12*(2), 48-54. http://dx.doi.org/10.1145/1052438.1052465

Johnson, B. D. (2010). Science Fiction for Scientists!! An Introduction to SF Prototypes and Brain Machines. In *Workshops Proceedings of the 6th International Conference on Intelligent Environments* (pp. 195-203). Amsterdam, NL: IOS Press. http://dx.doi.org/10.3233/978-1-60750-639-3-195

Johnson, B. D. (2011). *Science Fiction Prototyping: Designing the Future with Science Fiction*. San Rafael, California, USA: Morgan & Claypool.

Johnson, H., & Carruthers, L. (2006). Supporting creative and reflective processes. *International Journal of Human-Computer Studies, 64*(10), 998-1030. http://dx.doi.org/10.1016/j.ijhcs.2006.06.001

Johnstone, J. (2007). Towards a creativity research agenda in information ethics. *International Review of Information Ethics, 7*, 1-10. Retrieved May 5, 2013 from http://www.i-r-i-e.net/inhalt/007/34-johnstone.pdf

Kain, E. (2013, March 9) As 'World Of Warcraft' Bleeds Subscribers, Free-To-Play Is Already Winning The Future. *Forbes*. Retrieved July 11, 2013 from http://www.forbes.com/sites/erikkain/2013/05/09/as-world-of-warcraft-bleeds-subscribers-free-to-play-is-already-winning-the-future/

Kaplan, J., & Yankelovich, N. (2011). Open Wonderland: Extensible Virtual World Architecture. *IEEE Internet Computing, 15*(5), 38-45. http://dx.doi.org/10.1109/MIC.2011.76

Klüwer, T., Adolphs, P., Xu, F., Uszkoreit, H., & Cheng, X. (2010). Talking NPCs in a Virtual Game World. In *Proceedings of the ACL 2010 System Demonstrations* (pp. 36-41). Stroudsburg, PA, USA: Association for Computational Linguistics.

Kohler, T., Fueller, J., Matzler, K., & Stieger, D. (2011). CO-CREATION IN VIRTUAL WORLDS: THE DESIGN OF THE USER EXPERIENCE. *MIS Quarterly*, 35(3), 773-778.

Kohno, T., & Johnson, B. D. (2011). Science Fiction Prototyping and Security Education: Cultivating Contextual and Societal Thinking in Computer Security Education and Beyond. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 9-14). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1953163.1953173

Kremer, M. (2013, March 10). World of Warcraft verliert 1,3 Millionen Spieler in drei Monaten. *gulli*. Retrieved July 11, 2013 from http://www.gulli.com/news/21493-world-of-warcraft-verliert-13-millionen-spieler-in-drei-monaten-2013-05-10

Kymäläinen, T. (2013). Dreamnesting – Co-created future vision of an intelligent interior design experience. *Futures, 50*, 74-85. http://dx.doi.org/10.1016/j.futures.2013.03.013

LAMS. (2012, March 1). *What is LAMS?* Retrieved August 8, 2013 from the LAMS Wiki: http://wiki.lamsfoundation.org/display/lamsdocs/About+LAMS#AboutLAMS-whatIs

Lastowka., G. (2010). *VIRTUAL JUSTICE: the new laws of online worlds*. New Haven, CT, USA: Yale University Press.

Leidl, M., & Rößling, G. (2007). How Will Future Learning Work in the Third Dimension? In *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education* (pp. 329-329). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1269900.1268897

Li, K. (n.d.). *Building 3D Worlds with Project Wonderland* [PDF document]. Retrieved March 28, 2013 from https://kenai.com/downloads/osum/Wonderland.pdf

Linden Lab. (2013, July 20). *Second Life Celebrates 10-Year Anniversary*. Retrieved July 12, 2013 from http://lindenlab.com/releases/second-life-celebrates-10-year-anniversary

Liou, F.W. (2008). *RAPID PROTOTYPING AND ENGENEERING APPLICATIONS: A Toolbox for Prototype Development*. Boca Raton, Florida, USA: CRC Press.

Liu, H., Bowman, M., Hunt, W. A., & Duffy, A. M. (2012). ENABLING BEHAVIOR REUSE IN DEVELOPMENT OF VIRTUAL ENVIRONMENT APPLICATIONS. In *Proceedings of the 2012 Winter Simulation Conference* (pp. 1621-1632). New York, NY, USA: IEEE. http://dx.doi.org/10.1109/WSC.2012.6465246

Marcus, T. D. (2007). Fostering Creativity in Virtual Worlds: Easing the Restrictiveness of Copyright for User-Created Content. *New York Law School Law Review, 52*(1), 67-92.

Mazuryk, T., & Gervautz, M. (1996). *Virtual Reality, History, Applications, Technology and Future*. Retrieved November 15, 2013 from http://www.cg.tuwien.ac.at/research/publications/1996/mazuryk-1996-VRH/

Mehta, M., & Ram, A. (2009). Runtime Behavior Adaptation for Real Time Interactive Games. *IEEE Transactions on Computational Intelligence and AI in Games, 1*(3). 187 – 199. http://dx.doi.org/10.1109/TCIAIG.2009.2032416

Messinger, P. R., Stroulia E., & Lyons, K. (2008). A Typology of Virtual Worlds: Historical Overview and Future Directions. *Journal of Virtual Worlds Research, 1*(1). Retrieved April 8, 2013 from http://journals.tdl.org/jvwr/index.php/jvwr/article/view/291

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. http://dx.doi.org/10.1016/j.dss.2009.02.014

Minocha, S., Tran, M. Q., & Reeves, A. J. (2010). Conducting Empirical Research in Virtual Worlds: Experiences from two projects in Second Life. *Journal of Virtual Worlds Research, 3*(1). Retrieved April 8, 2013 from http://journals.tdl.org/jvwr/index.php/jvwr/article/view/811/882

Morie, J. F., Verhulsdonck, G., Luria, R. M., & Keeton, K. E. (2011). *Operational Assessment Recommendations: Current Potential and Advanced Research Directions for Virtual Worlds as Long-Duration Space Flight Countermeasures* (NASA/TP-2011-216164). Houston, TX, USA: NASA Johnson Space Center. Retrieved April 8, 2013 from http://ston.jsc.nasa.gov/collections/trs/_techrep/TP-2011-216164.pdf

Moro, A., Mumolo, E., & Nolich, M. (2010). Building Virtual Worlds by 3D Object Mapping. In *Proceedings of the 2010 ACM workshop on Surreal media and virtual cloning* (pp. 31-36). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1878083.1878092

Nakakoji, K., Yamamoto, Y., & Aoki, A. (2002). Interaction design as a collective creative process. In *Proceedings of the 4th conference on Creativity & cognition* (pp. 103-110). New York, NY, USA: ACM. http://dx.doi.org/10.1145/581710.581727

Nauha, J. (2012, October 10). *Tundra based hosting service Meshmoon launched*. Retrieved July 16, 2013 from http://realxtend.org/2012/10/10/tundra-based-hosting-service-meshmoon-launched/

O'Conner, J. (2010). *The cultural and creative industries: a literature review* (2nd ed.). Newcastle upon Tyne, UK: Creativity, Culture and Education.

Octavia, J. R., Raymaekers, C., & Coninx, K. (2009). A Conceptual Framework for Adaptation and Personalization in Virtual Environments. *In Proceedings of the 2009 20th International Workshop on Database and Expert Systems Application* (pp. 284-288). Washington, DC, USA: IEEE Computer Society. http://dx.doi.org/10.1109/DEXA.2009.15

OECD. (2011), Virtual Worlds. Immersive Online Platforms for Collaboration, Creativity and Learning, *OECD Digital Economy Papers* (184), Paris, France: OECD Publishing. http://dx.doi.org/10.1787/5kg9qgnpjmjg-en

Open Wonderland. (2013a). *About Open Wonderland*. Retrieved July 16, 2013 from http://openwonderland.org/index.php/about

Open Wonderland. (2013b). Open Wonderland FAQ. Retrieved July 16, 2013 from http://openwonderland.org/index.php/about/faq

Open Wonderland. (2012, March 12). *Learning the Basics of Open Wonderland*. Retrieved July 16, 2013 from the Open Wonderland Community Wiki: http://wiki.openwonderland.org/Wiki.jsp?page=Learning%20the%20Basics%20of%20Open%20Wonderland

OpenSimulator. (2013, June 18). *What is OpenSimulator?* Retrieved July 15, 2013 from the OpenSimulator Wiki: http://opensimulator.org/wiki/Main_Page

Pandey, P.M. (n.d.). *RAPID* PROTOTYPING *TECHNOLOGIES, APPLICATIONS AND PART DEPOSITION PLANNING*. Retrieved March 27, 2013 from http://web.iitd.ac.in/~pmpandey/MEL120_html/RP_document.pdf

Pape, D., Anstey, J., Dolinsky, M., & Dambikc, E. J. (2003). Ygdrasil—a framework for composing shared virtual worlds. *Future Generation Computer Systems, 19*(6), 1041-1049. http://dx.doi.org/10.1016/S0167-739X(03)00081-5

Papp, R. (2011). Virtual worlds and social networking reaching the millennials. *Journal of Technology Research, 2*. Retrieved July 8, 2013 from http://www.aabri.com/manuscripts/10427.pdf

Pellens, B., de Troyer, O., & Kleinermann, F. (2008). CoDePA: A Conceptual Design Pattern Approach to model Behavior for X3D worlds. In *Proceedings of the 13th international symposium on 3D web technology* (pp. 91-99). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1394209.1394229

Pirker, J., & Gütl, C. (2012). Iterative evaluation of a virtual three-dimensional environment for start-up entrepreneurship in different application scenarios. In *Proceedings of the 15th International Conference on Interactive Collaborative Learning*. New York, NY, USA: IEEE. http://dx.doi.org/10.1109/ICL.2012.6402035

Pirker, J., Gütl, C., Weghofer, P., & Feichtner, V. (2014). Interactive Science Fiction Prototyping in Virtual Worlds: Fundamentals and Applications. *International Journal of Recent Contributions from Engineering, Science & IT, 2*(3), 46-52. http://dx.doi.org/10.3991/ijes.v2i3.3824

Plucker, J. A., Runco, M. A., & Hegarty, C. B. (2011). Enhancement of Creativity. In S. R. Pritzker & M. A. Runco (Eds.), *Encyclopedia of Creativity* (2nd ed., vol. 1, pp. 208-213). London, UK: Elsevier Inc.

Potstada, M., & Zybura, J. (2014). The role of context in science fiction prototyping: The digital industrial revolution. In *Technological Forecasting & Social Change*, 84, 101-114. http://dx.doi.org/10.1016/j.techfore.2013.08.026

Qui, L., Tay, W.W., & Wu, J. (2009). The impact of virtual teamwork on real-world collaboration. In *Proceedings of the International Conference on Advances in Computer Enterntainment Technology* (pp. 44-51). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1690388.1690396

Quintella, F., Soares, L. P., & Raposo, A. B. (2010). DWeb3D: A toolkit for developing X3D applications in a simplified environment. In *Proceedings of the 15th International Conference on Web 3D Technology* (pp. 45-54). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1836049.1836056

Sanchez-Lozane, C. (2013). Storyweavers. In *Proceedings of the 3rd European Immersive Education Summit* (pp. 75-85). London, UK: iED.

Sarsani, M. R. (2011). Computers and Creativity. In S. R. Pritzker & M. A. Runco (Eds.), *Encyclopedia of Creativity* (2nd ed., vol. 1, pp. 208-213). London, UK: Elsevier Inc.

Scales, J. A., & Snider, R. (1999). Computers and creativity. *Geophysics*, 64(5), 1347-1348. Retrieved April 8, 2013 from http://inside.mines.edu/~rsnieder/computers.pdf

Schwarz, J. O., Liebl, F. (2013). Cultural products and their implications for business models: Why science fiction needs socio-cultural fiction. *Futures, 50*, 66-73. http://dx.doi.org/10.1016/j.futures.2013.03.006

Seidel, S. (2009). A THEORY OF MANAGING CREATIVITY-INTENSIVE PROCESSES. (Doctoral dissertation). Retrieved April 8, 2013 from http://miami.uni-muenster.de/servlets/DocumentServlet?id=4831 (urn:nbn:de:hbz:6-71519585023)

Shedroff, N., & Noessel, C. (2012). *MAKE IT SO: Interaction Design Lessons from Science Fiction*. Brooklyn, NY, USA: Rosenfeld Media.

Singh, G. (2012, January 24). *Brief exploration of creativity in virtual world of Second Life*. Retrieved July 22, 2013 from http://www.examiner.com/article/brief-exploration-of-creativity-virtual-world-of-second-life

Smelik, R. M. (2011). *A Declarative Approach to Procedural Generation of Virtual Worlds.* (Doctoral dissertation). Retrieved August 1, 2013 from http://repository.tudelft.nl/view/ir/uuid%3Ad6a97083-35a1-411c-bfb0-5f8714dbc2ef/

Smelik, R., Galka, K., de Kraker, K. J., Kuijper, F., & Bidarra, R. (2011). Semantic constraints for procedural generation of virtual worlds. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (Article No. 9). New York, NY, USA: ACM. http://dx.doi.org/10.1145/2000919.2000928

Smelik, R. M., Tutenel, T., de Kraker, K. J., & Bidarra, R. (2011). A declarative approach to procedural modeling of virtual worlds. *Computers & Graphics, 35*(2), 352-363. http://dx.doi.org/10.1016/j.cag.2010.11.011

Stanton, J. M. (2010). Virtual Worlds, the IRB and a User's Bill of Rights. *Journal of Virtual Worlds Research, 3*(1). Retrieved April 8, 2013 from http://journals.tdl.org/jvwr/index.php/jvwr/article/view/1567/875

Tassini, K. (2011). The Magician's Assistant. In *Workshop Proceedings of the 7th International Conference on Intelligent Environments* (pp. 267-278). Amsterdam, NL: IOS Press. http://dx.doi.org/ 10.3233/978-1-60750-795-6-267

Thompson, C. W. (2011). Next-Generation Virtual Worlds: Architecture, Status, and Directions. *IEEE Internet Computing, 15*(1), 60-65. http://dx.doi.org/10.1109/MIC.2011.15

Trescak, T., Esteva, M., & Rodriguez, I. (2010). A Virtual World Grammar for Automatic Generation of Virtual Worlds. *The Visual Computer: International Journal of Computer Graphics, 26*(6-8), 521-531. http://dx.doi.org/10.1007/s00371-010-0473-7

Tutenel, T., van der Linden, R., Kraus, M., Bollen, B., & Bidarra, R. (2011). Procedural filters for customization of virtual worlds. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (Article No. 5). New York, NY, USA: ACM. http://dx.doi.org/10.1145/2000919.2000924

UNCTAD. (2010). *CREATIVE ECONOMY Report 2010. Creative Economy: A Feasible Development Option*. Geneva, CH: UNCTAD.

Unity Technologies. (2005, June 30). *Unity 1.0.1 is out*. Retrieved July 17, 2013 from http://unity3d.com/company/public-relations/news/unity-101-out

Unity Technologies. (2013, July 9). *Unity Technologies Doubles Community to Two Million Developers*. Retrieved July 17, 2013 from http://unity3d.com/company/public-relations/news/unity-technologies-doubles-community-two-million-developers

Vatjus-Antilla, J. M., Hickey, S., Koskela, T. (2013). Adaptive Content Management for Collaborative 3D Virtual Spaces. In *Proceedings of the 13th Conference of FRUCT Association* (pp. 132-142). St. Petersburg, RU: State University of Aerospace Instrumentation.

Wang, G. G. (2003). Definition and Review of Virtual Prototyping. *Journal of Computing and Information Science in Engineering, 2*(3), 232-236. http://dx.doi.org/10.1115/1.1526508

Ward, T. B., & Sonneborn, M. S. (2009). Creative Expression in Virtual Worlds: Imitation, Imagination and Individualized Collaboration. *Psychology of Aesthetics, Creativity, and the Arts, 3*(4), 211-221. http://dx.doi.org/10.1037/a0016297

Warfel, T. Z. (2009). *PROTOTYPING: A Practitioner's Guide*. Brooklyn, NY, USA: Rosenfeld Media.

Warr, A., & O'Neill, E. (2005). Understanding Design as a Social Creative Process. In *Proceedings of the 5th conference on Creativity & cognition* (pp. 118-127). New York, NY, USA: ACM.

Wood, D., & Hopkins, L. (2008). 3D virtual environments: Businesses are ready but are our 'digital natives' prepared for changing landscapes? In *Proceedings ascilite Melbourne 2008: Hello! Where are you in the landscape of educational technology?* (pp. 1136-1146). Tungun, QLD, AU: Ascilite.

Wright, S. (2010). *Creative and Cultural Economy series / 2 Mapping the Creative Industries: A Toolkit*. London, UK: British Council (Accepted Paper No. 2012-26).

Wu, H. Y. (2013). Imagination workshops: An empirical exploration of SFP for technology-based business innovation. *Futures, 50*, 44-55. http://dx.doi.org/10.1016/j.futures.2013.03.009

Wu, H. Y., & Callaghan, V. (2011). The Spiritual Machine. In *Workshop Proceedings of the 7th International Conference on Intelligent Environments* (pp. 155-166). Amsterdam, NL: IOS Press. http://dx.doi.org/10.3233/978-1-60750-795-6-155

Zheng, P. & Callaghan, V. (2012), Creative Science: A New Way of Learning Innovation and Entrepreneurship? In *Entrepreneurial Learning in Organizations*. Retrieved December 17, 2013 from http://www.isbe.org.uk/Creative-Science-a-new-way-of-learning-innovation-and-entrepreneurship

Appendix A: Questionnaire

Pre-Questionnaire

Age	_____
Sex	O female O male
Field of Study	_____
Highest Level of education	_____
Working field	_____
Research interests	_____
How many hours per week do you usually use a computer?	O 0 O 1-5 O 5-10 O 10-20 O more than 20

How many hours per week do you usually use technologies for Communication/Collaboration with others?

Email	0, 1-5, 5-10, 10-20, >20
Instant-messaging (ICQ, Facebook, …)	0, 1-5, 5-10, 10-20, >20
Voice-Chat	0, 1-5, 5-10, 10-20, >20
Video-Chat	0, 1-5, 5-10, 10-20, >20
Forums	0, 1-5, 5-10, 10-20, >20
Newsgroups	0, 1-5, 5-10, 10-20, >20
Google Docs	0, 1-5, 5-10, 10-20, >20
other	

How many hours a week do you use internet?	0, 1-5, 6-10, 11-20, more than 20
How many hours a week do you play video-games?	0, 1-5, 6-10, 11-20, more than 20
How many hours a week do you play MMOS?	0, 1-5, 6-10, 11-20, more than 20
How many hours a week do you use virtual Worlds (Second Life, OWL, etc)?	0, 1-5, 6-10, 11-20, more than 20
How many hours a week do you use tools/programs to build or create 3 dimensional items?	0, 1-5, 6-10, 11-20, more than 20
How many hours a week do you use 2D drafting tools/programs, like AutoCAD?	0, 1-5, 6-10, 11-20, more than 20
How would you define a Virtual World?	

What do you expect from a Virtual World?	
What would you use a Virtual World for?	
Where do you see advantages using a Virtual World?	
Where do you see disadvantages using a Virtual World?	
What do you expect from a virtual world editor?	
What would you see a most important necessity for a world editor?	
What could the benefits of a Virtual World Editor be?	
What could the disadvantages of a Virtual World Editor be?	
How do you think such a tool may look like?	
An easy to use interface is important for me, in order to build a world.	(1, 2, 3, 4, 5)
It is important to me, to be able to inspect my creations in the virtual world.	(1, 2, 3, 4, 5
It is important to me, to see my changes immediately.	(1, 2, 3, 4, 5)
Collaboration is important, when building virtual worlds.	(1, 2, 3, 4, 5)

Post-Questionnaire

How was your first impression?	good, neutral, bad
Why do you feel that way?	
Would you use a virtual world?	
To which fields do you think can virtual worlds applied?	
I would use it for communication	(1, 2, 3, 4, 5)
I would use it for collaboration	(1, 2, 3, 4, 5)
I would use it for meetings	(1, 2, 3, 4, 5)
I would use it for learning	(1, 2, 3, 4, 5)
The response time is accurate	(1, 2, 3, 4, 5)
The interface is very stimulating	(1, 2, 3, 4, 5)
The design of the single interface elements is consistent	(1, 2, 3, 4, 5)
I did enjoy creating a virtual room	(1, 2, 3, 4, 5)
I would build my own world, if I had the time	(1, 2, 3, 4, 5)
What kind of world would that be?	
What do you think is easier to use?	O Virtual World tools O Editor
What would you use to build a world? The editor, the Virtual World tools, or something else?	
Why?	
What are the strengths of the editor compared to the tools?	
What are the strengths of the tools compared to the editor?	
What did you like about the virtual world tools?	
What did you not like about the virtual world tools?	

What would you improve?	
Overall the virtual world tools were easy to use.	(1, 2, 3, 4, 5)
Overall, the tool's features were self explanatory.	(1, 2, 3, 4, 5)
Overall, the design of the tools was logical.	(1, 2, 3, 4, 5)
Overall, I felt familiar with the way the tools were functioning.	(1, 2, 3, 4, 5)
Why? Why not?	
I think, someone without any knowledge about virtual worlds could build a world using the tools.	(1, 2, 3, 4, 5)
Using the tools was time consuming.	(1, 2, 3, 4, 5)
Moving objects was easy	(1, 2, 3, 4, 5)
Copying objects was easy	(1, 2, 3, 4, 5)
Setting the rights was easy	(1, 2, 3, 4, 5)

Answer only, if you completed Task 4 with the Virtual World Tools

Saving the world was easy	(1, 2, 3, 4, 5)
Loading the world was easy	(1, 2, 3, 4, 5)
Importing new objects was easy	(1, 2, 3, 4, 5)

What did you like about the editor?	
What did you not like about editor?	
What would you improve?	
Overall the editor was easy to use.	(1, 2, 3, 4, 5)
Overall, the editor's features were self explanatory.	(1, 2, 3, 4, 5)
Overall, the design of the editor was logical.	(1, 2, 3, 4, 5)
Overall, I did feel familiar with the way the editor was functioning.	(1, 2, 3, 4, 5)
Why? Why not?	

I think someone without any knowledge about virtual worlds could build a world.	(1, 2, 3, 4, 5)
I think a 2D representation of the world is easier to understand and to manipulate	(1, 2, 3, 4, 5)
Using the editor was time consuming.	(1, 2, 3, 4, 5)
I think, I have seen all features of the editor.	(1, 2, 3, 4, 5)
Which features would you like to have in a tool such as the editor?	
Which features do you see as unnecessary?	
I would prefer to see the objects in 2D rather than a representation of them.	(1, 2, 3, 4, 5)
I liked the graphical design of the editor.	(1, 2, 3, 4, 5)
What would you change on the graphical design?	
The design is consistent.	(1, 2, 3, 4, 5)
I would prefer to be able to change object and selection colors, text sizes and other visuals.	(1, 2, 3, 4, 5)
I think a tool like the editor is necessary to build a world.	(1, 2, 3, 4, 5)
The editor can accelerate the task of building a world.	(1, 2, 3, 4, 5)
Do you have any other suggestions?	
I was interested in seeing how the world would look like that I have built.	(1, 2, 3, 4, 5)
Moving objects was easy	(1, 2, 3, 4, 5)
Copying objects was easy	(1, 2, 3, 4, 5)
Setting the rights was easy	(1, 2, 3, 4, 5)

Answer only, if you completed Task 4 with the Virtual World Editor

Saving the world was easy	(1, 2, 3, 4, 5)
Loading the world was easy	(1, 2, 3, 4, 5)
Importing new objects was easy	(1, 2, 3, 4, 5)

I was not concerned about whether I would be able to complete the tasks.	(1, 2, 3, 4, 5)
The tasks were fun.	(1, 2, 3, 4, 5)
The controls were easy to pick up.	(1, 2, 3, 4, 5)

There were not any particularly frustrating aspects of the controls to get the hang of.	(1, 2, 3, 4, 5)
How immersed did you feel? (10 ... very immersed; 0 … not at all immersed)	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
I was challenged, but I believed my skills would allow me to meet the challenge	(1, 2, 3, 4, 5)
I knew clearly what I wanted to do	(1, 2, 3, 4, 5)
It was really clear to me that I was doing well	(1, 2, 3, 4, 5)
I felt I was competent enough to meet the high demands of the tasks	(1, 2, 3, 4, 5)
I had total concentration	(1, 2, 3, 4, 5)
I had a feeling of total control	(1, 2, 3, 4, 5)

Virtual World tools

I think that I would like to use this system frequently	(1, 2, 3, 4, 5)
I found the system unnecessarily complex	(1, 2, 3, 4, 5)
I thought the system was easy to use	(1, 2, 3, 4, 5)
I think that I would need the support of a technical person to be able to use this system	(1, 2, 3, 4, 5)
I found the various functions in this system were well integrated	(1, 2, 3, 4, 5)
I thought there was too much inconsistency in this system	(1, 2, 3, 4, 5)
I would imagine that most people would learn to use this system very quickly	(1, 2, 3, 4, 5)
I found the system very cumbersome to use	(1, 2, 3, 4, 5)
I felt very confident using the system	(1, 2, 3, 4, 5)
I needed to learn a lot of things before I could get going with this system	(1, 2, 3, 4, 5)

Virtual World Editor

I think that I would like to use this system frequently	(1, 2, 3, 4, 5)
I found the system unnecessarily complex	(1, 2, 3, 4, 5)
I thought the system was easy to use	(1, 2, 3, 4, 5)
I think that I would need the support of a technical person to be able to use this system	(1, 2, 3, 4, 5)
I found the various functions in this system were well integrated	(1, 2, 3, 4, 5)
I thought there was too much inconsistency in this system	(1, 2, 3, 4, 5)
I would imagine that most people would learn to use this system very quickly	(1, 2, 3, 4, 5)
I found the system very cumbersome to use	(1, 2, 3, 4, 5)
I felt very confident using the system	(1, 2, 3, 4, 5)
I needed to learn a lot of things before I could get going with this system	(1, 2, 3, 4, 5)