Intelligent Generation and Control of Interactive Virtual Worlds

Tomáš Trescák

May 31, 2012

Dissertation submitted to obtain the degree:
Doctor en Informàtica
(Ph.D. in Computer Science)

Advisors:
Dr. Inmaculada Rodriguez and
Dr. Anton Bogdanovych

Departament de Ciències de la Computació — Escola d'Enginyeria
Universitat Autònoma de Barcelona

In memory of Dr. Marc Esteva Vivanco

To my dear Valeria, my parents, my friends, and all the dear people I met and helped me and supported me during my study.

And to the members of IIIA.

Choose a job you love and you will never have to work a day in your life.

Confucius.
The Confucian Analects

Contents

List of Figures

Abstract

This thesis advocates the use of non-gaming virtual worlds as a significant future technology for the domains of e-* applications (e- learning, e-commerce, e-government) and social simulations. In such systems, a 3D virtual environment is often populated by a large number of inhabitants that can be either human-controlled avatars or intelligent virtual agents (avatars controlled by autonomous software entities) who engage in advanced interactions with their virtual environment and other participants.

One significant problem that impedes wide adoption of the virtual worlds technology for these problem domains is that virtual worlds in general are difficult to build, and significant effort has to be put into designing the 3D virtual environment and programming virtual agents; but even harder is to ensure the validity of participant interactions in such environments and enforce social norms on their inhabitants so that unauthorized behavior can be prevented.

To address this problem, we have developed a comprehensive technological solution that automates the design of such virtual worlds and its population with virtual agents. Our approach is based on the utilization of Virtual Institutions, which are virtual worlds with normative regulation of participant interactions.

The key focus of the thesis is on explaining how existing methods of formal specification of Virtual Institutions can be extended to automatically translate the institutional specification into an interactive 3D environment using the shape grammars approach and automatically populating such environments with virtual agents.

Shape grammars represent a powerful visual technique for creating procedural 2D and 3D designs, but existing work was not immediately suitable for our problem. Existing shape grammar solutions are normally restricted to very specific scenarios, do not normally address interactivity of the generated designs and rarely consider facilitating agent enactment of the generated environments as well as their normative regulation.

Thus, we have extended existing work and developed the Shape Grammar Interpreter framework, which addresses the limitations of existing solutions. This framework was further utilized for developing the concept of virtual world grammar, which is a sub-set of shape grammars targeting automatic generation of normative virtual worlds. As the result of this dissertation, Virtual Worlds Grammars constitute a strong formalization and a development environment

not only enabling automatic generation of normative virtual worlds, but also their platform independent deployment (using the VIXEE infrastructure that has been developed as an important part of this dissertation).

Another significant contribution of this thesis is developing a mechanism of automatic population of the generated environments with an arbitrary number of software agents, which are capable of intelligent interactions with 3D objects placed in the environment. Moreover, these agents are able to collaborate with human-controlled avatars, facilitate their problem-solving and ensure that all agent actions strictly adhere to social norms of the given institution.

For this purpose we have developed a generic model of virtual agents, which enables an agent to be situated within any normative virtual world, generate its own goals based on its current physiological and psychological needs, as well as to dynamically generate plans for satisfying these goals using the underlying institutional specification. The institutional specification in this instance provides a high-level representation of an agent's interaction possibilities.

To illustrate the usefulness of methods and techniques presented in this thesis, we have applied it to the domain of historical simulation, re-enacting everyday life of ancient people in one of humanity's first cities, the city of Uruk. The existing design of the Uruk city was enriched with dynamically generated artefacts and a large crowd of virtual agents simulating ancient citizens from different classes of the Uruk society. We showed how our approach allows to create a desired number of visually and behaviorally diverse agents, as well as dynamically generating food, tools and other items that they can utilise to satisfy their goals, while acting in a historically authentic manner.

Acknowledgements

This thesis would not be possible without all of the extraordinary people who have helped me with its completion. First, I would like to thank Dr. Marc Esteva, my supervisor and a great friend, whom we have tragically lost in 2011. Marc was a marvellous person and an inspiration to me and many others. I want to thank him for introducing me to the fascinating world of science and skillfully guiding me through it. Marc, you will be remembered in my heart till the very end, and I hope that this thesis makes you proud. Rest in peace buddy.

Great thanks go also to my other supervisors Inma Rodriguez and Anton Bogdanovych. To Inma, for her strong support and guidance and help on all the publications we have done together, you made them shine! To Anton for all the inspiration that feeds my hunger for exploring new ideas, for guidance in writing and a possibility of working on very exciting projects.

My gratitude also goes to Simeon Simoff and John Debenham, for the possibility to visit them in Sydney, where I spent six delightful months, during which many exciting new ideas surfaced, making my stay a great success. Thanks for assisting me with a scholarship without which it would be impossible to come. Thanks to all the lovely friends in Australia, dear Danko and Zuzana, that helped me to get started.

I would also like to thank Alessandro Farinelli for providing me the possibility to visit him in Verona, to learn italian and to eat as much italian pizza and pasta as possible. It was a hectic period, during which I wrote this thesis; thus I would like to thank for his patience and thanks to Meritxell for making this visit possible!

Thanks to Maite Lopéz-Sanchéz, Carlos Carrascosa, Gustavo Aranda, Javier Morales and Pablo Almajano for cooperation on our papers. I would like to thank Pablo for keeping my work "alive" and extending it to the new frontiers. Thanks to the shape grammar community for helping me to enter and understand exciting world of computational design. Thanks to Juan Antonio Rodríguez and Jesús Cerquides, for their help after Marc was gone.

Equally important has been the support from my gorgeous fiancee Valeria Toscani. I would probably already have gone crazy, without all the love she has surrounded me with during writing of this thesis. She made this world a best place to be for me, and I cherish every moment we spend together. Thank you for your patience with me, and I am looking forward to living our happily ever

after.

I would also like to thank my parents, my brother Michal and his lovely wife Diana for supporting me and encouraging me during this scientific journey, which in times was not easy. Many thanks also go to Ondrej Svora for his support and friendship. Thanks to my superb friends in Brno, Prague and Kosice, although we have not seen each other much I could feel your presence during difficult times.

Many thanks go to all the great friends from Barcelona and friends and colleagues from IIIA and CVC. Without the possibility to blow off some steam after work with a couple of horrible spanish beers I would probably explode. Thanks to Tomas Zelina, Andrew Koster, Javier Tur, Jose Rubio my Czechoslovak crew and many others.

This thesis would not have been possible without the Spanish government's generous funding through the Agreement Technologies project (CONSOLIDER CSD2007-0022, INGENIO 2010), and I am grateful to Carles Sierra for his leadership of this project, giving me the opportunity to write this thesis and present parts of it at various conferences and workshops. I am also grateful for the funding provided by European projects from MICIN (TIN2009-14702-C02-01), MEC (TIN2006-15662-C02-01) and spanish project from CSIC (2006 5 OI 099). My visit to Australia was funded by the Endeavour Research Fellowship scholarship, awarded by the Australian government, thank you for this opportunity. My visit to Italy was funded by the European Union's COST Action on Agreement Technologies (ICO801), great thanks to you too.

Finally, I would like to thank everybody who I have forgotten to thank. If I have forgotten to acknowledge you, I am in remiss. I want you to know that I am terribly sorry and will be sure to rectify this oversight in any future thesis I write.

–Tomas Trescak

Chapter 1

Introduction

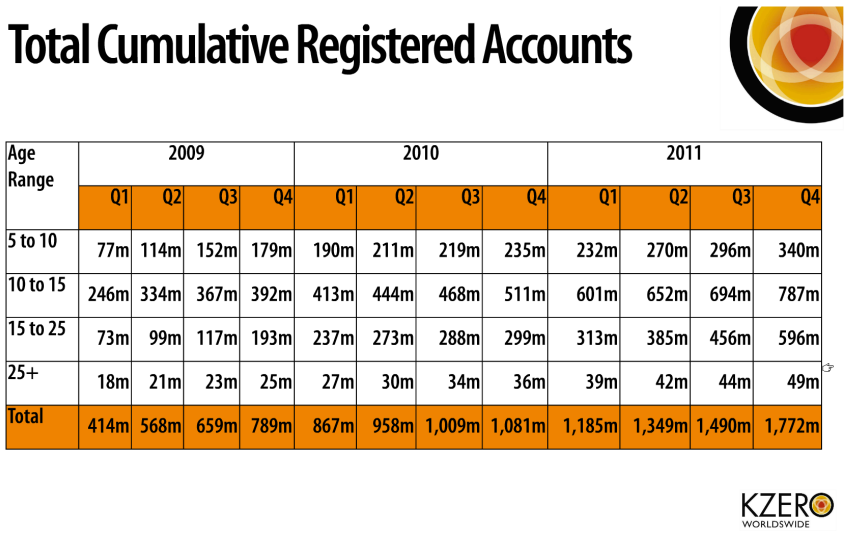
In this thesis, we are interested in 3D virtual worlds, and particularly in *normative virtual worlds*, which are virtual spaces that provide participants an immersive experience and specify a set of rules to control the validity of their interactions. We define methods and algorithms that allow *intelligent generation and control* of such virtual worlds, where humans and autonomous agents participate in order to achieve their common or individual goals.

This chapter introduces the motivation for our research, explains our goals and contributions, and introduces two application examples, which we use throughout this thesis. Further, it provides details on problems that virtual worlds face, and describes how solving these problems can facilitate their definition, execution and deployment.

1.1 Motivation

Not long ago, computers existed only as separate units, or they were connected to small office networks or home networks. Then, people started to connect them to larger networks, culminating to the World Wide Web and the start of the first era of the Internet. The second era of the Internet started with search engines, transforming the Internet into the information superhighway. Now, with massive success of social applications like Facebook or Twitter, we find ourselves in the era of social networks where user interactions moved from emails and private message boards into large-scale online social environments.

Virtual worlds (VW) are a particular kind of online social applications where users interact in a simulated computer-generated environment. Such environments are gaining a lot of hype with more than 1.7 billion virtual world accounts registered by the end of 2011 and a quarterly growth of 280 million new user accounts between the third and fourth quarter of 2011 (see Figure 1.1). Currently, the most popular virtual worlds are 2D and 2.5D virtual environments for kids between 5-15 years, making virtual worlds a new, vibrant market as the young generation is growing up in parallel with them.



Total Cumulative Registered Accounts

Age Range	2009				2010				2011			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
5 to 10	77m	114m	152m	179m	190m	211m	219m	235m	232m	270m	296m	340m
10 to 15	246m	334m	367m	392m	413m	444m	468m	511m	601m	652m	694m	787m
15 to 25	73m	99m	117m	193m	237m	273m	288m	299m	313m	385m	456m	596m
25+	18m	21m	23m	25m	27m	30m	34m	36m	39m	42m	44m	49m
Total	414m	568m	659m	789m	867m	958m	1,009m	1,081m	1,185m	1,349m	1,490m	1,772m

KZERO WORLDSWIDE

Figure 1.1: Number of virtual world users in Q4/2011

3D virtual worlds are a specific class of virtual worlds where participants experience a sense of presence by collaborating and interacting with others in the simulated 3D environment. 3D virtual worlds are distinct from other social applications in a way that VW participants can create and manipulate virtual world content.

Interactivity in virtual worlds is achieved using a multimodal graphical user interface, i.e., visual, audible, or textual, through which users receive multimodal clues from their environment with possibility to respond in the same manner. This interface allows users to interact with each other and collaborate on their common or individual tasks. A common scenario for a virtual world application, also considered in this thesis, is collaborative learning, which has proven to be an efficient way of knowledge comprehension in a modern networked era [Dillenbourg, 1999].

Interactions and collaborations in virtual worlds are possible through the use of *avatars*. An avatar is a graphical representation of a virtual world participant. In 3D virtual worlds, avatars are animated 3D characters that can represent anything from a talking fish to an exact real-world appearance of a participant. According to Liao, people tend to create their virtual alter egos that do not mirror their physical body, but reflect their imaginations and desires [Liao, 2008]. This creates a strong link between an avatar and a person it represents. Such link exists in both directions, when avatar behavior in the virtual world affects the behavior of a person in the real world [Yee, 2007].

Interactions using avatars are the key aspect of the subclass of virtual worlds, called *social* virtual worlds, where participants chat in 2.5D and 3D environments. Popular 3D chat platforms are IMVU[1], used mostly by teenagers, and WeeWorld[2], directed to younger kids. Other types of virtual world applications can be broken down into the following domains [Bartle, 2003] [Cavazza, 2007]: *gaming* - focused on online games (e.g. World of Warcraft[3]), *entertainment* - focused on multimedia and show business (e.g. UpNext[4]), and *business* - focused on e-commerce activities (e.g. Centre du Monde[5]).

Considering the successful deployment of VWs to business, social, entertainment and game fields, we believe that VWs can be deployed in more advanced scenarios such as e-* applications (e-commerce, e-learning and e-government) and social simulations, considered in this thesis. Current e-* applications are mostly web-based applications where stakeholders have no visual clues helping them to carry out their interactions. These applications have a limited possibility to deal with user interaction. This lack of multi-user and visual awareness in web-based systems can be handled by the 3D virtual world (VW) technology. Virtual worlds provide effective communication among participants and let them focus their attention on the who, where, or when of events. We are interested in the following scenarios:

- **Social simulations** with different levels of sophistication, from basic crowd simulations to advanced social dynamics simulations. Population of such simulations can form ethnic groups, which are closely integrated with their environment, and have the capability to evolve.

- **E-commerce** application where participants can engage in commercial activities in a 3D virtual world, what can improve his shopping experience over traditional online methods.

- **E-learning** applications where human participants participate in a 3D virtual world in order to achieve knowledge of a specific subject in an interactive way. For example, they can observe and interact with a population of a VW and learn about the context (e.g. historical society) in which this population acts and what it represents.

- **E-government**: We are interested in applications in which VW participants perform institutional actions in a structured way, what leads to the accomplishment of their specific goals.

In the context of **social simulations**, [Bainbridge, 2007] notes, "virtual worlds have a great potential as sites for research in the social, behavioral, and economic sciences, as well as in human-centered computer science".

[1] `http://imvu.com` (05/2012)
[2] `http://www.weeworld.com/` (05/2012)
[3] `http://us.battle.net/wow/en/` (05/2012)
[4] `http://www.upnext.com/` (05/2012)
[5] `http://www.lecentredumonde.com/galerie/` (05/2012)

[Chesney et al., 2009] showed how social 3D virtual worlds can effectively be used for performing in-world experiments. He performed five economical experiments in Second Life[6] (see Section 2.1.4 for details on Second Life virtual world) and compared the results to a study on human values that had collected data from several European countries with 1,500 samples from each country. He has found out that human values of Second Life residents are similar as those of a standard UK student sample; therefore, it can be used as a possible alternative for the conduction of certain types of social experiments. Virtual worlds are a natural candidate for such social experiments, either as a testbed prior to the actual test or as an experiment itself. In our work, we use virtual worlds in a cultural and social simulation of the historical city of Uruk (see Chapter 8).

In the **e-commerce** area, many business models rely on intensive internal and external collaboration. The success of these models is favored by the use of adequate communication tools. Nevertheless, current 2D and web-based tools, such as video conference, chats and forums, offer limited levels of interaction between stakeholders. An alternative to these approaches proposes to perform collaborative business activities using 3D virtual worlds, where an immersive participation can improve the experience [Bartle, 2003].

In the **e-learning** domain, traditional single-user e-learning systems seem to fail, as they do not address the social needs of users. These environments do not motivate users enough to explore all the possibilities of an e-learning system [Monahan et al., 2008]. Empirical research showed that collaborative and group work can help students to reach a higher achievement level [Laister and Kober, 2002]. Social collaboration among classmates is recognized as an important factor for the success of education [Joyce et al., 1980]. Virtual worlds can provide an immersive learning experience by combining the 3D visualization with the possibility to communicate in real-time, using chat, audio or even gestures. In particular, it helps students to sense belonging to a supportive e-learning community, which vitally improves interest and motivation to study the given topic [McInnerney and Roberts, 2004]. E-learning using 3D virtual worlds has been thoroughly evaluated [Bouras et al., 2001] [Bouras and Tsiatsos, 2006] [San Chee, 2001]. The majority of users found virtual environments intuitive and highlighted the social presence as the major advantage. Collaboration using audio and chat have also proven to be particularly useful. Overall, the feedback, both from professors and students was very positive and supported the usefulness of 3D paradigm.

The last e-* application type is **e-government**. E-government is defined as the use of ICT to improve the services of government [Layne and Lee, 2001]. Government has the opportunity to educate and interact with their citizens, as well as to promote new information using an entertaining and interactive way. This can be used to gain support for passing new bill, or educate people about the new changes in the political scene. Recently, American President Obama has already applied this approach and ran a part of his presidential campaign in

[6] `http://www.secondlife.com` (05/2012)

Second Life[7].

Such e-* applications often represent dynamic and complex processes, where frequently we need to create a new application scenario or change/adapt the existing one. An example from the e-learning domain is a history application explaining a historical subject in an interactive way. In such a scenario, we need to create a new intricate environment representing this historical subject. Another example can be found in an e-government application, where often we need to create new or change existing procedures for information processing by modifying existing specifications. In these examples, each new scenario requires a new 3D design of a virtual world, and often changes in the existing application specification require subsequent manual update of its design. The process of creation and modification of 3D design is complicated and tedious, where manual updates are not practical; thus, we seek for a more automated solution.

An automatic solution depends on Virtual World platforms (e.g. Second Life, OpenSim) allowing to generate and modify VW content programmatically, as well as on the possibility of automatically adding interactivity to the generated content, e.g., interactive 3D objects. Then, we can automatically generate a virtual world from the specification of activities defining an e-* application or a social simulation. In Chapter 4 we present the Virtual World Grammar mechanism that uses shape grammars [Stiny and Gips, 1972], a computational design technique, to generate many different VW designs out of a formal specification of activities.

The model generated by the Virtual World Grammar is static, with no possible interactions. This is not sufficient for e-* applications and social simulations as they depend on the interactivity between participants and the interactivity with the environment. Moreover, specific interactions have to be structured and controlled. An example is the control of a protocol execution in an e-government application, or the execution of an auction protocol in an e-commerce application. A popular method of such control is to use organization-centered or normative systems, which define roles and interaction protocols between roles containing norms and constraints, which structure participant behavior [Esteva, 2003]. In this thesis, we present a method of causally connecting an existing organization-centered system with multiple virtual worlds, allowing collaboration of their participants.

Although generating interactive, normative virtual worlds is sufficient for many applications, in the domains of our interest these worlds are lacking a world population. Populating the world with avatars is a key aspect of social simulations, allowing us to study participants' behavior. It is also highly beneficial for e-* applications, emphasizing the interactive and collaborative approach to task execution.

In many occasions, it is not practical to have humans controlling these avatars. This can happen when (i) we are simulating large crowds; (ii) when individual behavior can be automated; (iii) when execution of a task requires

[7] `http://thecaucus.blogs.nytimes.com/2007/03/31/obama-is-first-in-their-second-life/` (03/2012)

precise execution, and possible errors present a significant threat to a task execution. A popular substitution mechanism is to replace human presence with software agents, represented by their avatars. Agents acting in a virtual environment for the class of Intelligent Virtual Agents (IVA) and their distinction from other types of agents is that they try to mimic a human-like behavior.

Existing models for IVA focus on their execution within game scenarios [Gemrot et al., 2009] or sophisticated army scenarios [Silverman et al., 2006b] [Swartout et al., 2006]. Game IVAs such as Pogamut [Gemrot et al., 2009] have a basic, generic structure, which allows build custom add-ons, yet they are closely connected to existing game engines and game scenarios. Army IVAs possess highly complex features. They communicate in natural language, have human emotions and deal under stress. Complexity of model parts requires a high effort to re-configure application to a different scenario, usually taking months to accomplish [Swartout et al., 2006]. Thus, as existing models for IVA resulted inadequate for the purposes of e-* applications, there is a need to create a new model, that would be generic, lightweight, reusable, extensible and easily applicable to new scenarios. Using such model, we want to generate massive crowds for social simulations, where all crowd members have unique appearance and behave believably, that is human-like.

Considering presented motivations for an automatic generation of virtual world content and its population with autonomous agents, in this thesis, we present a new concept, named Virtual World Grammar (VWG) that can automatically generate a 3D virtual world model out of the specification of activities performed in the VW. A Virtual World Grammar can generate content for various virtual worlds; thus, we also present a mechanism that connects multiple worlds to the normative infrastructure allowing the participation of users from different virtual worlds in the same e-* application. Finally, we propose new methods to populate a normative VW with crowds of unique, believable agents.

We have presented the motivation for our research, as well as scientific and practical applications that this research provides, concerning the definition, control and execution of normative virtual worlds. Next, we present the research problem, and explain why we took related decisions to face challenges of this problem.

1.2 Research Problem

The initial research problem we faced was: how to facilitate a creation of a virtual world model, considering that this world represents an e-* application or a social simulation having a formal specification of activities performing in the application. Such applications are dynamic in their nature, thus we need an efficient way of reacting to changes in the specification, as well as to changes in the application state during runtime. Therefore, we decided to automatically generate the 3D virtual world from the formal specification of the application it represents. This problem can be addressed by fully automatic methods, such as rectangular dualization of planar graphs [Ancona et al., 2008]. In this approach, authors use

a specification of activities introduced in the form of planar graphs, to automatically these generate activities as rectangular spaces [Bogdanovych, 2007]. This approach, as well as any other fully automatic approaches, face the problem of tightly bonding the specification to the design of the virtual world (please also see Section 2.1.3 for details of rectangular dualization).

To address this problem, we seek to employ a semi-automatic process that would separate the conceptual specification of the application functionality from the creative design of a 3D virtual world. We found inspiration in works of [Gu and Maher, 2003] and [Smith et al., 2007], who used computational design and procedural architecture techniques, and introduced shape grammars to the generation process. Shape grammars [Stiny and Gips, 1972] were an excellent match for purposes of design generation, but existing work was not immediately suitable for our problem. Existing shape grammar solutions are normally restricted to very specific scenarios, do not normally address interactivity of the generated designs and rarely consider facilitating agent enactment of the generated environments as well as their normative regulation. Therefore, we designed a new shape grammar framework that included also our modified version of [Krishnamurti, 1981] sub-shape detection algorithm, which allowed real-time execution of recti-linear shape grammars. We implement this framework in the Shape Grammar Interpreter (SGI), presented in Chapter 3.

Furthermore, we needed a way to transform a design generated by the shape grammar into a 3D virtual world model, as the shape grammar generates a 2D floor plan which contains only basic geometrical data. We extended shape grammars with the possibility to add semantic information to shapes that the grammar generates. Moreover, we introduced several methods, such as validations and heuristics, which assured correct execution of the generation process. This effort resulted in the definition of Virtual World Grammar (VWG), which provides the possibility to generate a representation of multiple 3D virtual worlds out of a formal specification of activities performing in these virtual worlds. We implemented the Virtual World Grammar concept in a tool named Virtual World Builder Toolkit, which provides interfaces to define all VWG parts, and it is integrated as plug-in of Shape Grammar Interpreter.

The use of normative virtual worlds led to the next problem of the causal connection of a generated 3D virtual environment and the normative infrastructure. In this work, the normative infrastructure is represented by the Electronic Institution Execution Environment (AMELI) [Esteva et al., 2004]. Considering the vast amount of existing virtual world platforms, and the possibility of VWG to generate content for them, we set the goal to connect multiple virtual worlds with a single Electronic Institution This allows the participation of users from different virtual worlds in the same e-* application or simulation. The first challenge in this approach was processing of virtual world events, notification of Electronic Institution events and manipulation content in various environments. First, we needed a way to monitor, filter and process events coming from multiple virtual worlds. Second, for events coming from the normative runtime, we needed an efficient way to monitor and distribute these events to related virtual

worlds. Third, specific events require an update of connected virtual worlds, possibly with different architectures. First and second challenge was tackled by the Movie Script mechanism, proposed in Chapter 5. Incorporating Virtual World Grammar to the proposed Virtual Institution eXEcution Environment (VIXEE) completes the third challenge.

Having generated an interactive virtual world and connecting it with a normative infrastructure is still not sufficient for the application domains of our interest, as they miss the population. E-* applications stress the collaborative approach to task solving, while social simulations focus on a study of a population. In many cases, we desire to substitute human participants by software agents, e.g., actions can be automated. When substituting humans by agents, we require that agents act believably, human-like. Thus, the final problem we faced was to populate generated interactive spaces with believable virtual agents.

We have studied existing agents' models, e.g., [Swartout et al., 2006] [Silverman et al., 2006b], [Gemrot et al., 2009], and although they were inspiring, all of them proved to be inadequate for the purposes of applications considered by this thesis, due to their domain-specific aim, extensive complexity or lacking features. Thus, we set the goal to define a new generic agent model, appropriate for such applications, by combining approaches from Artificial Life and Intelligent Virtual Agents. We have done a rigid review of existing works in both fields, and collected information to face challenges during the definition of virtual agent model and its subsequent execution in e-* application and social simulations. The first challenge concerned the generation of a believable massive crowd, where each crowd member has a unique appearance, role and behavior style. This poses a problem for automatically generated agents having identical appearance. We solve this problem by storing agent properties into genetic structures and applying evolutionary mechanisms using a genetic algorithms based approach. The second challenge is in a believable, human-like behavior of agents. We approached this problem by adopting existing models for agent personality, emotions and physiology. The third challenge we faced was the requirement that in cultural simulations, agents carry cultural information, which affects their appearance and behavior. We solved this problem by adding the Virtual Culture [Bogdanovych et al., 2010a] information to the agent model. The final challenge was to integrate agents in the environment, what allowed them to reason and create plans depending on the current state of the environment without pre-encoding this data into agent behavior. We have separated agent definition from the definition of environment and addressed this problem by defining a Virtual World Object model carrying semantic information on interactive objects. As a result, we proposed a new generic agent model that defines sophisticated "digital organisms," represented by embodied, believable, autonomous virtual agents.

Figure 1.2 shows an overview of the approach presented in this thesis to the mentioned research problems. Virtual World Grammar (*VWG*) targets the problem of automatic generation of a virtual world design. *VIXEE* addresses the second problem of creating efficient causal connection between single normative

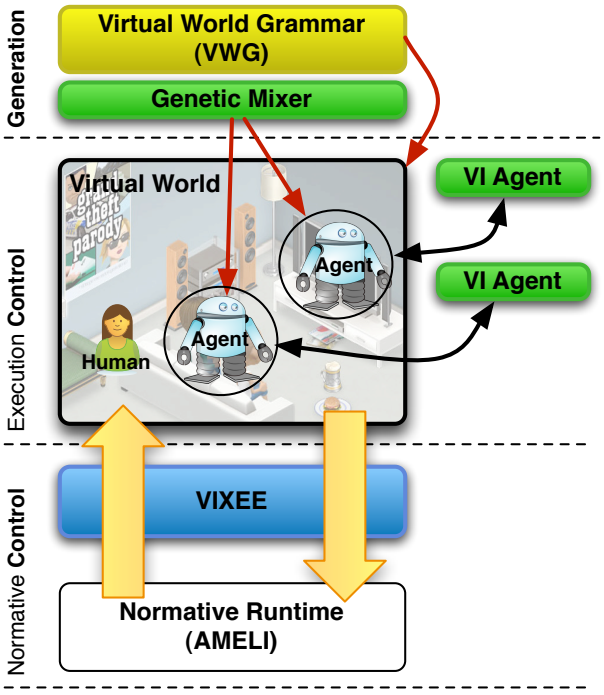


Figure 1.2: Overview of models for intelligent generation, execution and control of normative virtual worlds, presented in this thesis

infrastructure and multiple virtual worlds. The last problem was tackled by the *Genetic Mixer* and a new generic agent model, named *VI Agent*, which we have specifically designed for Virtual Institutions.

Summarizing all of the above challenges, we formulate the main research problem this thesis addresses:

Problem: *How to automatically generate and control normative, interactive, 3D virtual worlds, populated by humans and believable virtual agents?*

Next, we present our objectives for solving the research problem and describe the research method that we use to achieve these objectives.

1.3 Objectives

The aim of the thesis is to propose different models, techniques and methods for the automatic generation and control of interactive 3D virtual worlds populated by humans and autonomous virtual agents. We decompose this objective into a number of lower-level objectives, which target generation and control processes.

These objectives are:

- To design a new shape grammar framework having a real-time algorithm for sub-shapes detection, and its implementation in a general shape grammar interpreter.
- To develop a mechanism, named Virtual World Grammar, which provides the possibility to generate 3D virtual worlds from the formal specification of activities performing in the virtual space.
- To extend Virtual World Grammar with a general purpose agent model for intelligent virtual agents combining Intelligent Virtual Agents and A-Life features, acting in normative 3D virtual worlds representing e-* applications and social simulations.
- To develop an algorithm that allows the generation of avatars with unique appearance, while respecting the ethnic origin of their ancestors.
- To apply designed models, techniques and methods of virtual world generation, control and crowd generation to the domain of e-learning and social simulations.

1.4 Research Methods

To illustrate how the identified research objectives are addressed and how we solved the research problem, we again present the list of objectives, and identify corresponding research hypotheses and the research method selected to achieve each of the objectives:

- *To design a new shape grammar framework having a real-time algorithm for sub-shapes detection, and its implementation in a general shape grammar interpreter.* The research hypothesis associated with this objective is that it is possible to define a shape grammar framework having an efficient algorithm that detects sub-shapes in real-time, and that it is possible to successfully implement this framework in a general shape grammar interpreter. This hypothesis was validated by means of the creation of the Shape Grammar Interpreter (SGI) which implements a real-time sub-shape detection algorithm. This algorithm was evaluated according to its performance.
- *To develop a mechanism, named Virtual World Grammar, which provides the possibility to generate 3D virtual worlds from the formal specification of activities performing in the virtual space.* The research hypothesis associated with this objective is that it is possible to define a mechanism that provides the possibility to automatically generate a 3D virtual world from the formal specification of activities performing in this world. This hypothesis was validated by means of designing Virtual World Grammar mechanism, and implementing it in Virtual World Builder Toolkit (VWBT). We

implemented VWBT as a plug-in of SGI. This tool was used as a proof of concept by generating the representation of an e-auction application.

- *To extend the virtual world grammar with a general-purpose agent model for intelligent virtual agents combining Intelligent Virtual Agents and A-Life features, acting in normative 3D virtual worlds representing e-* applications and social simulations.* The research hypothesis associated with this objective is that it is possible to specify a general-purpose agent model for agents acting in normative 3D virtual worlds, which combines features from Intelligent Virtual Agents and A-Life. Agents from this model belong to a specific virtual culture, and their individual properties are encoded in their genes, allowing them to replicate and evolve. This model integrates agents closely with their environment, letting them automatically reason about their plans depending on observed data. This hypothesis was validated by means of defining the VI Agents model, and evaluating it in the context of a sophisticated social simulation, where VI Agents represented the population of the virtual city of Uruk. This simulation is running in Open Simulator and Second Life.

- *To develop an algorithm that allows the generation of avatars with unique appearance while respecting the ethnic origin of their ancestors.* The research hypothesis associated with this objective is that it is possible to automatically generate unique avatars while respecting the distinctive properties of parents' ethnics. This hypothesis was validated by mean of the definition of an algorithm that uses approaches from genetic algorithms. For a given base population, using such evolutionary approach, we can generate large crowds of unique avatars, respecting parents' ethnics. This algorithm was evaluated according to the diversity of generated avatars as well as to the preservation of distinctive visual features of parents.

- *To apply designed models, techniques and methods of virtual world generation, control and crowd generation to the domain of e-commerce, e-learning and social simulations.*The research hypothesis associated with this objective is that our proposed methods for automatic generation and control of interactive virtual worlds can be successfully applied to the domain of e-* applications and social simulations. This hypothesis was validated by means of applied explanatory research. The detailed literature review provided evidence and supporting information in favor of executing e-* applications and social simulations using virtual worlds and the need for the automatization of their definition and execution. The benefits of application of our automated methods were then illustrated in two different scenarios. First, we presented an e-auction application running in a virtual world, where using Virtual World Grammar we automatically generated and modified the world design during its execution. Second, we presented a social simulation of life in the virtual city of Uruk 3000 BC. This city is populated by our VI Agents, and operated using methods, models and techniques presented in this thesis.

1.5 Contributions and Significance

In this section we show the major contributions of this thesis and highlight the significance of these contributions in various areas of Computer Science.

1.5.1 Contributions

With this thesis, we made the following contributions:

- Defined a new shape grammar framework having an algorithm for real-time execution of sub-shapes detection and its consequent implementation in our Shape Grammar Interpreter.
- Designed an extension to shape grammar concept, called Virtual World Grammar, which allows automatic generation of a 3D virtual world from the formal specification of activities performing in the virtual space.
- Designed and developed a middleware layer model connecting multiple virtual worlds with a normative runtime infrastructure using our proposed Movie Script mechanism. This middleware is able to dynamically generate and manipulate the virtual world model depending on the current state of the normative runtime infrastructure.
- Designed a general purpose intelligent virtual agent model, called VI Agents, for agents executing in the context of normative virtual worlds, in our case Virtual Institutions. Each VI Agent is a member of a virtual culture, which can be disseminated within their environment.
- Developed an algorithm that generates avatars with unique appearance, while respecting the ethnic origin of avatar parents.
- Demonstrated how our models, techniques, and methods can be applied to the domains of e-commerce, e-learning and social simulations.

1.5.2 Significance

The research work presented in this thesis contributes to different fields of computer science: Virtual Worlds, Computational Design, Intelligent Virtual Agents, Artificial Life, Multi-Agent Systems, Simulations and e-* applications. Below we highlight the significance of thesis contributions for each of the aforementioned fields.

1. **Virtual Worlds:** In this thesis, we present different methods, models and techniques for automatic generation and control of interactive virtual worlds. Our proposed mechanism, named Virtual World Grammar, allows to dynamically react to changes in the specification, where each subsequent change only requires to re-generate the virtual world design, with no further manual modifications. Moreover, this mechanism allows to generate

many designs, letting a VW designer select the desired one. This dramatically cuts down the VW design and maintenance efforts. Targeting the control of interactions, VIXEE is a fast and reliable solution for simultaneously connecting different environments to the same normative runtime infrastructure, in our case an Electronic Institution. Thus, VIXEE allows developers to deploy one application to many virtual worlds. Concerning the presented general purpose virtual agent model, VI Agents can populate existing virtual worlds and use them to substitute humans for automated tasks, such as virtual guides, poll collectors, or to make virtual spaces more appealing for their visitors.

2. **Computational Design:** Shape grammars represent a prominent field in the world of computational design. Our shape grammar framework with a modified version of an algorithm for sub-shapes detection, and its subsequent implementation in our Shape Grammar Interpreter contributed to the world of shape grammars in four aspects. First, it can be used to efficiently execute rectilinear shape grammars with the use of sub-shape detection. Second, it becomes a teaching and presentation tool for shape grammar. Third, researchers working with shape grammars use SGI for experiments and to prove their concepts. Fourth, SGI can be used by industry, e.g., product designers or architects.

   Another contribution to the computational design field is our algorithm for the generation of agent appearance. Generated agents can be deployed not only in virtual worlds, but also in any graphical environment, such as games or movies.

3. **Intelligent Virtual Agents and Artificial Life:** Our proposed general-purpose agent model combines approaches from Artificial Life and Intelligent Virtual Agent fields. This model introduces a higher level of sophistication for A-Life "digital organisms." Agents implementing this model can be employed in e-* applications or perform sophisticated simulations with embodied, believable, cultural agents capable of evolution and adapting to a changing environment. This model is extensible, allowing researchers to adapt this model for their needs.

4. **Multi-Agent Systems:** In this thesis, we work with Virtual Institutions, which combine Electronic Institutions, an Organization Centered Multi-Agent System (OCMAS), and virtual worlds, to support human participation in MAS. Proposed VWG, VIXEE and VI Agent model facilitates the deployment and control of Virtual Institutions in e-* applications and simulations with a different level of sophistication. In these applications, humans can directly participate in Electronic Institution execution through their avatars, opening MAS to humans. The conducted study, literature review and developed prototypes present significant evidence in favor of applicability of our mechanism to these applications.

5. **Social Simulations:** VI Agent model is designed for deployment within

sophisticated social simulations to perform specific automated tasks. VI Agents carry virtual culture allowing their execution in complex cultural simulations. This model includes aspects supporting agent believability, such as psychology and physiology. To create a believable crowd of VI Agents, we take a genetic approach and encode agent properties into genes, generating agents with unique behavior and appearance.

1.6 Examples

In this thesis, we use two different examples to illustrate and prove our techniques, models and methods.

1.6.1 E-auctions

The first example is an auction system which allows both in-house users (bidders present in a real auction room) and internet users to participate in real auctions happening all around the world. This proved to be useful for specific types of auctions, like fish market auctions [Noriega, 1999], happening over a short time period, in the exact hour on the exact place. These types of auctions use extensive visual information for auctioneers, e.g., fish quality, that decide the final price of the auctioned item. However, how to accomplish the presence in all these places and achieve an effective and comfortable communication between in-house and internet users? Our answer is a hybrid environment which combines 3D virtual worlds, augmented reality and multi-agent system technologies. We generate an auction as a virtual space, either as a room in a big auction building or a separate building in the virtual world. All auction participants are displayed as avatars. Internet users move around the building and visit different auctions by entering auction rooms. In-house users are tracked either by cameras or some communication device, and their actions are constantly updated in the 3D representation. The auction system displays auction progress to every type of participant in the following format:

- In-House user sees an announcement board with the auction progress, or this information is displayed in custom glasses that he wears, allowing to augment reality.
- Web user sees a dynamic page control displaying the auction progress.
- Virtual world user observes the actual environment and the behavior of avatars to see the auction progress.

We use this example in Chapter 2 to explain Electronic Institutions concept and also in Chapter 4 to contemplate the automatic generation of the 3D virtual world representing the e-auction system. Using this example, we also present the possibilities of a dynamic manipulation of a virtual world model by adding and removing auction rooms depending on the current state of the auction system.

1.6.2 Uruk 3000 BC

The second example uses the 3D virtual environment developed by Anton Bogdanovych as a part of Authentic Interactive Re-enactment of Cultural Heritage with 3D Virtual Worlds and Artificial Intelligence [Bogdanovych et al., 2011]. In this project the author, using Second Life, recreated the life of the first city on earth, Uruk 3000 BC. In this example, we populate Uruk by VI Agents (Chapter 6). Agent appearance is generated using the algorithm described in Chapter 7, and their definition and execution is handled using VIXEE (see Chapter 5). VIXEE is also responsible for updating the virtual world content. A detailed specification of this example can be found in Chapter 8.

1.7 Structure

The remainder of this thesis is structured as follows:

Chapter 2 outlines the main components of the problem domain: virtual worlds, Electronic Institutions, Virtual Institutions, and their key characteristics. It also provides background on Shape Grammars, Intelligent Virtual Agents and Artificial Life and lists the relevant works in these domains.

Chapter 3 presents our algorithm for the real-time sub-shape detection during execution of shape grammars, evaluates its speed and complexity and describes its implementation in our Shape Grammar Interpreter (SGI).

Chapter 4 describes our Virtual World Grammar, a technique for the automatic generation of 3D virtual world content and presents results in an e-auction Virtual Institution.

Chapter 5 introduces a model of a communication infrastructure, named Virtual Institution Execution Environment (VIXEE) that allows the connection of multiple virtual worlds to a running instance of a single Electronic Institution. This chapter presents an evaluation of VIXEE according to its performance.

Chapter 6 presents our general purpose intelligent virtual agent model for agents acting in normative virtual worlds called VI Agent. This model is evaluated in Chapter 8.

Chapter 7 describes an algorithm for automatic generation of appearance of avatars, using approaches and techniques from genetic algorithms. This algorithm is evaluated by the variety of generated avatars and the preservation of ethnic features of parents.

Chapter 8 evaluates our work in a sophisticated simulation of the city of Uruk 3000 BC, running in OpenSim, a 3D virtual world platform. This simulation uses mechanisms introduced in this thesis.

1.8 Summary

In this chapter:

* We have introduced our research and presented the motivation to conduct it.
* We have stated the research problem and explained related challenges related to this problem.
* We have stated our key contributions of this research to various academic disciplines and showed how we advance the state of the art in those.
* We have presented two application examples that we use throughout this thesis.
* We have presented the structure of this thesis.

In the next chapter, we present the background on related subjects that we use in this thesis and may be unknown to the reader.

Chapter 2

Background and Related Work

The previous chapter introduced our research concerning intelligent generation and control of interactive virtual worlds. In this chapter, we present background information that helps in detailed understanding of concepts used in this thesis. Moreover, for selected concepts we provide the state-of-the art related to this research. We start with details on virtual worlds, and introduce their specific applications, where participant interactions must be controlled. Then, we present Virtual Institutions, which combine Electronic Institutions, a well-established Organization-Centered-Multi-Agent System, and virtual worlds [Bogdanovych, 2007]. We follow with a description of Electronic Institutions [Esteva, 2003]. Finally, we present the background and related work in A-Life and Intelligent Virtual Agents fields that inspired us to design our generic purpose agent model.

2.1 Virtual Worlds

Virtual worlds are a special class of online multi-user applications. They have been thoroughly described in many books and publications [Bartle, 2003] [Bogdanovych, 2007] [Messinger et al., 2008]. Their history, current use and future scientific potential are well documented [Bainbridge, 2007] [Messinger et al., 2009]. Thus, for readers unfamiliar with this concept, we provide a general overview of virtual worlds concept and then introduce only those parts related to our research. We start with a definition of virtual worlds according to [Bartle, 2003].

Definition 2.1. Virtual world (VW) is a computer based simulation environment through which participants can interact with each other and with objects in the environment.

Virtual worlds use graphics with different level of sophistication. Particularly,

(a) Poptropica (b) Betteverse

Figure 2.1: Children focused virtual worlds

2D and 2.5D[1] virtual world such as Poptropica or Betteverse are popular between kids (see Figure 2.1). This thesis is concerned with 3D virtual worlds.

Virtual worlds represent a broad class of online applications. These applications can be broken down into the following four domains (Figure 2.2) [Bartle, 2003] [Cavazza, 2007]:

- **Social** - focused on community.
- **Gaming** - focused on online games.
- **Entertainment** - focused on multimedia and showbusiness.
- **Business** - focused on e-commerce activities.

Interactions using avatars are a key aspect of the subclass of virtual worlds called **social virtual worlds**, where participants chat in 2.5D and 3D environments (see Figure 2.2 for virtual world types breakdown). Such virtual worlds are represented as simulated environments populated with avatars, where participants can change the look of the chat room, which represents their home or a clubroom. The limitation of such virtual worlds is that the user usually cannot move within his environment, or he can move only in a predefined way. One of the most popular 3D chat platforms is IMVU[2], used mostly by teenagers and WeeWorld for younger kids. Social platforms represent another type of social virtual worlds, which most resemble the classical social networks. Typical example of a social platform is Cyworld[3] where more than 90% of young South-Koreans interact on a daily base. Another type of social virtual world is an avatar-centric application such as NeoPets[4], where participants create, manipulate and share

[1] 2.5D application is a 3D application restricted to 2D plane, also known as fake 3D (see Figure 2.1b)

[2] `http://imvu.com` (05/2012)

[3] `http://us.cyworld.com/` (05/2012)

[4] `http://www.neopets.com/` (05/2012)

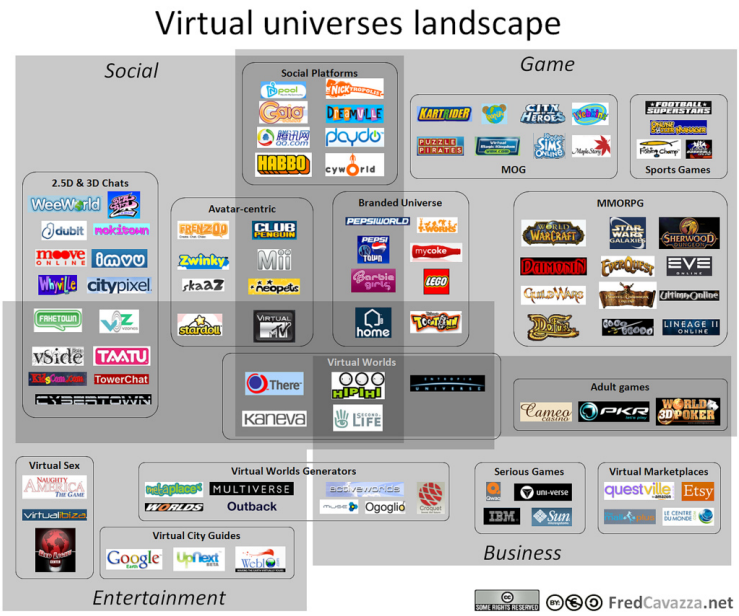


Figure 2.2: Breakdown of virtual worlds applications

their avatars with other participants. Similar to avatar-centric worlds are brand universes, e.g., Barbie Girls[5], where real world brands, such as Hasbro, create a specific virtual world that represents their virtual products creating a stronger bondage between participants and real-world company products, often allowing them to purchase these products directly from the virtual world environment.

Combining the social aspect of virtual worlds with entertainment forms the class of **entertainment virtual worlds**, using which participant can generate their own environments (e.g., Muse[6]), become immersed in virtual city tours (e.g., UpNext[7]), or engage in adult activities (e.g., Virtual Ibiza[8]).

Adding gaming possibilities on top of social interactions and virtual entertainment we obtain a class of **game virtual worlds**. These worlds represent many successful multiplayer online games (MOG), such as sport and gambling games and also very popular multiplayer online role-playing games (MMORPG), e.g., World of Warcraft[9] (WoW) and Runescape[10]. The WoW is (as of 05/2012) the most successful game VW with more that 11 million of registered users. Game VWs create enormous profits for their providers ($1 billion in year 2011 for Blizzard, provider of WoW). These profits come from subscriptions, online

[5] `http://games.barbiegirls.com/virtualworld/en/` (05/2012)
[6] `http://www.musecorp.com/` (05/2012)
[7] `http://www.upnext.com/` (05/2012)
[8] `http://www.virtualibiza.com/` (05/2012)
[9] `http://us.battle.net/wow/en/` (05/2012)
[10] `http://www.runescape.com/` (05/2012)

operations, e.g., gambling, as well as from virtual goods, that is content created within virtual worlds and exchanged for either real or virtual currency between world participants.

Profits themselves are a key aspect of the class of **business centered virtual worlds** where participants can engage and collaborate in business activities, such as virtual marketplaces, e.g., Centre du Monde[11]. In these virtual marketplaces, participants can browse a shop selection in simulated 3D environments.

Authors of Figure 2.2 separated the four just mentioned virtual universes (social, gaming, entertainment and business) into different subgroups but they named "virtual worlds" only those virtual universes that allow participants to create and manipulate their content. [Singhal and Zyda, 1999] summarized key aspects of virtual worlds that make them distinct from other types of computer applications:

- The ability to support multiple users differentiates virtual worlds from standard virtual reality and classical (single player) games.
- The ability to share and manipulate virtual world content (objects) differentiates virtual worlds from traditional chat rooms.
- The ability to support real-time interactions differentiates Virtual Worlds from email services and traditional web browsing.

Another important aspect of virtual worlds is their *persistence*. When a user leaves a virtual world it continues executing, allowing the participation of other users. Furthermore, 3D virtual worlds provide an immersive experience for their participants. They have a "sense of space," they can walk around, collaborate and interact with other participants and the environment.

Participants act in virtual worlds through use of *avatars*, their graphical representation. Sometimes it required to automate actions of a human participant, thus we can substitute his presence with an automated counterpart. In games, these automated participants are called *bots*, while in virtual worlds and the text of this thesis, we refer to them as *agents*.

Next, we present two aspects of virtual worlds related to this research. First, we explore the world of avatars. Second, we take a look at content creation in virtual worlds, explain how it can be used to fulfil the objective to automatically generate virtual world content. Then, we present several other works related to this objective. Finally, we introduce Second Life virtual world and an open-source virtual world platform called OpenSimulator, and explain their advantages over other existing virtual worlds and platforms.

2.1.1 Avatars

"Avatar" is a very popular word nowadays, used by the movie industry, computer games and online applications. Although, the meaning of the word avatar in

[11] `http://www.lecentredumonde.com/galerie/` (05/2012)

these fields differs, in general we can refer to it as one's *alter ego*. Historically, it is derived from the Sanskrit *avatra*, meaning "descent". Avatar first appeared in English in 1784, as a Hinduistic term for a material manifestation of a deity, which is earthly incarnation of higher gods [Bartle, 2003].

In the multimedia field, the term avatar as the on-screen representation of the user was coined in 1985 during designing LucasFilm's online role-playing game Habitat. In the computer magazine Run, issue from August 1986, Margaret Morabito wrote, "Once a human being enters Habitat, he or she takes on the visual form of an Avatar, and for all intents and purposes becomes one of these new-world beings." Neil Stephenson, in his breathtaking cyberpunk novel "Snow Crash" [Stephenson, 2000] introduced the word avatar to the wide public. In his own jargon, he called his virtual world "the Metaverse" and its digital inhabitants "avatars." In 2009, James Cameron created the famous sci-fi movie Avatar, where avatars are genetically engineered Na'vi-human hybrid bodies, which a team of researchers uses to interact with the natives of Pandora.

(a) Human-like avatar (b) Fish avatar

Figure 2.3: Second Life avatars

In this work, *we refer to avatar as the graphical representation of a user in a virtual space*. This meaning is also the most common between Internet users. [Liao, 2008] makes connections between avatars and a person's identity in the real world; he wrote: "An avatar does not usually fully represent a person, but rather, represents an alter ego or pretend persona. The body images they create usually do not mirror their physical body but are accumulations of imagination and desires." This process is sometimes two sided, when avatar impersonation affects the appearance and the behavior of a human user, called the Proteus effect [Yee, 2007]. Levine mentioned the case of a shy, 40-years old teacher, who played the role of necromancer (master commander of the dead) in the popular online game called *Everquest*[12] and while playing: "she began to become much bolder, stronger and more assertive as a result of playing this character, and she was able to carry that over into her real-life interactions[13]." Levine also noted that sometimes the process of such impersonation can be scary for users: "I

[12] `http://http://www.everquest.com/` (01/2012)

[13] `http://www.npr.org/templates/story/story.php?storyId=12263532` (05/2012)

Figure 2.4: Avatar Design in Second Life

played one of these games for two days once, and it scared the hell out of me. I made the decision never to play one again. Why? Because, I would end up just vanishing into it!". As a conclusion, we can state that there exist a deep link between users and their avatars; therefore, selecting the graphical representation of their avatars is particularly important for them.

In 3D virtual worlds, users are represented by their 3D avatars. A 3D avatar is an animated, emotive, complex model representing anything from actual resemblance of the human user (see Figure 2.3a) to a sharp-teeth, talking fish (see Figure 2.3b).

Nowadays, there exist the possibility to generate your 3D avatar. SitePal[14] allows web site owners to add an interactive 3D character to their web pages. Such character, either real-life like or cartoon-like, actuates as a conversational agent, which interacts with web-site visitors. There also exist many avatar-centric virtual worlds, such as NeoPets[15], which allow users to create their avatars, share them with friends and even order a real-life toy with the look of their avatar.

To create a 3D avatar is usually an application dependent process, where a user can create his avatar for a specific game or a virtual world, making it impossible to transfer the avatar to other applications. Figure 2.4 shows one of the most sophisticated avatar appearance editors, which can be found within Second Life (see Section 2.1.4). Second Life users can select an avatar from a rich variety of predefined avatars and modify different parts of their body. A user can also define and modify custom body parts, (e.g., head, torso, eyes), as well as clothes (e.g., skirt, pants, socks). Moreover, different textures can be applied

[14] `http://www.sitepal.com/` (05/2012)
[15] `http://www.neopets.com/` (05/2012)

to both skin and clothing, as well as different accessories can be attached to an avatar's body (e.g. hats, earrings, handbags). Business with avatar accessories creates a substantial part of Second Life's economy, which is based on a custom currency, the Linden Dollar.

In this thesis, we present a genetic based mechanism that can produce new avatars independently of the virtual world considered. Moreover, regarding crowd simulations in virtual worlds, there is a lack of approaches for automatic generation of a large crowd of unique avatars. This genetic mechanism tackles also this problem.

2.1.2 Virtual World Content Creation

In the introduction chapter, we have described the objective to automatically generate 3D virtual worlds from the specification of activities taking place in this world. We are able to accomplish this task in those virtual worlds that allow users to create and manipulate their content. Even though this is a defining feature of virtual worlds, many of them lack this capability, or it is limited (e.g. possibility of manipulating only a predefined set of objects). A recent research study revealed that due to the possibility to create content, virtual worlds have the potential to become one additional environment, like school, home, and the playground, where youth can learn, play, and grow [Beals, 2010].

Figure 2.5: Content manipulation in Second Life

Apart from educational benefits, content creation opens the door to virtual economies by selling the created content to other participants. This emerging economy raised $3 billion in 2010 with the planned increase to $12 billion in 2012 [Mitham, 2010]. This content includes not only avatars and avatar accessories, such as clothing or different hair designs, but also whole environments with buildings and their equipment. Monetization of virtual objects brings massive profits for virtual world providers; thus, there is an increasing effort to include

this possibility in many other virtual worlds. Although, due to the strong complexity of the implementation of such a feature, only existing content can be bought from the virtual world provider.

Each existing virtual world handles content creation in its own way. Advanced content manipulation features can be found within Second Life, which provides a virtual land (island) and a set of world-specific tools to create and manipulate world objects and avatar appearance. Content can be created either on privately owned land (for a fee) or in open environments, called sandboxes. Figure 2.5 shows a Second Life participant manipulating virtual world content. In past years Second Life users were limited to create their content combining "prims", simple 3D objects such as a box or prism. Each user can use only a limited number of prims, or he can pay an extra fee to use more prims. Nowadays, users can import complex 3D meshes and use them to design their islands.

Nevertheless, automatic generation of a virtual world depends on the possibility of automatic manipulation of its content. In Second Life, and OpenSimulator, content can be manipulated using OpenMetaverse[16] library. In Open Wonderland[17] virtual world content is managed with JMonkeyEngine[18]. These libraries allow not only to interact and manipulate with world objects, but also to control and interact with their participants. Thus, we can use these libraries to generate a virtual world design and to control avatars in the virtual world, providing functionality to put agents in play.

In the next section we introduce several works that tackled the problem of virtual world design generation. They are related to our objective of automatically generating virtual world content.

2.1.3 Related Work: Automatic Generation of a Virtual World Design

Although virtual architecture represents a separate field of study, it has similar concerns as virtual worlds design. Maher et al. have explored in different works the design of virtual worlds [Maher et al., 2000] [Lou Maher and Gu, 2003] [Gu and Maher, 2004] [Lou Maher et al., 2005]. Authors approached the design from the functional point of view, where each design element serves a specific function. Thus, it can be operated by a special design agent, which depending on current preferences of activities, updates the virtual world design. [Gu and Maher, 2003] and [Smith et al., 2007] presented the possibility of using shape grammars to generate virtual world design, what inspired us to create a Virtual World Grammar (see Chapter 4).

Considering existing approaches in the automatic generation of a virtual world design from a formal specification, [Ancona et al., 2008] and [Bogdanovych, 2007] generated a 2D floor plan of a Virtual Institution (see Section 2.4 for details on Virtual Institutions) from the conceptual model described

[16] `http://openmetaverse.org/` (04/2012)
[17] `http://openwonderland.org/` (05/2012)
[18] `http://jmonkeyengine.com/` (05/2012)

in its performative structure. This approach used rectangular dualization of biconnected planar graphs. For this purpose, OCoRD software was developed. Problems, related to this approach, include (i) close binding between the specification and the generated floor plan (ii) problem with scaling of sizes of generated spaces, and (iii) difficult navigation in such generated spaces. Figure 2.6 contemplates these problems showing the generated floor plan for a simple virtual world with three activities, i.e., Trade Room, Meeting Room, Registration Room. Spaces labelled with T1-T5 are transitions that represent intermediary states between activities. We can imagine that for a virtual world with many activities, its automatically generated layout becomes confusing for participants.

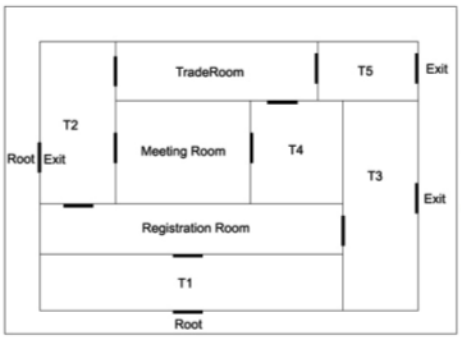


Figure 2.6: Generated floor plan using rectangular dualization

In the thesis, we explore an alternative to this approach using shape grammars, a computational design, rule-based technique (see Section 2.5 for details on shape grammars). The use of shape grammars allows much more freedom in the VI design, and it is possible to generate many different functional designs with limited additional cost. We do not generate transitions as separate rooms (as in OCoRD) but we map all transitions to a hallway. In this way, we can generate floor plans of buildings with much simpler navigation for the user. Furthermore, our system contemplates not only a generation of the 2D layout but also a complete 3D scene.

Our approach is similar to [Lou Maher and Gu, 2003], which used shape grammars to generate a floor plan of a virtual world. In their case, a design agent was responsible for manipulating virtual world content. [Duarte, 2001] used shape grammars to generate Siza's Malagueira houses using an online application that rendered house designs depending on user preferences. Duarte introduced the concept of discursive grammars that contain a shape grammar, a description grammar and a set of heuristics. While Duarte generated house designs from the discrete set of user preferences, we generate virtual worlds from a formal specification of any number of activities performing in the virtual world. Also, Duarte used markers and a descriptive grammar to pertain semantic data in generated forms, while we use user-defined ontologies to define concepts de-

scribing these forms.

Other works, such as VRID (Virtual Reality Interface Design) [Tanriverdi and Jacob, 2001] and VEDS (Virtual Environment Development Structure) [J.R. et al., 2002] presented methodologies that facilitate the designer's task either by dividing the design in high-level and low-level phases or guiding him in taking design decisions to get a usable virtual environment. A conceptual model of a virtual environment was presented by Ossa [Southey and Linders, 2001], the model considered conceptual graphs and rule based systems that were too complex to be managed by designers. i4D is another methodology based on the representation of conceptual models. This methodology contributed with a thin abstraction layer taking into account only a small space of the domain knowledge [Geiger et al., 2000]. Compared to the approaches in this thesis we provide a high level abstraction layer by means of the Virtual World Grammar, which enclosed both data and processes related to the 2D and 3D generation of designs.

VR-WISE system and Ontoworld tool have focused on the gap between the abstract model and the implementation prototype by proposing an approach to generate VW from high-level descriptions given by ontologies [Troyer et al., 2003] [Mansouri et al., 2009]. Objects in the domain and their relationships are described in a so-called domain ontology. The domain ontology is converted into a representable domain ontology which describes how objects in the domain can be represented in the virtual environment. As a main difference with our approach, the domain ontology does not have all the information needed to generate the 3D virtual world whereas our approach has it. We can generate the whole virtual world layout and situate 3D objects there while VR-WISE system situates objects in an already generated virtual scene. Currently, our mechanism supports the design generation for Open Wonderland[19] and Second Life[20] virtual worlds and also for OpenSimulator-based virtual worlds[21].

2.1.4 Second Life Virtual World and Open Simulator Platform

Second Life (SL) is one of the most popular 3D virtual worlds with more than 28 million sign ups since its launch in 2003 to this date. It can be considered as a massive technological and social experiment, where SL participants, called residents, interact in virtual environments called simulators (also called sims). Each sim represents a virtual land owned by some resident. It can host a limited number of residents and display a limited number of objects (called prims). Users can teleport between sims. Figure 2.7 shows the map of the SL universe, where each square represents one sim. Figure 2.8 displays a SL environment, with participating residents.

Residents explore the world by walking, running or even flying and teleport-

[19] `http://openwonderland.org` (05/2012)
[20] `http://secondlife.com` (05/2012)
[21] `http://opensimulator.org` (05/2012)

Figure 2.7: World map of Second Life

Figure 2.8: Second Life

ing. They can talk to other residents either via chat or by voice and interact with objects in the environment (e.g. touch, pick up and attach an object to their body). The distinguishing feature of Second Life is that residents can shape their environment according to their needs, particularly on their property. Thus, Second Life can be considered as a platform, and all its content is created by users. Content creation and environment interaction is protected by object and property rights. Residents can build objects, make them interactive (using a scripting language called Linden Scripting Language, LSL) exchange or sell these objects. They can manipulate their appearance, purchase or trade clothes and apparel, collaborate with other users, explore, work, or just have fun.

Second Life has its own economy system based on Linden Dollars, which can be exchanged with real dollars. Monetization of virtual objects and property in SL gained a lot of media attention when the SL resident Anshe Chung cashed out $1 million for her virtual real estate and virtual stock market investments[22]. Such success of a virtual economy draws attention of real-world businesses looking for fresh possibilities of their expansion and new forms of company marketing.

In the e-commerce domain, we have seen companies like IBM, Toyota, Honda and Sears presenting their products in SL. According to [Messinger et al., 2009] there is a strong opportunity for real-world companies to enhance their brand image or create higher levels of brand recognition by establishing their virtual presence in 3D environments such as SL. A survey revealed that users that shopped in a SL store are more likely to shop in the associated store in the real-world. Although, considering that the majority of SL residents visit the virtual world for social purposes, and wish to spend little, or nothing to modify their appearance, the success of full-time businesses is questioned, what resulted into recent closing of American Apparel store in Second Life. This does not degrade the importance of SL for e-commerce domain. According to the business analysis done by [Footprint, 2007], virtual worlds provide an environment for numerous business applications and opportunities, such as (a) laboratory market research, (b) test market, (c) large market for advertising, (d) retailing center and (e) brand recognition and marketing. With more existing applications and more possibilities, Second Life attracts still more users positively affecting the existing and emerging virtual businesses.

The vast number of Second Life residents has affected the decision of many real-world organizations to open their branches in Second Life as their message can be distributed to many potential clients, sometimes with "the right kind of media attention" (e.g. Obama's speech in Second Life[23]). We can find several government embassies in SL, including Maldives, Sweden and Estonia. Furthermore, religious organizations created their virtual disciples, when the christian church of Oklahoma opened its 12th campus, the first one in the virtual world. In 2007, radio Islam Online bought a virtual property where residents could perform the ritual of Hajj.

In the e-learning domain, Second Life presents a powerful educational platform for interactive and collaborative learning. This type of learning can help where traditional learning processes fail. The traditional learning model for interactive learning is the four-pronged model developed by [Kolb et al., 1974]: concrete experience, reflective observation, abstract conceptualizations and active experimentation. Collaborative learning is particularly popular between young generation that grew up in constant contact with digital media, named by [Tapscott, 1998] Net Generation or Generation Y. Members of this generation, also known as *digital natives*, seek every opportunity to use digital media to facilitate their education. According to [Steinkuehler, 2004] virtual worlds fit the needs of digital natives as they "use the scientific habits of mind better than

[22] `http://www.anshechung.com/include/press/press_release251106.html` (05/2012)
[23] `http://www.cbsnews.com/8301-503544_162-5151594-503544.html` (05/2012)

traditional teaching methods by letting participants to experiment with unfamiliar alternatives, rationally calculate problem outcomes, and develop complex reasoning about the subject of study". In this thesis, we implement an e-learning scenario, and recreate life in the mesopotamian city of Uruk. Exploring Uruk, students can better understand habits, values and history of the ancient Sumerian culture.

Apart from e-learning, Second Life provides excellent research possibilities in multiple scientific fields. Considering the number of users of different population groups and profiles, SL is especially well designed for formal experiments in social science and cognitive science. Researchers can construct virtual laboratories that can automatically recruit potentially thousands of research subjects. Also, examining the existing social networks and economic systems, we can extract diverse data. For creative scientists, virtual worlds provide a powerful testbed, before their experiments can be applied to real life, such as comparing socioeconomic consequences of alternate government regulations [Bainbridge, 2007]. Apart from social sciences, Bainbridge points out possibilities of virtual worlds such as Second Life in education, history, humanities and even sexuality.

Considering the vast number of applications that have been used with Second Life, we conclude that this virtual world platform fits well for the e-* applications and social simulations concerned by this thesis. For practical reasons, we have decided to use Open Simulator, an open-source counterpart of SL. Open Simulator provides an excellent testbed for applications of our concern, before they will be deployed to Second Life or other virtual worlds.

Second Life has many complex features but it misses a fundamental characteristic to the deployment applications with controlled interactions. The "runtime" control of participant interactions is limited to the "terms of service," which is a document that contains what the user agrees to comply with. For the application domains of our interest, we need a way of structuring and normatively control participant interactions, what allows to implement different procedures for e-* applications or even specify scenarios for social simulations. For this purpose, we employ Organization Centered Multi-Agent Systems presented in the next section.

2.2 Virtual Worlds as Hybrid Multi-Agent Systems

In the previous section, we have introduced virtual worlds and its potential to deploy e-* applications and social simulations. For these applications, it is often practical to populate the virtual space both with humans and agents, this is the reason we use the term hybrid. In this context, agents can be used to automatize human tasks. Agents who act in virtual spaces are called autonomous, intelligent virtual agents. Formally:

Definition 2.2. An **Autonomous Intelligent Agent** is a software agent (i.e. a software program which acts on behalf of a user or other program), which

observes through sensors, acts upon an environment using actuators and directs its activity towards achieving goals [Russell et al., 1995].

For some authors, an agent being autonomous is more important than being intelligent; therefore, often they are referred to as "autonomous agents" rather than "intelligent". Agents differ widely in their architecture. An example is a *reflex agent*, which reacts to a world state depending on a set of rules. According to [Russell et al., 1995], there are other types of architectures. In this thesis, we are concerned with a combined architecture of *goal-based agents* and *learning agents*. *Goal-based agents* take decisions depending on the information stored in the model of the observed environment and agents' current goal. Goals allow agents to choose among multiple possibilities of actions. *Learning agents* start initially in an unknown environment, and during execution they observe the environment, and adapt their actions depending on its current state. The advantage of this approach comes with the usage of a learning element, which rewards or punishes agents when performing their actions, thus allowing agents to adapt and improve.

There exist different agent models depending on their field of application, e.g., semantic web agents, mobile agents, interface agents. In this thesis, we are concerned with agents that act in virtual environments, called intelligent virtual agents (also see Section 2.7). Formally:

Definition 2.3. An **Intelligent Virtual Agent** (IVA) must perceive the world in which it exists, both virtual and real, often including human participants' natural language and gestures. It must reason about those perceptions as well as decide on how to act on them in pursuit of its own agenda [Marsella and Badler, 2011]

While sometimes it is enough to deal with a single instance of an agent (e.g. inteface agent), we are interested in systems that can handle and coordinate many agent instances, called multi-agent systems.

Definition 2.4. Multi-agent systems (or MAS) are systems in which several interacting, autonomous agents pursue some set of goals or perform some set of tasks [Weiss, 1999].

In many cases, MAS represent systems needing only agent-agent interaction. Virtual worlds represent an excellent environment for human-agent interaction. Then, Multi-agent Systems are used to specify when and how should participants (i.e. humans and agents) interact. There are two approaches to this specification. *Agent-centered approach* focuses on searching for agent-level capabilities, while *system-centered approach* deals with searching for group-level rules, such as conventions or norms [Weiss, 1999]. We are interested in both approaches, where autonomous agents reason and adapt their actions according to a set of specified regulations, i.e., norms, constraints and conventions. The specification of regulations requires the use of formal methodologies and formal languages. A class of MAS that employs these methodologies and languages is called Organization Centered Multi-Agent Systems (OCMAS) [Ferber et al., 2004]. An

OCMAS is designed relying on organizational concepts such as roles, tasks (activities), interaction protocols and norms.

A Normative Virtual World is a virtual environment where interactions are controlled by an Organization Centered Multi-Agent System ("Normative" word comes from the earlier use of the Normative MAS for interaction control, currently better known as OCMAS). In this thesis, we employ Virtual Institutions [Bogdanovych, 2007], which are a specific class of Normative Virtual Worlds that combines strengths of 3D virtual worlds and Organization-Centered Multi-Agent Systems, in particular, Electronic Institutions [Esteva, 2003]. 3D virtual worlds are responsible for audio-visual presentation, and the Electronic Institution enables the formal rules of interactions between participants. In the next section, we present details on Electronic Institutions as its knowledge is substantial in understanding Virtual Institutions.

2.3 Electronic Institutions (EI)

An *Electronic Institution* is an Organization-Centered Multi-Agent System that structures agent interactions, establishing what agents are permitted and forbidden to do as well as the consequences of their actions [Esteva, 2003]. A more complete definition is provided in the following [Rodriguez-Aguilar, 2001]:

Definition 2.5. Electronic Institutions (EI) are software systems composed of autonomous entities, i.e. humans or autonomous agents, that interact according to predefined conventions on language and protocol and that guarantee that certain norms of behavior are enforced. This view permits that participants of Electronic Institutions behave autonomously and make their decisions freely up to the limits imposed by the set of norms of the institution. An Electronic Institution is in a sense a natural extension of the social concept of institutions as regulatory systems which shape human interactions.

In general, an Electronic Institution regulates multiple, distinct, concurrent, interrelated, dialogic activities, each one involving different groups of agents playing different roles. It is defined by a *dialogic framework*, a *performative structure*, *scenes* and *norms*. An EI is executed in a runtime infrastructure called AMELI.

Following sections provide details on all parts of EI using the "e-auction house" institution example introduced in the Section 1.6.1.

2.3.1 Dialogical Framework: Roles and Ontology

Communication between participants of Electronic Institutions is conducted through speech acts [Searle, 1969]. In general, a speech act consists of two components (1) a performative verb and (2) a propositional content. A performative verb explicitly contains the type of the action a "speaker" is trying to achieve by pronouncing the propositional content. To illustrate how this differs from human communication consider the following two speech acts: *request("window*

opened") and *inform("window opened")*. Although the propositional content is the same, "window opened", a speaker expects different results for each action. Specifying a unified language for such speech acts (by means of an ontology) allows communication between agents with otherwise unrelated architectures.

In the terminology of EI, speech acts are called *illocutions*. Each *illocution* consists of an *illocutionary particle* (a performative verb) and a message content (propositional content). The dialogic framework establishes the acceptable illocutions by defining an ontology (vocabulary) - the common language to represent the "world" and for communication and knowledge representation.

The dialogic framework also defines participant roles and their relationships. Each one of EI participants is required to adopt a role defined in the dialogic framework. For each defined role, it is possible to specify a list of *role properties*. Each role property is defined by (i) name, (ii) type, (iii) default value and a (iv) flag if this property is required.

Figure 2.9 shows all the components of the dialogical framework of the e-auction institution. This screenshot is taken from Islander tool, which allows the visual definition of Electronic Institution components [Esteva et al., 2002]. In the right part of this figure, we see a role hierarchy with main roles of guest and staff and their sub-roles buyer, seller and auctioneer. Staff and auctioneer are internal (system) roles always executed by autonomous agents. Buyers and sellers are external roles, which could represent human users or also autonomous agents. We see a *ssd* relation between roles of guest and staff and buyer and seller. Acronym *ssd* stands for *static separation of duty*, which means, that a user can only take one of these roles in the system and when decides to participate with one role, cannot change to the other. In the left part of this Figure, we see the properties of the *buyer* role. Each agent participating in the system with a given role obtains a copy of the role properties, and these properties become part of the agent state. These properties change depending on the agent's actions within the system.

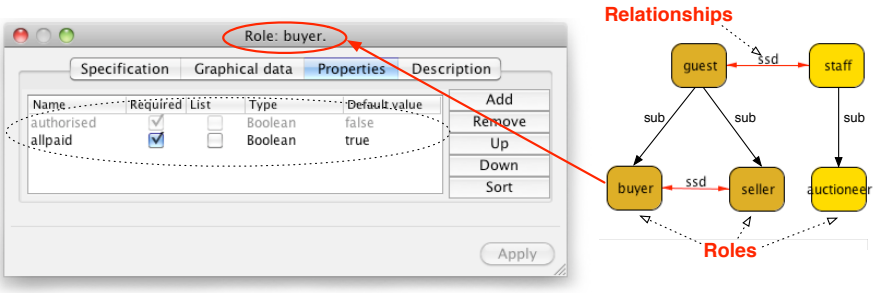


Figure 2.9: Dialogic framework components

Dialogic framework also contains a definition of the ontology and illocution particles. For the e-auction institution, we define *failure*, *inform* and *request*

particles. Figure 2.10 shows the ontology for the e-auction institution. On the left side, we see a list of functions and complex types that these functions use.

Complex types represent structures composed of atomic types, e.g., string, integer, and other defined complex types. In the e-auction example, Item and ItemList represent complex types.

Functions form a propositional component of the illocution. On the right side of Figure 2.10, we see the detail of the function *registerItem*. Sellers register auctioned items by specifying a name, description and initial price for the auctioned item. Thus, *registerItem* has three parameters, two strings and a float for specifying item details.

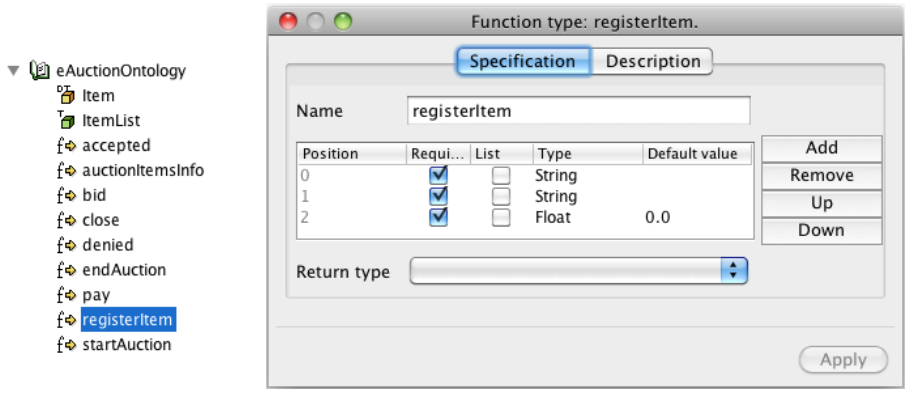


Figure 2.10: e-auction institution ontology and function detail

2.3.2 Scene

Scenes (also called Activities) represent agents group meetings that follow well defined communication protocols. All the interactions between agents at the institutional level happen only in scenes. The scene protocol represents the possible dialogic interaction between roles instead of agents. We can say that these protocols are patterns of multi-role conversation. The distinguishable feature of the institutional scene is that an agent can enter and leave the scene only in a particular moment, in a specific context, and only when he fulfils all his obligations.

Figure 2.11 shows the simplified English Auction scene protocol from the e-auction institution. In this protocol, buyers and auctioneer enter the scene in the START state, indicated by symbol "+buyer" and "+auctioneer" in the square above the START state. Buyers can exit before the auction has started, indicated by the "-buyer" in just mentioned square. Then, the auctioneer starts the auction and scene execution moves to state W1. In this state, buyers place their bids. Bidding imposes restrictions. Figure 2.12 shows the Islander window

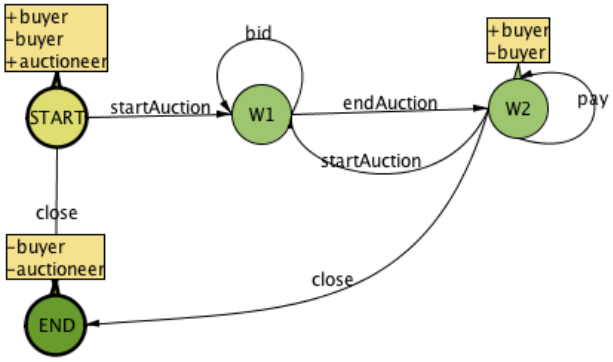


Figure 2.11: Scene protocol

where we define properties of bidding arc (recurring arc with label "bid" in the state W1). In the left part of the figure, we see basic arc properties, where we can define arc label, constraints, arc action and actions.

Figure 2.11 show the simplified English Auction scene protocol from the e-auction institution. In this protocol buyers and auctioneer enter the scene in the START state, indicated by symbol "+buyer" and "+auctioneer" in the square above the START state. Buyers can exit before the auction has started, indicated by the "-buyer" in just mentioned square. Then, auctioneer starts the auction and scene execution moves to state W1. In this state buyers place their bids. Bidding imposes restrictions. Figure 2.12 shows Islander window where we define properties of bidding arc (recurring arc with label "bid" in the state W1). In the left part of the figure, we see basic arc properties, where we can define arc label, constraints, arc action and actions.

- **Constrains** impose context-specific restrictions on agents which can perform the action specified in arc action. For bidding arc, the restriction states that a new bid has to be higher than the last bid and that the same bidder cannot bid twice in a row.

- **Arc actions** define the illocution or time out. The right side of Figure 2.12 contemplates the Islander window, where we define details of the illocution where a buyer informs all participants of a bidding scene that he is bidding a specific price.

- **Actions** define how context changes after arc action was performed. In the case of auction protocol, we record the last bidder and the last price.

Buyers continue bidding until there are no new offers. After buyers finish bidding, the auctioneer ends the auction, and scene protocol moves to state W2. In this state, all buyers can leave the scene, except for the buyer that won the

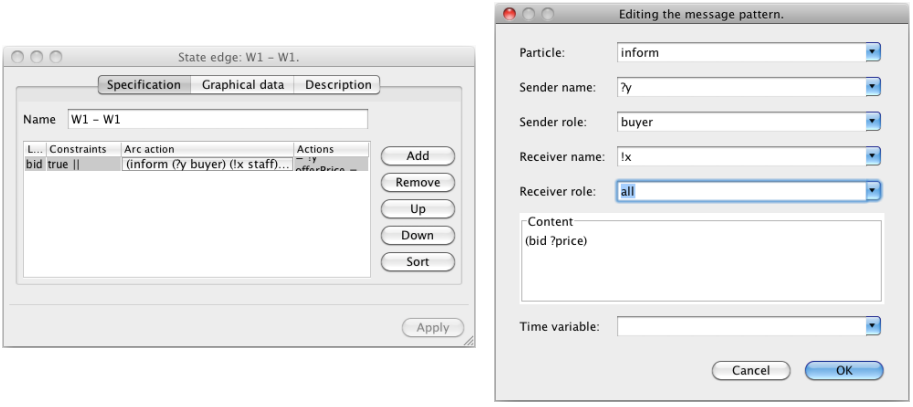


Figure 2.12: Details of the arc "bid" from the scene protocol in Figure 2.11

auction. This contemplates the state detail window in the Figure 2.13, where (in the area marked with the red ellipse) we see the constraint for the buyer exiting the auction. In order to leave the auction, the winning buyer has to pay for the product. When the buyer pays for the product, the auctioneer closes the auction and everyone leaves.

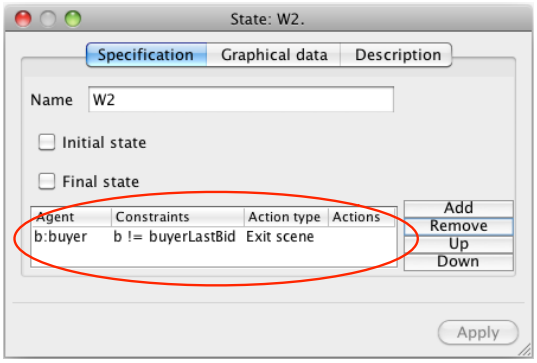


Figure 2.13: Scene protocol

2.3.3 Performative Structure

More complex activities can be specified by establishing networks of scenes (activities) in so-called performative structures. These define how agents can legally move among different scenes (that is from activity to activity) depending on their role. Furthermore, a performative structure defines when a new scene execution starts, and if the scene can be executed multiple times. A performative struc-

ture can be regarded as a graph whose nodes are both scenes and transitions (scene connectives), linked by directed arcs. The transition type allows to express choice points (*Or* transitions) for agents to choose which target scenes to enter, or synchronization/parallelization points (*And* transitions) that force agents to synchronize before progressing to different scenes in parallel. Labels on directed arcs define which roles can progress from scenes to transitions or from transitions to scenes.

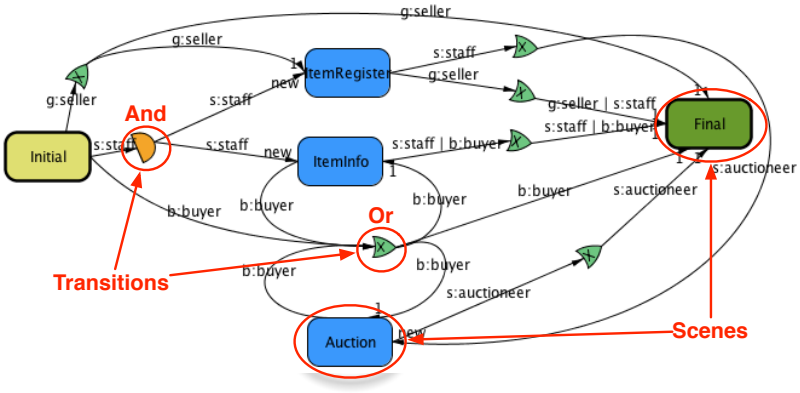


Figure 2.14: Performative structure of the e-auction institution

Figure 2.14 contemplates a performative structure of the e-auctions institution with scenes, transitions and arcs. For each arc, we define the list of roles that can enter this arc. Then, we specify the *type* of the arc that defines how an agent can enter the scene connected with this arc, i.e., the agent can either create a new instance of the scene or join one or multiple instances of the running scene. Then, for each arc, we can define *context-dependent constraints* and *actions*. Constraints and actions of an arc are expressions, which use role properties (current value is stored in an agent state) and institutional properties. The arc constraint is evaluated prior to an agent passing through the arc, and the action is evaluated after passing through this arc.

In the performative structure from Figure 2.14, all roles enter the e-auction institution through the Initial state. A staff agent then creates an *ItemRegister* and an *ItemInfo* scene. After these scenes were created, sellers can join the *ItemRegister* scene where they register products for the auction. After a product is registered, the staff agent changes its role to auctioneer and creates a new *Auction* scene. From this moment, buyers can enter the *Auction* scene and participate in the auction. Buyers can also visit the *ItemInfo* scene and ask about currently registered items and performing auctions.

2.3.4 Norms

Electronic Institution defines the possibilities and restrictions on agent actions while acting within scenes. As mentioned earlier, these actions are either illocutions or scene movements. Norms define the commitments, obligations and rights that agents acquire while performing their actions within the institution. Commitments restrict future actions of agents and may limit agent access to specific scenes and the illocutions that can be uttered. For the e-auctions institution, we do not define any norms.

2.3.5 Summary of Institutional Data

This section summarizes the institutional data that define an Electronic Institution. From what we have described so far, the behavior of an institution participant is constrained by an Electronic Institution at two levels:

- *Inter-scene*: Arcs in the performative structure define the possible paths that agents may follow depending on their roles. Furthermore, the constraints over output arcs impose additional restrictions on an agent attempting to reach a target scene, as well as specify how the agent state and EI context is modified when the agent passes through this arc.

- *Intra-scene*: Scene protocols define what, can be said by whom (with a specific role) to whom and when. Moreover, arcs are further extended by constraints and actions for agents passing through the arc.

Moreover, it is possible to constrain participant behavior by norms, which define commitments, obligations and rights of an agent. Constraints and norms are defined using agent properties (defined for each role), scene properties and message parameters. Table 2.1 contemplates the institutional data that an institution designer uses to define an Electronic Institution. In this table, the acronym DF stands for Dialogic Framework, while PS for Performative Structure. The definition of an institution is static and does not change during its execution, only the institutional context changes by the participation of agents in the institution.

2.3.6 EIDE Framework

EIDE[24] (Electronic Institutions Development Environment) is used to support the specification, design and deployment of Electronic Institutions [Esteva et al., 2008]. EIDE comprises the following components:

- **Islander** provides graphical interfaces for the specification of Electronic Institutions. It can also verify the specification according to both integrity and protocol correctness.

[24] `http://e-institutions.iiia.csic.es/eide/pub/` (05/2005)

Source	Part	Name	Description
DF	Role	**Role Property**	Property of a role
Norms	Norm	**Norm**	Norm defining obligations, commitments and rights of participating agent
PS	-	**Properties**	PS Properties
	Arc	**Agent Variable**	Allowed role for this arc
		Type	Defines if an agent can create a new scene connected with this arc or join existing instance(s) (i.e. one/many/new)
		Constraints	Context-based constraints for entering PS arc
		Actions	Updates context after passing arc
	Scene	**Protocols**	Scene protocols associated with given scene
		Properties	Properties of the scene
Scene	Arc	**Arc Action**	Message said by agent
		Constraints	Context-based constraints for entering scene arc
		Actions	Updates context after passing this scene arc
	Node	**Action**	Enter/Exit scene action
		Agent	Agent/Role allowed to perform the action
		Constraints	Context-based action constraints
		Actions	Updates context after the action

Table 2.1: Institutional Data

- **AMELI** is a runtime infrastructure for Electronic Institutions, which loads an institution specification and then mediates participant interactions while enforcing institutional norms.
- **aBuilder** facilitates agent programming for agents participating in an Electronic Institution. It structures the development of agents along the dimensions of the specification and automatically generates agent "skeletons".
- **SIMDEI** provides a simulation environment for quick testing of Islander specifications. With SIMDEI institutions can be tested with different agents populations and under different circumstances [Esteva et al., 2008].

Figure 2.15 contemplates all EIDE. The definition of the EIDE framework concludes our explanation of Electronic Institutions. In the following section we explain how we can use Electronic Institution to create Normative Virtual Worlds (i.e. Virtual Institutions).

2.4 Virtual Institutions (VI)

The concept of combining Electronic Institutions with 3D virtual worlds was introduced in [Bogdanovych et al., 2005] as Normative Virtual Worlds and named

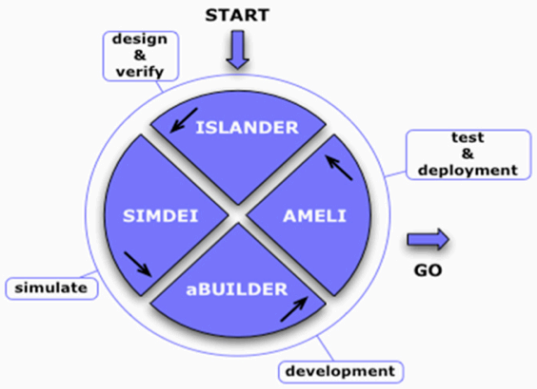


Figure 2.15: EIDE Framework Components

Virtual Institutions. In this context, Electronic Institutions are used to specify the rules that govern participants' behaviors, while 3D virtual worlds are used to facilitate human participation in the institution. Therefore, participants of Virtual Institutions can be both human and software agents. A Virtual Institution is separated into a *Normative Control Layer* and a *Visual Interaction Layer*. This provides support to the conceptual separation between the normative control of interactions and the design of the virtual world, i.e., the design of the 3D graphical user interface. The Normative Control Layer is responsible for the institutional control of interactions among participants, while the Visual Interaction Layer focuses on the 3D representation of the institution. Regarding participants, humans participate in the system by controlling an avatar on the Visual Interaction Layer, while software agents are directly connected to the Normative Control Layer, visualized as "special" avatars in the Visual Interaction Layer, and participate by exchanging messages.

Both layers are causally connected, whenever one of them changes, the other one changes in order to maintain a consistent state [Maes and Nardi, 1988]. In the case of our Virtual Institution, a Causal Connection Layer keeps a consistent state between the model, represented by the Normative Control Layer, and its view, represented by the Visual Interaction Layer. Figure 2.16 shows an overview of the three layered architecture of Virtual Institutions.

There is an important conceptual difference between Electronic Institutions (EI) and Virtual Institutions (VI). In EIs everything is regulated in the sense that it is defined what is permitted, and everything else is prohibited. In VIs the situation is opposite: there are some actions provided by the virtual world platform that have institutional meaning, and hence, they are regulated (e.g. participants are obliged to pay for obtained goods before leaving the auction room), while the rest of actions are permitted (e.g. there is no regulation for walking around a room). This is similar to traditional organizations or institutions, where participants are able to perform many actions, but only some of

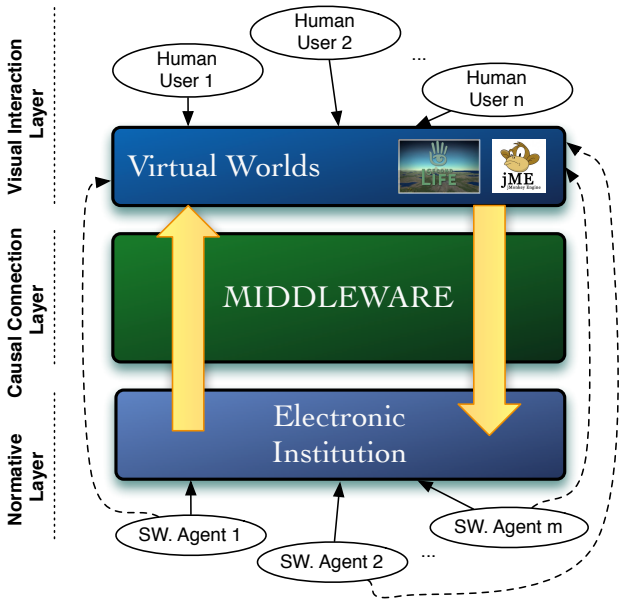


Figure 2.16: Overview of VI architecture.

them are regulated and have organizational or institutional meaning.

2.4.1 Related Work: Causal Connection Between Virtual Worlds and Multi-agent Systems

In this research work, we advocate the use of Virtual Institutions as Normative Virtual Worlds for the deployment of e-* applications and social simulations. For the purposes of such applications, we found that current architectures of the middleware (Casual Connection Layer) lack features, therefore we proposed a new one. This section presents the state-of-the-art in causal connection architectures, as well as in works related to Normative Virtual Worlds.

First attempts to establish a *causal connection* between Electronic Institutions and 3D virtual worlds (in this case Adobe Atmosphere) led to the creation of Causal Connection Server (CCS) [Bogdanovych, 2007]. CCS implemented two basic causal actions: (i) creation of an external agent connected to AMELI when a new human user logs in the institution and (ii) visualization of a new avatar in a virtual world when a software agent connects to AMELI. Additionally, CCS used an action/message table mapping each virtual world action to an institutional message. Later, [Bogdanovych et al., 2008] presented an architecture that allowed causal connection between multiple environments (e.g. mobile devices, virtual worlds and the real world) and AMELI.

Subsequently, the implementation of the causal connection between a 3D vir-

tual world and an Electronic Institution was further extended during the development of the Itchy Feet prototype [Seidel, 2010] which used Virtual Institutions in a virtual tourism environment. For this purpose, authors implemented the Connection Server, bound to the Itchy-Feet implementation. Authors later removed the binding to their solution and presented a Generic Connection Server (GCS). The architecture of this system is depicted in Figure 2.17. GCS is a robust extension of CCS, that uses its own communication protocol between the virtual world and AMELI. Messages using this protocol are specified in XML format and validated against XSD schema. We have used the ideas from both GCS and CCS to design our new architecture.

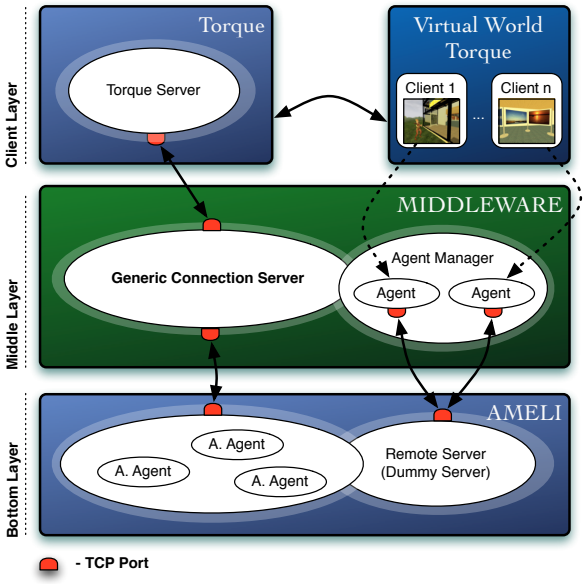


Figure 2.17: Architecture of the Itchy Feet solution

Previous approaches to causal connection have some limitations: (i) they can only use a single virtual world (GCS) or the execution with multiple environments is difficult to control (CCS); (ii) they cannot manipulate the virtual world content at runtime; (iii) there is no mechanism for a fail-safe communication.

We have removed these limitations and created Virtual Institution eXEcution Environment (VIXEE). VIXEE is a generic solution, which handles the causal communication with multiple environments using the Movie Script mechanism (see Section 5.2.2). Moreover, VIXEE dynamically reacts on changes in the institution and changes the virtual world content accordingly (see Chapter 5).

In works related to other types of Normative Virtual Worlds, [Cranefield and Li, 2009] describe the implementation of a tool that structures social interactions (inspired by human society) between software agents and adapts them for use with the computer-mediated human communication pro-

vided by virtual worlds. [Ranathunga et al., 2010] addressed issues of collection, filtering and processing of the sensor data. This research led into the development of a tool that connects Second Life with a MAS programming language called Jason. Jason is one of the most known agent programming languages and platforms [Bordini et al., 2007]. In this work, authors demonstrate a new framework in conjunction with an extension of the Jason BDI interpreter that allows agents to specify their expectations of future outcomes in the system and to respond to fulfillment and violations of these expectations.

The use of Normative Virtual Worlds has been explored in the context of computer games as well. In the most recent ones, [Aranda et al., 2010] use Virtual Institutions in the definition of Quests for the Massive Multi-player Online Games (MMOG). The idea of this work is to apply multi-agent systems technology to deploy MMOGs. [Napagao et al., 2010] modelled gaming scenarios using social structures. Authors of this work intended to create a methodology and tools for Game AI developers. This would allow decoupling the implementation and the design, allowing reusable solutions.

2.5 Shape Grammars

In previous sections, we introduced virtual worlds, Electronic Institutions and Virtual Institutions concepts. Now, we present a technique named shape grammars, which serves as the basis for our proposal of Virtual World Grammar concept (see Chapter 4).

As described by McCromack and Cagan [McCormack and Cagan, 2003], a shape grammar is a method of generating designs by using primitive *shapes* and *rules* of interaction among them. Hence, it is composed of shapes and rules. One of the shapes is marked as *starting shape*, while rules are composed of left-side shapes and right-side shapes. Designs are generated from a shape grammar by starting with an initial shape and recursively applying its rules. A shape grammar rule takes the form A $\rightarrow$ B, where A and B are both shapes. The rule is applicable to shape C if there exists a transformation τ, such that $\tau(A) \leq C$ where $\leq$ is the operator that determines the existence of a sub-shape. The rule is then applied by subtracting the transformed instance of shape A from shape C and adding a similarly transformed shape B, hence C - τ(A) + τ(B).

An example of a shape grammar rule is displayed in Figure 2.18. Figure 2.19 displays five steps of the shape grammar derivation process using the rule from the Figure 2.18. This rule simply adds a rotated copy of a rectangle to its origin. We can see that the derivation process uses rectangle shapes that were produced by the intersection of already generated ones. These shapes are called *emerged shapes*. The process of finding such shapes is called *sub-shape detection* and it is a complex problem for computer based implementations of shape grammars. The use of sub-shape detection leads to the richer variety of designs and allows us to use full power of shape grammar technology. Our sub-shape detection algorithm is based on the [Krishnamurti, 1981] algorithm. Our modification significantly decrements the search space, thus allows real-time execution for

relatively complex shapes.

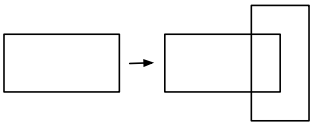


Figure 2.18: Shape grammar rule

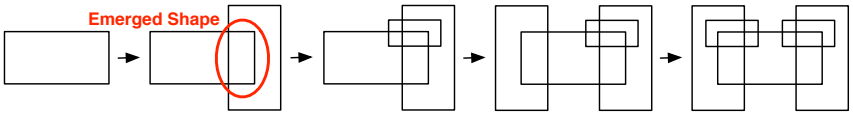


Figure 2.19: Shape grammar derivation process

Furthermore, *markers* (also called labels) can be used to control how rules are applied to the left-side shape. Rules with markers are called *labelled rules*. Markers control the positioning of the shapes according to its symmetries. An example of markers and different positioning of markers with the resulting derivation can be seen in Figure 2.20. In this figure, we see two different positions of markers that lead to very different derivations. Furthermore, markers can be used as carriers of semantical information about a given point. This information can provide not only data about the functionality of the shapes but it also can hold values of parameters used in the shape grammar generation process [Agarwal and Cagan, 1998].

Once described all parts of a shape grammar, we are ready to provide a formal definition:

Definition 2.6. As defined by Stiny [Stiny and Gips, 1972], a *shape grammar* (SG) is a 4-tuple: SG $= (S_T, S_M, R, I)$ where

1. S_T is a finite set of terminal shapes. S_T^* is a set of shapes formed by the finite arrangement of an element or elements of S_T in which any element of S_T may be used multiple number of times with any scale or rotation operation.

2. S_M is a set of shapes used as *markers*, such that $S_T^* \cap S_M = \emptyset$. Markers permit to control how rules are applied to the left-side shape.

3. R is a finite set of *rules*, that are ordered pairs (u, v) such that u is a left-side shape consisting of an element of S_T* possibly combined with an element of S_M and s is a right-side shape consisting of:

   i An element of S_T^* contained in u or

   ii An element of S_T^* contained in u combined with an element of S_M or

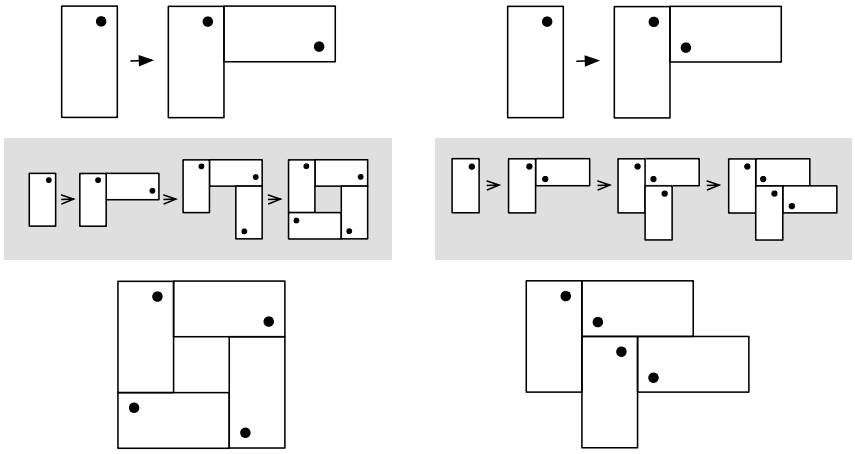


Figure 2.20: Two different labelled rules and their derivations

iii An element of S_T^* contained in u combined with an additional element of S_T^* and an element of S_M

4. I is the *starting shape* consisting of elements of S_T^* and S_M.

Having defined all important parts of shape grammars, we present some interesting examples of shape grammar use from different fields such as architecture or product design. Initially, shape grammars were introduced in the early 1970s as a way of describing and creating paintings and sculptures [Stiny and Gips, 1972]. This first attempt led to a wide-spread use of this kind of grammars into multiple artistic, scientific and industrial fields. They were used to analyze and recreate works of various artists like Piet Mondrian, Georges Vantongerloo and Fritz Glarner potentially with great accuracy.

In architecture, Frank Lloyd Wright's prairie houses [Koning and Eizenberg, 1981], Palladio's villas [Stiny and Mitchell, 1978] or Mughul gardens [Stiny and Mitchell, 1980] were analyzed using shape grammars so that new designs could be generated based upon the originals and with similar style. José P. Duarte used them to generate Siza's Malagueira houses [Duarte, 2001] and created an online application that rendered houses depending on user preferences. Duarte also introduced the concept of *discursive* grammars that contain a shape grammar, a description grammar and a set of heuristics. Furthermore, shape grammars also entered non traditional fields when they were used to derive cellular automata rule patterns [T. Speller, 2007].

Shape grammars have also been used in the design of industrial products. Prats [Prats et al., 2009] used shape grammars to analyze sketches of industrial designers and let them have the freedom to explore a design space, whilst providing quantitative information about produced designs. They presented car design examples to illustrate the application of the shape grammar tool. Orsborn

[Orsborn and Cagan, 2009] proposed a multiagent shape grammar implementation (MASGI) to automatically generate product forms according to a preference function that could represent designer or consumer design preference. This system used the multiagent system to enable modifications to the shape grammar as the form design space changes. Soni, Khanna and Tandon [Soni et al., 2010] made an effort to develop a design system to support the automatic concept generation and evaluation for the mass customization of industrial products. They presented examples in mobile phone design.

2.5.1 Related Work: Computer Implementation of Shape Grammars

In this thesis, we present a new shape grammar framework for definition and generation of rectilinear forms, which contains our algorithm for efficient sub-shape detection and is implemented in the Shape Grammar Interpreter. In this Section, we present the state-of-the-art in the area of the computational implementation of shape grammars.

Shape grammars were initially executed either manually (i.e with pencil and paper) or through ad-hoc computerized approaches intended for a particular grammar. Early generic shape grammar systems were implemented in Prolog [Chase, 1989] [Flemming, 1987]. Nevertheless, the definition of shapes and rules in these early systems was declarative, thus not interactive. One of the first significant visual implementations of shape grammars was done by Tapia [Tapia, 1999]. The framework we present in this thesis has been designed to be a general tool to create and work with any rectilinear 2D shape grammar. By general we mean that our tool is not tied to any particular type of shape grammar, and it can be used in many different domain applications. Our tool covers the entire process of shape grammar management that goes from the grammar conception (i.e design of shapes and rules) until the generation of the design (i.e generation algorithm and parameters).

Nevertheless, one of the most notable features of our framework is the implementation of our *sub-shape detection algorithm*. Sub-shape detection is the process of finding sub-shapes in a shape. It is a complex problem for computer implementations of shape grammars. Most of the current implementations of shape grammar interpreters do not support sub-shape detection. Using sub-shape detection, we gain the power to use emergent shapes in the generation process. A first description of sub-shape detection algorithm for rectilinear shapes was done by Krishnamurti [Krishnamurti, 1981]. In Chapter 3, we present a modified version of Krishnamurti algorithm, which allows sub-shapes detection in real time. McCormack and Cagan [McCormack and Cagan, 2002] focused on creating an efficient parametric shape grammar interpreter that would support creativity through shape emergence. If a shape grammar tool wants to support shape emergence, it has to have a possibility to detect such emerged shapes. In their approach, they divided the shape into a hierarchy of sub-shapes based on specific geometric relationships within the shape. Recent years brought attempts

to create a sub-shape detection algorithm for curved shapes [Jowers, 2006] during which a shape grammar interpreter for curved shapes was produced (named QI). Later, this algorithm and his versions were used in computer vision projects, where shape grammars were applied to analyze sketches of product designers, in order to easily browse the possible design space [Lim et al., 2008] and [Prats et al., 2009]. Although the dissertation of Patrick Min does not focus on shape grammars, he introduces interesting approaches on shape detection using the matching of ovals [Min, 2004].

In our research, we focus on 2D shape grammars producing rectilinear shapes, but there were efforts to create a 3D shape grammar, such as 3D Shaper from [Wang and Duarte, 2002]. From more recent works we mention the 3D shape grammar implementation by [Chau et al., 2004]. [Hoisl and Shea, 2011] presented a basic 3D grammar interpreter for the interactive, visual development and application of three-dimensional spatial grammar rules. In our work [Trescak et al., 2010b] we took a different approach and used specific 3D transformation mechanisms, that produced a 3D output using a semantic description of 2D shapes.

An alternative to shape grammars are so-called Lindenmayer systems (L-systems) [Prusinkiewicz et al., 2001] where rules are codified using symbols. The Computer Generated Architecture (CGA) introduced the idea of grammar-based modelling using rules that iteratively refine a design by creating more and more detail [Parish and Muller, 2001] [Muller et al., 2006]. Rather than building on string replacement as L-systems do, CGA rules replace shapes with shapes. CityEngine, a program allowing automatic population of cities, was inspired by hybrid systems that combine shape grammars and L-systems. A recent research, also using CGA shape, has presented a real-time visual editor which works on usability issues such as the possibility of doing interactively local modifications on buildings [Lipp et al., 2008].

In contrast to L-systems based approaches, we present a pure shape grammar framework where the rules and the process of their application is performed using geometrical shapes. This leads to some advantages of shape grammar systems over L-systems. First, because of the direct definition of shapes via rules no associations (i.e symbol-shape) are needed. Therefore, the user has immediate overview of the behavior. Second, the use of markers allows better control of shape grammar rules execution. Last, shape grammars lead to more general use, while L-systems usually aim to generate biological forms or any repetitive patterns. Nowadays, shape grammars continue being an interesting research topic and trends are oriented to provide the designer both an attractive visual interface and realistic results.

Shape grammar explanation closes explanation of topics related to the generation and execution of virtual worlds. In the next section, we take a look at A-Life field, which inspired us to create the general purpose agent model for Virtual Institutions. This model is introduced in Chapter 6.

2.6 Artificial Life

Artificial life (A-Life) attempts to understand the essential general properties of living systems by synthesizing life-like behavior in software, hardware and biochemical systems [Bedau, 2003]. It is the study of synthetic systems that behave like natural living systems in some way. A-Life is a young field that has been gaining acceptance over the past two decades, but already in 1911 Leduc postulated:

> *The synthesis of life, should it ever occur will not be the sensational discover which we usual associate with the idea. If we accept the theory of evolution, then the first down of the synthesis of life must consisted in the production of forms intermediate between the inorganic and the organic world forms which possess only some of the rudimentary attributes of life, to which other attributes will be slowly added in the course of development by the evolutionary action of the environment [Leduc, 1911].*

There are many important questions that evolutionary scientists confront while using A-Life simulations [Zimmer, 2005]: What good is half an eye? Why does a forest have more than one kind of plant? Why be nice? Why do we need sex to reproduce? What does life on other planets look like? What will life on earth look like in the future? A-life is devoted to understanding such questions by attempting to abstract the fundamental dynamic principles underlying biological phenomena (e.g. evolution) and recreating these dynamics in other physical media, thus making them accessible to new kinds of experimental manipulation and testing [Sipper, 1995]. A-life allowed evolutionary biologists to move from experiments "in vitro" to "in silica" and observe a *evolution* of many generations of *digital organisms* in a very small time period comparing to "in vitro" experiments. A reason for this is the spectacular ability of digital organisms, the ability to evolve. They replicate; they mutate; they compete with each other. Robert Penncock, from Michigan State University notes: "A-life systems are not simulations of evolution, they are the instance of it. It may seem strange to talk about a chunk of computer code in the same way you talk about a cherry tree or a dolphin. However, the more biologists think about life, the more compelling the equation becomes." The resemblance of computer programs and DNA is in their representation of sets of instructions. Computer programs tell a computer how to process information, while DNA instructs a cell how to assemble proteins [Zimmer, 2005].

A-life is based on the well-known intersection of evolutionary biology with computer science, that is genetic algorithms or their many variants (genetic programming, evolutionary strategies, and so on). All these variants share the same basic technique:

1. Create random potential solutions.

2. Evaluate each solution assigning it a fitness value to represent its quality.

3. Select a subset of solutions using fitness as a key criterion.
4. Vary these solutions by making random changes or recombining portions of them.
5. Repeat from step 2 until a solution is found that is sufficiently good.

First "evolving" computer system was presented by Steen Rasmussen in 1990 [Rasmussen et al., 1990] although with limited success. The first successful A-Life experiment was performed by Thomas Ray using his "Tierra" system [Ray, 1991] [Ray, 1993]. Adami, used Tierra to create conditions in which computer programs evolved the ability to solve simple mathematical problems (such as addition), without forcing them to use a pre-defined approach [Adami, 1998]. This project imposed many limitations, thus Adami, together with Charles Ofria and C. Titus Brown developed a new A-Life platform called Avida [Ofria and Wilke, 2003].

Avida platform was used to investigate the evolution of complex features. Researchers set up an experiment in Avida to document how equals operation evolves. Evolution in Avida produced organisms that could carry out the equals operation. Moreover, all the successful equals tests were done in a completely different way. This provides a partial answer to the question "what good is half of an eye," mentioned in the beginning of this section, when considering the evolution of the "eye" organ, a fly and a human can both produce a visual image of their environment, but the structure of the eye is completely different [Zimmer, 2005].

Another significant A-Life simulator that moved digital organisms to 3D space was introduced by Yeager and named PolyWorld [Yaeger, 1993]. Digital organisms in this artificial life simulator have several biological features. They "live" and "move" in a simulated 3D space, represented by extruded 3D polygons. Simulated organisms reproduce sexually, fight and kill and eat each other, eat the food that grows throughout the world, and either develop successful strategies for survival or die. An organism entire behavioral suite (move, turn, attack, eat, mate, light) is controlled by its neural network "brain"; thus, it has the ability to learn. Organisms can perceive their environment using their "eyes" with limited visibility. Even physiological information is encoded into an organism's chromosome; thus, brain, body and all components of behavior evolve over time. In Polyworld, a variety of species evolved over time, displaying such complex ethological behaviors such as swarming, flocking, foraging and avoidance[25]. Polyworld project remains active until nowadays. [Yaeger et al., 2010] used Polyworld to study the evolutionary selection of neural network structure and function. Authors applied graph theoretical tools to the analysis of the topology of artificial neural networks known to exhibit evolutionary increases in dynamical neural complexity.

In summary, *digital organisms* motivate researches from many different fields due to several reasons[26]:

[25] `http://www.beanblossom.in.us/larryy/polyworld.html` (last accessed 03/2012)
[26] `http://avida.devosoft.org/about/` (05/2012)

1. Artificial life forms provide an opportunity to seek generalizations about self-replicating systems.

2. Digital organisms enable us to address questions that are impossible to study with organic life forms.

3. Other questions can be addressed on a scale that is unattainable with natural organisms (e.g. extensive study of millions of types of genotypes of digital organisms vs. one genotype of bacteria E. Coli).

4. Digital organisms possess the ability to truly evolve, unlike mere numerical solutions.

5. Digital organisms can be used to design solutions to computational problems.

In one of the most famous applications of A-Life theory, Karl Sims designed a system that generates virtual creatures that move in simulated three-dimensional physical worlds [Sims, 1994]. Creatures were made of a set of blocks of different sizes. Both, the morphology of block organization and a neural system for controlling the movement of these blocks were generated automatically using genetic algorithms. Different fitness evaluation functions were used to direct simulated evolutions towards specific behaviors such as swimming, walking, jumping, and following. The author presented an unprecedented variety of interesting locomotion strategies, where some of them would be difficult to invent, build or design without using an evolutionary strategy.

A-Life theory was also exploited by the world of games where users could play with a simplified version of evolution of digital creatures using their computer. The first successful A-Life based game was Creatures[27], which has four sequels. In 2008 EA presented The Spore[28] which allowed users to evolve their own 3D modeled creatures that inhabited different planets.

In this thesis, A-Life and digital organisms inspired us to create a new intelligent virtual agent model with a high level of sophistication as this model possesses A-Life features, but it also develops believable, human-like behavior. Theory on believable agents is the focus of Intelligent Virtual Agents field explained in the next section.

2.7 Intelligent Virtual Agents

Intelligent virtual agents (IVAs) are animated characters capable of autonomous interactions in dynamic social environments. Works in this area seek to create virtual characters with capacities for perception, cognition and action, often including the ability to engage in dialogue with human users. A virtual agent must perceive the world in which it exists, both virtual and real,

[27] `http://creatures.wikia.com/wiki/Creatures_Wiki_Homepage` (03/2012)
[28] `http://www.spore.com/ftl` (03/2012)

often including human participants' language and gestures. It must reason about those perceptions as well as decide on how to act on them in pursuit of its own agenda [Marsella and Badler, 2011]. In some works we may find references to Intelligent Virtual Agents as to Virtual Humans [Badler, 1997] [Kshirsagar and Magnenat-Thalmann, 2002] [Swartout et al., 2006] or Digital Humans [Badler et al., 2002].

Intelligent Virtual Agent are used to substitute human presence in virtual environments; thus, it is essential that they act believably. Such believable agents are personality-rich, emotional characters, acting and behaving human-like [Loyall, 1997]. Human behavior is a complex subject of study, thus its simulation is separated into several different sub-fields. [Kasap and Magnenat-Thalmann, 2008] reviewed the state-of-the-art for these IVA fields and separated the research interest of IVA believability into several categories:

1. **Personification** is the application of human properties to non-human creatures and objects and deals with simulation of human *personality* and *emotions*.

2. **Interaction** study focuses on the intelligent representation of agent's verbal and nonverbal behavior. Studies in this field deal with *facial expressions*, *gestures* and *dialogue management*.

3. **Autonomous behavior** deals with the ability of agents to autonomously act within their simulated environment. Research interests in this category include methods of *perception* of an environment, *decision-making and adaptation* of agent actions depending on perceived data and agent state and *action control* dealing with believable visual representation of agent actions (e.g. agent cannot be sitting and running in the same time)

In the following paragraphs, we review some existing works in previously mentioned domains of IVA research, related to our work. We start with works on agent personification, because emotional behavior and personality have a significant impact on agents' believability.

Well known models on human psychology, are the *Big Five framework* for human personality [Costa and McCrae, 1992] and the appraisal theory for human emotions, represented by the Ortony, Clore & Collins (OCC) model [Ortony et al., 1988].

The *Big Five framework* of personality traits has emerged as a robust model for understanding the relationship between personality and various academic behaviors. The Big Five factors, known as personality traits, are Openness, Conscientiousness, Extraversion, Agreeableness, and Neuroticism (OCEAN). Openness reflects the desire for novelty and variety, the intellectual curiosity. Conscientiousness represents the goal-driven thinking, the ability to be disciplined and organized. Extraversion defines the level of sociability, preference to solve the goal by social interactions, assertiveness and talkativeness. Agreeableness reflects the will of being cooperative, helpful and sympathetic to others. Finally,

neuroticism refers to the level of emotional instability, anxiety and to react impulsively rather than rationally. The OCEAN model has been refined and adapted since its initial design, so that it provides us the possibility to model different personality traits.

A popular model for simulation of human emotions is the appraisal-based model OCC [Ortony et al., 1988]. Appraisal theory is based on the idea that emotions are based on the subjective representation of perceived events; thus, the same event produces different emotions for every individual. OCC model isolated 22 core emotion types (e.g. joy, distress) and designed the structure of their evaluation [Steunebrink et al., 2009] depending on the perceived event.

Several systems that implemented OCEAN and OCC were presented in the past few years [Egges et al., 2003] [Bartneck, 2002] [André et al., 2000] [Van Dyke Parunak et al., 2006] [Doce et al., 2010] [Hudlicka, 2005] [Liu et al., 2009]. All these systems share a common aspect of quantifying the personality and emotion values and evaluating them according to the external event. Other available personality and emotion models have been thoroughly reviewed and evaluated by [Gratch and Marsella, 2004] [Gratch and Marsella, 2005] [Marsella et al., 2010].

While *personification* represents a substantial part of an agent believable act, agents will not be believable without intelligent nonverbal behavior during *interactions*. Scientists studied different aspects of non-verbal behavior, which included facial expressions and speech [Cassell et al., 1994] [Gratch et al., 2002], eye movement and gaze [Bickmore and Cassell, 2005] and gestures [McNeill, 1996]. Authors even focused on the study of head nods during interactions [Lee et al., 2010]. They followed studies that showed that head nods serve a variety of communicative functions and that the head is in constant motion during speaking turns. They concluded that there is a significant effect on the perception of head nods in terms of appropriate nod occurrence, especially between the data-driven approach and the rule-based approach.

When many different components form the believable behavior, e.g., gestures, head nods and speech, it is difficult to keep track of their relations and execute them in a correct way. For example, nod head to agree and move closer to a person when showing aggression. [Kipp et al., 2010] suggest to distinguish realization planning, where gesture and speech are processed symbolically using the behavior markup language (BML), and presentation, which is controlled by a lower-level animation language (EMBRScript).

In another category of IVA research, authors focused on correct spatial behaviour, which is essential for virtual agents acting in human-like environments, such as buildings or cities. An important part of correct spatial navigation is memory. Different authors have explored possible models of memory organization. The most common memory model is *episodic memory* with *spatial* [Brom et al., 2009] or *spatio-temporal* [Schill and Zetzsche, 1995] memory organization. Brom et al. investigated on a simplified model of a virtual character living in a virtual house, how memory representation is formed, and how it evolves based on how objects are moved within the environment. He compares

agent behavior with behavior of real humans and concludes that his memory model is feasible for virtual agents.

At last we mention several long-running projects that consider investigation of different aspects of Intelligent Virtual Agents. First, the *Social Agents*[29] project aims to create sociable virtual characters that facilitate human-computer interaction. Project's interests vary from imitation mechanisms, motor cognitions, dialogue coordination to adaptive embodied communication. Second, the *Embots*[30] research group focuses on embodied agents and their correct representation of verbal and nonverbal behavior. In one of the last projects of this group authors studied the possibility of having avatars showing sign language as a visual aid for hearing disabled people. Last project group[31] focuses on agent multi-modal behavior and the coordination processes between specific parts of such behavior.

2.7.1 Related Work: Models for Intelligent Virtual Agents

In this thesis, we present a generic model which intends to create believable intelligent virtual agents that automatically reason and act in Normative Virtual Worlds. In this section, we present the related work to this research and present several others existing models of IVA, which inspired this work.

In the previous section, we have introduced several lines of research on agent believability. Authors presented specific implementations to demonstrate feasibility of such research. [Kopp et al., 2005] created a conversational agent called Max and put him to test real-world settings in a museum, where the agent engaged in a natural face-to-face communication with museum visitors. People interacting with Max were aware that Max was an artificial being and adjusted their interaction to small talk. Authors evaluated their system according to the conversational capabilities of the virtual agent and the resemblance to the human-like dialogues. Results indicated that Max engaged people in interactions where they were likely to use human-like communication strategies, suggesting the attribution of sociality to the agent.

[Weitnauer et al., 2008] integrated Max into Second Life and let him communicate with Second Life residents. Max could multi-modally (i.e. voice, chat) communicate with other avatars. Residents reported that they suspected the agent to be controlled by an artificial system at a certain point during the conversation, or they got frustrated without realizing that they were not interacting with a human and thus ended the conversation. In another research, [Jan et al., 2009] created a virtual tour guide for Second Life, which combined the abilities of a conversational agent with the navigation in the world. The tour guide was a US army marshal the gave island visitors information about the island as well as giving a guided tour as it goes through his rounds. Tour guide's avatar can be controlled either by an agent, or by a human operator.

[29] `http://www.techfak.uni-bielefeld.de/ags/soa/` (05/2012)
[30] `http://embots.dfki.de/projects.html` (05/2012)
[31] `http://www.ru.is/~hannes/ru_main_projects.html` (05/2012)

These implementations represent goal specific implementations, producing not-reusable solutions. In recent years, several general purpose agent models were proposed. These models, along with their implementations represent reusable solutions, which allow to use IVA in different environments, simulations, social experiments and computer games.

Currently available general virtual agent models are focused on games and simulations. In the rest of this section, we present following existing systems and explain why they do not properly fit to our purposes (except for Uruk 3000 BC):

- **Pogamut** [Gemrot et al., 2009]
- **NonKin village** using PMFserv [Silverman et al., 2011]
- **Virtual Humans** from Mission Rehearsal Exercise project [Swartout et al., 2006].
- **Uruk 3000 BC** [Bogdanovych et al., 2011]
- **ALIVE** [Napagao et al., 2010]

(a) Agents Interacting in EmoHawk (b) Pogamut with UT2004

Figure 2.21: Pogamut

Pogamut 3 is an open-source platform for rapid development of behavior for virtual agents embodied in a 3D environment of the Unreal Tournament (UT) videogame [Gemrot et al., 2009]. Pogamut 3 is designed to support research as well as educational projects. Developers can program application-specific agents. They can implement agents' control mechanisms or use the reactive planner POSH. This project in its history was used to: (i) study evolution of UT bot behavior embodied using genetic algorithms [Kadlec, 2008]; (ii) research possibilities of episodical memory of game bots [Brom et al., 2008] [Brom et al., 2009]; and to (iii) study possibilities or virtual storytelling[32]. While Pogamut 3 introduces many interesting features, it is aimed as the student platform for design

[32] `http://artemis.ms.mff.cuni.cz/emohawk/doku.php`, (last accessed 03/2012)

and execution of virtual agents, where the agent model includes only simple functionalities (e.g. path-finding) and agent reasoning is based on declaratively specified decision trees.

A complex model for believable intelligent virtual agents was presented by Silverman et al. with the definition and implementation of PMFServ [Silverman et al., 2006b] [Silverman et al., 2006a] and more recently in a simulation environment called NonKin Village [Silverman et al., 2011]. Silverman et al. created a highly sophisticated model for socio-cognitive intelligent virtual agents, which included either newly designed or adapted models of agent physiology [Gillis, 2000] [Witmer et al., 2002], stress assessment [Janis and Mann, 1977], personalities, emotions (cognitive appraisal - OCC), perception, social functions (relations, identity, trust) and cognition (affect- and stress-augmented decision theory, bounded rationality). This agent implementation was developed to train the army corps to act in foreign cultures. Silverman evaluated goals according to the Performance Modification Functions (PMF) to access the next action to execute. PMF is a function that models a behavioral response of an agent depending on related input. Silverman designs many generic types of PMF, such as decision, emotion, perception or stress. PMFserv is an implementation of such PMF theory. It is an open agent architecture that allows to research and explore alternative PMFs to add realism to a software agent. PMFserv incorporates four domains of psychological theories and models: (i) psychobiological, (ii) personality, culture, and affect, (iii) social, and (iv) cognitive.

In other works, the *Mission Rehearsal Exercise* project joined efforts of several dozens of authors from different research fields to create a model for intelligent virtual humans [Swartout et al., 2006]. Authors stated their biggest contribution in integrating several existing models, learning lessons during this integration, what resulted into improved state-of-the-art in related fields. Virtual humans from this project were able to communicate in natural language, correctly assessed their environments, had different personalities, presented their emotions, interacted verbally and nonverbally and dealt with stress.

Both presented agent models represent highly complex models that integrate several sophisticated components, where the main focus of agent modelled in this system is its believable interaction with a human user. Both projects were used in army scenarios, where goals of every agent are strictly pre-defined and agent execution is started only by entering agent's area of interest, thus simulating the game-like life and not our desired artificial life.

Cultural simulation in a e-learning scenario, using Virtual Institutions, was presented by [Bogdanovych et al., 2011] in the project of Authentic Interactive Re-enactment of Cultural Heritage with 3D Virtual Worlds and Artificial Intelligence. In this project, authors, using 3D virtual world, re-created life in the mesopotamian city of Uruk 3000 BC. This simulation included accurately reconstructed models of historical buildings (e.g. ziggurat), models of authentic historical objects and avatars representing Uruk inhabitants, wearing traditional clothes. Authors presented the role of Fisherman, which performs his actions in Second Life virtual world in 15 minutes cycles. Agents are capable of dynamic

planning, adjusting their actions depending on their state and the state of the environment. In this thesis, we extend the features of Uruk agents and define a general purpose virtual agent model for agents acting in normative virtual environments.

So far, we have introduced models for individual agents, but other authors also focused on coordination and organization of virtual agents, in order for them to be able to collaborate on their common or individual tasks. *ALIVE* project [Napagao et al., 2010] uses existing formal models of *coordination* and *organization* mechanisms to deliver a flexible, high-level means to describe the structure of interactions between services provided by agents in the environment. Agents represent components of a dynamic ecosystem, where each component provides some services. Agents collaborate in order to achieve a common goal. All components of ALIVE system provide semantic description of their functionalities through web services. ALIVE project is an example of service-based computing. Its structure is split into three different levels: (i) the *service level* introduces social context to existing services by connecting them with semantic technologies, (ii) the *coordination level* specifies workflows that represent high level of interaction among services, and (iii) the *organization level* specifies the organization rules and norms that govern interactions. Such separation allows agent designer to think in *why-what-how manner instead of the most common when-what practice*. At the organization level, the designer specifies goals, that specify why to do something. The coordination level specifies what to do and the service level how to do it. Authors of ALIVE platform connected their system to some existing computer games such as Grand Theft Auto IV[33], Warcraft III [34] and Lincity [35].

We approach the problem of agent coordination and organization by using Electronic Institutions, a well established Organization-Centered Multi-agent System, which structures agent collaborations.

2.7.2 Related Work: Crowd Simulation

So far, we have talked about single instances of Intelligent Virtual Agents. In many occasions, we need to generate a crowd of these agents which look and behave believably. To achieve believability of a generated crowd each crowd member, i.e., agent, should have a unique appearance and behavior.

Crowd generation methods vary on how to model a single individual and the approach of making every crowd individual unique. In one of the first attempts to generate a population of unique 3D characters, [DeCarlo et al., 1998] created a system that generated facial models. The model was based on randomization of anthropometric measures applied to B-spline surfaces. Later, [Blanz and Vetter, 1999] used Principal Component Analysis (PCA) to analyze datasets of facial features to extract base vectors from the face, and used these vectors to generate new, unique faces. [Allen et al., 2003] extended this work

[33] `http://www.rockstargames.com/IV/`
[34] `http://us.blizzard.com/en-us/games/war3/`
[35] `http://lincity-ng.berlios.de/`

to generate the whole body of an avatar. A different approach was taken by [Maim et al., 2009] and [Thalmann, 2007] who used variance of attachments and textures over a predefined set of avatars, where avatars mainly varied in size and the type of textures they used while still appearing as clones that undergone a minor modification.

In other works, authors considered different means of crowd generation, also considered by this thesis, and they isolated specific body and clothing parameters which deform related 3D models. The values of these parameters can be randomized in order to produce unique crowd members [Seo and Magnenat-Thalmann, 2003] [Magnenat-Thalmann et al., 2004a].

Isolating visual features, quantifying and operating with them inspired researchers to encode their values into genetic structures and apply genetic algorithms theory to generate unique crowd members. Ventrella was one of the first to explore the possibilities of storing and modifying the avatar properties in a "chromosome," represented by an array of integer values. Genes from this array can be modified in order to generate a sketch of an avatar with different appearances. Ventrella presented his results using the developed Genetic Customization Tool [Ventrella, 2000].

While Ventrella worked with 2D sketches, [Lewis, 2000] [Lewis and Parent, 2000] applied the genetic approach to 3D avatars. In these works, authors presented the way of mapping body parameters (e.g. hip size, back arch) to 3D models, encoding them into avatar genes and generating new avatars using these genes. Another important aspect of this work was the specification of how to capture dependencies between body parameters. Maintaining their dependencies allowed authors to define high level parameters, such as height, whose modification performed hierarchical sizing of 3D models in a natural way.

Taking the previous work one step further, [Vieira et al., 2008] and [Vieira et al., 2010] closely followed the genetic inheritance processes from the biological perspective, where each child chromosome holds a copy of mother's and father's chromosome. During the reproduction process, child chromosomes are duplicated, combined and then split into four "gametes". Through the process called "fecundation" a father gamete and a mother gamete are combined in order to produce the new child's chromosomes. The parents' gene values from the chromosome are combined to visualize the final value. This approach naturally models biological inheritance and it can be used to study the distribution of dominant or recessive genes.

In our approach of avatar generation, we do not follow the biological evolution like [Vieira et al., 2008], rather we apply approaches from genetic algorithms. Similar to [Lewis, 2000], we encode visual properties to genes, which form chromosomes. We use genetic operators to generate new, unique individuals. Our contribution is the definition of new techniques applied during replication, such as deep inheritance, which provides the possibility of inheritance of features from our far ancestors. Another contribution is the definition of genotype rules, which provide better control over features of generated individuals. Using geno-

type rules, we can define key ethnic features that have to be preserved during replication, or we can define dependencies between visual features, e.g., making an avatar fat. As a result our work allows the generation of ethnic crowds with a high degree of variation across hundreds of body features, while also maintaining the similarity with avatar ancestors in a classical genetic sense.

2.8 Summary

In this chapter:

* We have presented the background information on virtual worlds and explained their parts related to our research, such as avatars and content creation. We have also introduced Second Life virtual world and open-source virtual world platform OpenSimulator.
* We have presented all concepts related to the use of virtual worlds as hybrid multi-agent systems, specifically Organization-Centered Multi-Agent Systems (OCMAS).
* We have introduced a specific OCMAS called Electronic Institutions.
* We have introduced Virtual Institutions and explain the reasons why they are of our particular interest.
* We have introduced shape grammars, explained their use and related work.
* We have presented the background on Artificial Life and Intelligent Virtual agents fields and gave details of state-of-the-art related to our research.

In the next chapter, we introduce a new shape grammar framework, which includes the algorithm for the real-time sub-shape detection. Then, we present our Shape Grammar Interpreter (SGI), a general interpreter for generating rectilinear forms.

Chapter 3

Shape Grammar Interpreter (SGI)

In the previous chapters, we have explained the concept of shape grammars and presented our motivation for using this technique for automatic generation of normative virtual worlds. But existing shape grammar focused approaches are not immediately suitable for being applied to Virtual Institutions, are lacking means for creating general-purpose interactive designs and are not sufficiently supported with practical implementation tools. So, in this chapter, we deal with the objective of designing a new shape grammar framework having a real-time algorithm for sub-shapes detection, used during the execution of shape grammars, and its implementation in a general shape grammar interpreter, named SGI. We introduce the architecture of this framework and provide a performance evaluation of proposed algorithms, showing a significant gain in performance over previous algorithms.

3.1 Motivation

Shape grammars play a vital role in a new generation of tools for the analysis and design of products (see Section 2.5 for details on shape grammars). In computation, they represent a class of production systems that generate geometric shapes or designs [Stiny and Gips, 1972]. Instead of using sequenced instructions as a basic unit of computation, production systems use unordered and data-sensitive rules called production rules.

Shape grammars are capable of representing knowledge about both the shape of a product or design and represent knowledge about the functionality. Functionality knowledge is represented by either labels or markers, and by *parametric shape grammars* in advanced scenarios [Tapia, 1992]. Additionally, shape grammars generate forms not previously defined, i.e emergent shapes. Using sub-shape detection it is possible to use emergent shapes in the generation process.

Sub-shape detection is a vital function in the application of shape grammars during design synthesis tasks or design classification tasks.

Our interest is centered on synthesis problems, which involve the generation of geometrical designs. While many existing shape grammars (e.g. Palladian grammar [Stiny and Mitchell, 1978], Prairie houses grammar [Koning and Eizenberg, 1981]) were designed and evaluated "by hand", this is not adequate for synthesis tasks. These tasks require generation of many designs and selecting the best candidates. Thus, it is practical to generate shape grammars using a computer implemented interpreter.

Up until now, there have been numerous attempts to create a general shape grammar interpreter, but most of the existing tools are either highly specific in their purpose, have a limited functionality, do not include sub-shape detection, sub-shape detection is slow, or they are programmed for a single operating system.

Therefore, we designed a new shape grammar framework, which includes an efficient algorithm for sub-shapes detection used in the shape grammar generation process, and implemented in our Shape Grammar Interpreter (SGI). This framework allows designers to automatically synthesize designs and to actively participate in the generation process.

Potential applications of this research can be found in the educational field (i.e. architecture and arts) and in the automatic generation of architectural, mechanical and product designs. Also, universities use SGI to introduce the shape grammar concept (e.g. shape grammar classes in Carnegie Mellon University[1]). In Chapter 4, we introduce the Virtual World Grammar, a shape grammar extension for the intelligent generation of 3D virtual worlds.

3.2 Implemented Generation Algorithms

In this section, we describe two algorithms for the generation of designs using shape grammars, which are included in our shape grammar framework. They represent two different execution protocols, which manage both rule selection and execution during the design generation process. First, we present a simpler implementation using tree structures. This approach does not use emerged shapes. Then, we introduce our sub-shape detection algorithm, which is a modified version of the algorithm proposed by [Krishnamurti, 1981]. We evaluate these algorithms in Section 3.4.

3.2.1 Tree-Search Based Algorithms

The *tree-search* mechanism stores a state of the generation process in a tree structure and uses traditional tree-search algorithms, i.e., breadth first and depth first, to find the next rule to apply. This mechanism does not detect emerged shapes, only uses the knowledge of shapes used in the left and right side of the rule. The tree structure holds the generation execution state. Each node

[1] `http://www.andrew.cmu.edu/course/48-747/subFrames/schedule.html` (05/2005)

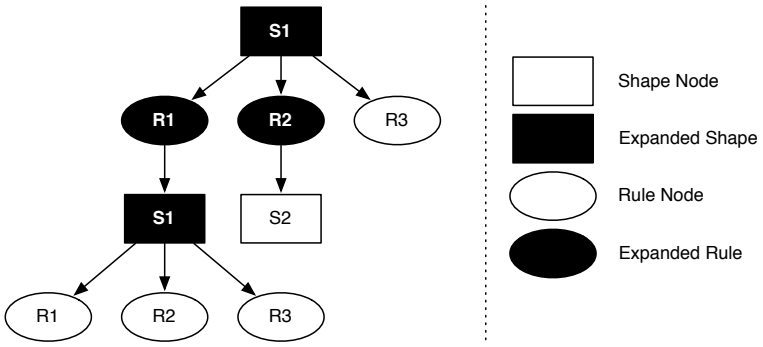


Figure 3.1: An example of execution tree using tree-search, breadth-first search protocol

represents either a shape or a rule. When a shape node is expanded, it has as many children as there are rules with this shape on the left side of the rule. Figure 3.1 shows an example of such a tree. Rectangles represent execution states defined by shapes, while ovals represent execution states defined by rules. The black-colored nodes of the tree in Figure 3.1 have been expanded, i.e., executed. We can see that rule-based nodes have 0 or 1 children, depending whether they have been executed or not. A child represents the right side shape of the rule. To select the next rule to execute, the framework can use strict breadth/depth first search, or it can decide randomly.

The advantage of this approach is that time and space complexity of finding a next rule to execute is the same as finding a tree node using the breadth-first or depth-first search algorithm. The time and space complexity of breadth first search algorithm is $O(b^d)$, where b is a maximal branching factor and d is the depth of the tree. For depth first search, the time complexity is the same $O(b^d)$, but the space complexity is only $O(d)$.

Parameterization of the Generation Process Using Tree-Search Mechanism

The following input parameters can be introduced to the tree-search mechanism to affect its generation process and output:

- *Level of randomization* - defines the probability of using currently selected node. The framework provides five different levels of randomization (none, low, medium, high and extreme). Level "none" represent strict breadth/depth first search. All other levels decide the probability of using the currently selected node in the generation process (e.g if high randomness is selected, there is little probability of using the selected node). Randomization is used to better explore the execution tree.
- *Number of iterations* - an *iteration* is the execution of a shape grammar

rule in the current shape. Number of iterations indicates how many rules will be executed during the current generation process.

- *Step iteration* - executes only one iteration (i.e. one shape grammar rule is executed).
- *Next shapes* - returns a list of possible shapes after the application of shape grammar rules in one iteration, in the current shape. One of them is selected to proceed in the generation process.
- *Use of markers* - we can decide if we want to use markers or no.

The use of randomization and markers allows a user to browse the design space by generating multiple designs, and select only those he likes. Figure 3.3 shows two different outputs of the generation process for the shape grammar depicted Figure 3.2 using the breadth-first search algorithm. Figure 3.3a) shows the output for 15 iterations while Figure 3.3b) for 40 iterations, where the shape with the lighter color shows the shape from 15 iterations.

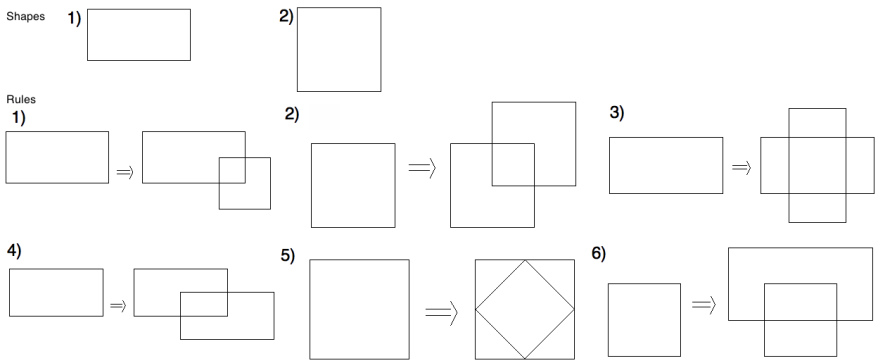
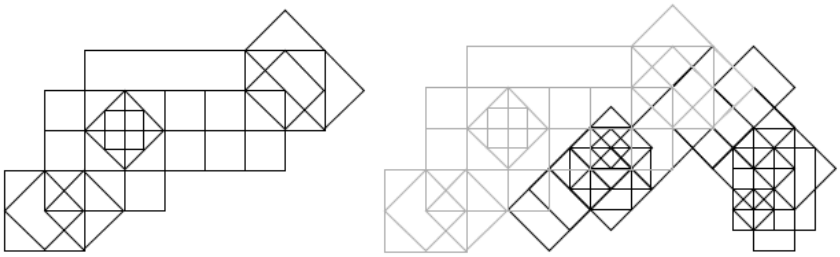


Figure 3.2: Shape grammar used in the generation process presented in Figure 3.3

3.2.2 Subshape Detection Algorithm

The Shape Grammar Interpreter (SGI), presented in this chapter, implements a modified version of Krishnamurti's algorithm for sub-shape detection [Krishnamurti, 1981]. This algorithm processes rules as the affine transformations of a left side of the rule to the right side of the rule.

Krishnamurti's algorithm is capable of solving most of the problems related with sub-shape detection, but it also has some limitations. It misses the detection of infinite sub-shapes (such as finding line sub-shapes in line segment); thus, it processes only shapes with at least three points. However, the biggest drawback of this algorithm is its performance as any three points of the sub-shape are transformed to any three points of the input shape. To overcome this



a) b)

Figure 3.3: Two examples of designs generated using the tree search algorithm using the shape grammar in Figure 3.2. Outputs for a) 15 iterations and b) 40 iterations.

performance weakness, we have modified the algorithm by allowing detection of sub-shapes with at least one intersection point. This provides a real-time capability of rendering designs using sub-shape detection. This algorithm works with the *maximal shape*, *maximal line* and *viable intersections* concepts which we explain next.

Maximal Shape and Maximal Line

A maximal shape is a new shape created from the original one using its maximal lines, which is the minimum set of lines maintaining the original form of the shape (function $CreateMaxLns$ in Algorithm 1). Figure 3.4 shows how the maximal shape is created by joining lines into maximal lines. In the original shape 1) lines a and b, and c and d are joined by the algorithm to maximal lines e and f to create a maximal shape 2). This allows the algorithm to work with a minimal set of intersections and also to detect correctly if a subshape is within the boundaries of an original shape (as explained in Section 3.2.2 below). The time complexity of finding a maximal shape is $O(n^2)$, where n is the number of lines of a shape.

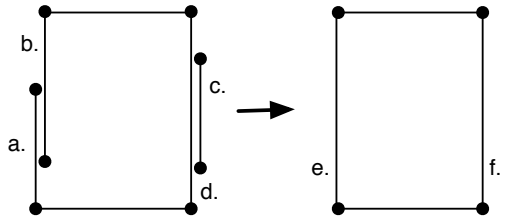


Figure 3.4: Maximal lines: 1) original shape 2) maximal shape

Viable Intersections

A viable intersection is any inner or outer intersection of two segments of the shape (this process is done by the function *CreateInts* in Algorithm 1). By outer intersection, we mean an intersection of two segments that is positioned on the lines containing the segments but outside of the boundaries of at least one of these segments. The algorithm finds all intersections by checking each endpoint with all other endpoints. This can be done with the time complexity of $O(n^2)$, where n is the number of end points of an input shape.

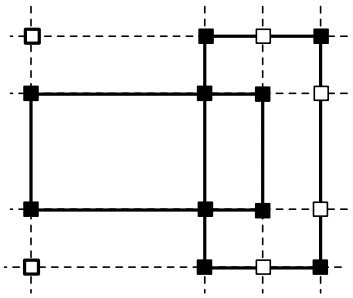


Figure 3.5: Intersections

In Figure 3.5, we see all viable intersections of the shape. Internal intersections are shown as black boxes, external as white boxes and dotted lines display the lines containing the segments.

Algorithm

In this section, we use a simple example to explain the execution of the algorithm and what modifications we have performed. Our modified version of Krishnamurti's algorithm is presented in Algorithm 1. In this example, we detect a *subShape* in an *inputShape* displayed in Figure 3.6 1) and 3.6 2) respectively.

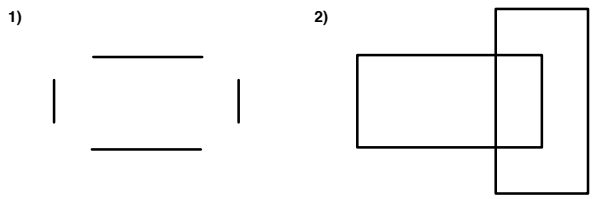


Figure 3.6: Algorithm input: 1) *subShape* 2) *inputShape*

In the first step, maximal shapes (sets of maximal lines) are created from both, input shape and subshape, as presented in Figure 3.4. When this process is finished we find all the viable intersections in both shapes as shown in Figure

Algorithm 1: Subshape detection algorithm

```
Input: inputShape, subShape
Output: Collection of subshapes
begin
    maxLines ← CreateMaxLns (inputShape)
    subMaxLines ← CreateMaxLns (subShape)
    inters ← CreateInts (maxLines)
    subInters ← CreateInts (subMaxLines)
    transfs ← FindTransfs (subShape, inters, subInters)
    forall transfs do
        if ∀subMaxLines ⊆ maxLines
        then subShapes ← TransShape (subShape)
    return subShapes
end
```

3.7. Figure 3.7 emphasizes the importance of finding external intersections as without them the subshape 1) would not be detected in input shape 2).

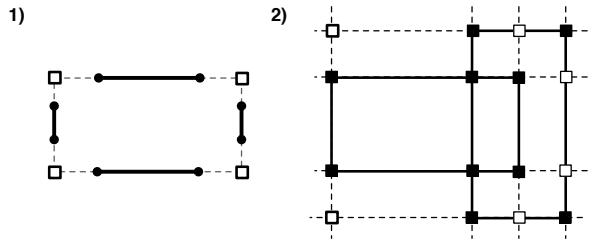


Figure 3.7: Intersections: 1) *subShape* 2) *inputShape*

The most time-consuming part of the algorithm is to find the correct transformations (function *FindTransfs*) of the *subShape* to the *inputShape*. In the original version of the algorithm, any three points are taken from the *subShape* to create the transformation to any three points in the *inputShape*. This transformation is used to check if the remaining points of the *subShape* are transformed to some points of the *inputShape*. This search space is exponential to the amount of intersections in the *inputShape*. Our proposal reduces this search space by using *intersection triplets*, that is a structure containing an intersection point, two guiding points, the angle and the ratio of lengths between related segments. Figure 3.8 shows an example of an intersection triplet. The intersection point is represented as a white box and guiding points are represented as black boxes. The angle is calculated using the containing lines of the two segments, and the ratio is obtained from the lengths of the segments. To find all triplets means to select all combinations with three members from the set of intersections. This can be done in time $O(3!\binom{n}{3})$, where n is the number of found intersections.

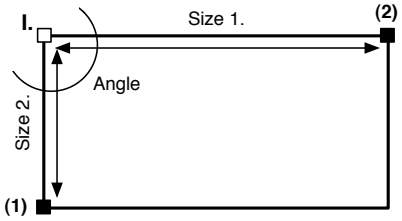


Figure 3.8: Intersection triplet

The process works as follows: first, we find triplets in the *subShape*, order them by the angle (ordering a set can be done in time $O(nlog(n))$) and select the one with the smallest angle. Second, we start exploring triplets in the *inputShape*. All triplets that do not have the same angle and the same ratio are thrown away, while the *good triplets* ones are stored. For each of the stored triplets, we find an affine transformation, and check if the remaining points of the *subShape* fall onto points of the *inputShape*. We store all the transformations satisfying the condition and discard the rest.

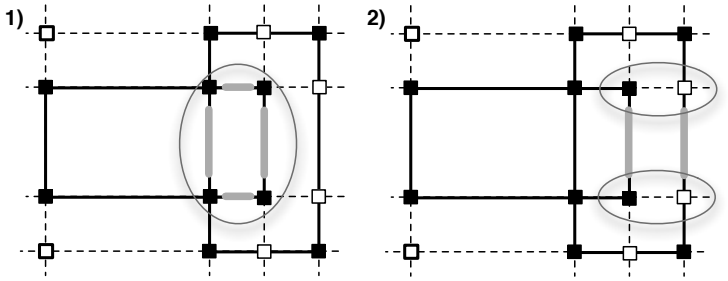


Figure 3.9: Boundary detection: 1) Passing detection 2) Failing detection, missing boundary

In the last step, we check if the transformed maximal lines of the *subShape* fall within the boundaries of the maximal lines of the *inputShape* (function $TransShape$ in the algorithm). Figure 3.9 shows the passing condition 1) in the left part of the shape and a failing condition 2) in the right part. In case 2) we see that we could find the transformation of the points of the *subShape* to the *inputshape*, but the test on the boundaries fails. In 1) the thick gray lines represent the detected subshape and in 2) we see subshape's top and bottom line missing. The white boxes represent outer intersection points, and the black boxes are inner intersection points.

Figure 3.10 depicts the flow chart of the subshape detection process. It starts with the maximalization (having time complexity $O(n^2)$) of both the subShape

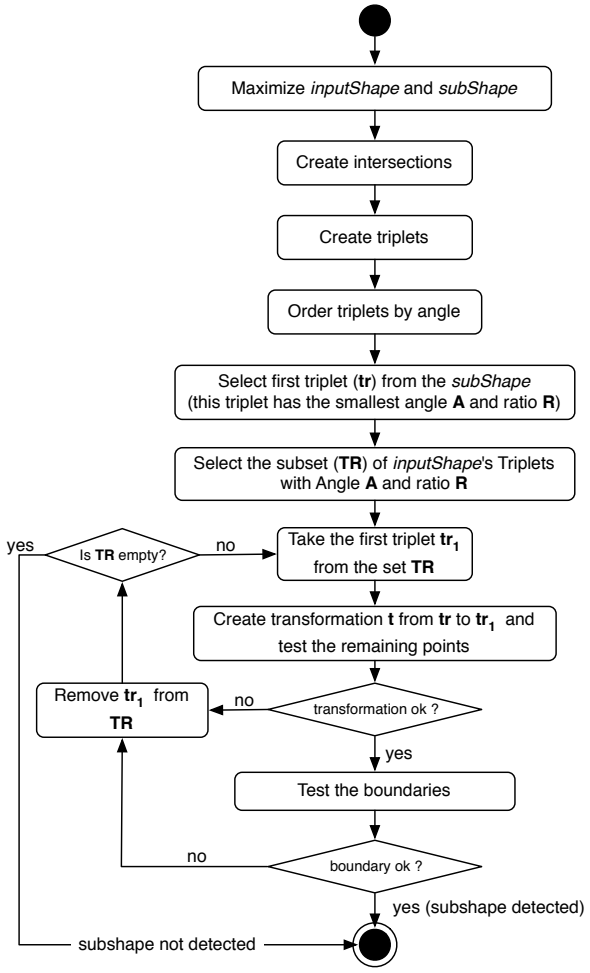


Figure 3.10: Flow chart of the subshape detection process

and inputShape, continues by finding intersection points ($O(n^2)$) from which it creates triplets ($O(3!\binom{n}{3})$) and orders them by the angle ($O(nlog(n))$). The algorithm then selects the subShape triplet with the smallest angle and ratio and filters the set of inputShape triplets according to this angle and ratio ($O(n)$). Then, for each triplet from the filtered set a transformation is created onto subShape triplet, and using this transformation, we test the remaining points ($O(3!\binom{n}{3})$). The correct transformation is used to test the subShape boundaries in the inputShape. If the boundary test passes, the algorithm can finish, otherwise it removes the current triplet from the set and continues with the next one, until the set is empty. Given presented complexities, the worst-case time complexity of our algorithm is $O(k!\binom{n}{k})$ where n is the number of shape's inter-

sections and k is the number of intersections, we use to create a transformation. In our case, we create transformations for triples, thus $k = 3$.

In the worst-case scenario, this algorithm has the same complexity as the original algorithm, although this occurs only when all the angles of the shape and all the triplet's ratios are the same, and shapes tend to have different angles. In the standard scenario, we have significantly improved the performance of the sub-shape detection algorithm. Original algorithm has to test $3!\binom{n}{3}$ combinations (3 is the number of points used to create the transformation and n is the number of intersections of the *inputShape*). In the example from Figure 3.5, original algorithm tests 3360, while with the proposed modification it tests only 48 combinations. Section 5.5 provides a more detailed evaluation of performance in different iterations of the generation process.

3.2.3 Parameterization of Generation Process Using Subshape Detection

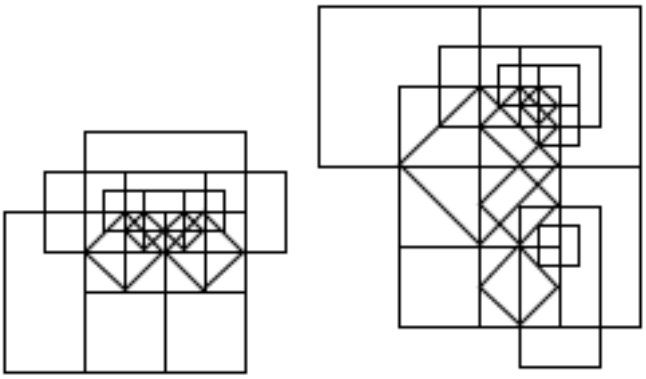
Most of the input parameters of subshape detection are the same as in tree-search algorithm (see Section 3.2.1). They are the *level of randomization*, *number of iterations*, *step iteration*, *next shapes* and *use of markers*. The parameter *level of randomization* now decides the probability to use the currently found subshape. If this sub-shape is not used, the algorithm tries to search for another one. For sub-shape detection we have introduced a new parameter *size of the search space* which limits the search space by defining how many sub-shapes should be checked and returned in case of the "step iteration".

The use of randomization and markers gives us freedom to browse the design space by generating multiple designs and selecting only those designs we like. Limiting the search space increases the performance of the application. Figure 3.11a) and 3.11b) show two different outputs during random generation in 14 and 20 iterations. These outputs were produced using the shape grammar defined in Figure 3.2.

3.3 Shape Grammar Interpreter

We have focused on bringing a complete and robust framework that allows the user to interactively specify any shapes and rules and also to have a complete control over the design generation process. The developed Shape Grammar Interpreter (SGI) is generic in the sense that it allows to create and process any shape grammar with full support of *labeled rules* and *subshape detection*. Another important aspect of SGI is its object-oriented design that allows future programmers to easily extend current functionalities.

Great effort has been devoted to providing an interactive way of defining shapes, which are used in shape grammar rules and designs' generation process. SGI implements two different types of algorithms for the generation of designs. First, tree-search algorithms, which store the state of the generation process in a tree structure and use traditional tree-search algorithms to find the next rule



a) b)

Figure 3.11: Generation process using the subshape detection algorithm. Outputs for a) 14 iterations and b) 20 iterations using the shape grammar in Figure 3.2.

to apply. Second, and most importantly, SGI includes our sub-shape detection algorithm. Hence, sub-shapes of the existing shapes can be detected and used in the generation process obtaining not only a wider set of designs but potentially more appealing ones.

3.3.1 Framework description

We offer SGI as open-source[2] (distributed under GNU license) to provide possibilities of joint development in the shape grammar community.

The user creates shapes by drawing them on the canvas using a mouse. Rules are operated in similar manner, the user creates rules by specifying the spatial relation among shapes either parametrically or by the mouse. All modifications of the current grammar are persisted in a XML file for future use. Appendix B shows an example of such a XML file containing the definition of a simple shape grammar.

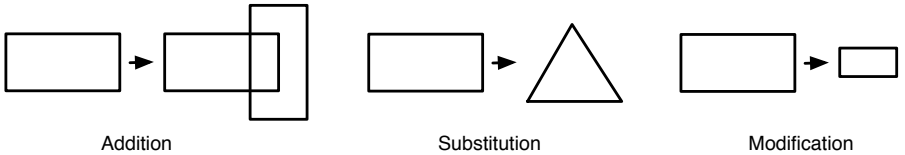


Figure 3.12: Three shape grammar rule types supported by SGI.

[2]http://sourceforge.net/projects/sginterpreter/

SGI supports the following types of rules (depicted in Figure 3.12):

- *addition*: adds a new shape in the right part of the rule in spatial dependency to another shape which is in the left part of the rule
- *substitution*: substitutes an existing shape by another shape
- *modification*: modifies the proportions of an existing shape

The framework contemplates nondeterministic shape grammars, characterized by the possibility of applying several rules in one generation step. It implements several mechanisms, also referred in this chapter as execution protocols, to select a candidate shape and the rule to proceed in the generation process. These mechanisms were described in Section 3.2, they are *tree search* and *subshape* detection algorithms. These algorithms can be further parameterized as described in Section 3.2.1 and Section 3.2.3.

3.3.2 SGI Architecture

In this section, we present the architecture of SGI. It is programmed in Java using the Eclipse Rich Client Platform (RCP) framework. RCP is based on bundles, that is a dynamic plug-in model. Using bundles it is easy to add or remove parts in the system. In the Chapter 4, we describe the extension of SGI called Virtual World Builder Toolkit, that uses the Virtual World Grammar concept to generate 3D virtual worlds.

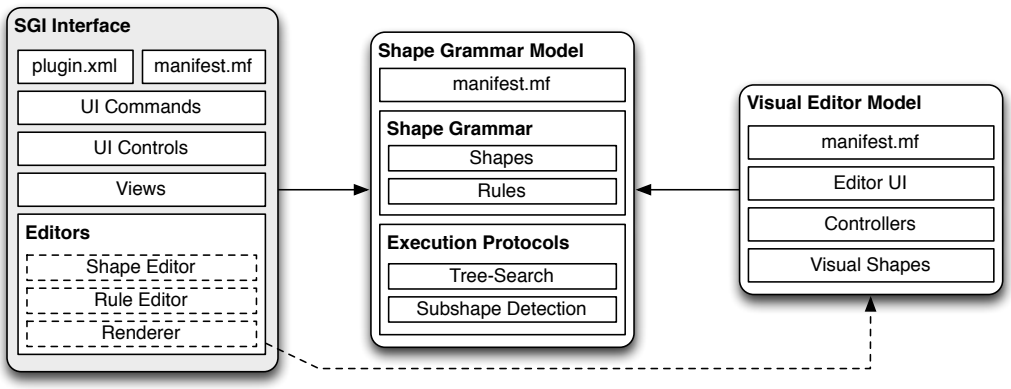


Figure 3.13: SGI Architecture

With RCP, programmers can build their own applications on existing platforms by extending the existing systems with plug-ins. To create the Shape Grammar Interpreter, we implemented three plug-ins. Figure 3.13 shows these three plug-ins, their components and relations between them. We see, that SGI system respects the Model-View-Controller (MVC) architecture.

The model is implemented as a stand-alone plug-in, *Shape Grammar Model*, where shape grammar and its parts are defined (i.e shape models and rule models). Moreover, the model contains the definition of *Execution Protocols*, that use *tree-search* and *subshape detection* algorithms to execute the grammar. If a developer wants to include a new execution protocol, he needs to define a new plug-in that adds this functionality into the model. Solid lines in Figure 3.13 represent dependency, while dashed lines represent inheritance.

Visual Editor Model defines controllers and views for the *Shape Grammar Model* parts. It also defines a visual control, an editor, that allows to visually define and manipulate *Visual Shapes*. Keeping it in a separate plug-in provides the possibility to replace the visual editor with another one. The *Shape Editor*, *Rule Editor* and *Renderer* extend the functionality of the *Visual Editor* and facilitate modifications of shapes, rules and display rendering results.

The core of the Shape Grammar Interpreter is implemented as a bundle that extends the RCP and depends on the Shape Grammar Model and *Visual Editor Model*. Within this plug-in we define all the visual controls and actions. Several views (e.g. render chain view, shape list view) facilitate the understanding of the design generation process.

Using the architecture of the Eclipse bundles (based on OSGi framework) we allow developers to easily extend current functionalities by adding new plug-ins. These plug-ins can use or extend the functionality of the SGI core.

3.3.3 SGI User Interface

Figure 3.14 displays the SGI graphical user interface. It was designed to provide an intuitive user experience. The main focus was put on the interactive definition of shapes and rules by using the mouse. Shapes and rules are defined on separate canvases giving the SGI user complete control and overview of ongoing work. Figure 3.14 also points to the most important parts of SGI:

1. The list of all the existing shapes and rules of current grammar that can be operated either by context menu or by double-clicking, which opens the currently selected item in the *editor area*.

2. The main application toolbar that dynamically changes depending on the currently selected editor and that includes standard input/output functions such as *load grammar* and *save grammar*, as well as editor functions, such as *create line*.

3. The outline view allows the user to (i) preview currently modified shape in the shape editor and to (ii) preview a rule in the standard shape grammar rule form.

4. The Editor area is where the user can modify the selected element, that is either a shape or a rule.

5. The renderer is used to visualize generated designs using the current grammar. The generation is done in real time and the result can be exported to

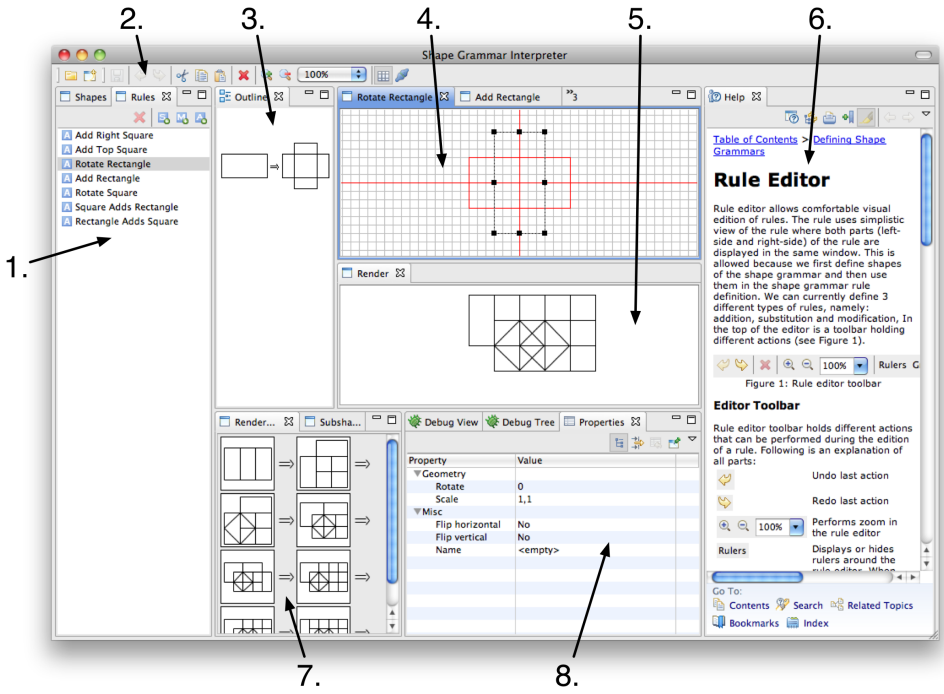


Figure 3.14: SGI: User Interface

different graphics formats (e.g. bmp, jpg). The user selects parameters of generation such as *randomization* level or number of *iterations*. The renderer also allows to render designs step by step and observe the emergence of shapes.

6. The dynamic help view provides to the user help on how to use currently selected part of the SGI interface.

7. The render line view shows current derivation line. It provides the user the possibility to trace the execution of the shape grammar, i.e how he obtained the current design. In this part, during subshape detection, we can also display all possible next shapes and select the next one to use.

8. The list of all the properties of the currently selected item. A property of the item is, for example, its position (in [x,y] coordinates) or name. The property list allows the user to manually set these values.

Figure 3.15 shows the rule editor with an outline view where we see the left and right side of a rule. In the bottom part of this window, we see a derivation using this rule. Figure 3.16 shows the rule editor with a shape's outline. In the bottom part of this figure, we see a list of possible next steps of generation using the subshape detection algorithm.

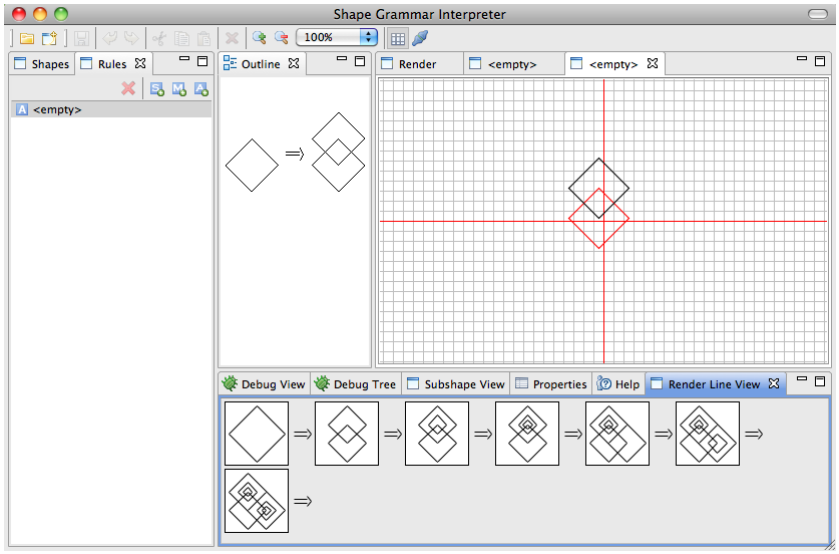


Figure 3.15: SGI: Derivation process view

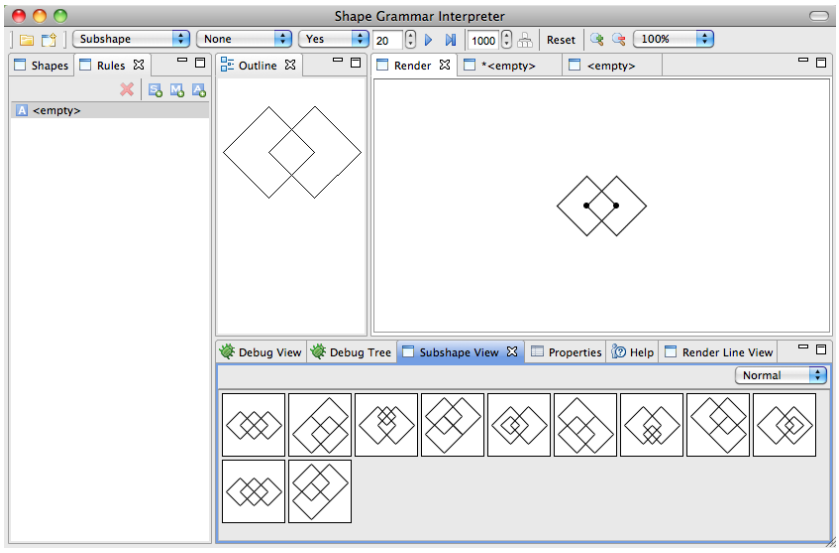
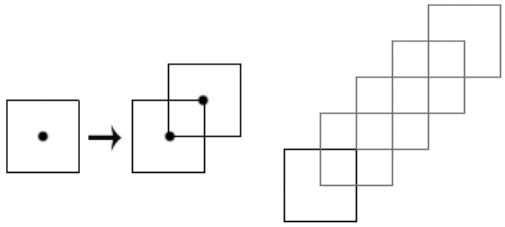


Figure 3.16: SGI: Subshape detection view

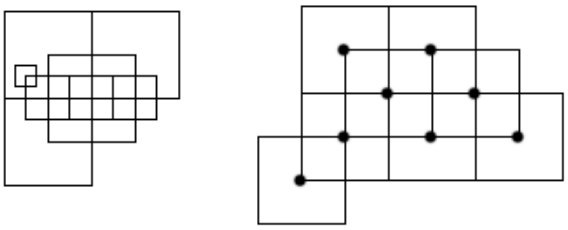
Next, we present results of design generations using SGI. Figure 3.17a represents a spatial rule, where we add another square to an original square, creating a new square in their intersection. Figure 3.17b shows the result of the generation using a tree search protocol. This generation respects the original shape and its orientation, creating a simple design.



(a) (b)

Figure 3.17: a) Definition of rule b) Generated design using tree search protocol

Figure 3.18a shows the result of a generation using the simple rule from Figure 3.17a with *subshape detection* and *without markers*. Figure 3.18b depicts the result of the generation with subshape detection and using markers. Please observe that it generates much more shapes creating various, complex designs from this one simple rule. The use of markers limits the number of emerged shapes that the subshape algorithm may find. When placing a marker on some shape, the generator not only has to find the subshape in the current shape but also has to find this marker on the same position in the detecting shape. In Figure 3.18b, we see that the missing presence of a marker in the center of an emerged shape does not allow us to use these smaller rectangles in the generation process, and it is limited to those having a marker in its center.



(a) (b)

Figure 3.18: Generated design using rule from Figure 3.17a using subshape detection. a) Without markers b) With markers

3.4 Evaluation

In this section, we evaluate the performance of the Shape Grammar Interpreter (SGI) using the generation algorithms presented in Section 3.2. All measures were taken using a generation with medium level of randomization and aggregated in several iterations. First, we take a look at the tree generation, and then we present the performance of the sub-shape detection algorithm.

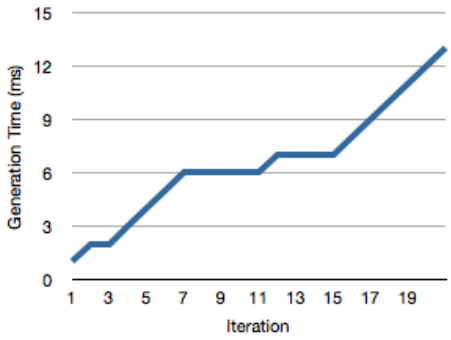
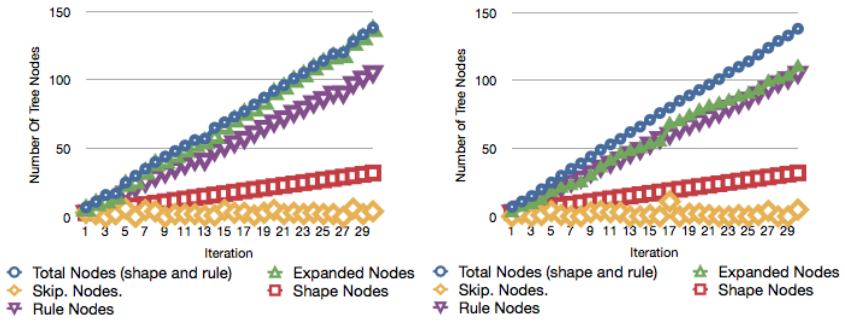


Figure 3.19: Aggregated generated time using the breadth-first search algorithm

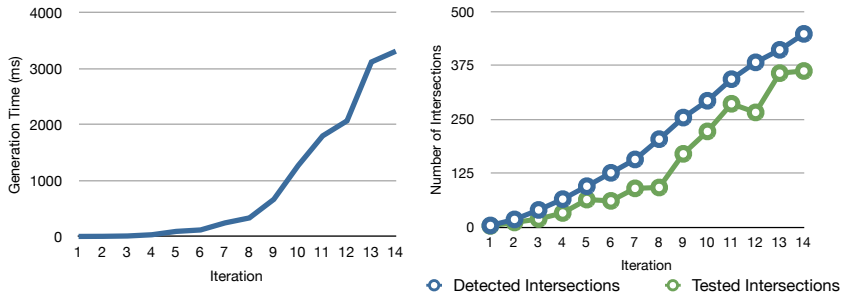
3.4.1 Tree Search Algorithm

We have used basic breadth-first and depth-first search algorithms with possibility of randomization. Therefore, our algorithm inherits time and space complexity of these algorithms. Figure 3.19 contemplates the graph displaying the execution times of the generation process for twenty iterations. As we can see, each iteration is executed in constant time of one millisecond, making it powerful yet simple generation process. Figure 3.20a) shows the different numbers of tree nodes generated during the breadth-first search algorithm execution, while Figure 3.20b) shows the same for depth-first search algorithm. In the breadth-first algorithm, the number of expanded nodes (Figure 3.1 shows an example of a partial execution tree with shape and rule nodes, as well as with expanded and not expanded nodes) is almost equal to the total number of nodes, meaning that we always first check all nodes in one level of the tree, before proceeding to the new level. On contrary, the depth-first algorithm expands the tree to its depth following always the leftmost node. As a result, we see in the Figure 3.20b) that the number of expanded nodes is almost equal to the number of rule nodes. The number of skipped nodes relates to the level of randomization, when we randomly decide if we use the currently selected node.



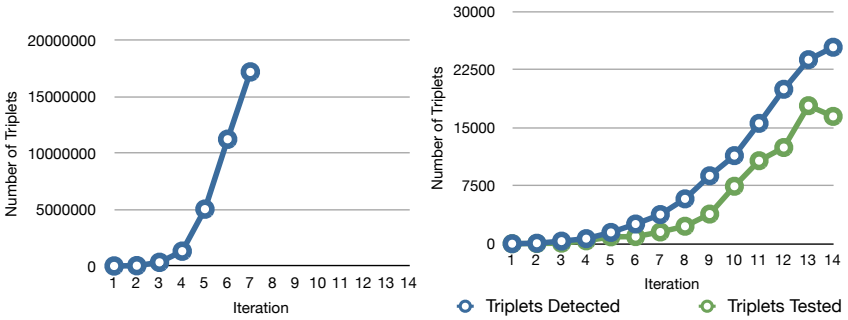
(a) (b)

Figure 3.20: Numbers of tree nodes during the generation process using (a) the breadth-first or (b) depth-first algorithm for different iterations



(a) (b)

Figure 3.21: a) Time measure of shape grammar generation using the sub-shape detection algorithm and (b) number of detected and tested intersections



(a) (b)

Figure 3.22: Sub-shape detection: numbers of triplets detected and tested during the generation process using (a) the original Krishnamurti's algorithm and (b) our proposed algorithm

3.4.2 Subshape Detection Algorithm

In this section, we focus on performance measures of the sub-shape detection algorithm, and we also present the differences between the original by [Krishnamurti, 1981] and the proposed algorithm. Figure 3.21a) shows the execution time of the sub-shape detection algorithm in different iterations. We can see that the time rises exponentially but in a steady manner. Figure 3.21b) shows the number of intersections that were detected in the current shape in a given iteration. These two figures allow us to study how the execution time raises depending on the number of detected intersections.

Figure 3.22a) shows the number of triplets that would have to be tested using the original approach. We see that in the seventh iteration, we would have to test more than 17.000.000 triplets. Figure 3.22b) show the number of triplets that we have to test using our approach. In comparison to the original approach, in the seventh iteration we possibly would test only 5.000 triplets. The grey line

in Figure 3.22b) also shows how many triplets were tested in order to detect the first sub-shape. We see that in the tenth iteration they were only 7.500.

In conclusion, we can say that tree-search based algorithm is much faster, as the time of each iteration step is nearly 0 ms, while the time for the sub-shape detection is growing exponentially with every step. Our proposed change in sub-shape detection algorithm allows efficient execution (under 4 sec) for shapes with around 400 intersection points.

3.5 Summary

In this chapter:

* We have presented the generic shape grammar framework and our Shape Grammar Interpreter, which supports the interactive creation and execution of rectilinear designs.
* We have presented the optimized version of sub-shape detection algorithm allowing fast detection of sub-shapes.
* We have presented the architecture of SGI that allows plug-in extensions.
* We have evaluated the performance of SGI tool.

In the next Chapter, we introduce the Virtual World Grammar, our proposed extension to shape grammars, which allows the automatic generation of 3D virtual worlds designs from the formal specification of activities performing in these virtual spaces. The Virtual World Builder Toolkit (VWBT) implement the Virtual World Grammar concept. VWBT is implemented as an extension plug-in of SGI.

Chapter 4

Virtual World Grammar (VWG)

In the previous chapter, we have introduced the concept of shape grammars and we have presented a Shape Grammar Interpreter (SGI) for the generation of rectilinear designs. In this chapter, we propose a mechanism, inspired by shape grammars (see Section 2.5) and virtual world (see Section 2.1) paradigms, named Virtual World Grammar (VWG) that aims to automatically generate a 3D virtual world from a formal description of activities taking place in the virtual space. We also present the Virtual World Builder Toolkit which implements the VWG generation process and provides a graphical user interface to define and execute VWG components. Finally, we present some generated designs and give performance measures. This chapter uses the e-auction house example, introduced in Section 1.6.1, to explain the generation process.

4.1 Motivation

In Chapter 1, we presented our motivation for deploying e-* applications and social simulations in 3D virtual worlds. Such applications often represent dynamic and complex processes, where frequently we need to either create a new application scenario or change/adapt the existing one. Every new scenario and every change requires a manual update of virtual world model so that virtual world creation is a time-consuming task for designers and makes it difficult to manage its dynamic update at runtime. The latter issue is crucial for us as we are interested in the dynamic nature of virtual worlds. Thus, we need an efficient method to generate and update 3D virtual scenes in an automatic way.

We propose to automatically generate virtual world designs from the formal specification of activities taking place in the virtual space. In this approach, we advocate the use of normative virtual worlds, which structure participant interactions depending on an organizational specification, which establishes roles,

tasks (activities) and interaction protocols. Using such specification, we map activities to virtual spaces and actions to interactive objects or gestures. Using this mapping, we generate a full-scale interactive model of the virtual world. An example is mapping the Auction scene, introduced in the e-auction house example, to a virtual auction room and saying a "bid" message to the gesture of raising a hand or clicking on the auctioneer, represented by his avatar. However, such mapping of activities to virtual objects does not contain spatial information of objects in the virtual world. Encoding spatial information directly into shapes leads to scaling problems (e.g. adding a new activity to current spatial configuration triggers space redistribution among other activities). Thus, we need a mechanism that can automatically position objects creating a floor plan of a virtual world design.

A popular computational design technique also used for analysis and synthesis of floor plans are shape grammars [Stiny and Gips, 1972]. Shape grammars have proven to be an adequate approach for generating architectural structures [Stiny and Mitchell, 1978] [Koning and Eizenberg, 1981]. While there exist the possibility of manipulating directly 3D shape grammars, we take approach similar to [Duarte, 2001], and first generate the 2D floor plan and then translate this model to 3D. This approach allows us to generate many different designs and then select the desired one. Shape grammar generation process uses shapes and rules, where right-side shape of the rule replaces the left side shape of the rule. Generated designs contain geometric forms.

Generated forms contain only spatial information, what is not sufficient for transforming a 2D design into 3D. We need a way to specify semantic data about shape components (e.g. if a line represent a wall, we need to know its height or texture). Also, shape grammars do not contain any mechanism of controlling the execution of the generation process and discarding the invalid designs already during generation (e.g. we do not want to generate intersecting forms). Another requirement is to place objects in a specific order, or on related locations (e.g. place together activities that are executed in sequence).

To overcome just presented limitations, we propose an extension to shape grammars named Virtual World Grammar (VWG). A VWG automatically generates a 3D virtual world from the specification of activities. It contains mechanisms for providing semantic data to geometrical forms, mapping these geometrical forms to specification activities, generating new designs with specified logic and validating this design during the generation process as well as in the end. The output of a VWG is a 2D floor plan, which is transformed by a specific 3D transformation engine to a 3D model. Currently we can generate designs for Open Wonderland, Second Life and OpenSim based virtual worlds.

4.1.1 Motivation Example

To introduce the Virtual World Grammar mechanism, we follow the e-auction example introduced in Section 1.6.1, which is a hybrid system that combines 3D virtual worlds and multi-agent systems (MAS) technologies. Figure 4.1 contemplates an overview of our approach, which facilitates the generation of such type

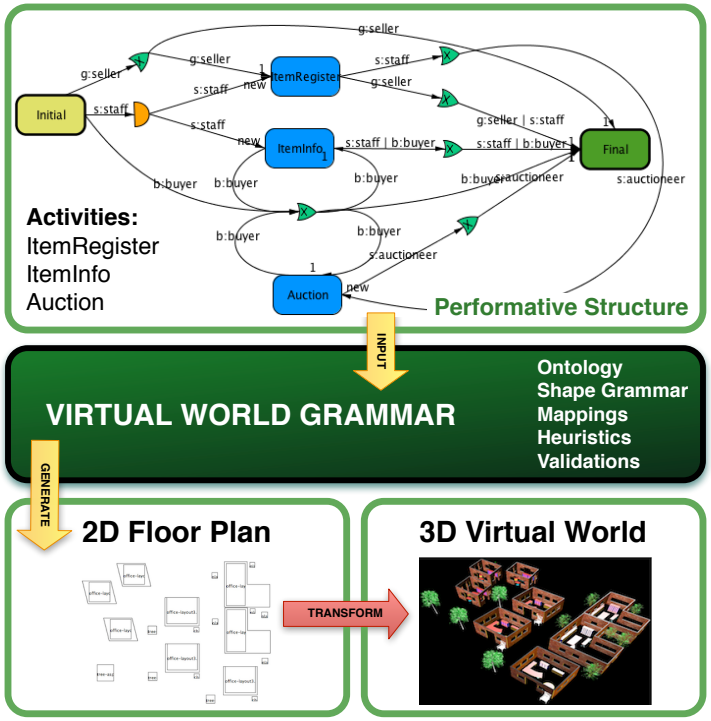


Figure 4.1: 3D virtual world generation process

of hybrid environments out of a formal specification. In particular, we focus on the generation of a Virtual Institution (see Section 2.4) from its performative structure. A specification of the Auction House Virtual Institution is depicted in the top rectangle of Figure 4.1. Rounded rectangles represent *activities* (also called *scenes*). In this performative structure, we see the following scenes: Item Registration, Auction and Auction Info. The initial and final scenes represent the institution entrance and exit points, which are mapped to the entry and exit of the generated 3D VW. In the proposed e-auction house scenario, there are many "Auction" scenes, and their number depends on currently active auctions.

To apply a VWG, we specify object types and their properties from both the performative structure and the shape grammar, which form a general vocabulary, that is the *ontology* of the Virtual World Grammar. For each selected object from the specification and all shape grammar shapes, we create an *instance* of the related ontology object. Then, we specify *mappings* that define, which shape grammar object can represent which specification instance. In Virtual Institutions, we are focusing on activities that can be mapped to virtual spaces, such as stand-alone buildings or rooms in an institution building.

When we have successfully defined the ontology, created all instances of specification and shape grammar objects, and defined the desired mappings, we can

proceed to the generation of a VW design. This is where *heuristics* take a decisive role. They decide the next specification element to process and the applicable rule of the shape grammar for the selected specification element.

To make sure that we are generating functional and correct designs, we use *validations* during every step of the generation. We can also evaluate the final design. For example, we do not want designs where rooms intersect each other or rooms that have no entry or exit. As shown in the bottom of Figure 4.1, the automatic generation of a Virtual Institution is done in two steps. First, a 2D floor plan of the institution is generated. Then a 3D transformation mechanism transforms this floor plan into a final 3D scene.

Please note that the e-auction example is using the performative structure of an Electronic Institution as the specification document, but our approach is applicable to any system where activities and their interactions can be formally specified.

4.2 Virtual World Grammar (VWG)

In this section, we introduce in detail the concept of Virtual World Grammar (VWG). First, we formalize all the necessary elements, such as ontology, validation and heuristics, to conclude the section by giving a formal definition of a VWG. For each of the VWG parts, we present a solution related to the motivation example.

4.2.1 Ontology

An ontology is a formal definition of the relevant concepts of a domain. In the context of a Virtual World Grammar, the ontology contains two different kinds of concepts. On the one hand, those related to the description of the activities that will take place in the virtual world. They define how activities are conceptualized, the relationships among them and in combination with a shape grammar determine the *layout* of the virtual world. On the other hand, there are the concepts that define the *properties* of the virtual world elements. Those are the properties of shapes in the virtual world design. Notice that a shape grammar contains geometrical information about shapes, but it does not contain any semantic information about them. Hence, an ontology defines the properties containing semantic information, such as texture or size, about those shapes that are later used during the generation process and to validate the obtained design.

In order to define an ontology, we take an object-oriented approach. The different concepts are defined by classes, and there exists a hierarchical relationship between them. We define $B = \{integer, real, boolean, string\}$ as the set of basic data types and I_C as a set of indexes.

Definition 4.1. We define an *ontology* as a tuple $o = (C, \prec)$ where:

- $C = \{(c_i, A_i, \sigma_{c_i})\}_{i \in I_C}$ is a set of class definitions (*concepts*), each one defined as a tuple, where c_i stands for the class identifier, A_i is a set of attribute identifiers, and $\sigma_{c_i} : A_i \longrightarrow T$ maps each attribute to its type, where T is recursively defined by the following rules:
  - $(B \cup \{c_i\}_{i \in I_C}) \subset T$
  - if $t_i, t_j \in T$ then $t_i \times t_j \in T$
  - if $t_i \in T$ then $t_i\, list \in T$
  - Nothing else belongs to T.
- $\prec$ is a class hierarchy such that if $c_i \prec c_j$ then $A_j \subseteq A_i$.

We distinguish between the concepts describing the activities and their relationships related to the specification (C_{Spec}), and those related to properties of shapes from shape grammar (C_{SG}). Hence, $C = C_{Spec} \cup C_{SG}$.

While the previous definition establishes how the domain concepts are formalized, by $terms_o$ we denote the actual instances of the concepts defined in an ontology o. Furthermore, by $terms_o^{C_{Spec}}$ we denote the instances of concepts in C_{Spec}, while by $terms_o^{C_{SG}}$ the instances of concepts C_{SG}.

E-auction House Ontology

From the specification of the e-auction House institution, only scenes that define activities are used in the generation (see blue rounded rectangles in Figure 4.1). Hence, the specification concepts (C_{Spec}) just contain the scene (activity) class. Attributes of this class come from the Virtual Institution specification. For instance, attributes defining the maximum number of participants of an activity. Specification elements for a Virtual Institution are obtained by searching within the specification document.

To perform the 3D transformation of a 2D floor plan we need data such as a texture, size, or information if some 2D object is substituted by a 3D model or it is *procedurally generated*. An example of such procedurally generated structure is a *wall*. Walls can be rendered as solid walls with texture, or walls with opening for windows and doors. We introduce the following shape grammar concepts (C_{SG}) and their properties:

- **Design wall** is the basic design element which forms higher-level objects. It represents an actual separation wall between some virtual spaces. Every wall holds a basic set of *geometrical properties* such as position, length, width and height. *Design properties* specifying how many doors and windows should be created. *Texture properties* include standard texture mapping properties, such as texture path and tiling.
- **Design space** represents an area that is substituted by a functional or non functional (see *office-layout, tree* in Figure 4.8 a)) 3D model. A design space holds information, such as path to the 3D model and its size, used during the 3D transformation phase.

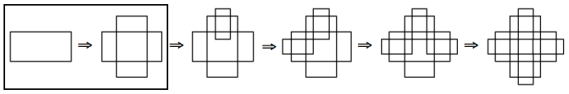


Figure 4.2: Shape grammar derivation process

- **Design block** is a collection of walls and design spaces that represents one "shape" of our shape grammar. We can look at the generation process as a lego-like building process, where different blocks are spatially placed together to create the final design. This placement is validated using validation rules.

4.2.2 Shape Grammar

Shape grammar is a method of generating designs by using primitive *shapes* and the *rules* of interaction among them. One of the shapes is marked as the *starting shape*. Shape grammar rules are composed of a left-side shape and a right-side shape, where the right-side shape replaces the left-side shape. Designs are generated from the shape grammar by starting with the initial shape and recursively applying its rules. As an example Figure 4.2 shows a rule and steps of the shape grammar derivation process. The shape grammar rule is marked with a black square. For details on shape grammars, please see Section 2.5.

E-auction House Shape Grammar

We define two different shape grammars to present the possibilities of our mechanism. The first grammar, displayed in Figure 4.7, creates one institution building, and for each activity, it creates a room within this building. The second grammar, depicted by Figure 4.9, creates a separate building for each activity. Shapes of a shape grammar represent blocks (rooms) of the building and placeholders (spaces) for the 3D models. Activities from the specification are associated with these spaces, and they are automatically resized to fit to the number of participants in the activity.

Both shape grammars use two different rule types. The first one positions rooms into locations within the outline. The second one distributes the rooms depending on the position of the previous one. We have used breadth-first search tree protocol (see Section 3.2).

4.2.3 Validations

Validations are used for testing and evaluating the execution of a shape grammar. We define a *validation language* that will serve as a basic representation for the *validation terms*. First, we define the set of binary operators $\Omega = \{<, \leq, =, >, \geq\}$. Second, we define an open set of functions $\Phi = \{range, in, not\}$. This set can be extended by designers by adding new functions.

Definition 4.2. Given an ontology $o = (C, \prec)$, a set of basic operators Ω and a set of functions Φ, we define the *validation language* $\mathcal{L}_V$ as the language generated by the following grammar with starting symbol E:

$$
\begin{array}{lll}
E ::= & E\,op\,E & \text{with } op \in \Omega \\
& \mid fun(M) & \text{with } fun \in \Phi \\
& \mid p.P & \text{with } p \in C_{Spec} \\
& \mid q.Q & \text{with } q \in C_{SG} \\
& \mid c & \text{with } c \in \{terms_o^{C_{Spec}} \cup terms_o^{C_{SG}}\}
\end{array}
$$

$$
\begin{array}{lll}
M ::= & E \mid M, M & \\
P ::= & a \mid P.a & \text{with } a \in A_{C_{Spec}} \\
Q ::= & a \mid Q.a & \text{with } a \in A_{C_{SG}}
\end{array}
$$

where A_i stands for the set of attributes of concepts of type i.

Definition 4.3. A *validation term* $T_{\mathcal{L}_V}$, also called *validator* is a term created using validation language $T_{\mathcal{L}_V} \in \mathcal{L}_V$

Validations can be evaluated at two different stages of the generation process. Specifically, they can be evaluated after each generation step (*step validations*) or at the end of the generation process (*final validations*). Step validations provide control mechanisms for shape grammar execution so that no invalid path of execution is selected (e.g. test for correct placement of rooms so the walls do not cross). Final validations serve for evaluating the final design, and we can regard them as *goals* or *objectives* of the generation process.

E-auction House Validations

We define a new validator *intersect*, which is executed after each execution step, and checks that (i) design blocks do not *intersect*, but they can *touch*, but only with walls that do not contain any doors or windows.

4.2.4 Heuristics

Heuristics guide the process of generation of a virtual world design. They have two fundamental roles. First, to decide in which order to process the elements from the specification. Second, how to find candidate execution nodes in the shape grammar execution tree for currently selected specification element. The generation process stores information in a tree structure, where each node holds information about the state of generation (see Section 3.2). This tree structure holds execution states, which represent either a shape or a rule. If a node represents a shape, it has as many children nodes as there exist rules with this shape on the left side. If a node represents a rule, it has only one child, which represents the right-side shape of the rule.

Definition 4.4. *We* define heuristic next as a function $h_{next} : terms_o^{C_{Spec}} \times 2^{terms_o^{C_{Spec}}} \times 2^{terms_o^{C_{Spec}}} \rightarrow terms_o^{C_{Spec}}$, which for any $x \in terms_o^{C_{Spec}}$, a set of already processed specification elements and a set of all specification elements returns element $y \in terms_o^{C_{Spec}}$, which will be the next processed element. Function $h_{next}(nil, \emptyset, SE)$, returns an initial element.

Definition 4.5. *We* define heuristic exec as a function $h_{exec} : S_T \times tree_{exec} \rightarrow n_{tree}$, that given a shape $x \in S_T$ and an execution tree $t \in tree_{exec}$ returns the next node to expand $y \in n_{tree}$ (S_T is a set of terminal shapes of a shape grammar).

Definition 4.6. *We* define heuristics as a tuple $\mathcal{H} = (h_{next}, h_{exec})$ where h_{next} is a *heuristic next* function and h_{exec} is a *heuristic exec* function.

In other words, function h_{next} is responsible for defining the order in which specification elements are processed. On the other hand, function h_{exec} is also responsible for finding the correct node in the execution tree that represents the possible rule that can be executed. If more than one node is returned, we can randomly decide which one to choose.

E-auction House Heuristics

In the e-Auction House example h_{next} is a straightforward function that returns the next element in the list of specification elements given the last processed one. The h_{exec} function searches for the non expanded nodes of the execution tree that can be used with the current specification element. Notice, that in the Virtual World Grammar, it is defined which shapes can be used to represent a specification element. Thus, the function searches for those rule nodes whose right-side shape is one of these shapes. When several candidate nodes are found the function just randomly selects one of them.

4.2.5 Virtual World Grammar (VWG)

With all the presented definitions, we are now ready to define Virtual World Grammar. It includes an ontology specifying all specification and shape grammar concepts. Then, it includes instances of these concepts, i.e., concrete elements that are used to generate a virtual world. It also includes a shape grammar that contains the different shapes and the rules used to generate the final design. Each specification element is mapped to a set of shapes that can represent it in the generated virtual world. During the generation process, it is decided which one will represent the element in the generated design. Each terminal shape S_T from shape grammar is associated to a shape grammar element defining the properties of that shape. These properties are applied during the 3D transformation process. The Virtual World Grammar also includes a set of heuristics that guide the generation process and validations that control and evaluate this process. At last, it includes a function that for each validation term defines its execution time, either after rule execution, or in the end of the generation process to validate final design.

Definition 4.7. We define a *virtual world grammar* (VWG) as a tuple: $VWG = (o, G, E, f_E, f_s, \mathcal{V}, f_t, \mathcal{H})$ where

1. o is an ontology that defines the relevant concepts for the generation process; that is multi-agent system *specification elements* C_{Spec}, and *shape properties* C_{SG}.
2. $G = (S_T, S_M, R, I)$ is shape grammar describing shapes and rules.
3. $E \subseteq terms_o^{C_{Spec}}$ is a set of instances of specification elements.
4. $f_E : E \rightarrow S_T^+$ returns for an specification element the set of shapes that can represent it in the generated design.
5. $f_s : S_T \rightarrow C_{SG}$ maps each terminal shape S_T of shape grammar to the ontology class defining its properties.
6. $\mathcal{V}$ is a set of validation terms $\mathcal{V} = \{T_{\mathcal{L}_V}\}^*$
7. $f_t : T_{\mathcal{L}_V} \rightarrow \{STEP, END\}$ is a function that assigns a value $STEP$ to the validator if it has to be evaluated after each step of shape grammar execution, or value END if it is evaluated at the end of generation.
8. $\mathcal{H}$ is a set of heuristics

4.2.6 Design Generation Process

Algorithm 2 generates a 2D floor plan and summarizes how and when are used all defined parts of a Virtual World Grammar. This algorithm first initializes variables, where SE is a list of all specification elements, *specElems* is a list of already processed specification elements, a is the currently processed specification element, t_{exec} is the execution tree and n is the currently processed node of the execution tree t_{exec}.

Function *initTree* initializes an execution tree t_{exec} by inserting a starting shape as the root node. Then, using function h_{next} searches for the next specification element to process, assigning it to variable a, and adds it into the list of executed elements *SpecElems*. Using function f_{SE} it finds the set of shapes that can be used to represent this element. It loops over this whole set till it finds a valid design. In this loop, it searches for the execution tree node n using heuristic function h_{exec} and executes it by calling function *Execute* creating new shape. It validates the result of execution. If the result is valid, it proceeds to the next iteration. The process finishes when it has processed all nodes from the SE. The process fails and returns nothing if no valid design was found.

In our case, we use the performative structure of an Electronic Institution as a specification of activities. In this specification, we isolate activities/scenes and also illocutions from scene protocols as our specification elements. We map activities/scenes to virtual spaces, and illocutions to interactive objects, which automatically perform illocutionary actions during interaction. Using this approach we can automatically generate interactive virtual worlds (this requires connecting an Electronic Institution to a virtual world, tackled in Chapter 5).

Algorithm 2: Virtual World Builder Algorithm

```
Input: Specification, Virtual World Grammar
Output: 2D draft (floor plan) of the virtual world
begin
    // initialize variables
    a ← nil; specElems ← ∅; t_exec ← initTree(); n ← ∅
    while (size(specElems) != size(SE)) do
        // get element from specification
        a ← h_next(a, specElems, SE)
        // put this element in control set specElems
        specElems ← a
        // search for valid design
        valid ← false
        // find associated shapes
        S ← f_E(a)
        foreach (s ∈ S) do
            while (not(valid) ∨ n = ∅ ) do
                // find unexecuted node in the exec. tree
                n ← h_exec(s, t_exec)
                // execute rule and store right shape
                c ← Execute(n, t_exec)
                // validate
                valid ← Validate (c)
            if valid then break
        if valid then appendChild(t_exec, n, c)
        else return ∅
    return t_exec
end
```

4.3 Virtual World Builder Toolkit

The Virtual World Builder Toolkit (VWBT) provides visual interfaces and mechanisms to define and execute Virtual World Grammars. The toolkit loads the specification of an Electronic Institution and combines it with information stored in the *Virtual World Grammar* to produce the final output. It is integrated in our Shape Grammar Interpreter. An intermediate output of the generation process is a 2D draft of the virtual world (floor plan). Using a 3D engine (jMonkeyEngine), this draft is transformed into a 3D model, used for preview purposes. The tool allows to implement different renderers that export the 3D model into different virtual worlds (e.g. Second Life, Open Wonderland). Furthermore, this solution allows to:

- dynamically react to changes in the MAS specification and simply regenerate the virtual world visualization.

- separate artistic (graphical) design of the institution from the functional implementation.
- make the generation process transparent to an institution designer and 3D virtual world designer.
- browse the design space and easily explore different designs.

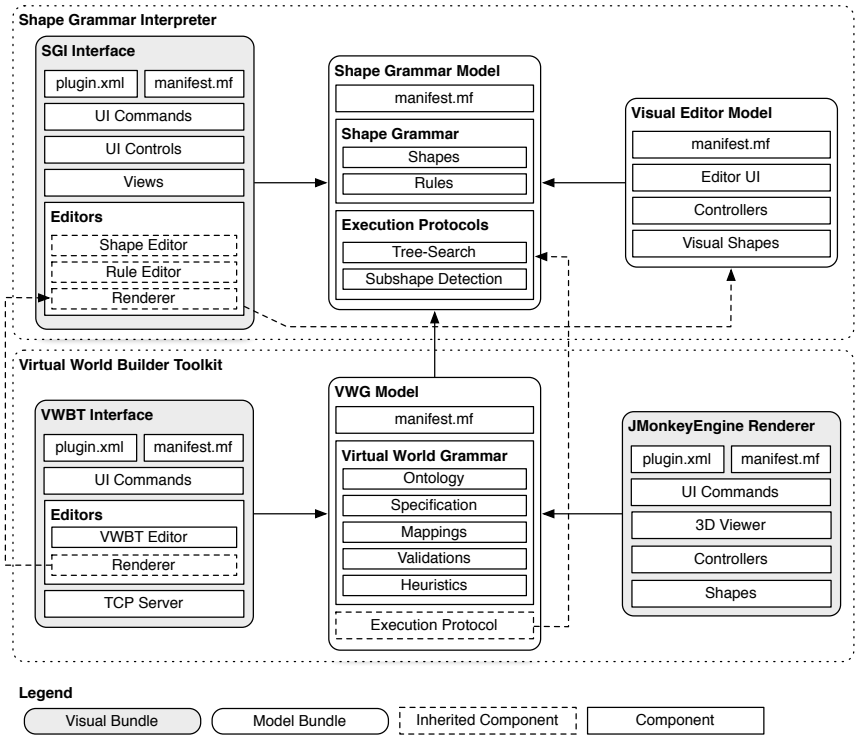


Figure 4.3: Architecture of the VWBT system

Figure 4.3 shows the architecture of the VWBT, which was implemented in three plug-ins that extend the SGI system shown in the previous chapter, Figure 3.13. There are two visual, *VWBT interface* and *JMonkeyEngine Renderer*, and one model plug-in, *VWG Model*. The model plug-in implements the VWG and all its parts. Moreover, it implements the new *execution protocol*, which extends the existing tree search protocol and adds new features to it, such as validation of steps, or custom selection of rules to apply.

VWBT interface is a visual plug-in that provides all commands and interfaces needed to define and modify a VWG. Using these interfaces a grammar designer can specify related parts of the grammar, such as (i) which specification instances are generated in the 3D space, (ii) properties of objects defined

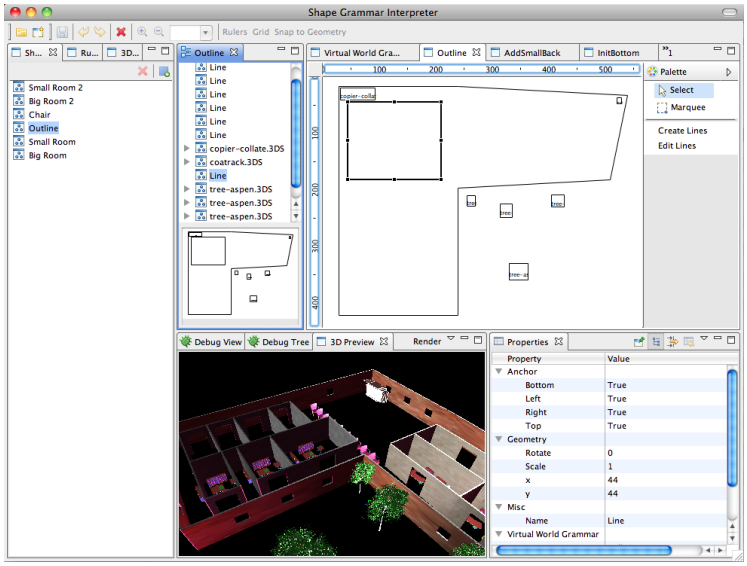


Figure 4.4: SGI interface with WVBT extensions

by the ontology and (iii) validation terms. It also includes a custom renderer which shows extra information about the generated shape, such as the meta-data stored within the description of shapes in the VWG. The second visual plug-in is the *JMonkeyEngine renderer* that allows us to preview the generated 3D virtual world using the JMonkeyEngine, which is a 3D engine programmed in Java. Figure 4.4 shows the interface of the VWBT, where in the top right part, we see the editor for the VWG and below in the middle, we see the JMonkeyEngine 3D renderer.

4.3.1 Workflow for Definition and Execution of Virtual World Grammars

Virtual World Builder Toolkit brings many creative possibilities to the Virtual World design process. Designers may explore many different designs based on a shape grammar. Shape grammar elements serve as a visual style sheet for the generation process. Trying different values for parameters, or even having prepared multiple sets of instances brings possibilities of *styling* or *skinning* in virtual worlds. Figure 4.5 describes the workflow process for the definition and execution of a Virtual World Grammar. Depending on the results of the *draft* or the *final* generation, we can readapt the grammar.

A grammar designer can either browse through possible designs or modify existing parts of the shape grammar to obtain satisfiable results. The workflow is divided into three main parts. First, in the *preliminary definition*, it defines the ontology and the shape grammar. Second, in the *instance definition*, it

loads the specification, creates and defines all specification elements (SE) and shape grammar elements (SGE) and specifies mappings between them. Then validations and heuristics are defined. Finally, in the *execution* part, it browses through random designs and modifies instance parameters (i.e. step "Set up SGE" where, for example, we decide actual sizes of shape grammar elements) to produce the 2D floor plan, which is then transformed to 3D.

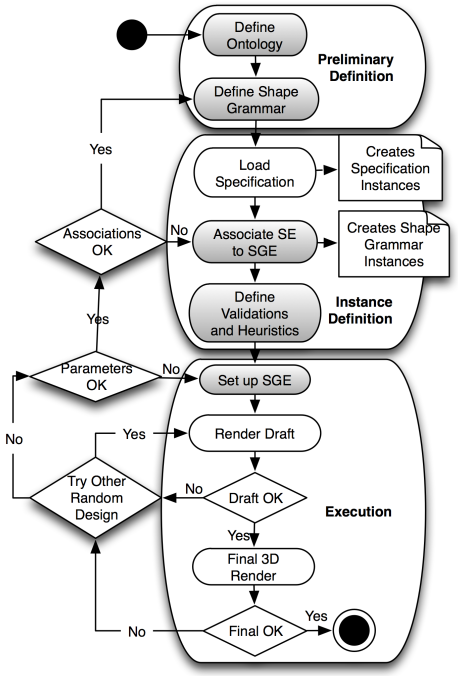


Figure 4.5: Workflow for definition and execution of VWG

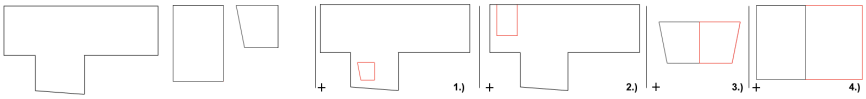
4.4 Results

In this section, we present different results of the generation, and we measure generation performance. As an input, we take the e-Auction house grammar, and we vary the number of the auction rooms. We also define two different shape grammars. We use a simplified display of rules presented in Figure 4.6 where the left-side of the rule is shown in black and the right-side in red.

The first shape grammar, depicted in Figure 4.7, generates an institution building and positions all rooms inside this building. The initial shape of this grammar is the outline shape. This shape grammar distributes the rooms within this outline. The floor plan and the 3D model for five auction rooms (we have selected five rooms for a demo example of a small institution) and two remaining

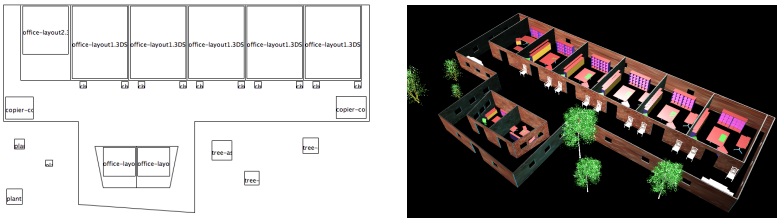
(a) Normal (b) Simplified

Figure 4.6: Rule display simplification



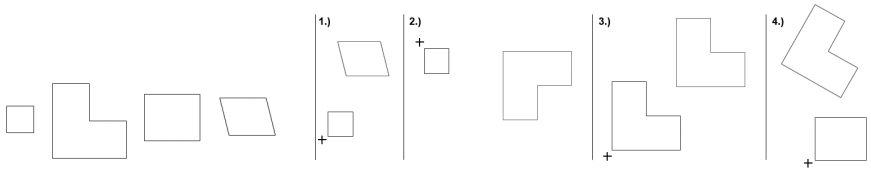
(a) Shapes (b) Rules

Figure 4.7: Shape grammar 1 for the Auction House institution



(a) Floor plan (b) 3D render

Figure 4.8: An output of the Virtual World Grammar using shape grammar 1



(a) Shapes (left one is the initial shape) (b) Rules

Figure 4.9: An excerpt from the shape grammar 2 for the Auction House institution

rooms (Item Register, Auction Info) are displayed in Figure 4.8. This output was produced using three shapes (outline, rectangle room and iso-room) and four rules (two rules place the rooms within this outline, and two rules distribute rooms depending on the position of a previous one). The drawback of this grammar is that it is rather simple and the design space it generates is rather small.

The second shape grammar for the auction house institution does not limit

Figure 4.10: Two different outputs of the Virtual World Grammar using shape grammar 2

the design by the initial outline, and it generates large design spaces. Figure 4.9 displays an excerpt from this grammar. We can see four shapes that represent three possible room designs and an initial shape. In the right part of this figure, we see examples of rules that place rooms according to the position of the previous shape. Rooms in this grammar are generated as stand-alone buildings. Figure 4.10 shows two generated floor plans and the corresponding 3D models for five auction rooms. The small rectangles and the rectangles within the shape grammar shapes represent placeholders (office-layout in Figure 4.8) for the 3D models that are substituted during the 3D transformation phase.

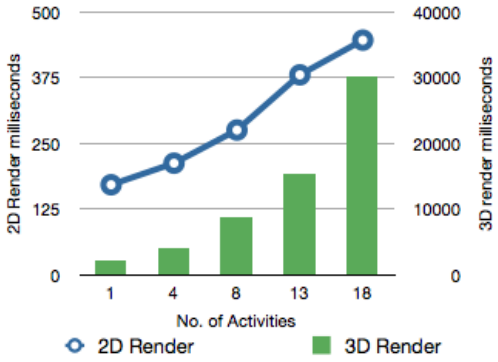


Figure 4.11: Performance measurements of VWG

Figure 4.11 shows the graph showing the generation time depending on a

number of scenes. We have scaled the institution up to 30 scenes and in these scenes we used some complex models to measure also the possibilities of the jMonkeyEngine. The generation of the floor plan for a large institution was under one second. The 3D render grew from two seconds for five rooms to 30 seconds for 25 rooms. The reason for increased time is the use of complex models, such as trees, which in total made more that 1.4 million of faces for 30 rooms.

An Auction House institution is a typical example of the use of Virtual Institutions. Our approach allows a comfortable separation of parts of a Virtual Institution into design subsets. This allows to produce designs for large institutions or confederations of institutions.

4.5 Summary

In this chapter:

* We have defined the Virtual World Grammar mechanism for automatic generation of 3D virtual worlds designs from a specification of activities.
* We have described the 3D virtual world generation algorithm.
* We have presented the Virtual World Builder Toolkit, a tool implementing the Virtual World Grammar functionality.
* We have presented examples of generated 3D virtual world designs using a Virtual World Grammar with two different shape grammars.

In the next chapter, we present our Virtual Institution Execution Environment which provides causal connection between virtual worlds and an Electronic Institution, providing the possibility to enforce at run-time participant interactions.

Chapter 5

Virtual Institution eXEcution Environment (VIXEE)

In the previous chapter, we introduced the Virtual World Grammar (VWG) as a mechanism for automatic generation of virtual worlds' designs. In this chapter, our concerns shift from design to execution. We propose Virtual Institution eXEcution Environment (VIXEE) as an innovative communication infrastructure for Virtual Institutions (VIs). The main features of the infrastructure are i) the causal connection between several Virtual Worlds and an Electronic Institution using our Movie Script mechanism, ii) the automatic generation and update of VIs' 3D visualization and iii) the simultaneous participation of users from different virtual world platforms. We illustrate the execution of VIXEE system on the previously introduced e-auction house example and use this example to evaluate the performance of our solution.

5.1 Motivation

Nowadays there is an increasing demand for e-* applications (where * stands for learning, commerce, government, etc.). These applications support the participation of humans that engage in different activities, to achieve their goals. Whenever some tasks can be delegated and automated, these applications can be enriched with software agents. As a consequence, human and agent interaction must be handled. The internet based and distributed software technologies, such as Virtual Worlds (VW) and Multiagent Systems (MAS), may support the engineering of this type of applications.

Specifically, we advocate to take a MAS approach for designing these systems and to use 3D Virtual Worlds to get humans-in-the-loop by facilitating their participation in the system. First, a 3D real-time representation of the

system facilitates a better understanding of what is happening at both agent and the entire system levels. Second, thanks to the regulation imposed by the MAS, the 3D environment becomes a normative Virtual World, where norms are enforced at runtime. This automatic regulation contrasts with the way it is done in current virtual worlds where norms are restricted to the user's acceptance of the terms of service. Third, system participants can be both humans and software agents. In other words, it is an effective way to facilitate direct participation of humans in MAS, instead of just allowing them to customize agent templates with their preferences. This approach is taken in Virtual Institutions [Bogdanovych et al., 2005], which have proven [Seidel, 2010] [Bogdanovych, 2007] [Bogdanovych et al., 2009] [Bogdanovych et al., 2011] to be an adequate platform to support this type of hybrid multi-agent systems, by combining Electronic Institutions [Esteva, 2003], which is an Organization-Centered MAS (OCMAS), and 3D Virtual Worlds. In this context, Electronic Institutions are used to establish the regulations that structure interactions and support software agent participation while virtual worlds facilitate human participation.

As described in Section 2.3, the formal specification of an Electronic Institution, called performative structure, is used by our Virtual World Grammar (VWG) to automatically generate the virtual world design. Activities of performative structure are displayed as virtual spaces, while illocutions from the scene protocols are transformed to specific world interactions and gestures (see Chapter 4). Also, Virtual World Grammar mechanism allows to generate virtual world model for several different virtual worlds (e.g. Open Wonderland, Second Life).

Nevertheless, the generated design is only a 3D model of a virtual world, a visual layer, separated from the EI runtime infrastructure (called AMELI), the normative control layer. This separation does not allow normative control of world interactions at run-time. Thus, layers have to be connected, and in a way that would assure their consistent state according to the other layer. Therefore, the desired connection has to keep their causal dependence, when related actions in virtual worlds are processed by EI and related EI events are visualized in a virtual world, by manipulating the virtual world design (e.g. opening doors).

However, with the existence of such a large number of virtual worlds, it is often practical to let participate users from several different virtual worlds. This increases the possible user base or allows to perform experiments with different user groups (e.g. kids, teens or adults). In some cases, as in the case of the presented e-auction house, it is even desired to join the execution of the virtual world application with the non virtual environment, such as the web application or even the real world (see Section 1.6.1).

Combining several different environments and their simultaneous execution raises several inter-operability issues, such as:

1. Parallel presence, movement and interactions of avatars in different virtual worlds.

2. Virtual world participants that speak different languages try to participate in the same "single language" application.

3. Heterogeneous architectures of virtual worlds make it difficult to monitor and react to virtual world events and interactions, and to causally update the virtual world model according to the EI state.

Considering point (1) solving parallel avatar presence and movement is out of scope of this research. However, we demand only a basic level of virtual worlds' interoperability. This means that participants from other virtual worlds are visualized as limited avatars, which only perform institutional actions. Thus, for instance, it is not shown how they walk around the room. Although this may bring visual problems related to group communication. An example application using multiple virtual environments is an auction where users from Second Life, Active Worlds or PS3 Network participate in the same auction room with a fixed amount of chairs. Hence, as soon as a participant takes one of the chairs, his avatar appears sitting in a chair, in all other universes (i.e. Virtual Worlds).

The popular solution to solve the problem of multilanguage environments, mentioned in point (2), is to define a common ontology, that each of the participating virtual worlds adopts for controlling and executing world's interactions. In our approach, we rely on Electronic Institutions, which define such common ontology for all institutional interactions.

Considering heterogeneous architectures in point (3), we need a mechanism that creates a mapping between virtual world dependent actions and institutional messages. In reverse, it has to define mappings between institutional events and target virtual environments, where such an event should be visualized. Concerning the content manipulation, the virtual world model is generated prior to the execution of the Virtual Institution, and then it is dynamically updated during the execution of the institution (e.g. launching of a scene in the normative layer can add a new room to an institutional building in the visual interaction layer). In our approach, we use Virtual World Grammar (see Chapter 4), which can dynamically manipulate 3D content of multiple virtual worlds.

[Bogdanovych et al., 2008] presented the architecture of a causal connection server, which was able to create a causal connection between different environments (e.g. virtual world and mobile application). The drawback of this solution is a simple action-message table which makes it difficult to route events between different environments and an Electronic institution. Therefore, we propose VIXEE as an innovative Virtual Institution eXEcution Environment which adds important extensions to previous Virtual Institution infrastructures. These extensions address generic and dynamic features. That is, VIXEE allocates at run-time participants from different VW worlds, and it modifies on the fly the content of a virtual world (e.g a new virtual room can be added during the execution of the infrastructure). An important factor of any middleware is its agility that is its ability to respond quickly and safely to both layers event during heavy loads. Therefore, we have evaluated our solution by measuring response time with a large number of agents (up to 500 agents).

5.2 Virtual Institution Execution Infrastructure

[Bogdanovych, 2007] proposed a three-layer architecture for Virtual Institutions. Figure 5.1 shows an overview of this architecture. Causal Connection Layer represents the middle layer (implemented by a middleware) of the three-layer architecture. In this chapter, we propose a Virtual Institution eXEcution Environment (VIXEE), which defines a new architecture of a middleware of a Virtual Institution, connecting users in the Visual Interaction Layer to the Normative Control Layer. Normative layer is in charge of regulating participant actions by mean of AMELI, the EI execution infrastructure. VIXEE supports the participation of software agents by visualizing them as bots in connected Virtual Worlds; thus, human-software agent interaction is enhanced. Its main component consists of an Extended Connection Server (ECS) and a Virtual World Manager (VWM).

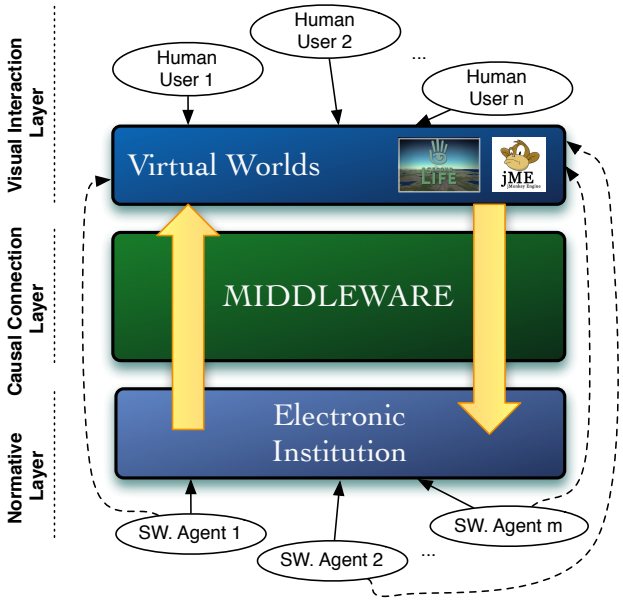


Figure 5.1: Overview of the Virtual Institution architecture.

5.2.1 Solution Architecture

We design VIXEE respecting the 3-layered architecture of Virtual Institutions, depicted in Figure 5.2. The top layer consists of several 3D Virtual Worlds. The bottom layer is represented by the Electronic Institution execution environment (AMELI). Both layers are causally connected by our middleware, which consists of the Extended Connection Server (ECS) and the Virtual Worlds Manager (VWM).

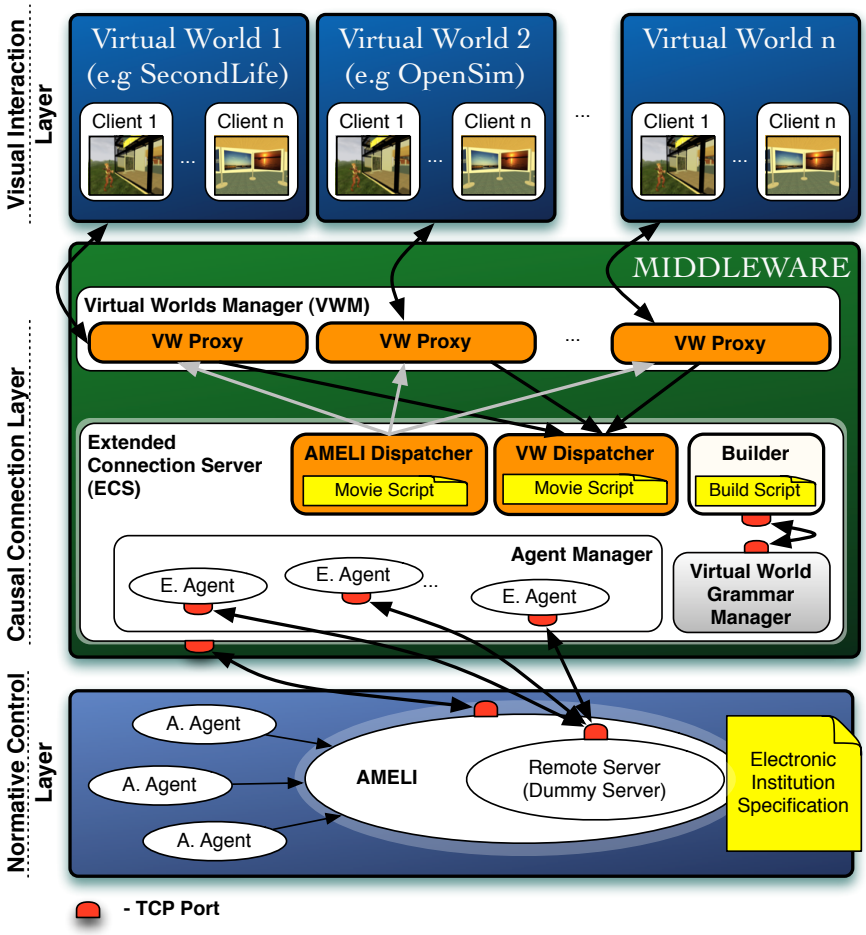


Figure 5.2: Architecture of the Virtual Institution Execution Environment

Visual Interaction Layer

The Visual Interaction Layer consists of several 3D virtual worlds. Human users and selected software agents are represented by their avatars (i.e. 3D virtual characters). Each of the virtual worlds can be implemented in a different programming language and using different visualization technologies. The virtual world communicates with the middleware using a standard network protocol (e.g. TCP, HTTP).

The case study introduced in Section 5.4 uses OpenSim as the virtual world platform for the e-auction house Virtual Institution. The Visual Interaction Layer may be composed of different types of interfaces allowing users to participate from a 3D Virtual World as well as from a web page.

Normative Control Layer

The Normative Control Layer is represented by an Electronic Institution running in AMELI, which mediates agents' interactions while enforcing institutional rules. AMELI is a general-purpose MAS infrastructure, as it can interpret any institution specification generated by ISLANDER [Esteva et al., 2002], the EIs specification editor. Therefore, it can be regarded as domain-independent. The combination of ISLANDER and AMELI provides full support for the design and development of Electronic Institutions [Esteva et al., 2004]. For each participant within an institution, AMELI creates a governor, which mediates its participation in the institution, and coordinates with the rest of other agents in AMELI to guarantee the correct execution of the institution. Autonomous software agents (A. Agent in Figure 5.2) participating in the institution are directly connected to AMELI. AMELI uses two TCP ports to communicate with the middleware. The first one is used to announce changes in the institutional state (event) to the causal connection layer (e.g. started institution, agent entering or exiting a scene). The second one is used to receive messages, corresponding to VW actions, from the middleware.

Causal Connection Layer

The Causal Connection Layer (CCL) (hereafter we may refer to it as *middleware*) provides the causal connection between different 3D virtual worlds and an executing Electronic Institution. As Figure 5.2 shows, it is split into two main components: (i) Extended Connection Server (ECS), responsible for communication with AMELI; and (ii) Virtual Worlds Manager (VWM), responsible for communication with 3D virtual worlds. The rest of this section describes them.

Extended Connection Server (ECS)

ECS mediates all communications between AMELI and Virtual Worlds Manager. This layer introduces new features and improvements that were not available in previous designs of the Causal Connection Layer [Bogdanovych, 2007] [Bogdanovych et al., 2008] [Seidel, 2010]. The most notable features are: support for multiple environments, reaction on early EI events, and connection fail-safe mechanisms. Current design allows us to connect one running instance of an Electronic Institution to multiple 3D virtual worlds, thus allowing joint participation of users from different virtual worlds in the same Virtual Institution. Reaction to early EI events allows to catch events like *institutionStarted*, which can trigger the construction of the virtual world design. The connection fail-safe mechanisms deal with connection errors from both virtual worlds and EI to the Casual Connection Layer.

An important part of ECS is the Agent Manager. For each human participating in a 3D virtual world, the Agent Manager creates an external agent (E. Agent in Figure 2) in the middleware representing him within the institution. Thus, when the avatar tries to perform an action that requires institutional verification, this agent is used to send the corresponding message to AMELI. Hence, AMELI

perceives all participants as software agents. There are two different classes of external agents, one for human participants and another one for software agents. Software agents adopt our generic model for believable virtual agents called VI Agents (see Chapter 6). Both classes use AMELI for the normative control of their actions.

Moreover, ECS includes the *Builder* component, which communicates with the Virtual World Grammar Manager, and it is responsible for manipulating the virtual world content. Specifically, the Builder uses the Virtual World Grammar (see Chapter 4) to perform its tasks. Notice though, that whilst the Builder is responsible for creating a new geometry for a virtual world scene (e.g., virtual rooms), the Virtual World Manager is the component in charge of updating the virtual world visualization.

ECS uses the *VW Dispatcher* to mediate virtual world actions and the *AMELI Dispatcher* to mediate AMELI events. Both dispatchers use our proposed Movie Script mechanism to select which action to perform depending on the context of an action/event (see Section 5.2.2). As Section 5.2.3 details, when the VW Dispatcher receives an action from a virtual world, it sends the corresponding message to AMELI, and if necessary, waits for AMELI response (synchronous messages wait for a response, while asynchronous do not). The response to the event is communicated back to the virtual world (see Figure 5.4). 3D worlds are able to use either HTTP or TCP protocol to communicate its events. On the other hand, as described in Section 5.2.4, *AMELI Dispatcher* handles AMELI events. Such events provoke a causal change in the state of 3D virtual worlds, so AMELI Dispatcher triggers an action execution to each of the connected virtual worlds (see Figure 5.5).

ECS uses three TCP ports. First is used one to communicate between the Builder and Virtual World Grammar Manager. The second one is used to listen for AMELI events. The third one is used by the Agent Manager to send messages to AMELI (all external agents share the same port).

Virtual Worlds Manager (VWM)

Virtual Worlds Manager is used to mediate all communications between 3D virtual worlds and ECS and to dynamically manipulate the 3D representation of all connected virtual worlds. It consists of a set of the virtual world Proxies, one for each connected virtual world (see Figure 5.2). These proxies allow to use a language specific for a given virtual world to communicate with VIXEE. Thus, for example, in Second Life we use OpenMetaverse[1] library to update the state of the virtual world. The content is manipulated after receiving a selected AMELI event (e.g. *SceneStarted*), when as a result a new room is created in the 3D virtual world. Next section details how this is done based on our *Movie Script* mechanism, which can map any event/action from a specific context (AMELI event or virtual world action) to a *Movie Script* action.

[1] `http://openmetaverse.org/`, (last accessed 05/2012)

5.2.2 Message Handling: Movie Script Mechanism

Virtual World interactivity is accomplished through *virtual world actions* that human users and software agents perform within a virtual world. Such actions are either *institutional* or *non institutional*. An institutional action has to be validated by the Electronic Institution running in AMELI (e.g. entering some scene). On the contrary, execution of non institutional actions does not have to be validated by the institution (e.g. walking).

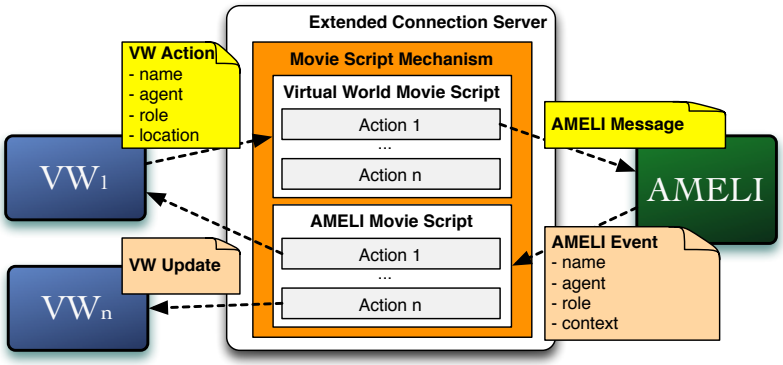


Figure 5.3: Movie Script Mechanism Conceptualization

When an institutional action is performed in a virtual world, an Electronic Institution needs to be causally updated by receiving the corresponding AMELI message (e.g. agentEnterScene). In reverse, when the Electronic Institution produces an event (we refer to such an event as *AMELI event*), all virtual worlds need to be causally updated. The original mechanism that handles this mechanism, proposed by [Bogdanovych, 2007], is an Action/Message table. This table holds mappings between virtual world actions and an institutional message. An example from the e-auction house example is the mapping of the gesture raising hand to the institutional message "bid". The problem arises when we need to assign two different meanings for the same gesture, or the same action. That is, actions or gestures can have different meaning in a different context (e.g. police man raises hand to stop traffic). Another issue of the original approach is that it is not possible to route events to and from multiple environments, meaning, that we cannot decide which events to process in which environment and in what manner.

Thus, we propose our *Movie Script* mechanism to process the transformation of virtual world actions to AMELI messages and in reverse transformation of AMELI events to virtual world updates. Figure 5.3 describes how the Movie Script mechanism uses *Movie Script actions* to handle such transformations. This mechanism supports the virtual world independence and facilitates a simple and consistent definition of the expected behavior of a 3D virtual world. A

Action Type	Action	Description
Motion	enterInstitution	Request to enter the institution
	moveToTransition	Request to move from a scene to a transition
	moveToScenes	Request to move from a transition to several scenes
Illocutionary	saySceneMessage	Request to say a message in a scene
Information Request	accesScenes	Ask for the scenes the agent can join from a transition
	accesTransitions	Ask for the transitions the agent can join from a scene
	agentObligations	Ask for pending agent's obligations
	sceneState	Ask for a scene's current state
	scenePlayers	Ask for agents in a scene

Table 5.1: Institutional Actions

virtual world Movie Script is defined for all virtual environments and a AMELI Movie Script for an Electronic Institution. Both are processed by the VIXEE environment. A Movie Script contains lines, where each virtual world Movie Script line specifically defines the context for which this Movie Script line applies (e.g. participant role, virtual world location, action name). Each AMELI Movie Script line defines for which AMELI event this line applies (e.g. event name, institution name, sender role) and which virtual worlds should be notified about this event using what Movie Script action.

Virtual World Actions

VIXEE differentiates between three types of institutional actions: *illocutionary* (illocutions that agents try to utter within scenes), *motion* (movements between scenes and transitions), and *information request* (scenes reachable from a transition, transitions reachable from a scene, agent's obligations, scenes' states, and scenes' participants).

Institutional actions are initiated in a specific virtual world and validated by sending the corresponding message to its Proxy and observing the response. Table 5.1 shows examples of institutional actions. The *Movie-script mechanism* maps the institutional action to the defined *Movie Script action*, which depending on the action context received from the Proxy, creates the AMELI message and sends it to AMELI. For each received message, AMELI replies with one of the following messages: *agree* (correct message), *refuse* (incorrect message), or *unknown* (message not understood).

We seek inspiration in a movie production, where depending on a current scene and its state, an actor performs an action. Like a regular Movie Script, it contains script lines. Each line holds the definition of the context, upon which a defined action is executed. Formally:

Definition 5.1. Virtual World Movie Script is a function:
$vwf : A \times R \times L \times VWA \rightarrow MSA$ which maps a virtual world action $vwa \in VWA$

to a corresponding Movie Script action $msa \in MSA$ depending on its context, where:

1. A is a set of Virtual Institution participants
2. R is a set of roles that participants can take while participating in the Virtual Institution
3. L is a set of locations, that is either transitions or scenes
4. VWA is a set of virtual world actions being sent from the virtual world
5. MSA is a set of Movie Script actions

An example of a script line from the eAuctions institution (see Section 5.4) is: vwf_1(agentId, buyer, auctionRoom, saySceneMessage("Bid", AgentId, amount)) = makeBid(AgentId, buyer, auctionRoom, saySceneMessage("Bid", AgentId, amount)). This lets a user with role buyer bid a specific amount of money and sends the bid request to AMELI.

AMELI events

The causal connection of AMELI with all connected virtual worlds is maintained by the AMELI Dispatcher, which reacts to received AMELI events by executing a corresponding Movie Script action. Then, for each of the connected virtual worlds, and for each of the relevant AMELI events, we have to implement a Movie Script action, executed by the corresponding Proxy, which updates the virtual world visualization. Table 5.2 contains examples of typical AMELI events. Formally:

Definition 5.2. AMELI Movie Script is a function:
$amf : E \to \{VW \times MSA\}^*$ which depending on received AMELI event $e \in E$, for each of the connected virtual worlds $vw \in VW$ returns a corresponding Movie Script action $msa \in MSA$ where:

1. E is a set of AMELI events, where each event contains the name of the event, the event context (identifying the origin of the event) and message specific content.
2. VW is a set of identifiers of all connected virtual worlds.
3. MSA is a set of Movie Script actions. Specifically $msa \in MSA$ is a Movie Script action which updates the given virtual world $vw \in VW$ depending on the content of the message $e \in E$. In particular, msa can be empty, if no Movie Script action is defined for a specific virtual world.

An example of an AMELI Movie Script line from the e-auction house institution (see Section 5.4) for AMELI event $ae_1 = (agentEnteredScene[agent = buyer_1, scene = Auction])$ is: $amf_1(ae_1)$ = {SecondLife, openDoor(ae_1)}, {OpenSim, openDoor(ae_1)}. This opens the door of the *auction* scene in Second Life as well as Open Simulator.

5.2.3 VW Actions Implementation

There are different activities that participants perform in 3D virtual worlds. For example, they interact with objects and communicate with other participants. It is similar in Virtual Institutions where agents represented by avatars must be able to perform institutional actions. In this section, we present how our system handles 3D virtual world actions, maintaining the causal dependence with an Electronic Institution using a bidirectional connection.

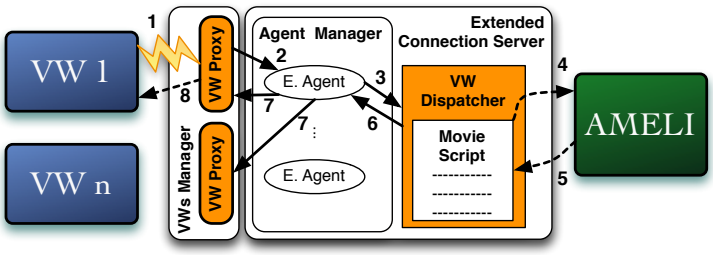


Figure 5.4: Message flow for VW generated action. Dashed lines represent the *message* passing, solid lines represent the *use* relationship.

Figure 5.4 depicts the message flow of a virtual world action. In this Figure, the dashed line represents a *message*, while solid line represents the *use*[2] relationship. When an action happens in a virtual world, it is sent to the Connection Server using a corresponding VW proxy (1). Next, the received message is passed to the external agent representing the user who has performed the action (2). The external agent uses the VW Dispatcher to send the message to AMELI. Specifically, VW Dispatcher uses the Movie Script mechanism (function vwf) to find the action (3). The Movie Script action is executed, and VW Dispatcher sends the corresponding message to AMELI (4). Then, agent actions are evaluated by AMELI. As a result, an AMELI event may be generated (see Section 5.2.4), and all connected virtual worlds are notified about agents' actions. For an asynchronous message, the process ends.

For synchronous messages, the ECS waits for the response, which is either the confirmation of the action execution or an error message (5). The response is sent to the Agent Manager (6). Finally, The Agent Manager contacts all the connected VW proxies (7) and each one of them informs its related 3D virtual world about the result of the action (8). If the result is positive, the action is visualized in the 3D virtual world. Actions produce human readable or visual output so the participant can perceive the output of his action either by a change in a 3D world (opening door) or by receiving a message.

[2]We consider the UML 2.0 notation

5.2.4 AMELI Events Implementation

AMELI keeps the execution state of an EI, which evolves as the consequence of participant actions. In such a case AMELI informs participants about those changes by using a set of institutional events (i.e. AMELI events). Examples of such events are: institution started, scene started, entering or leaving a room or a transition. VIXEE provides support for all AMELI events.

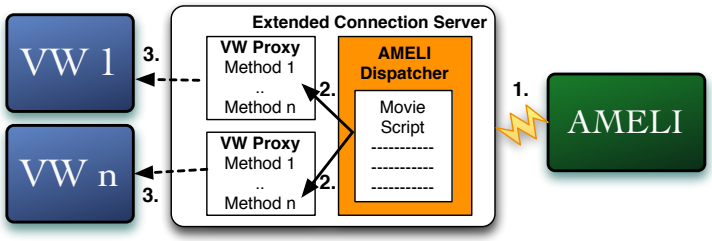


Figure 5.5: Message flow from AMELI to 3D virtual world

Figure 5.5 depicts how VIXEE processes an AMELI event. When AMELI triggers an event (1) it is communicated to ECS using TCP in the predefined AMELI format. AMELI has a fixed, predefined set of events. When ECS receives an event, it calls the AMELI Dispatcher that uses the Movie Script function amf to look for an action msa to execute (2). Then, for each of the connected VW Proxies, it dispatches the related action (3). If requested, ECS can use the *Builder* to dynamically update the 3D representation of the related virtual world.

Table 5.2 contains the list of typical AMELI events. Notice that some events require the use of Builder component to manipulate the virtual world content. These events and their corresponding VW updates are:

- *InstitutionStarted* - The system generates a 3D representation of an institution from scratch or reset the institution to the default state.
- *SceneStarted* - The system generates a 3D representation of the scene and set the virtual world parameters so that it is possible to enter the generated scene
- *SceneFinished* - The system removes a 3D representation of the scene and teleport all avatars out of the scene.

5.3 VIXEE Interface

While previous systems (CCS [Bogdanovych, 2007], GCS [Seidel, 2010]) needed to run several components and programs in order to connect a Virtual Institution to a virtual world, VIXEE runs as a stand-alone tool with its own user

Event	Description
InstitutionStarted	An instance of new Electronic Institution was created
SceneStarted	An instance of new scene was created in AMELI
SceneFinished	An instance scene was destroyed
EnteredAgentEI	An agent has entered an institution
ExitedAgentEI	An agent has exited the institution
MovedToScene	An agent has moved to a scene
ExitedScene	An agent has been exited of the scene
SaidMessage	A message has been said in a scene

Table 5.2: AMELI events

graphical user interface. Figure 5.6 shows VIXEE interface demonstrating its parts. (1) launcher is used to start and stop Virtual Institution execution. Button "Institution" launches AMELI infrastructure. (2) VIXEE components are defined in several different tabs of the application editor. Figure 5.6 depicts the "Movie Script" tabs (see Section 5.2.2). Other tabs (i.e. Avatars, Roles, Agents, Environments, Runtimes and Run) are used to specify, project properties (e.g. name, description or path to performative structure), or avatar data for different virtual worlds. Other tabs are related to the definition of behavior of VI Agents presented in Chapter 6. (3) Log component describes application log messages. It is also used to monitor the incoming and outgoing communication.

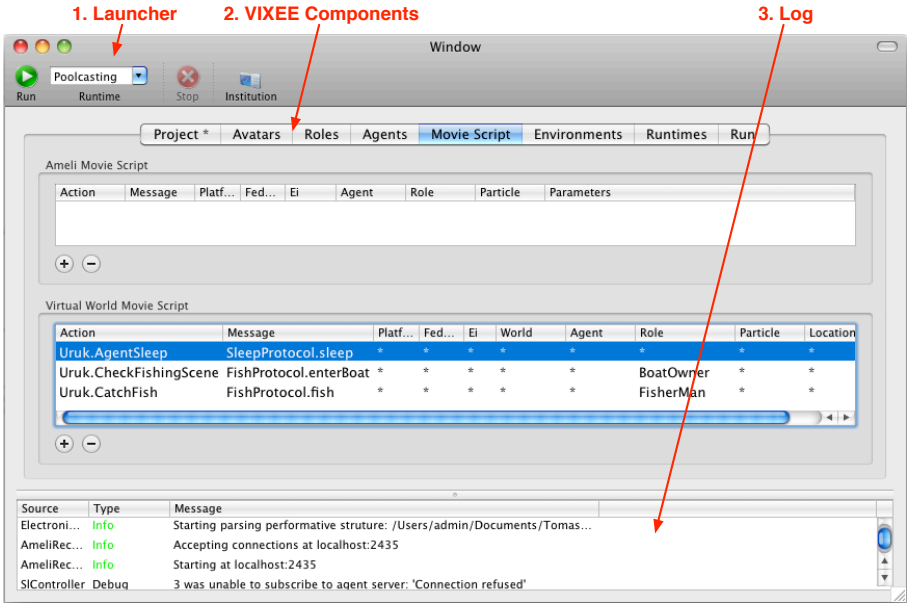


Figure 5.6: VIXEE interface

5.4 Case Study: eAuction House

In this Section, we present the use of VIXEE on the e-auction house example from Section 1.6.1. First, we specify the normative control layer of the Virtual Institution, which is an Electronic Institution specification. Figure 5.7 depicts a performative structure (set of connected scenes) of the e-auction house institution. As this figure depicts, institution supports roles of seller, buyer, staff and auctioneer. Staff agent is a software agent responsible for automatic processes, such as the creation of an auction room and controlling the auction execution protocol. To start the auction, staff agent changes role to auctioneer. Scenes in this institution are: *ItemInfo*, *ItemRegister* and *Auction*. *Initial* and *Final* scene are specific scenes through which participant can enter or leave the institution.

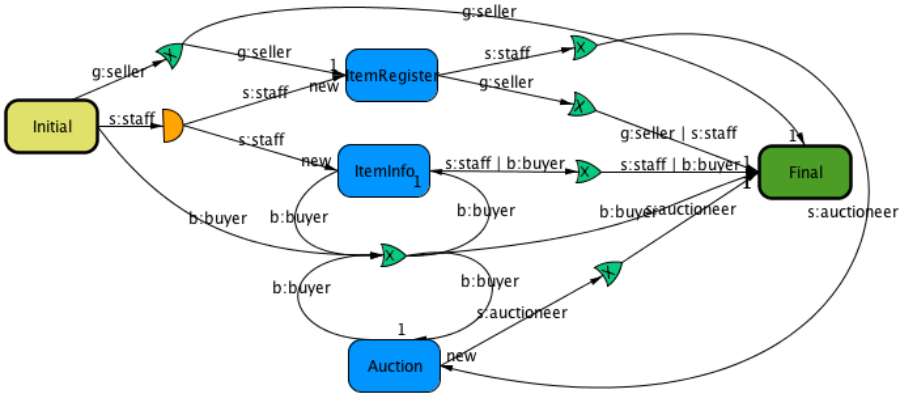


Figure 5.7: Performative structure of the eAuctionHouse institution

In this scenario, sellers first register items in an *ItemRegister* scene. If no auction is running, staff agent creates a new auction scene and waits for participants to start the auction. Buyers join the *ItemInfo* scene to check the list of items to be auctioned and the auction starting time. When required, a number of buyers enter the auction scene, the auction starts. Buyers can bid either by raising his hand, or by typing the *bid* command. When the auction finishes a winner cannot leave the auction room until he pays for the items he won. When no more items are to be auctioned, the auction room is destroyed (removed from virtual space).

Second, using the Virtual World Builder Toolkit, we define a Virtual World Grammar that allows us to generate the 3D representation of the Virtual Institution from the performative structure. In this tool, we can test the generated output. An example of the generated floor plan is displayed in Figure 5.8. To generate this floor flan, we map *ItemRegister* scene to the registration room, *ItemInfo* scene to the item information room and *Auction* scene to the auction room. In this Figure 'a' is the registration room, 'b' is the item information room, 'c' is some decoration (tree) and 'd', on the right side of the figure, represents

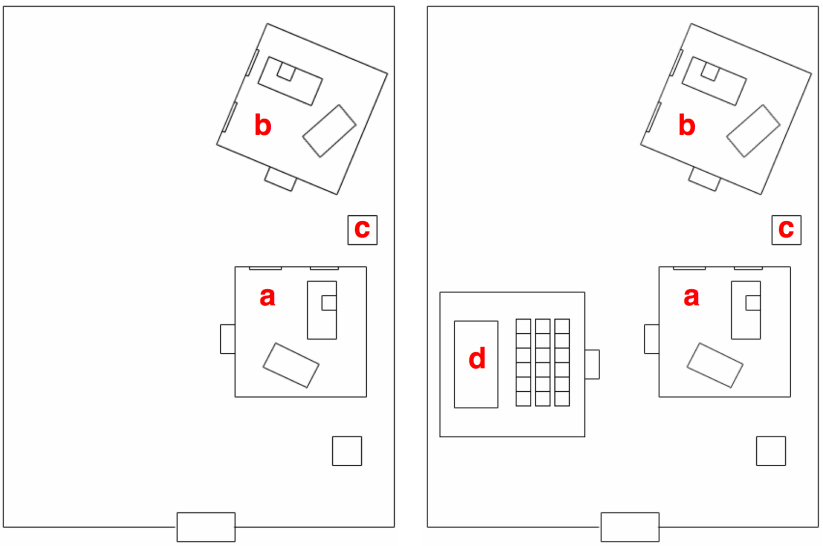


Figure 5.8: Initially generated floor plan (left) and the floor plan generated with the addition of the auction room (right)

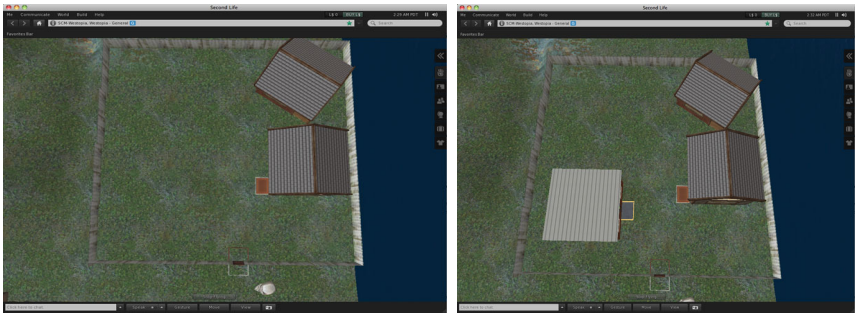


Figure 5.9: Initially generated 3D model (left) and the 3D model generated with the addition of the auction room (right)

the subsequently generated auction room.

Finally, once the Electronic Institution and Virtual World Grammar are specified, we map virtual world actions and AMELI events to the corresponding *Movie Script actions*. An example of virtual world action is an avatar raising his hand in the auction room that triggers a bidding Movie Script action. The bidding action is an example of the synchronous event, where its success or failure is visually announced to the other participants in the virtual world. We also map an AMELI event of a StaffAgent creating a new Auction scene to the Movie

Figure 5.10: Avatars participating in an ongoing auction

Script action where the system adds a new auction room to the visualization of the virtual world, where avatars will participate in the auction.

Once previous steps are defined, we can run VIXEE. Figure 5.8 depicts floor plans generated by the Virtual World Grammar, while Figure 5.9 depicts their visualization in Second Life. In the left part of Figure 5.9, we see a visualization of the eAuction House in Second Life after launching the institution and with *ItemRegister* and *ItemInfo* scenes. The right part of Figure 5.9 depicts the aerial view of the institution after the auction room has been generated. Additionally, Figure 5.10 shows avatars participating in the running auction, from the point of view of the auction manager. Figure 5.11 shows the AMELI interface: a top set of highlighted lines indicate that a seller agent registered a new item for the auction; a single highlighted line shows an event describing that staff agent entered (and thus created) a new Auction scene.

5.5 Evaluation

In this section, we evaluate the performance of VIXEE using the e-auction house Virtual Institution example. Participants of this institution are humans and software agents. Humans participate playing the roles of buyers and sellers. Software agents play either as staff, or as buyers or sellers. During the test we simulate virtual world actions of humans and software agents and measure

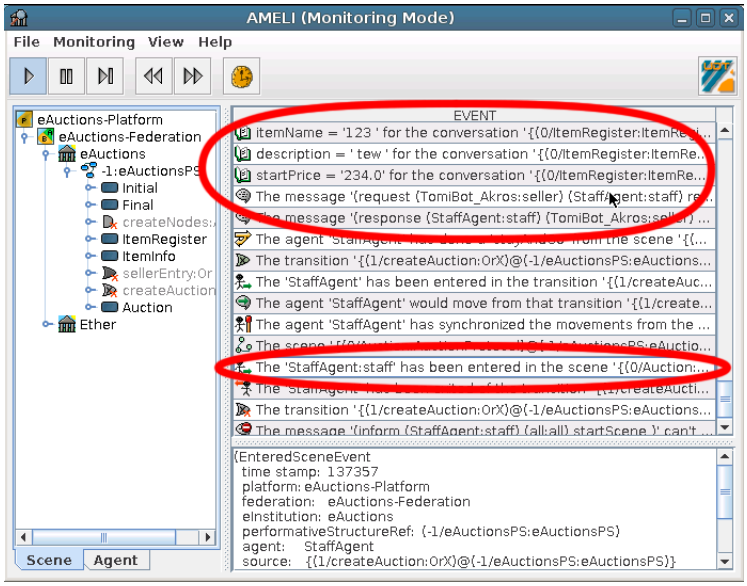


Figure 5.11: AMELI interface

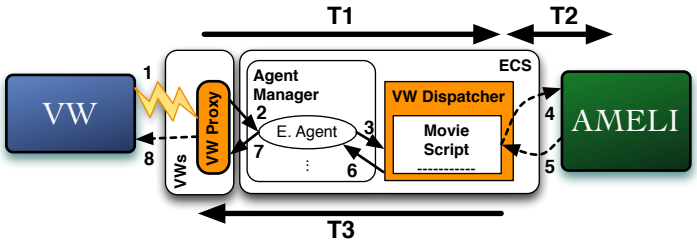


Figure 5.12: Measured response time intervals

VIXEE's response time to such actions (e.g. VIXEE's response time to validating the entrance to a particular scene or response time to a bidding action). The simulation of actions is done by feeding VIXEE's proxy with VW event calls (step 1 in Figure 5.12) and measuring time of receiving a response (step 8 in Figure 5.12). This test was repeated ten times recording obtained values. We have measured this response time in three intervals:

- **T1** is the time interval between receiving the message from virtual world and sending it to AMELI server (that is time of step 1, step 2 and step 3 in Figure 5.12). In this interval VIXEE parses the message from text format into the form understood by VIXEE, finds appropriate Movie Script action and executes it and depending on the result of the action it then sends this message to AMELI.

- **T2** is the time interval between sending and receiving the message from AMELI (that is time of step 4 and step 5 in Figure 5.12).

- **T3** is the time interval VIXEE needs to process the received AMELI response and send it back to the virtual world (that is time of steps 6, 7 and 8 in Figure 5.12).

To simulate actions of humans and software agents, we have created two different sets of actions (i.e. plans) that VI users typically perform within the eAuction institution. First plan is performed by the simulated human user with the *buyer* role. In this plan, the human user enters the institution, obtains the list of items registered for an auction, and then enters the auction. In order to get the list of auctioned items, the human avatar communicates with the *staff* agent. Second plan is executed by a simulated software agent with the role of *seller*. We assume that a human user programs the software agent to register the items for him. In this plan, agent enters the institution, registers the item and then leaves the institution.

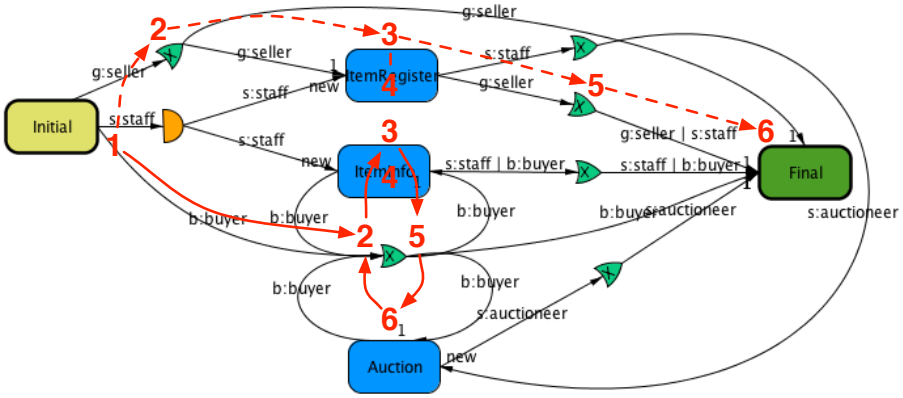


Figure 5.13: Two different plans that VI participants follow during the test (dashed lines are for SW agents and solid lines for human users).

Figure 5.13 shows the actions that follow both simulated humans and software agents. In the plan for simulated human users (marked with solid lines), humans enter the institution and move to the *Initial* scene (1). Then, they exit the *Initial* scene (2) and move to the *ItemInfo* scene (3) where they request the information about currently auctioned items (4). Then, they exit the *ItemInfo* scene (5) and move to the *Auction* scene (6). After moving to the *Auction* scene, agents decide not to participate, so they exit the scene (2) and move back to *ItemInfo* scene (3). This creates an infinite loop of actions. In step (4) we simulate the execution of a complex Movie Script action (e.g. finding specific auctioned items, considering a huge amount of registered items). We set the execution time of this action to 1500 ms. We use this simulated action to prove

that scalability of VIXEE depends only on the implementation of the specific VI that is, its Movie Script actions.

The plan for software agents in Figure 5.13 is marked with a dashed line. In this plan agents enter the institution and move to the *Initial* scene (1). Then, they exit the *Initial* scene (2) and move to the *ItemRegister* scene (3), where they register some item (4). Then, agents leave this scene (5) and exit the institution (6). When their plan is completed, agents restart it.

We ran the test in multiple threads where each thread ran a predefined number of agents. The test randomly decides how many human users and how many software agents will simulate. Threads run in parallel, where each thread executes actions in the following manner: (i) randomly select an agent; (ii) execute one step from agent's plan; (iii) wait; then thread waits a random time from an interval of $[0, 3]$ seconds. Executions of random agents in random time simulates real-world behavior where different actions from different agents are executed simultaneously.

To evaluate the system scalability we ran two different tests with different amounts of threads and agents and compared the results. We test the system response time from two different aspects:

- Testing the average response time separated to presented three intervals (T1-T3) of each of the steps of the plan (1-6), for 100 and 500 agents.
- Testing the total average response time while incrementally increasing the number of active agents

	T1	**T2**	**T3**	**Total**
With step 4	87,79 ms	2,92 ms	0,03 ms	90,75 ms
Without step 4	2,22 ms	3,18 ms	0,04 ms	5,45 ms

Table 5.3: Average response times for 100 agents (in milliseconds)

First test ran with 10 threads, each running 10 agents (100 agents in total). First row of the Table 5.3 shows the average response time of each interval T1, T2 and T3, along with an average *total* response time. Second row of the Table 5.3 shows the average response times without the step 4 (i.e. complex Movie Script action). Then, we see that the average response time drops to 5 ms. This shows that the limits of VIXEE are bound to the complexity of the Movie Script actions, which is domain dependent, since it corresponds to the implementation of the specific Virtual Institution. To further illustrate this, Figure 5.14 shows the average execution times for each of the six actions for 100 agents, clearly showing that action 4 takes the longest execution time.

In the second test, we ran 25 parallel threads, each running 20 different agents, that is 500 agents in the same virtual environment. Agents were joining VIXEE in zero to three seconds interval. First row of the Table 5.4 shows the average time of each interval T1, T2 and T3, along with the average *total*

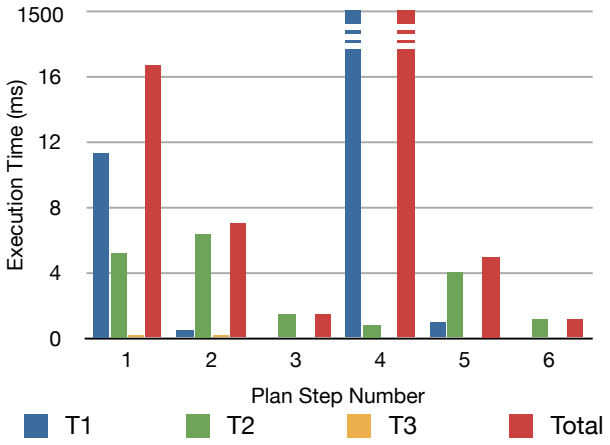


Figure 5.14: Average step execution time for all steps for 100 agents

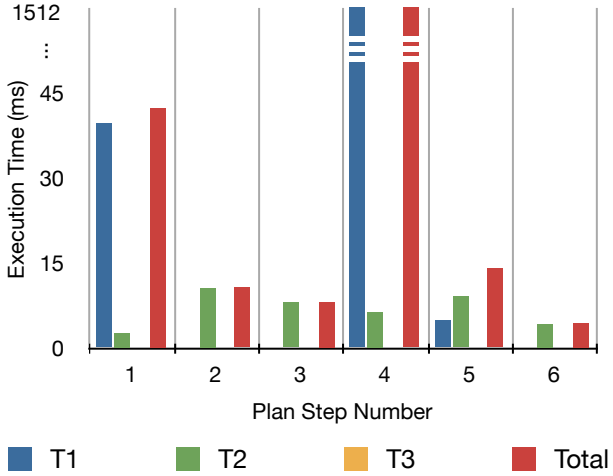


Figure 5.15: Average step execution time for all steps for 500 agents

	T1	**T2**	**T3**	**Total**
With step 4	91,54 ms	6,83 ms	0,05 ms	98,44 ms
Without step 4	8,81 ms	6,86 ms	0,05 ms	15,73 ms

Table 5.4: Average response times for 500 agents (in milliseconds)

response time with all six actions included. Second row of the Table 5.4 shows the average response time without step 4 (that is the step where we perform a Movie Script action taking 1500 ms). By comparing both tables, we can see, that even that we have increased the number of agent five times, VIXEE's average

response time increased from 5 ms to 15 ms (making the relation sublinear). Figure 5.15 shows the average execution times for each of the steps. We can note that in comparison with the times from Figure 5.14, the sublinear relation is kept for all actions.

Another aspect that we have evaluated was the total response time ($T1+T2+T3$) during the incremental load. We have been adding agents one by one till we were running 500 agents. We let all agents execute some actions in parallel, and we measured the VIXEE's response time. Figure 5.16 shows the graph of average response times depending on the number of connected agents. We can observe that even with the very high number of connected agents VIXEE was slowing down steadily with the average response time around 20 ms. We have approximated the performance decrease by a linear function $y = 0,0095x+15,34$, that just corresponds to 0.9% performance decrease by each connected agent. The computed coefficient of determination is $R^2 = 0.0003$[3].

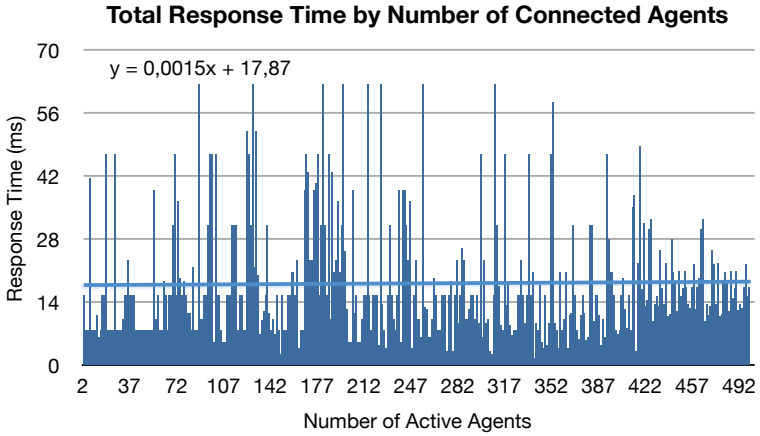


Figure 5.16: Average response time with different number of connected agents

5.6 Summary

In this chapter:

* We have presented the design of the middleware that provides a causal connection between several 3D virtual worlds and a runtime infrastructure that enforces interaction rules, AMELI.

* We have proposed the *Movie Script mechanism* that extends the original Action/Message table. Movie Script defines the context for which a selected Movie Script action will be executed. This context is formed by

[3]The y function and R^2 function were computed by the standard functionality of the Numbers program for Mac

the description of related AMELI event or a virtual world action. Such Movie Script action either transforms a virtual world action to an AMELI message, or an AMELI event to a virtual world update.

* We have presented an e-auction example contemplating VIXEE features.
* We have evaluated VIXEE's performance.

In the next chapter, we start our discussion on crowd simulation in 3D virtual worlds. We introduce a generic model for intelligent virtual agents that are designed to participate in Virtual Institutions.

Chapter 6

VI Agents

Simulating large crowds of virtual agents has become an important problem in virtual reality applications, video games, cinematography and training simulators. In this chapter, we approach this problem and present a new general-purpose agent model which provides the possibility to generate crowds of agents, where each agent has a unique behavior and appearance. This model *combines features of Artificial Life's digital organisms with features of Intelligent Virtual Agents*. It is designed to be executed in the context of Virtual Institutions, which are normative 3D virtual worlds, where participant interactions are controlled by an Electronic Institution, a well-established organization centered Multi-agent System (OCMAS). We use the Uruk city example, presented in Section 1.6.2, to explain particular concepts.

First, we introduce the motivation for the creation of a new virtual agent model. This section also presents motivations for the work on automatic generation of avatar appearance, presented in Chapter 7. Then, we present our approach and explain what are our contributions. In the rest of the chapter, we present all model components following by the description of their implementation.

6.1 Motivation

Nowadays, most virtual worlds represent online communities, where human users multi-modally interact in a simulated 3D space. In many cases (e.g. IMVU[1], Smeet[2]), 3D virtual worlds serve little function beyond 3D environments for online chatting. In this thesis, we are concerned with using 3D virtual worlds in more advanced scenarios, such as e-* applications and social simulations (as described in Chapter 1).

A significant class of such scenarios represents e-* applications, allowing e-commerce, e-government and e-learning participants to perform their tasks in

[1] `http://www.imvu.com/` (05/2012)
[2] `http://en.smeet.com/` (05/2012)

simulated 3D environments. [Bogdanovych, 2007] notes that 3D virtual worlds provide an immersive experience and support valuable real-world attributes like social interaction, location awareness, advanced visualization, collaborative shopping and impulsive purchases, which can improve existing practices in e-commerce portals [Bogdanovych, 2007]. Attributes, such as immersive experience, social interaction and collaborative approach to solving problems could facilitate more interactive and practical applications in e-learning and e-government.

Another class of virtual worlds' applications we are concerned with are social simulations. Social simulations represent a research field that applies computational methods to study issues in social sciences, as well as to other disciplines that study complex human behavior. The issues explored include problems in sociology, political science, economics, anthropology, geography, archaeology and linguistics [Takahashi et al., 2007]. In social simulation, computers support human reasoning activities. This field explores the simulation of societies as complex non-linear systems, which are difficult to study with classical mathematical equation-based models [Weidlich, 2000]. To highlight the practical benefits of using 3D virtual worlds for social simulations, in this thesis, we focus on the e-learning domain and simulate the society of the ancient Mesopotamian City of Uruk (see Chapter 8).

Social simulations, e-* applications and their execution in 3D virtual worlds are the main application domains we are concerned with in this thesis. We introduce methods that facilitate their definition and execution. In previous chapters, we have focused on automatic generation of such 3D virtual worlds. For this purpose, we have defined the Virtual World Grammar, which generates a 3D environment of the virtual world from the formal specification of activities being performed in this virtual world. Later, using VIXEE, we have connected this world with an Electronic Institution, making the virtual world normative, where interactions can be structured and controlled. At present, this generated environment contains no initial population.

Having empty 3D spaces with no population is not adequate for e-* applications and social simulations. In e-* applications, we stress the importance of social interactions and the collaborative approach. In social simulations, the issue is even stronger as it depends on execution and study of artificial societies. Therefore, we need to populate generated 3D space with avatars.

Very often, these avatars are required to complete frequent repetitive tasks where making errors has significant negative consequences. An example is an e-commerce application, which uses avatars to approach world visitors and collect poll data from them. Recording incorrect data could lead to the corruption of poll results. Moreover, poll data often has to be collected in any given moment during the day, thus "poll avatar" availability has to be assured 24 hours a day. Moreover, in social simulations, we often simulate large societies; therefore, it is needed to populate space with a large number of avatars. In all these instances, it is not practical to control avatars by human users, therefore we need a mechanism that would substitute human presence.

The popular substitution mechanism is the replacement of human presence by agents. In artificial intelligence, an intelligent agent (IA) is an autonomous entity that observes through sensors, acts upon an environment using actuators and directs its activity towards achieving goals [Russell et al., 1995]. Using agents with agent-based modelling (ABM) is a powerful simulation modelling technique that has seen a number of applications in the last few years including applications to real-world business problems [Bonabeau, 2002] [Farmer and Foley, 2009]. In agent-based modelling, a system is modelled as a collection of autonomous, decision-making agents. Each agent individually assesses its situation and makes decisions on the basis of a set of rules. Agents may execute various behaviors appropriate for the system they represent, e.g., producing, consuming, or selling. Repetitive competitive interactions between agents are a feature of agent-based modelling, which relies on the power of computers to explore dynamics out of the reach of pure mathematical methods [Bonabeau, 2002]. Agents and ABM are often used in social simulations, where agent-based social simulation (ABSS) [Li et al., 2008] models the different elements of the social systems using artificial agents, and placing them in a computer simulated society to observe the behaviors of the agents. Deploying ABM to the domain of 3D virtual worlds, we represent agents by avatars and let them act in a simulated 3D environment. Avatars autonomously navigate in the 3D virtual world, using data from their sensors and interact with the environment, human users and other agents. Thus, by representing agents with avatars and using ABM, a well-established system modelling technique, we are assured that agents effectively substitute human presence in 3D virtual worlds.

Substituting humans by agents introduces several challenges for related applications [Thalmann et al., 2009]. In the area of social simulations, we are concerned with the simulation of large crowds. We require that the simulated crowd is believable. We approach crowd believability on two different levels. On the macro (crowd) level, we want each crowd member (agent) to be visually and behaviorally unique; thus, assuring the believability when overlooking the crowd. On the micro (agent) level, we want each agent to behave believably, mimicking the human behavior. With regards to believability on the macro level, the simplest method to generate a large crowd is by replicating (i.e. cloning) a single individual until we reach a desired number of individuals. Naturally, this leads to identical appearance and behavior, undesired for believable crowds. Therefore, we need a mechanism that automatically alters individual appearance and behavior, thus producing unique agents.

The most straightforward method of creating unique agents is the manual modification of their visual and behavioral features. A manual method for designing hundreds (or even thousands) of such avatars is a time-consuming and inefficient task, and a higher level of automation is desired. A common automation approach is a definition of several appearances and behaviors and their consecutive variance between crowd members [Barbara, 1998] [Maim et al., 2009]. This is also a most common approach in role-playing computer games[3]. This

[3]World of Warcraft, Assassins Creed

approach requires a lot of effort in generating sufficient predefined appearances and behavior patterns, so that a large crowd remains believable and appearance repetition and behavior repetition is not visible.

Another popular option [Lewis and Parent, 2000] [Thalmann, 2007] [Magnenat-Thalmann et al., 2004b] is to isolate and quantify specific visual and behavioral features (e.g. height, foot size, friendliness) and then, for every agent, randomize their values (e.g. producing agents with different height, foot size, having different levels of friendliness). Although very effective, randomization of features sometimes leads to unacceptable results (e.g. generating unusually small avatars with giant feet). Moreover, it is difficult to maintain the visual properties of a specific ethnic during reproduction (e.g. asian eyes). All the mentioned approaches seem to be lacking important features, thus inadequate for our purposes. When looking for alternatives, we found inspiration in evolution, responsible for creation of unique organisms, even within single species.

Using a genetic approach, we first manually develop a small representative number of individual agents capable of autonomous actions. This initial sample models all required population groups and defines the initial population of a virtual world. Then, we use the genetic approach to generate a large diverse crowd of a desired size, where each generated agent is an offspring of agents from the existing population.

In Chapter 7, we mimic the genetic reproduction principles found in nature. In the artificial genetic model, we propose to encode an avatar's visual features and behavioral features into its genes, which form agent chromosomes. Chromosomes of two different avatars are combined and mutated, to create a new, unique avatar. This provides a practical solution for introducing an automatic way of behavior and appearance variation in large avatar crowds

Using a genetic approach, we first manually develop a small representative number of individual agents capable of autonomous actions. This initial sample models all required population groups and defines the initial population of a virtual world. When initial population is defined, we use the genetic approach to generate a large diverse crowd of a desired size, where each generated agent is an offspring of agents from the existing population.

Generating a large crowd using genetic approach assures believability on the macro level, overlooking the crowd from the distance, by generating agents with unique appearance and behavior. On the micro level we want individual agents to behave believably, that is human-like [Loyall, 1997]. Such believable behavior is achieved by modelling human personality and emotions [André et al., 2000], human processes such as physiology (e.g. hunger, energy) [Gillis, 2000], decision making under stress situations [Janis and Mann, 1977] [Silverman et al., 2006b] or using non-verbal behavior, such as facial expressions and speech [Cassell et al., 1994] [Gratch et al., 2002], eye movement and gaze [Bickmore and Cassell, 2005], gestures [McNeill, 1996] and head nods [Lee et al., 2010] during agent interactions.

Personality and emotions are of our particular interest. Emotions are af-

fected by personality, when two individuals with different personalities react differently to the same situation. In the physical world, each person has a unique personality. In the virtual world, encoding personality values into agent chromosomes assures generation of agents with unique behavior, thus believable on the "macro" level. Execution of psychological processes defined by these values introduces believability on the "micro" level.

Agent believability presents a separate scientific field, which belongs to the bigger research field of IVA. This work uses existing knowledge from IVA field (e.g. [Kasap and Magnenat-Thalmann, 2008] summarizes state-of-the-art for IVA), along with information from evolutionary computing and A-Life. Thus, based on the literature review from these fields, we have created a general-purpose virtual agent model where physiological, psychological and social needs of humans are simulated in a believable way, while strongly relying on existing theories about human behavior. This generic model combines properties both from A-life digital organisms (see Section 2.6) and Intelligent Virtual Agents (see Section 2.7). A-Life inspired us to encode agent properties into chromosomes and to use the genetic algorithm approach for generation and evolution of avatars. Then, we used the IVA theory on believable agents and defined models for agent appearance, psychology and physiology. This generic model allows us to perform A-Life evolutionary simulations with a high level of sophistication.

Such generic models, however, do not take into account specific cultural characteristics of the simulated human population and established social norms. For the purposes of social simulations, when simulating different cultural groups and their interactions, we need a way of encoding culture-specific properties and culture-specific rituals. To address this drawback, we embed the general-purpose agents into the model of virtual culture [Bogdanovych et al., 2010b]. Virtual culture holds a definition of locations, actions, objects and rituals specific for given culture. Agents can learn new cultural information, allowing us to simulate dissemination of cultures.

Another drawback of such generic models is that they are independent of the external environment. In standard scenarios (e.g. games, simulations), agents are explicitly defined by a list of actions, usually specified as decision trees [Gemrot et al., 2009] [Adobbati et al., 2001], and objects that they use to accomplish these actions. This approach allows fast but restrained reasoning over existing sets of objects. In the applications domains of our interest, such approach is not adequate. In social simulations, agents reason and plan their actions using complex data from the changing state of an environment. Previously mentioned approach would explicitly give agent knowledge about all possible objects, their location in the world and all the possible interactions. This significantly increases the complexity of agent definition. We seek a more dynamic approach, where an agent "discovers" possibilities of the environment in which it acts. Therefore, we propose a model for Virtual World Objects (VWO), which defines the meta-data for 3D objects, which allow an agent to reason about them and use them in their planning. Using VWO, we are able to shape agent actions by dynamically changing the environment, or even incrementally introduce new

objects to the simulation.

VWO complements the design of our generic purpose agent model. Considering all of the presented properties of this model, we are capable of accurate simulations of large human societies. Furthermore, individual virtual agents are suitable for being employed in all the application domains this thesis is concerned with.

Considering our motivations, we conclude that, for e-* applications and social simulations executed in normative virtual environments, one needs a way of automatically populating them with a desired number of virtual agents. Those agents must be intelligent, believable and aware of their dynamic environment, while it is desired not to design them in a manual fashion, but have a high degree of automation. Therefore, this chapter will investigate how to build an agent model that contains information on agent culture, allows to utilize the knowledge about the environment that comes with a Virtual World Grammar and automatically populates this environment with avatars.

Thus, further in this chapter we explain how our work extends existing state-of-the-art presented in Section 2.7 and state our contributions. Then, we present all parts of this generic agent model. This model is evaluated in Chapter 8.

6.2 Approach Overview

In this section, we present our approach to the creation of a generic model for believable Intelligent Virtual Agents (IVA) that can be automatically generated with a unique appearance and behavior. In Section 2.4, we presented the background on IVA and explained how to approach agent believability. Then, we introduced the state-of-the-art for the IVA models and stated why existing models are inadequate for applications concerned by this thesis.

In general, all existing IVA models depend on the declarative definition of agent behavior. In this approach, agent behavior specifies when and where to perform an action. What exact action to perform is decided depending on several external (e.g. state of the environment) and internal (e.g. emotional state) factors. Agent decision on a current action is deduced either directly from a decision tree [Gemrot et al., 2009] or using a custom evaluation function [Silverman et al., 2006b] that considers both, an external and an internal agent state. In the case of decision trees, we are forced to manually specify goals for each agent or a group of agents. Using the functional approach, we need to "tune" the configuration of functions so that agents behave accordingly to implementation specifics, while a given function configuration needs not to apply to an implementation in a different domain and needs to be reconfigured.

In our approach, we aim for a more automated solution. The focus of this thesis is the automatic generation of virtual environments from a formal specification of activities performing in this environment. In this work, we consider a performative structure of an Electronic Institutions, an Organization-Centered Multi-agent System (OCMAS), which structures and normatively controls agent execution and interactions. Agents, using information stored in such specifica-

tion can automatically reason about their goals, and also plan collaboration with other agents in order to achieve common or individual goals. Goal, in our case, is an illocution that has to be uttered in a specific scene. For example, illocution $eat(calories)$, in the scene Eat, tells an agent to consume food with a specified number of $calories$.

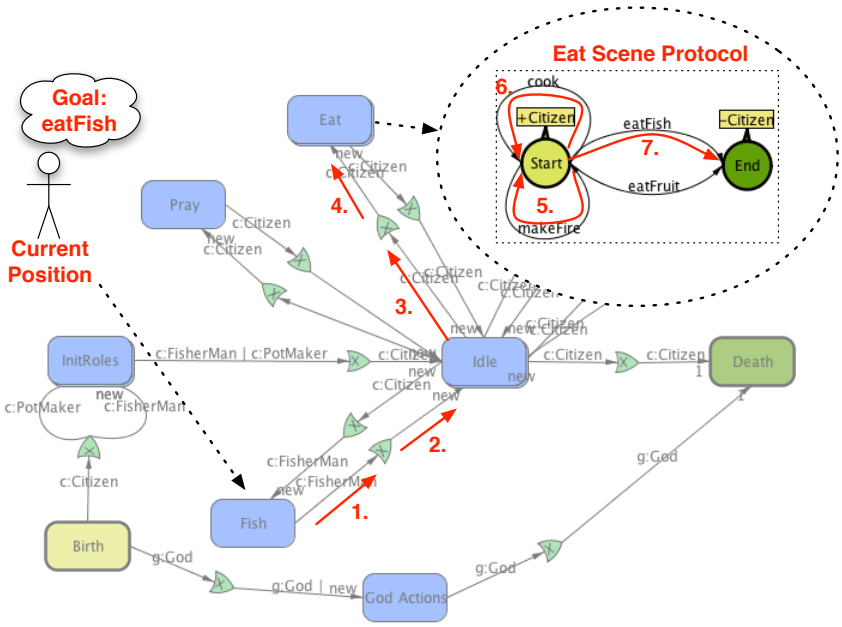


Figure 6.1: Plan creation process overview

Thus, to achieve a goal means to navigate in a performative structure of an Electronic Institution from a current location (i.e. scene or transition) to the scene state that is the output node of a scene arc containing the illocution defining this goal. Figure 6.1 uses a performative structure of the Uruk Electronic Institution (see Chapter 8) to explain how a plan is created. In this figure, a Fisherman agent is currently in the Fishing scene, and its goal is to $eatFish$. Thus, the agent performs a dynamic search for a sequence of actions that would lead to satisfying the goal. As a result it creates a list of seven institutional actions that lead him to the arc in the Eat scene containing the $eatFish$ illocution. Those actions include catching the fish, waiting for the $Fish$ scene to evolve into a state when the agent can leave this scene, exiting the scene, progressing to the Eat scene (through the $Idle$ scene) and in the Eat scene performing the illocution that results the fish being cooked in the fireplace and finally eating the fish. Having the institutional specification in place allows the agent to conduct high level reasoning about its options and then dynamically construct a plan that can satisfy its current goal, while the underlying VIXEE infrastructure provides

an efficient and powerful way of connecting the institutional specification with the sequences of atomic actions the agent must perform to execute its plans and progress toward its goal by performing institutional illocutions via actions in the virtual world.

The most straightforward way of specifying agent goals is to define a list of temporal goals for each role in the system. Temporal goals specify a time and a goal that has to be executed at a particular time. More importantly, in our approach we integrate several existing models of agent believability, what allows us to specify *high level goals*, which are dynamic processes modelled by a related part of the agent model, such as physiology, psychology or non-verbal behavior (see Section 2.7). An important part of agent believability is its spatial behavior, for which we advocate the use of the episodic memory [Nuxoll and Laird, 2007], which stores memories with their temporal and spatial factors. Integrating agent believability models allows us to specify goals such as "be happy," modelled by agent emotions or "survive," modelled by its physiology. Using this approach we need to include in the specification how possibly can an agent obtain food, or what makes it happy. With high level goals, agents focus rather on *why* to perform an action than *when*, *where* and *what* action to perform.

Using the *why* approach for goals, agents reason about their internal state to set their goals, which are executed in the virtual environment by interacting with other agents or environment. A plan for the current goal is created depending on the current state of the environment; thus, the agent needs to have a way of understanding the functionality of objects in the environment. In our approach, we have integrated agents closely with their environment by introducing the concept of Virtual World Objects (VWO). VWOs contain specific metadata that allows agents to reason about their goals (e.g. object apple, holds data saying that agent can use this object for eating, thus provides action of $eat(100)$). Thus, by changing the environment, we change the reasoning of agents about their goals. Unlike existing agent models, this approach allows to specify very high level goals (e.g. survive), and still it can model very specific aspects of agent behavior only by designing the environment. Such environment is generated using our Virtual World Grammar, thus achieving full automation of the generation process.

Apart from the benefit of the *why* approach, high level goals allow us to model so-called "fitness functions," which serve as a selection mechanism in a simulation of evolution using genetic algorithms. In our approach, we encode agent properties in genes, which form their chromosomes. These properties include their visual and psychological information (i.e. personality values). Unlike in existing agent models, using genetic operators and mutation, we can generate agents with a unique appearance and personality during application runtime. This mechanism permits to perform evolutionary simulations, well known from A-Life field, but with much higher level of sophistication. In this approach, Electronic Institution is used to model all the possible actions performing in the virtual environment, while a genetic component, along with believability components, dynamically decide the preferences of their execution.

Using Electronic Institutions allows agents to adopt and disseminate specific Virtual Culture [Bogdanovych et al., 2010b]. Virtual Culture models the social behavior and social decisions of an agent. It also models what rituals, with what object at which locations are performed, if an agent is a member of a given culture.

Thus, our main contribution is the generic Intelligent Virtual Agent model, which we developed based on existing work and extended it in order to accomplish our requirements on automatization of the virtual environment generation process. We also show, how this model can become culture specific by incorporation of Virtual Cultures and how it can be used in the evolutionary simulations. Next, we provide details on this model.

6.3 VI Agent Model

In this section, we describe our generic intelligent virtual agent model for agents acting in normative virtual environments. In our case, agents perform their actions within Virtual Institutions (VI) [Bogdanovych, 2007]; thus, we named them VI Agents. This model consists of several components, where some of these components depending on a specific virtual world (e.g. sensors and actuators), while some depend on selected normative infrastructure (e.g. reasoning). Virtual Institutions use Electronic Institutions [Esteva, 2003] as the normative infrastructure. Most of the presented components were implemented and evaluated in Chapter 8.

VI Agent model takes inspiration from the Artificial Life (ALife) systems. Such systems are designed from bottom-up, from the simplest to the most complex parts. Therefore, we explain components in order of their use, explaining first components that define structures used by others.

The *timer* simulates the real-world time, allowing to speed up or slow down the execution of the current simulation. The time sensor is responsible for observing current system time and periodically informing related components, what can trigger the execution of a time-dependent function.

The *genetic* component stores selected agent data, such as its appearance and psychological profile values. Ordered sets of genes form agent chromosomes. Chromosomes are operated with genetic operators, such as crossover and mutation, during the process of agent replication, leading to generation of agents with a unique appearance and behavior (see next Chapter 7).

Several agent *believability* components, such as *psychology*, *physiology* and *non-verbal communication*, integrate existing models supporting agent human-like behavior. Considering the *physiology* component (i.e. the homeostatic model). It describes an agent specific physiological needs and evaluates physiological processes. An example would be agent metabolism, where food produces energy and agent needs food to survive. Survival can be modelled as agent's high level goal. An extension to existing *psychology* models is our concept of predispositions. Predispositions work as action modifiers (e.g. talent or handicap in performing some activity).

The *memory* component is responsible for storing the spatio-temporal data on the state of the environment, enforcing believable spatial behavior of an agent. We adopt the existing model of *episodic* memory [Rolls et al., 2002] [Nuxoll and Laird, 2007]. This model stores memories of objects and actions that took place at a specific time in a specific space. It also maintains custom associations between memories, allowing an agent to quickly reason about the state of the world.

The *culture* component defines structures and executes processes related to the culture of an agent, which (i) respects cultural rituals performed at special locations (e.g. christian prays in church, muslim prays in a mosque), (ii) uses culture-specific objects, and (iii) performs specific gestures (e.g. western-world shake hand to greet each other while asians bow). It also defines social protocols specific to a given culture. Culture component is an implementation of Virtual Culture proposed by [Bogdanovych et al., 2010b]. An agent can become part of different cultures and also extend its knowledge of other cultures.

The *sensor* and *actuators* are virtual world dependent components that observe environment and perform agent actions in the specific virtual world. Sensors observe currently visible objects, visible agents and actions that update the virtual world state (e.g. chat, voice, door opening). They are responsible for extracting the semantical data about interactive objects stored in the model of Virtual World Objects. This model contains semantic annotations, which allow agents to use object data during reasoning and adapt plans depending on the current state of the environment. Actuators perform agent actions (i.e. movement, chat, speech, gestures) in a virtual world virtual world. Such actions include moving avatar to new locations, moving objects, interacting with objects or even communicating with other avatars or human users through chat or audible channels.

The *reasoning* component is responsible for selecting a current goal, creating a plan of actions for this goal, and executing it. Goals are either specified manually as a list of temporal goals, or as high-level goals evaluated by a specific component (e.g. goal "survive"). Reasoning component depends on the selected normative infrastructure as it uses specification dependent data create a plan of actions. In our approach, we use a performative structure of an Electronic Institution. VI Agent creates plan depending on the specification and current state of an Electronic Institution, the state of the virtual world, the state of the agent and the data stored in its memory. Reasoning is triggered by the timer or an external event (e.g. interaction with another agent).

Figure 6.2 contemplates a conceptual view of the VI Agent architecture. It represents all components of the VI Agent model and shows how these components contribute to reasoning. Please note the Electronic Institution (EI) specification (yellow box). Electronic Institution is defined by a dialogic framework, performative structure and norms (see Section 2.3). Each VI Agent plays a selected role in the EI. This role is a part of the dialogic framework. Each role has a list of temporary goals. Agent goals are represented by an illocutionary message M from some scene protocol. In the rest of this text, for message M

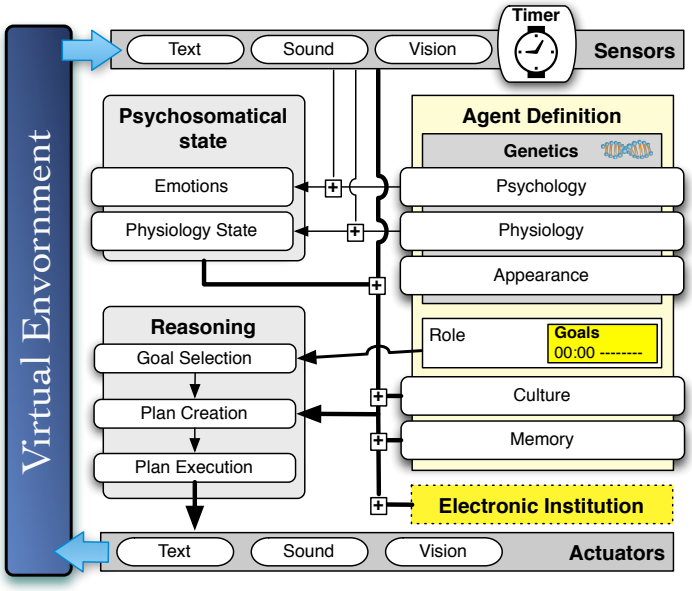


Figure 6.2: Conceptual view of the VI Agent architecture

we use Scene.MessageName notation. Message M can have n parameters. An agent plan consists of institutional actions forming the path in the EI performative structure from agent's current institutional location (i.e. scene or transition) to the arc containing a message M. An Institution specifies normative protocols of agent communication and actions, thus fulfilling a goal means to create a list of institutional interactions that will lead to the accomplishment of this goal.

Our proposed architecture follows the BDI - Beliefs Desires Intentions approach for agent modelling. In this context the *reasoning* component evaluates agent's *desires* and creates a plan executed by *actuators*, representing *intentions* of an agent. The rest of modules operate with *beliefs* of an agent.

In the rest of this section, we provide the detailed information on all components of a VI Agent model.

6.3.1 Genetics

Genetic component encapsulates the functionality of A-Life digital organisms, that is genetic reproduction and evolution. It contains structures for storing agent data, used during these genetic processes. These structures are represented by the set of chromosomes that contain genes holding qualities and properties of an agent. These chromosomes are:

1. **Appearance** chromosome holds appearance values of an agent.

2. **Psychology** chromosome holds personality traits and predispositions. Personality values depend on the selected personality model (e.g. the big-five model).

3. **Physiology** chromosome holds values that specify agent's physiological requirements and measures. For example, how fast an agent is burning energy.

During agent replication, the genetic component generates a new agent by using approaches from genetic algorithms (i.e. crossover, mutation and deep inheritance). As a result, generated agent is an evolved version of its parents from physical, psychological and physiological point of view. For more details on the genetic process, please see Chapter 7.

6.3.2 Believability

VI Agent model concerns believable agents. Agent believability is achieved by integrating selected existing models on humanlike behavior of agents. These models represent psychological processes (i.e. personality and emotions), physiological processes and non-verbal behavior during interactions, such as facial expressions and speech [Cassell et al., 1994] [Gratch et al., 2002], eye movement and gaze [Bickmore and Cassell, 2005], gestures [McNeill, 1996] or even head nods [Lee et al., 2010].

For personality model, we advocate the use of the big-five (OCEAN) model [Costa and McCrae, 1992] and OCC model for emotions [Ortony et al., 1988]. Personality shapes agents emotions affect agent's decisions. An example is a pair of agents with no money, no food and a big hunger level. An aggressive agent would try to rob other agents in order to obtain food, while a submissive agent would try to beg (that is, if we allow begging and robbing in the simulation).

We introduce a new concept of *predispositions*, which extends the personality model. These predispositions can have positive, or negative influence on the agent's performance. Positive ones are talents (e.g. skilled fisherman), and negative predispositions are handicaps (e.g. weak vision). These predispositions are stored in agent's psychological chromosome and passed to new generations. Predispositions can mutate as well; thus, they become an essential part of agent evolution.

The physiology forms basic survival needs and processes of an agent. Such process is metabolism, which changes energy resources (i.e. food) to agent energy. Each agent has specific requirements on the energy consumption. Some agents burn the energy faster, some slower; some need a lot of food to feed; others need to eat only small amounts to fully replenish their energy. Physiology component is split in two parts: (1) a model that holds agent's physiological needs and (2) agent's current physiological state. It is an analogy to psychology, where personality shapes emotions.

6.3.3 Culture

[Bogdanovych et al., 2011] introduced the project of Authentic Interactive Reenactment of Cultural Heritage With 3D Virtual Worlds and Artificial Intelligence. In this work, authors studied traditional ways of preserving historical and cultural knowledge and proposed a new, modern, interactive way, based on Virtual Institutions. In these "cultural simulations," culture-specific structures are modelled in 3D, avatars wear traditional clothes and perform culture-specific tasks. Agents are able to apprehend and disseminate information about cultures. Human users communicate with agents to learn about their culture. Authors define a *virtual culture* as follows:

Definition 6.1. *Virtual Culture* is defined as a tuple $C = (E, P, O, Ag, I, l)$ where:

1. E is the virtual environment
2. P stands for the set of virtual places occupied by a virtual culture
3. O stands for the objects produced by a virtual culture (buildings and artifacts)
4. Ag stands for a set of virtual agents
5. I stands for a set of Electronic Institutions structuring the interactions of virtual agents
6. $l : I \rightarrow P$ is a function mapping each Electronic Institution to its context (location), namely some virtual place

Using Definition 6.1, a VI Agent culture component contains *gestures*, *actions*, *objects*, *locations*, *rituals* where:

- *Gestures* are a set of gestures that agent can perform. Execution of gestures is a VW-dependent process
- *Actions* are a set of institutional actions, which agents can perform within Virtual Institution
- *Objects* are a set of annotated Virtual World Objects
- *Locations* are a set of locations where an agent can perform a specific action.
- *Rituals* are a set of rituals specified by an institution protocol.

Virtual agents in the context of virtual cultures are characterized by their appearance (i.e. avatar model, clothing), their cultural knowledge (i.e. beliefs) and by actions and gestures they perform (e.g. shaking hands instead of a deep bow). Each agent plays a specific role in the system, where each role has different

beliefs. Agents are able to exchange knowledge, their beliefs. This occurs after interacting with another agent in the context of an Electronic Institution. Thus, the content of the culture model is changing during the execution of a virtual world. Agents can learn new *actions* to perform their plans, they can learn about new *objects* and discover new *locations*.

6.3.4 Virtual World Objects

A virtual world is a computer simulated environment in which participants interact with each other and the environment. Such environment contains many objects. Most of these objects serve only decorative or informative purpose; therefore, participants cannot interact with them, or their interaction is not important for system execution. Those objects that somehow alter agent's state, or provide the possibility to execute an institutional action can be used to accomplish agent's goals; therefore, they can be used in its reasoning. Such change can either be visual (e.g. opening doors, grabbing an object) or non-visual (e.g. buying object deducts money from the user's account). We name this object model Virtual World Objects (VWO).

Each VWO is annotated with semantic information informing agents about (i) the owner of the object: no one, a specific agent, a group of agents and (ii) actions that can be performed with the object and how to execute these actions. Object's actions represent illocutions from a specific scene protocol of an Electronic Institution. For each possible action we define following:

1. **Name** - a name of the message (i.e. illocutionary action) M in form $Scene.MessageName$. This message can have n parameters.
2. **Parameters** - set of values for n parameters of the action M.
3. **Owner permission** - specifies if the owner can perform the action.
4. **Group permission** - specifies if agents from owner's group can perform the action.
5. **User permission** - specifies if all other agents can perform the action.
6. **Activation** - Indicates which Virtual World action that has to be performed in order to interact with the object (e.g. click, approach object, attach object, sit on object). Activation is performed only after successful execution of the message M in the Electronic Institution.
7. **Activation distance** - The minimal distance that an agent needs to have from the object in order to activate it.

In our approach, we map all illocutions of performative structure to the Virtual World Objects, and limit that any illocution can be uttered only by interacting with these objects. Thus, VWO forms fundamental part of agent reasoning, where the agent has to consider the spatio-temporal data of a VWO

(see Section 6.3.5). Also, each VWO contains a set of tags, that allow to categorize this object in agent's episodic memory.

Table 6.1 shows an example of an annotation for a Fireplace object from the Uruk example. In a fireplace, an agent can make a fire (represented by the message Cook.makeFire()) and cook specified amount of food (represented by the message Cook.cook(amount)).

Table 6.1 shows an example of an annotation for a *Fireplace* object from the Uruk example. In a fireplace an agent can make a fire (represented by message $Cook.makeFire()$) and cook specified *amount* of food (represented by message $Cook.cook(amount)$).

Name:	Fireplace					
Description:	A place to prepare food					
Owner:	no-one					
Tags:	foodProcess					
Message	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
Cook.makeFire	-	x			Click	1m
Cook.cook	amount	x			Click	1m

Table 6.1: Fireplace VWO annotation

Virtual World Object model provides the virtual world independency for agent reasoning, as agents use only VW independent data from VWO model and object position obtained by the sensor module.

6.3.5 Reasoning

In this section, we introduce the "brain" of an agent, a decision making component that depending on available data and the specified goal creates and executes a plan of actions. The process that includes selecting a correct goal, creating a plan to executing the plan is quite long and contains many functionalities. Therefore, we split the explanation of reasoning into three phases:

1. **Goal selection phase** - selects agent's current goal, that has the highest priority, it is relevant to its role; it is within the current time interval, and its execution did not fail in a recent time-frame.

2. **Plan creation phase** - creates a list of actions for the current goal, depending on the available data from other modules.

3. **Plan execution phase** - controls the execution of the plan of actions.

Goal Selection Phase

Each agent plays some role in the institution. Each role has assigned a list of goals, which are either high-level goals (e.g. survive) or specific temporal goals (e.g. from 8 AM till 1 PM "work"). A temporal goal is valid only within a specific

time interval. High-level goals are valid always. Purpose of the goal selection is to select a valid goal to execute and pass this goal to the plan creation phase. Each temporal goal is defined by a time interval and an institutional message in form Scene.MessageName which has to be uttered in order to achieve this goal (e.g. *Eating.eat*, Fish.catchFish).

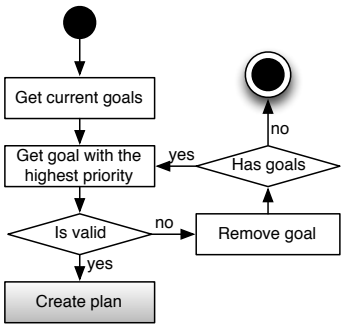


Figure 6.3: Activity diagram of a goal selection phase

Figure 6.3 shows the activity diagram of the goal selection phase. Selection phase starts by selecting goals valid for the current time and ordering them by a priority. Next, it selects the goal with the highest priority. Each goal can be marked either as *valid* or *invalid*. A goal becomes invalid if its execution fails a predefined number of times in a recent time interval. The execution of this phase can fail due to following reasons: (i) there exist no plan for current goal in this moment, or (ii) the execution of a plan failed. If the goal is invalid, it is removed from the selected list of goals, and the algorithm continues. If the goal is valid, a reasoning component tries to create a plan for the goal.

Plan Creation

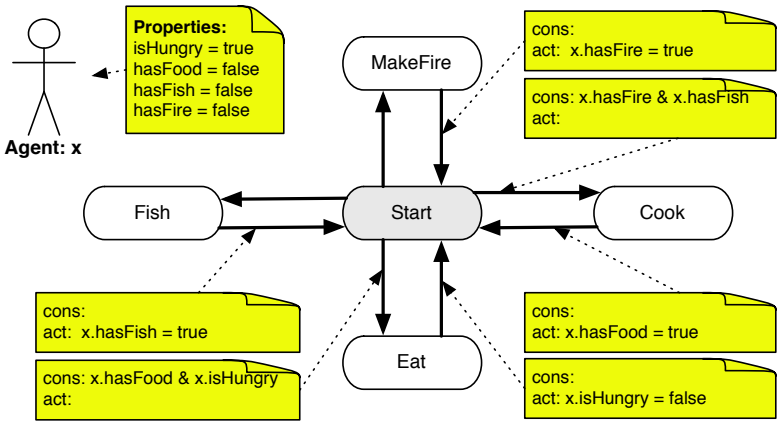
Creating a plan for an agent's current goal means creating a list of institutional interactions, represented by individual plan items, that leads to fulfilment of this goal. Creating this protocol means finding a path from agent's current institutional position to the arc containing the goal represented by an illocution. Thus, each plan item is represented by one of the following:

- Scene to Transition arc (ST) defines the way how to move from a scene to a transition.
- Transition to Scene arc (TS) defines the way how to move from a transition to a scene.
- Message/Illocution to be uttered (i.e. perform an action) in a scene protocol of a scene.

In Section 2.3 we contemplate that each arc in the performative structure (i.e. ST, TS, or message) can contain context-dependent constraints and actions. These constraints and actions provide the logic for the plan creation. Imagine a simple cooking example, where you cannot cook food (i.e. fish) unless you have something to cook and the fire is on. When we are creating a plan, we need to fulfill all these constraints. Constraints and actions contain (i) agent properties (ii) scene properties and (iii) message properties. Agent properties are part of agent state; they change their value during an EI execution. Scene properties are a part of institutional state, and their values change during agent participation in the scene. Message properties are a part of the semantic description of actions provided by the VWO. For example, message *eat* with parameter *quantity*, can be said in the scene *Eat*. Quantity specifies how many units of hunger we can subtract from our hunger level. For example, there are two edible VWOs: an apple and a cooked fish. Surely, they both are edible and both satisfy our hunger differently. Thus, they both provide action *Eat.eat*, but apple provides parameter 1 (*Eat.eat*(1)) while cooked fish parameter 5 (*Eat.eat*(5)) We can now easily imagine how VWOs are used during reasoning, when finding a path to the goal and satisfy all constraints. If no plan exists for all valid goals, agent can either start a random walk in order to discover new ways to plan its actions or try to execute another goal.

Figure 6.4 contemplates the plan creation process from the cooking example. In the upper part, we see a simple performative structure with scenes connected with arcs annotated with arc constraints and actions. For simplification of explanation, we left out transitions between scenes. In the bottom part of this figure, we see a table which displays the reasoning process of an agent to accomplish its goal: *Eat.eat*. In the first step of the plan creation, the agent tries to find a path to the scene *Eat*, but finds out that it does not fulfill the constraint *hasFood*. The agent uses the information stored in its memory and the information stored in the performative structure to find a path that fulfills the *hasFood* constraint. Looking at the performative structure, we see that if the agent passes through the *Cook* scene, its state will change to fulfill this constraint, but also it introduces two more constraints *hasFood* and *hasFire*. The agent continues reasoning and in four steps creates the plan: *Fish.MakeFire.Cook.Eat*.

As mention earlier, when an agent decides to perform an action, it first needs to find a Virtual World Object (VWO) that provides execution of this action. VWO provides semantical data on what actions can be performed, what is the distance this agent needs to have from the object in order to perform it, what is the activation type for this action and what are the action parameters (see Section 6.3.4). Activation is usually done by touching or approaching the object. In our design, *virtual culture* limits which objects can be used for specific actions. For example, to accomplish a goal *Pray*, agents of different religions (cultures) would go seek different places to pray (e.g. church, mosque). Thus, following our cooking example, the agent creates all possible plans of actions and selects the one using the following criteria (i) has the shortest list of actions and (ii) the VW distance to travel to accomplish the action is the shortest.



GOAL: Eat

Step #	Plan	Problem Constraints
1	Eat	x.HasFood = false
2	Cook.Eat	x.HasFire = false; x.hasFish = false
3	MakeFire.Cook.Eat	x.hasFish = false
4	**Fish.MakeFire.Cook.Eat**	-

Figure 6.4: Cooking plan creation (cons: constraints, act: actions)

Plan execution

We have described the way agents create their plans. Such plans consists of atomic institutional actions (i) move to transition, (ii) move to scene and (iii) say message. In Section 5.2.2, we have introduced a Movie Script which allows to specify which institutional actions are visualized in a virtual world. Moving to transition, entering and exiting a scene are executed in an EI immediately when requested by the agent, and they are visualized only if it is specified in the Movie Script. With messages, the situation is different, when the agent first needs to find a VWO that provides the possibility to execute this message and activate this VWO within the maximum allowed distance. The object is activated only if the related EI message was executed correctly.

To perform agent actions in a virtual world, an agent uses its *actuator* component, which implementation depends on the specific virtual world and supports the following actions:

1. **Motion** - move agent to a specific location.
2. **Interaction** - interact with objects existing in the virtual world.
3. **Communication** - communicate with other VI participants using text, voice and gestures.

6.4 Implementation

In this section, we explain details of the implementation of our general purpose agent model. We use VI Agents in the context of simulations executed using a Virtual Institution. Main components of such simulation are:

- *Timer* controls the time in the simulation. It allows to speed up or slow down the execution process.
- *Virtual Institution* consists of:
  - *Electronic Institution* running in *AMELI* execution environment provides normative control over participants actions. . Moreover, it defines the roles that EI participants play in the system and a common language (i.e. ontology) that participants use during their interactions.
  - *3D Virtual World* provides simulation participants a multimodal (e.g. visual, audible) interaction, immersive 3D environment
  - *VIXEE* causally connects both EI and a 3D Virtual World
- *Simulation Participants* are either *VI Agents* or *humans*, which either only observe the environment or directly participate in a simulation.
- *Virtual World Objects (VWO)* are 3D objects existing a virtual environment that contain semantic description of actions they provide.

A simulation follows a 24-hour day cycle, controlled by the *timer*. Each VI Agent plays some role, where each role has a list of temporal goals. Goals represent institutional actions, i.e., illocutionary message in a specific scene of an Electronic Institution. Plans are formed by atomic steps of moving from current institutional state to the required one. Consequences of institutional actions are visualized in a 3D virtual world.

During the execution of a Virtual Institution agent, institution and virtual world states change. All this information contained in the institution state, and the virtual world state helps agent to create its plan. This information is processed and evaluated by the reasoning component of the agent. A VI Agent reasoning is initiated every specified time interval (e.g. every 3 seconds) or upon an external event (e.g. by talking to an agent). The reasoning time interval should be as small as possible, so that an agent can dynamically update its goals and plans depending on the changing world state. An example is an agent that is going to pick the last apple from the tree, but at the same time another agent picks this apple. With a short enough interval, the agent can break its plan and go to search for apples elsewhere.

The reasoning component functionality was split between two managers, where each manager takes care of separate reasoning functionality. Goal manager selects correct goal, and Plan Manager creates and executes plans.

An agent reasons depending on available data (in BDI terminology this data is considered *beliefs* of the agent). This data is collected from several model components. Memory and Culture component of a VI Agent represent the beliefs of the agent about its *outer space*. This outer space is formed by the 3D world state. On the contrary, we use the *agent state* to represent the *inner state* of the agent. This state is separated into the agent's *institutional state, execution state* and *properties*. The institutional state holds the data related to agents participation in a specific Electronic Institution and the copy of all agent's institutional properties, which help the agent during the reasoning process. The execution state holds the data related to the execution of decisions of the agent, such as a goal and the plan history, a current goal, a current plan or a current location. Formally:

Definition 6.2. We define a *VI Agent's state* as a tuple $S = (IS, ES, P)$ where:

1. IS is a set of institutional states $IS = \{is_1, is_2 \dots i_n\}$ and institutional state is is a tuple $is = (i, r, p)$ where:
   (a) i is the Electronic Institution to which this state is related to.
   (b) r is the role of the agent in institution i.
   (c) p is agent's current position in the performative structure of the institution i, that is, current institutional activity in which the agent is participating.
2. Execution State ES is defined as tuple $ES = (l, GH, g, p)$ where:
   (a) l is the agent's current location
   (b) $GH = \{g_1, g_2..g_n\}$ is a set of goal history items, where a goal history item is defined as a tuple $g_n = \{(g,\ t_{start},\ t_{end},\ r)\}$ where g is the goal, t_{start} and t_{end} are a start and end time of the goal g and r is a result of an execution of the goal g.
   (c) g is the current goal.
   (d) p is the current plan for goal g.
3. P is a set of current values of agent properties.

An agent state along with an agent's memory and a culture provides all the necessary information for the agent to reason about its goals and to create plans to fulfill these goals. The execution state is continuously updated by the *sensor* module, while the institutional state is always updated by VIXEE. Using VIXEE (see Chapter 5), we connect a VI Agent with the EI runtime infrastructure. It translates agent actions to the AMELI format. Moreover, VIXEE uses our *Movie Script mechanism* to monitor AMELI events and using the *actuators* executes all related virtual world updates.

6.4.1 VIXEE Integration

We have integrated VI Agents into the Virtual Institution Execution Environment (VIXEE), presented in Chapter 5. VI Agents in VIXEE represent external agents that communicate with an Electronic Institution and their actions are visualized in a virtual world. Agent definition is facilitated by the set of VIXEE interfaces situated in several tabs. These VIXEE tabs provide following functionalities:

- *Project* tab facilitates the definition of project properties, such as a name and a description of the project. Here we also define which Electronic Institution we want to launch and where is the location of its configuration.
- *Avatars* tab facilitates the specification of virtual world servers and user accounts of avatars that connect to these servers.
- *Roles* tab (see Figure 6.5) facilitates the definition of temporal goals for each role in the system. We can modify time validity, priority of a goal and the goal type.
- *Agents* tab (see Figure 6.6) facilitates the definition of an agent, the specification of its parents and its avatar in the virtual world. Using the genetic mixer (see Chapter 7) we can generate its appearance and visualize it in the

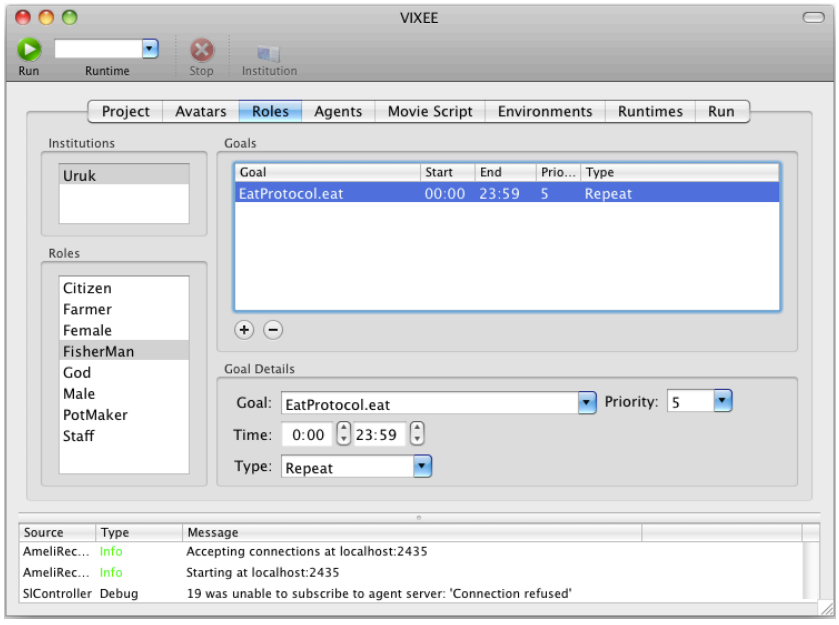


Figure 6.5: Definition of temporal goals in VIXEE

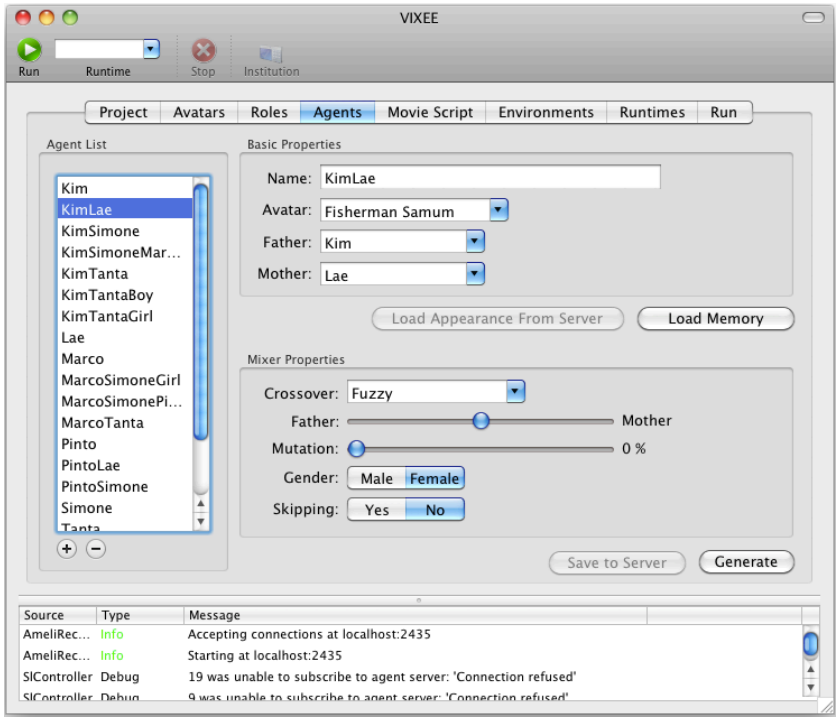


Figure 6.6: VI Agent definition in VIXEE

virtual world. We can also load current memory of the avatar representing the agent and recreate spatial information it sees.

- *Movie Script* tab facilitates the definition of a virtual world Movie Script and an AMELI Movie Script.

- *Environments* tab (see Figure 6.7) facilitates the annotation of a virtual environment using Virtual World Objects (VWO). We specify actions that given VWO provides, the type of the interaction and action parameters.

- *Runtimes* tab facilitates the definition of a runtime configuration, such as: what agents having which roles and representing which culture we launch in a virtual world.

- *Run* tab facilitates the control of agent execution by allowing to start, stop, and pause the execution and also to change current execution time. It provides a 2D map which shows current positions of all agents and paths they decided to take. We can also view current agent state, its goal and plan.

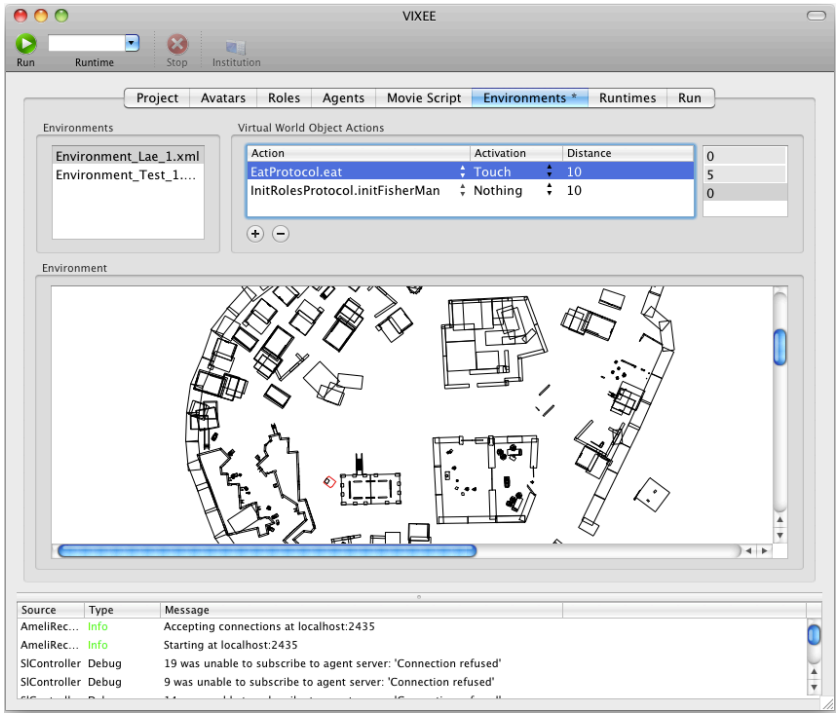


Figure 6.7: Definition of environments in VIXEE

6.5 Summary

In this chapter:

* We have presented the general purpose agent model for Virtual Institution called VI Agent.
* We have introduced all of VI Agent components and stated our contributions.
* We have described the implementation of this model.

In the next chapter, we present details on generation of unique avatars using the genetic algorithm approach.

Chapter 7

3D Avatar Generation

In the previous chapters, we have presented a method for the automatic generation of 3D virtual worlds from a formal specification. Later, we have connected the generated virtual world with the execution infrastructure of Electronic Institutions (AMELI), making the environment normative and interactive. Then, we have introduced a generic model for intelligent virtual agents named VI Agents, which populated generated virtual environment, forming large crowds.

In this chapter we show how to achieve a high degree of appearance variation in simulating large crowds and their 3D avatars through the use of genetic algorithms, while also manifesting unique characteristic features of a given population. We show how large virtual cities can be populated by diverse ethnic crowds of virtual agents, illustrate how our approach can be used to simulate full body avatar appearance, present a case study and analyze our results.

7.1 Motivation

Virtual worlds and 3D games use 3D avatars[1] for user's physical representation in the virtual space as well as for simulating computer-controlled non-player characters. The use of 3D avatars is not limited to video games or virtual worlds, but has a wide application, for example, in the movie industry, training simulators and health systems. In most instances, such avatars are manually designed, but often there are situations when such manual design is not practical. The most common case when avatar design automation is required is when a large crowd of computer-controlled avatars must be simulated to perform a particular group activity.

As an example of automatic crowd generation, one of the most popular solutions in the movie industry, is to utilize *Massive*[2] software that offers facilities for creating a given number of avatar clones and further modification of those clones

[1] 3D avatar is an animated, emotive, complex model representing a user in a graphical form that ranges from actual resemblance of the human user to a talking fish or a robot.

[2] `http://www.massivesoftware.com/` (last access 05/2012)

by introducing a slight variation into their appearance features. This technique was widely used in Peter Jackson's The Lord Of The Rings film trilogy (see Fig. 7.1) for simulating battles.

Figure 7.1: Lord of the Rings Crowd Simulation in Massive

The approach in simulating crowds, taken by systems like Massive, is to use a number of manually created avatar shapes and randomly modify some shape parameters and textures to introduce the variety. To avoid non-plausible distortions of the resulting shapes, the features being changed are often limited to randomly selecting a clothing texture from a predefined list or modifying the height and width of the avatar. Thus, such systems are limited in terms of introducing a variety into crowds.

What is often desired in crowd simulation - is to have a diverse crowd with representatives of various genders and age groups, having different facial and body appearance, while still consistently maintaining the distinct features of their ethnic group. To illustrate this idea, Fig. 7.2 outlines a group of alien tribesman from the Avatar movie[3], where each member of the group is a distinct individual, but they are all perceived as members of a single tribe due to similarities, for example in skin color, ear shape and hair style.

Simulating such diverse groups requires identifying the characteristic features that represent a given population and defining the acceptable range of variation in these features, as well as their intelligent manipulation. Without satisfying these conditions when generating crowds of avatars in an automatic fashion - either the believability of the crowd appearance or its diversity will be very limited.

In order to address this problem, rather than using standard randomization techniques for the crowd simulation [Thalmann, 2007], we suggest to introduce diversity into avatar appearance by mimicking genetic rules of reproduction present in nature. Under normal circumstances, humans and animals, when producing their offsprings, manage to achieve enough variety in the appearance of

[3] `http://www.avatarmovie.com` (last access 05/2012)

Figure 7.2: Example of crowd diversity in the Avatar Movie

their children, while also preserving their distinct personal and ethnic characteristics, as well as ensuring that their body shape and facial features remain within the acceptable range of variation for the given species.

Thus, in this chapter, we introduce an algorithm that generates visually unique avatars following the representations and techniques used in genetic algorithms theory [Holland, 1975] (e.g. crossover and mutation). This algorithm generates individuals that respect the visual, racial, cultural and behavioral features of a defined population as well as genetic inheritance of these features. Using genetic mutation, we add novelty to generated avatars. In addition to the visual aspects of crowd simulation, our approach is applicable to generating unique personalities of virtual agents.

Our work focuses on generating large crowds, where each individual is a unique, distinguishable member of some ethnic group (see Figure 7.3). It should not be treated as an attempt to closely mimic genetic reproduction found in nature. Simulating the underlying processes behind forming body tissues and bone structure based on the DNA code is quite complicated, as those mechanisms are not yet fully understood and are not computationally feasible at present. Instead, our algorithm uses the same basic principles, but deals with a much more simple high-level representation of genetic code and a very straightforward technique for manipulating it to produce the desired changes in the avatar appearance.

In our approach we first isolate and quantify visual features of an avatar (e.g. head size, jaw shape) and then use approaches and techniques from *genetic algorithms* rather than mimicking biological behavior. Genetic algorithms belong to the field of *evolutionary computing* where computational approaches in optimization seek inspiration from evolution and genetics.

Isolation and quantification of visual features of an avatar can be a tedious process, but by doing so we can achieve a great level of detail and variety in

Figure 7.3: Our Approach: Simulating Ethnic Crowds

generated avatars [Lewis and Parent, 2000]. For example, for heads of human avatars some can represent the craniofacial measures quantified by their minimal and maximal values [Vieira et al., 2008]. Limiting these features by an interval allows us to better control the appearance of an avatar, prohibiting the creation of unwanted (e.g. implausible) results. Also, we can combine them to represent more complicated visual aspects such as emotions (e.g. happiness). Another possibility is to define different intervals of quantification for visual features of children and adults.

A big advantage of isolating visual features is its reusability with different 3D models. We can easily imagine how the 3D model should change when manipulating a visual feature in non-human 3D models (e.g. manipulating a visual feature of eye-size in a 3D model of an animal). We are convinced that the visual feature approach is more intuitive than most of previous approaches.

When visual features are isolated and quantified, all current values for an individual are extracted to form the *chromosome*. A specific value of a visual feature is named *gene*. Genetic algorithms provide different operators that combine (i.e. crossover operator) and manipulate (i.e. mutation operator) parent chromosomes to generate their children with a defined level of novelty. Next, we describe our model of avatar generation based on representations and genetic algorithms.

7.2 Avatar Generation

In this section, we propose a general model for automatic generation of 3D avatars. By general, we mean that our model can be deployed in any current 3D game architecture or existing virtual world. Moreover, we do not want to limit this model to human avatars, but to use it with any kind of avatars (e.g. humans, animals, fishes, robots, orcs) with distinguishable *visual features*.

7.2.1 Genetic algorithms

Genetic algorithms (GA) belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as *inheritance*, *mutation*, *selection*, and *crossover*.

In the history of evolutionary computing, four important paradigms served as a base activity of the field: genetic algorithms [Holland, 1975], genetic programming [Koza, 1992], evolutionary strategies [Rechenberg, 1973] and evolutionary programming [Fogel, 1995]. Their differences lie within the terminology behind the algorithms, the reproduction operators and selection methods. In our work, we use the terminology and procedures from genetic algorithms.

Traditional genetic algorithm defines and manipulates individuals at the level of their *chromosomes*, where a chromosome is represented as a *fixed-length bit string*. Each position in the string is assumed to represent a particular feature of an individual, called *gene*. Usually, the string is "evaluated as a collection of structural features of an individual, which have little or no interactions" [Sivanandam and Deepa, 2007]. To produce a new generation of individuals, we combine parent's chromosomes using a *crossover* operator. Combining parent's genes allows gene *inheritance*. To introduce novelty in the population, we apply *mutation* to the current chromosome.

Genetic algorithms have been already used in numerous important fields, such as *search*, *optimization* and *machine learning* [Goldberg, 1989]. In most of these cases GA are used as a search heuristic that mimics the process of natural evolution. In this approach, the GA are treated as an optimization algorithm utilizing a fitness function to *select* a partial solution of the original problem, till reaching some predefined threshold for suboptimal solution.

In our approach, we are not using GA as an optimization algorithm. This is why we do not use the *selection* and focus only on *crossover*, *mutation* and *inheritance*. In the following sections, we provide the formal representation of genetic data and operators. Then we contemplate our design of deep inheritance, also called gene skipping, and explain the generation algorithm.

7.2.2 Formal Representation of Genetic Data

In this section, we formalize genetic concepts used during the avatar generation process. First, we define visual information or definable visual traits of an avatar. For the purpose of generation of unique avatars such information needs to: (i) express the specific appearance trait of an individual and (ii) must be quantifiable. Therefore, this information must hold the quantifiable visual descriptors of an individual. For example, "height" visual descriptor can be quantifiable by a numeric interval, from 0 to 100, with 0 being short to 100 being tall. Moreover, a visual descriptor can quantify the shape, texture or color of an avatar's body part. We define the visual descriptor as the *visual feature*:

Definition 7.1. *A* **visual feature** holds a quantifiable descriptor of a body part of an avatar. It is defined as $vf = \{\text{Id, Name, Value, Interval, Minumum, Maximum}\}$ where:

1. *Id* is a unique integer identifier, $id \in \mathbb{I}$
2. *Name*, is a string that identifies the visual feature, $name \in \mathbb{S}$
3. *Interval* defines the range of visual feature values
4. *Minimum* is a string that describes the visual feature when it reaches the minimal value, or when value is false (e.g. short), $minimum \in \mathbb{S}$
5. *Maximum* is a string that describes the visual feature when it reaches the maximal value, or when value is true (e.g. tall), $maximum \in \mathbb{S}$

Using our approach, each avatar is first described by a list of *visual features*. These features should thoroughly characterize its appearance. When such list is defined, we quantify the values of visual features into *genes*. Genes are real or boolean values of visual features, which allow us to perform genetic operations, such as crossover and mutation.

Definition 7.2. A **gene** $g \in \mathbb{R} \cup \mathbb{B}$ represents a real or boolean value of the *visual feature* $g = vf_{value}$:

(i) To quantify the *presence* of the visual feature, $g \in \mathbb{B}$

(ii) To quantify the *strength* or a *position* of a feature (e.g. height with values close to minimum representing short avatar and values close to maximum representing a tall avatar), $g \in \mathbb{R}$

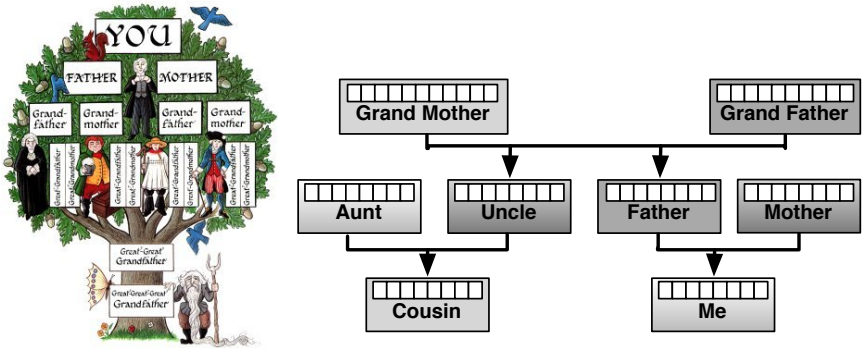
Table 7.1 shows an example of an avatar's visual traits, with defined *id*, *name*, descriptor of *minimum* and *maximum* for interval values, current *value* and a *gene* value.

Id	Name	Interval	Minimum	Maximum	Gene
33	Height	[0,100]	Short	Tall	81
80	Gender	{0,1}	Male	Female	1
93	Head Shape	[0,100]	Round	Pointy	84

Table 7.1: Examples of definitions of visual features

The ordered set of avatar's genes is called *chromosome*. A chromosome holds the complete genetic information of an individual. Order of the genes in a chromosome is specified by *genomic sequence*.

Definition 7.3. A **genomic sequence** defines the genetic composition of a chromosome of a specific group, forming a representative gene ordering of a species or group. A genomic sequence GS for a chromosome of length n is a set of indexes which define the ordering of genes in the chromosome: $GS = \{i_1, i_2 \dots i_n\}$.



(a) Graphical Family Tree (b) Schematic Family Tree

Figure 7.4: Examples of graphical representations of family trees

Definition 7.4. A **chromosome** of length n, c_n, is a set of n genes $g_1, g_2 \dots g_n$ in the order given by its genomic sequence GS, that is $c_n = \{g_{i_1}, g_{i_2} \dots g_{i_n}\}$ where $i_1, i_2 \dots i_n \in GS$.

Furthermore, to operate with the genetic inheritance in population we store all the parent-child relationship information in a *genealogy tree*. As a result, it allows us to navigate within the relationships of the population. This tree provides information on ancestors or siblings of the generated avatar and provides access to their chromosomes. Its name comes from the graphical representation where the family ancestors are visualized in the tree-like structure, also called as *family tree*. A *genealogy tree* is usually represented as a directed acyclic graph. Figure 7.4a shows an example of a genealogy tree. Figure 7.4b shows a schematic view of a family tree.

Definition 7.5. A **genealogy tree** is represented as a directed acyclic graph $GT = \{V, E\}$, where each graph node $v \in V$ holds both the identifier of the avatar and its chromosome. Each graph edge $e \in E$ represents the parent-child relationship.

7.2.3 Formal Representation of Genetic Operators

In our approach, we first define the initial population, that characterizes the base groups of population. Members of these groups have distinguishable visual features that define such group (e.g. asians with asian eyes, africans with african skin). Then, we use this characteristic information to create new individuals. Therefore, we may refer to our generation process as to *reproduction*. During reproduction, we use the information stored in *chromosomes* of parents and combine and modify them using a *genetic operator* to reproduce chromosomes of new unique individuals.

In the previous section, we have formalized genetic concepts related to the reproduction process, that is *genes*, *chromosome*, *genomic sequence* and *genealogy*

tree. In the following sections, we define all the genetic functions and operators. Such operators are responsible for *crossover*, *mutation* and *inheritance* of genetic information from parents. Moreover, we introduce the *genotype rule* a mechanism to explicitly control the generation process, preserve the group characteristic visual features and introduce relationships between genes. Our algorithm first combines parent chromosomes using the *crossover* operator and then uses *gene skipping* to inherit some amount of genes from its deeper ancestors. Next, to introduce novelty, it *mutates* the produced chromosome. Finally, using a set of predefined *genotype rules*, it adjusts this chromosome to respect the population group properties.

Algorithm 3: Naive algorithm for generation of 3D avatars

Input: Genealogy tree, Mother chromosome, Father chromosome
Output: Chromosome representing a new unique 3D avatar
begin
    chromosome ← `Crossover` (fatherChromosome, motherChromosome);
    `Inherit` (chromosome, genealogyTree);
    `Mutate` (chromosome);
    `Adjust` (chromosome, genotypeRules);
end

7.2.4 Crossover

Crossover is a genetic algorithm operator that is used to vary and combine the genetic information stored in chromosomes from one generation to the next one [Sivanandam and Deepa, 2007]. Analogous processes from biology are *genetic reproduction* and *biological crossover*. In this process, a crossover function selects and combines the genes from one or more parents to create a new child chromosome. We formalize different crossover operators. Some operators (i.e. clone, split) are traditional, coming from the genetic algorithm theory, some are defined by us to meet our needs (i.e. exchange, fuzzy). The first traditional crossover operator is *clone* operator, which copies the genetic information directly from one of the parents.

Definition 7.6. Given the mother chromosome c^m consisting of genes $c^m = g_1^m g_2^m \dots g_n^m$ and the father chromosome c^f consisting of genes $c^f = g_1^f g_2^f \dots g_n^f$ a **clone** operator $\oplus^q : C \times C \to C$ is defined as $c^m \oplus^q c^f = g_1^q g_2^q \dots g_n^q$ where q is either denoting mother (m) or father (f) genes.

Figure 7.5 shows an example of father cloning process. In this figure, we see chromosomes of both parents, mother and father, consisting of smaller parts, genes. Each parent is colored with different color. In the bottom part of this figure, we see that all genes in the chromosome were copied from father, they have the same color. We use this color notation in the following examples as well.

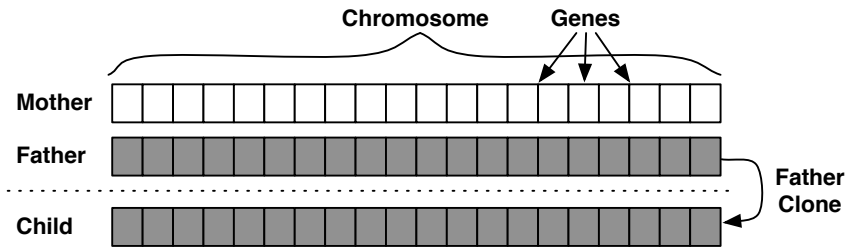


Figure 7.5: Crossover technique called "cloning"

Cloning is the simplest and fastest crossover technique, but it does not provide many functionalities in mixing genetic information from parents. The simplest crossover operator that allows us to combine this genetic information is the *split* operator (in GA terminology also known as one-point crossover [Sivanandam and Deepa, 2007]). Figure 7.6 contemplates the operation of the split operator. This technique splits the chromosome after a specific gene, called *split point*, and all genes on the left of this split point will be copied from one parent, and all genes on right will be copied from the other one. Formally:

Definition 7.7. Given the mother chromosome c^m consisting of genes $c^m = g_1^m g_2^m \dots g_n^m$ and the father chromosome c^f consisting of genes $c^f = g_1^f g_2^f \dots g_n^f$ and the split point $0 \leq s \leq n$, a **split** operator $\ominus : C \times C \to C$ is defined as $c^m \ominus c^f = g_1^m g_2^m \dots g_s^m g_{s+1}^f \dots g_n^f$

In fact, the split point specifies a *father-mother ratio*, r^{fm}, meaning what percentage of genes should be copied from mother and which from father. The split technique is applicable to more parents, where we select one split point for each additional parent. We name such a split the *multiple split* (in GA terminology also known as n-point crossover [Sivanandam and Deepa, 2007]). A multiple split example is displayed in Figure 7.7.

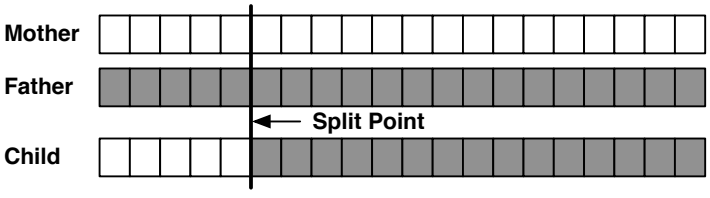


Figure 7.6: Crossover technique called "split"

Previous crossover operators allow none, or very limited mechanisms for combining and experimenting with genetic information from parents. A flexible

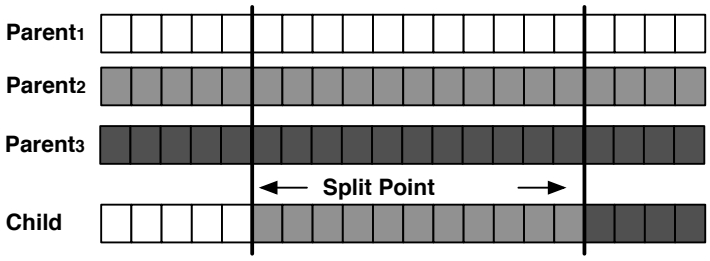


Figure 7.7: Crossover technique called "multiple split"

crossover operator, is the *exchange operator* (in GA terminology it is similar to the uniform crossover [Sivanandam and Deepa, 2007]). This operator combines parents' chromosomes by randomly selecting two complementary subsets of genes from mother and father and joining them. Formally:

Definition 7.8. Given the mother chromosome c^m consisting of genes $c^m = g_1^m g_2^m \dots g_n^m$, the father chromosome c^f consisting of genes $c^f = g_1^f g_2^f \dots g_n^f$ and the exchange selection function $e : G \times G \to G$ that for input father and mother genes outputs only one of them, we define a crossover **exchange** operator $\otimes : C \times C \to C$ as $c^m \otimes c^f = e(g_1^m, g_1^f) \cdot e(g_2^m, g_2^f) \dots e(g_n^m, g_n^f)$.

The process of gene exchange is shown in Figure 7.8, clearly marking genes selected from mother and from father. The number of genes selected from the father and mother is controlled by the *father-mother ratio*.

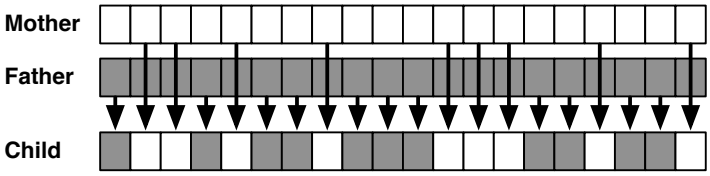


Figure 7.8: Crossover technique called "gene exchange"

For our purposes, we define a new crossover operator named *probabilistic fuzzy operator*. This operator is similar to *exchange* operator, but rather to copy the exact values of parent genes it combines values in the interval given by the values of genes from the father and mother. This operator brings even more variety into the generation process and produces avatars which closely resemble their parents. We use this operator to evaluate our approach in Section 5.5.

Definition 7.9. Given the mother chromosome c^m consisting of genes $c^m = g_1^m g_2^m \dots g_n^m$, the father chromosome c^f consisting of genes $c^f = g_1^f g_2^f \dots g_n^f$,

the parent *gene selection function* $s^i_{r^{fm}} : 2^G \to \{0, 1\}$ which for position i, where $0 \leq i \leq n$, selects either mother or father gene depending on probability given by the father-mother ratio r^{fm} and the *fuzzy function* $f : \mathbb{I} \to \mathbb{R}$ which for gene on position i selects a random value in the interval given by $f(i) = [s(i), (g^m_i - g^f_i)/2]$, we define a **fuzzy** crossover operator $\oslash : C \times C \to C$ as $c^m \oslash c^f = f(1) \cdot f(2) \ldots f(n)$.

We can define even more crossover operators, but for our purposes, the split, exchange and split are enough. An example of custom crossover operator would take the arithmetic average of mother and father gene values.

7.2.5 Inheritance and Gene Skipping

Looking at ourselves in the mirror and analyzing our visual appearance, most probably we find visual traits from our parents. Nevertheless, many of our traits go even deeper into our ancestry tree, and sometimes we look more similar to our grandparents than to parents. Inheriting visual and behavioral features from our predecessors is possible due to a process called *gene skipping*. In some extraordinary cases, it is even possible for black parents to have white (not albino) baby [TheSun, 2010], when the skin color gene from white ancestors becomes active generations later. Although, sometimes our visual features have nothing to do with our ancestors and are the result of *mutation* or altered by some external factor (see next Section 7.2.6).

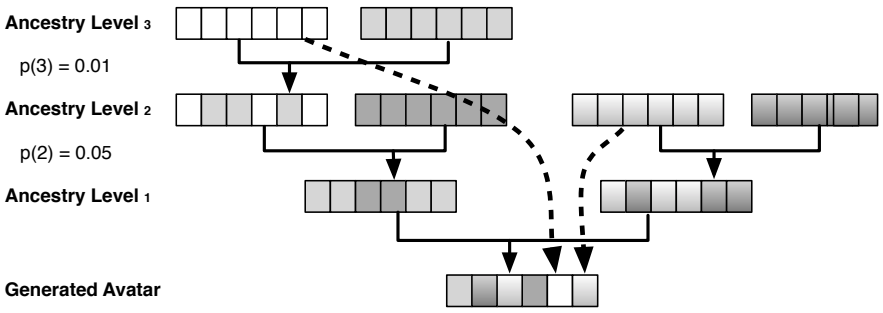


Figure 7.9: Gene skipping, where $p(n)$ denotes the probability of inheriting gene from ancestry level n

Gene skipping is a natural genetic process that occurs during the reproduction process, when we obtain the genetic information from our deeper ancestors. Such genetic information does not only alter our appearance, but also behavior, personality and predispositions. Inheriting skipped gene can, for example, lead to strong, unexpected predispositions to alcoholism, drug addiction or gambling [Comings et al., 1996].

Figure 7.9 contemplates the process of gene skipping. In our approach, this process becomes active after combining chromosomes using a crossover operator. For each gene in the chromosome, we evaluate the possibility of inheriting this gene from our ancestors. The probability value is bound to an *ancestry level*. Ancestry level of our parents is 1. Our grandparents are our second closest ancestors, their level is 2, great-grandparents have level 3 and so on. For a given gene, we define the probability of skipping it from a predecessor in ancestry level n as a quadratic function $p(n) = \frac{1}{5 \times 2^n}$, where $n \geq 2$. We have not found unified and exact values of such probabilities in the literature; therefore, we have modelled it to reflect the quadratic decrease with relation to older ancestors. In this manner, there is a 5% probability of inheriting a given gene from our grandparents, 2.5% from our great-grandparents, 1.25% from great-great-grandparents. The gene skipping technique introduces an interesting level of variability to the appearance of avatars. In the next section, we introduce a new genetic operator called *mutation*, which further extends the variability of generated results.

7.2.6 Mutation

In molecular biology and genetics, mutations are changes in a genomic sequence: the DNA sequence of a cell's genome or the DNA or RNA sequence of a virus. It can be defined as sudden and spontaneous changes in the cell. Mutations are caused by radiation, viruses, and mutagenic chemicals, as well as errors that occur during DNA replication [Bertram, 2000]. Mutation can either be beneficial or harmful to living organisms.

In genetic algorithms, mutation is a genetic operator used to provide genetic diversity from one generation to the next. Mutation alters one or more gene values in a chromosome from its initial state. Using mutation, the solution may change entirely from the previous solution. Hence a genetic algorithm can come to better solution using mutation [Goldberg, 1989].

We use mutation to bring more variability to generated avatars, allowing them to have new visual traits, unknown to their ancestors. In our case, mutation is performed by randomly selecting a specific number of genes and modifying their values. The new value is either random, or taken from a specific interval (e.g. $\pm$ 20% of parent value). The mutation level is expressed by the percentage of the mutated genes from the total number of genes. In the bottom of the Figure 7.13, we see a woman avatar, PintoLae2 that is significantly different from his parents due to the high level of mutation set to generate this avatar.

7.2.7 Genotype Rules

In genetics, the *genotype* is a genetic makeup of a cell, an organism, or an individual. In another words, it is a measurement of how an individual differs within a group of individuals or species. We have borrowed this specific term for a mechanism that allows us to define the characteristics of individuals or population groups, using *genotype rules*.

We propose *genotype rules* to provide precise control over values of individual genes during the reproduction process. It is a powerful mechanism, that allows us to define the characteristics of a population group and to maintain their significant traits (e.g. asian eyes, height of Pygmy tribe is never over 150 cm). Moreover, genotype rules can specify the relationship between genes. Such rule can, for example, force the inheritance of a group of genes only from one parent, depending on the gender of the generated avatar. In another example, which we explain in detail later in this section, we can generate female avatars with ideal breast-waist-hip ratio.

A genotype rule modifies the gene value of the reproduced chromosome depending on the other gene values stored in this reproduced chromosome, as well as gene value stored in any of its ancestors. Genotype rules use the *selection function* that allows them to select the gene value from its own chromosome or from a chromosome of a specific relative. The input of this selection function for gene n is a string, which encodes the full path to the given individual. Formally:

Definition 7.10. A **gene selection function** $s_g : \mathbb{S}^* \to \mathbb{R}^*$ is a recursive function which returns the value of a specific gene g from one or several relatives. Relatives are selected depending on the *input string* s in the form $s = X_1 \cdot X_2 \cdot \ldots \cdot X_m$ where $X_{k|k \leq m} \in \{self, parent, child, sibling\}^*$. Values of X_k represent standard tree node selection functions and their concatenation specifies the path to the specific relatives.

Examples of *input strings* S^*, to the gene selection function follow the subscript notation (i.e. $selectionFunction_{geneName}$) with the gene name $geneName$ of some selected individual appearing under the selection function name $selectionFunction$:

- Avatar: $self$
- Father: $parent_{gender}$ = 'male'
- Grandmother from mother's side: $parent_{gender}$ = 'female' $\cdot$ $parent_{gender}$ = 'female'
- All my grandparents: $parent \cdot parent$ (note the recursive evaluation, where the $\cdot$ operator starts the evaluation for all returns value on the left from the dot)
- All my brothers that are taller than me: $sibling_{gender}$ = 'male' $\wedge$ $sibling_{height} > self_{height}$

Using the selection function the genotype rule is:

Definition 7.11. A **genotype rule** for gene g_i is a function $r_{g_i} : S_1 \times S_2 \times \ldots S_n \to \mathbb{R}$ which for a given input $s_1, s_2 \ldots s_n$, where $s_{k|k \leq n}$ is a gene selection function, returns the value of gene g_i.

Definition 7.12. A **genotype** is a set of genotype rules that characterizes some specific population group or an individual.

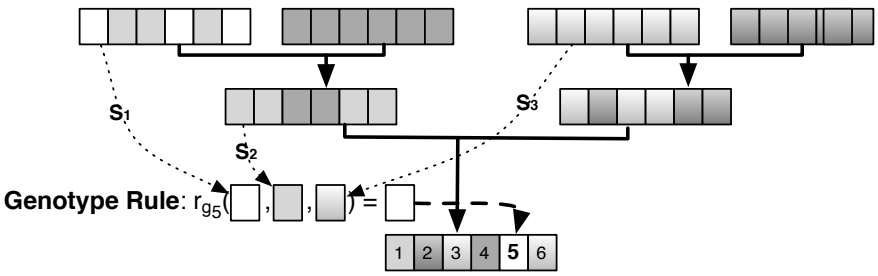


Figure 7.10: Selection process for rule $r_{g_5} : S_1 \times S_2 \times S_3$

Figure 7.10 contemplates the process of selection of genes for the genotype rule evaluation. Next, we show an example of how the genotype is used to characterize a population group and to specify a gene relation. In this example we use $random(x)$ function which returns a random value $i \in \mathbb{I}$ in the interval $0 \leq i \leq x$. If we look for more uniform population, we can model the $random()$ function as a fuzzy function with a gaussian distribution. For clarity, we do not use the number index of a gene r_{g_i} but the name of related visual feature r_{gender} that represents gene g_i.

Example 1: Pygmy Tribe

We want to specify that members of Pygmy tribe have black skin and their height never passes over 150 cm. We characterize this fact using two genotype rules:

1. r_{height} = 150 - $rand(30)$
2. $r_{skinColor}$ = "black", where black color is represented by some numeric value

In our algorithm, genotypes rules are applied after the reproduction process. This is the reason why no matter what the result of the reproduction is (e.g. after mutation), genotype rules adjust the final chromosome to the defined height and skin color.

Example 2: Ideal ratio

In this example, we want to generate female avatars which breast-waist-hip ratio is (90-60-90). We can specify this fact with four rules, where we even

respect the diversity of generated values by using them as measure to the new "ideal" proportions.

1. r_{gender} = "female"
2. $r_{breasts} = \frac{3}{8}(s_{breats}(self) + s_{waist}(self) + s_{hip}(self))$
3. $r_{waist} = \frac{2}{8}(s_{breats}(self) + s_{waist}(self) + s_{hip}(self))$
4. $r_{hip} = \frac{3}{8}(s_{breats}(self) + s_{waist}(self) + s_{hip}(self))$

The first rule defines that all generated avatars will be female. The remaining rules encode that breasts should be $\frac{3}{8}$ (same as $\frac{90}{240}$) of the ratio and hip $\frac{2}{8}$ of the ratio and waist $\frac{3}{8}$ of the ratio. This example shows how easily we can model relations between genes. It is easy to imagine how we would model "correct" tall people, by specifying the relations between their corporal length and the length of their arms and feet.

Example 3: Eye Color

In this scenario, we define the color of the eyes of a new avatar depending on the color of the eyes from his relatives. The proposed rules do not reflect the real-life scenario, as in the real life the eye color is decided by the combination of *alleles* (part of the gene), which we do not model. To model the eye color we specify one rule:

1. $r_{eyeColor}$ =
   - $s_{eyeColor}(parent) \equiv$ "blue" $\rightarrow$ "blue"
   - $\exists(s_{eyeColor}(parent) \equiv$ "blue") $\wedge$ $\exists(s_{eyeColor}(parent) \equiv$ "brown") $\rightarrow$ "brown"
   - $\exists(s_{eyeColor}(parent) \equiv$ "blue") $\wedge$ $\exists(s_{eyeColor}(parent) \equiv$ "brown") $\wedge$ $\exists(s_{eyeColor}(parent \cdot parent) \equiv$ "blue") $\rightarrow$ "green"

These rules use the information stored in the genealogy tree and depending on values from ancestors decide what will be the eye color of the generated avatar. Genotype rules are powerful mechanism that controls or "fine tunes" the generated population.

7.2.8 Algorithm

We have presented all the components and operators of our genetic approach for generation of unique 3D avatars. In this section, we present the generation Algorithm 4. The input of this algorithm is:

1. A *genealogy tree* of the population which contains the relationships between ancestors.

2. Two graph nodes containing the *mother and father chromosomes* of the newly generated avatar.

3. *Crossover operator* (e.g. exchange), that defines how the genetic information from parents is combined.

4. *Mother-father ratio* r^{mf} defines what percentage of genes should be inherited from father and what percentage from mother.

5. *Mutation* level introduces the novelty and uniqueness in generated avatars by specifying what percentage of genes will be mutated.

6. *Gene skipping flag* that either allows or disables deep inheritance.

7. *Genotype rules*, that allow precise control over reproduction of individual genes. Executing a genotype rule returns a boolean value if this rule has modified the chromosome or not.

Algorithm 4 uses all the mechanisms introduced in the previous chapters. First, it combines the chromosomes from *father* and *mother*, using a specific *crossover operator* and creates a new child *chromosome*. Second, if allowed, performs a *gene skipping* using the new chromosome and information stored in the *genealogy tree*. Gene-skipping uses pre-defined probabilities to decide if a given gene should be skipped from some ancestry level. Third, the algorithm *mutates* a specific amount of genes in the new chromosome. This amount is set by the *mutation level*. At last, the chromosome is modified according to *genotype rules*. The algorithm repeatedly executes all rules till no change in chromosome is detected. The generated chromosome contains all the information about visual appearance of a new unique 3D avatar. The values of genes of generated chromosomes are used to render the new avatar.

Figure 7.11 shows the interface of the Genetic Mixer tool[4], that implements all the features of the presented Algorithm 4. We used the Genetic Mixer to generate all the examples in Figure 7.13. Next, we present how the presented algorithm can be applied to Second Life avatars.

7.3 Second Life - Genetic Mixer Application

Second Life and open-source virtual world platform Open Simulator provide one of the most sophisticated systems for avatar creation. In Second Life, the appearance of each avatar can be modified in the appearance editor by changing values of appearance features (e.g. height, head size, eye size). It lets the user to modify a vast amount of visual parameters, attach objects to the different parts of the body (e.g. head, eyes) and define different clothing. Our tool, called Genetic Mixer uses OpenMetaverse[5] library, that allows to change these values programmatically. We can modify 269 visual features of Second Life avatar,

[4]Genetic mixer video: `http://youtu.be/HOw_nyjI3RA` (last access 05/2012)
[5]`http://openmetaverse.org/`

Algorithm 4: Algorithm for generating an avatar

Input: genealogyTree, mother, father, crossover operator, fatherMotherRatio, mutationLevel, skipGenes, genotype rules
Output: Chromosome representing a new unique 3D avatar
begin
  // combine parent chromosomes
  chromosome ← `Crossover` (crossoverType, fatherChromosome, motherChromosome, fatherMotherRatio);
  // gene skipping
  if skipGenes **then**
    chromosome ← `SkipGenes` (genealogyTree, chromosome);
  // mutate chromosome
  chromosome ← `Mutate` (chromosome, mutationLevel);
  // adjust chromosome according to genotype
  // execute the rules till no change is performed
  repeat
    adjusted ← *false*;
    foreach *(rule* ∈ genotype*)* **do**
      adjusted = adjusted ∨ `ExecuteRule` (rule, chromosome);
  until *adjusted* = *false* ;
  return chromosome
end

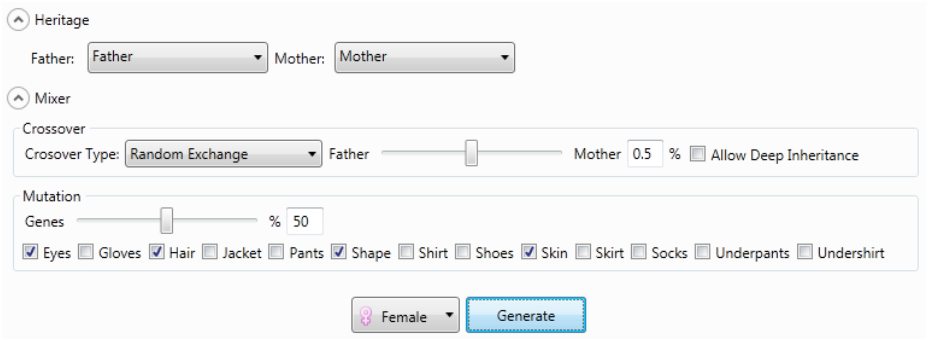


Figure 7.11: Interface of the Genetic Mixer tool

identified by *id*, *type* and *name* and *minimal*, *maximal*, *default value* and their labels. The chromosome of an avatar is a concatenation of gene values, where each gene represents a value of one of these 269 visual features. *Id* is an integer value uniquely identifying this visual feature. We use the id value to set the order of the gene in chromosome, thus the genomic sequence is given by the order of visual features ids. *Type* specifies the part of the avatar this property belongs to (e.g. shape, eyes, skin). *Name* describes the property (e.g. head

Id	Name	Type	Minimum	Maximum
33	Height	shape	Short	Tall
34	Body Thickness	shape	Body Thin	Body Thick
35	Ear Size	shape	Small	Large
36	Shoulders	shape	Narrow	Broad
37	Hip Width	shape	Narrow	Wide
38	Torso Length	shape	Short Torso	Long Torso
80	Gender	shape	Male	Female
93	Glove Length	gloves	Short	Long
98	Eye Lightness	eyes	Darker	Lighter
99	Eye Color	eyes	Natural	Unnatural
105	Breast Size	shape	Small	Large
106	Torso Muscles	shape	Regular	Muscular
108	Rainbow Color	skin	None	Wild
110	Ruddiness	skin	Pale	Ruddy

Table 7.2: Examples of appearance parameters

Id	Name	Gene
33	Height	81
34	Body Thickness	52
35	Ear Size	31

Table 7.3: Gene values and encoding

size). *Minimal* and *maximal* value define the limits for a given appearance trait, and their labels represent the meaning of such limits (e.g. for height minimal value would be "short" and maximal "tall", or for torso muscles the minimal value would be "regular" and maximal would be "muscular"). Table 7.2 shows examples of Second Life appearance features that we specify as genes. These example parameters belong to different avatar parts, such as the body (e.g. shape, eyes, skin) or clothing (e.g. gloves).

A value of an appearance parameter defines the corresponding gene value. Gene is a real value is range [0,100], where boolean values are represented as 0 for false and 100 for true. In the following example, we assume that we have only three visual properties: height, weight and head size with values displayed in Table 7.3.

The corresponding chromosome is the concatenation of the genes:

c^m: 81|52|31

In the following example we present a random split crossover operator with the posterior mutation. We consider that chromosome M is father's chromosome and F for mother.

c^m: 81|52|31
c^f: 73|40|60

We use the *split* operator, and we split after second gene. The new child c^c after crossover is following:

c^c: 81|52|60

We set the level of mutation to 33% and select second gene to mutate:

c^c: 81|52|60 (before)
c^c: 81|63|60 (after)

We have not used the gene skipping, nor defined any genotype rules, thus the generation process is over. The output values for the child chromosome are shown in Table 7.4.

Id	Name	Gene
33	Height	81
34	Body Thickness	63
35	Ear Size	60

Table 7.4: Second Life: Output gene values

Figure 7.12 shows the process of avatar creation. First, parent chromosomes are mixed to produce the child's chromosome. Second, the gene vales from produced chromosome are stored into visual parameters of the avatar. Last, the visual parameters are used by the OpenMetaverse library to create a new avatar.

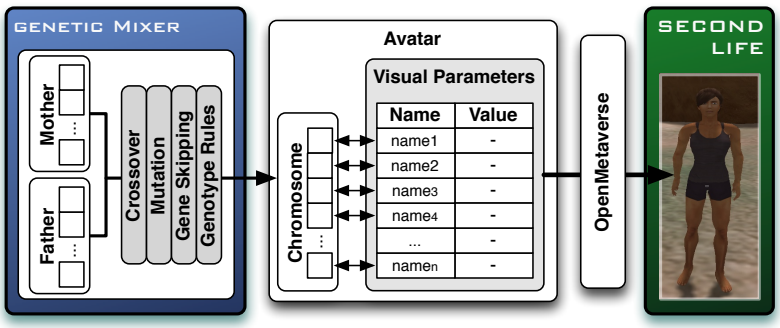


Figure 7.12: Avatar generation process, using the genetic mixer tool

Next, we evaluate our algorithm according to two important aspects, that is the variability of generated avatars and the speed of the algorithm.

7.4 Evaluation

We have evaluated the generation algorithm on the variety of generated avatars depending on different input parameters of the algorithm, that is crossover operator, father-mother ratio and mutation level. We have also checked if distinctive features of individuals are preserved within their children. Results are shown in Figure 7.13.

For our purposes, we have defined six avatars, three females and three males that form the base population of our virtual world. Two avatars are asian (Kim and Lae), two are african (Pinto and Tanta), one caucasian (Simone) and one arab (Marco). We have isolated 200 visual features of each avatar, that are provided by OpenSim and that define shape, skin, eyes and hair. Each of these visual features represents a gene in an avatar's chromosome.

For the generation, we have used two crossover operators, *fuzzy* (marked as F in the avatar captions) and *exchange* (marked as E in the avatar caption). We have used different father-mother ratios and different mutation level. Each generated child is named by combining the father's and mother's name (e.g. child of Kim and Lae is named KimLae).

In the first row, of Figure 7.13 we see all six avatars from our base population. In the second row, we see their children. KimLae was generated using the *fuzzy* with the ratio set to 50%. We can clearly see the resemblance from both father and mother. She has her father's eyes, but her mother's chin. Nose width is somewhere between father and mother. We can clearly see that she is asiatic. The same parameters were set for TantaPinto and we can see that she is black, resembling both parents. If we now focus on KimSimone and her brother KimSimone2, we see that they are both generated using the same parameters with significantly different results. This is due to the random nature of fuzzy operator and the use of *mutation*, which was set to 2%. When we look at the grand-kids of the base population avatars we can see how their visual features are combined.

In the last row we present a selection of avatars that we generated during the generation of 3000 avatars from our base population. PintoLae2 is significantly different from others due to the high level of mutation set to 10%. It's evident from the results that the algorithm generates a large variety of visually acceptable avatars respecting the genetic inheritance of ancestor features. Although we were able to generate 3000 avatars, we are unable to display them all at once as we are limited by the capabilities of the OpenSimulator allowing to host a maximum of 100 avatars, but recent research from Intel on using Distributed Scene Graphs (DSG)[6] allowed simultaneous participation of thousands of users.

We have also measured the average speed of generation of one avatar during the generation of crowd of avatars. The speed of the algorithm linearly depends on the length of the chromosome, that is its time complexity is $O(n)$ with respect to the length of a chromosome. We have generated 3000 different avatars, using

[6] `http://www.hypergridbusiness.com/2011/06/intel-increases-opensim-avatar-capacity-20-fold/` (last access 05/2012)

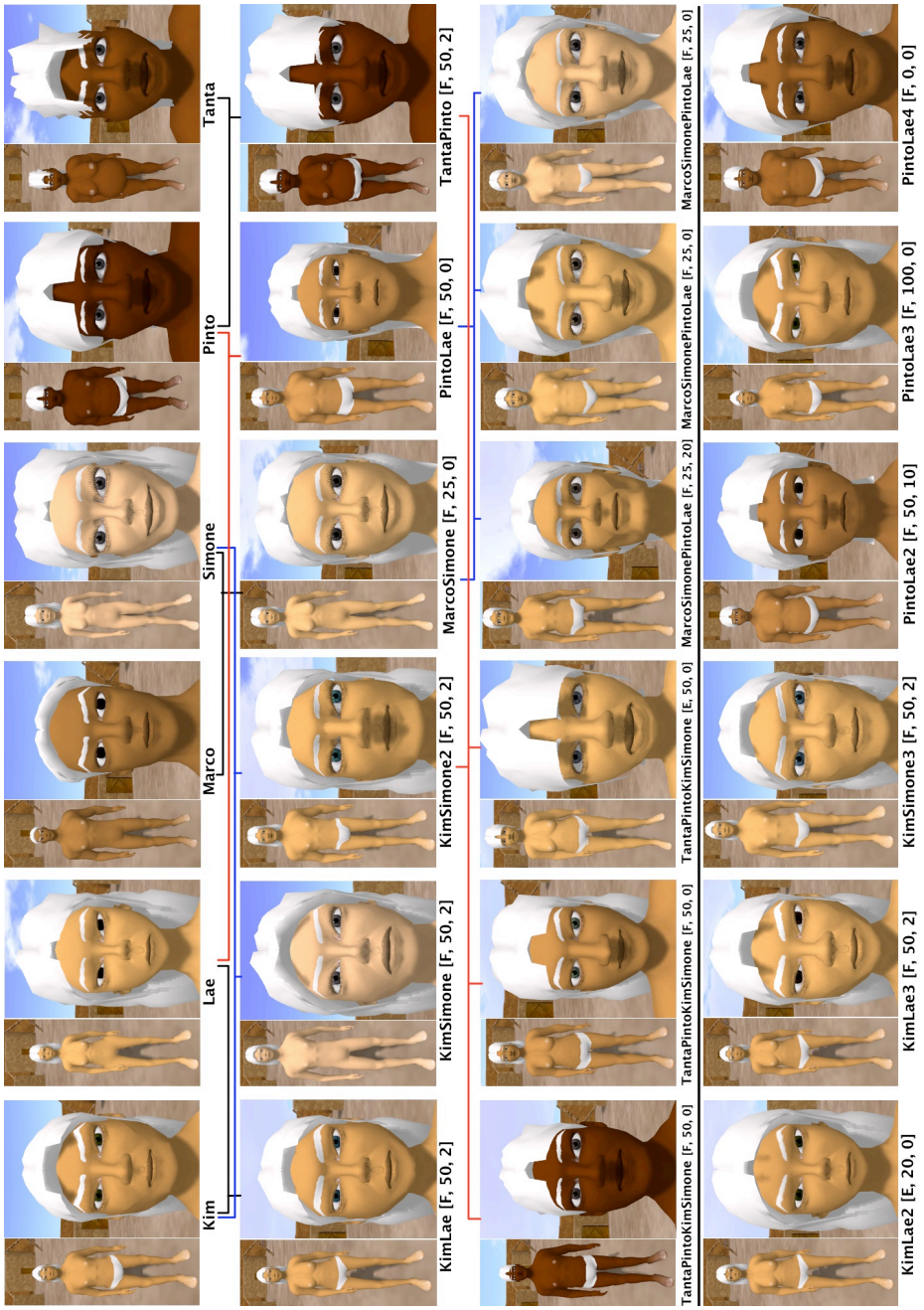


Figure 7.13: Avatars generated using our method. The top row forms the start population, bottom rows are the children. The label of every figure contains following information: *Name [crossover, father-mother ratio, mutation level]*

the most complex *fuzzy* operator with mutation level set to 2% and on average the algorithm generated a new avatar in 305 ms. The generation was performed on a MacBookPro with 2.6 GHz Intel Core 2 Duo processor and 4Gb of RAM.

7.5 Summary

In this chapter:

* We have formalized data and operators used by our genetic-based approach to generate 3D avatars.
* We have presented an algorithm for automatic generation of 3D avatars use techniques and formalizations from genetic algorithms.
* We have presented examples of generated avatars using our approach
* We have evaluated the performance of our approach when generating large number of avatars according to the variance of produced avatars and speed of algorithm.

In the next chapter we present a complex example in which VI Agents populate the city of Uruk 3000 BC, the first city on earth. This simulation was designed as part of Virtual Heritage Project [Bogdanovych et al., 2011]. Our work uses the knowledge collected during the development of this project and evaluates our model for agents that participate in this cultural simulation.

Chapter 8

Case Study: Uruk 3000 BC

In this chapter, we provide a case study in which we have applied mechanisms and techniques presented in the thesis to the domain of interactive e-learning. This case study concerns the execution of cultural and social simulation of life in the first city on earth, Uruk 3000 BC. This simulation was previously introduced by [Bogdanovych et al., 2010b]. It included an interactive 3D environment executed in Second Life, with exact archeological recreations of the main city buildings, such as Ziggurat or the Uruk temple. Human participants observe and learn historical facts about life in Mesopotamia by interacting with Uruk environment and its inhabitants.

We enrich this system by connecting the Uruk environment to an Electronic Institution using VIXEE and enforcing the control of participant interactions. Each computer-controlled participant is executed by a VI Agent. VI Agents perform their tasks depending on the specified set of temporal goals. Their appearance is automatically generated using the Genetic Mixer tool. Moreover, using Virtual World Grammar, we dynamically manipulate the virtual world content and insert or delete objects in the virtual world at run-time.

8.1 Introduction

Inhabitants of Uruk belong to the Sumerian culture, wear traditional clothes and use culture-specific objects. For the purposes of this simulation, we isolated several sub-cultures where each of them represents a specific role in the system. These roles are Gods, Fisherman, Potmaker and Farmer.

Gods are divine entities, and in this simulation, they do not need to sleep; thus, they work 24 hours a day. We employ two gods, Enki, god of freshwater, male fertility, and knowledge and Ninhursag, goddess of earth. God Enki is responsible for the creation of a population, and he is visualized by the Uruk temple. Ninhursag is responsible for growing crops, and she is visualized close to the fields. God Enki is worshiped by Fishermen and Potmakers and goddess Ninhursag by Farmers.

Inhabitants of Uruk, formed by Fishermen, Potmakers and Farmers, follow the same daily routine: (1) in the morning, they wake up, pray, eat and then go to work; (2) before lunch, they pray, then eat; after lunch, they go back to work; (3) after work they rest, pray, have dinner and go to sleep. Although this schedule is common for all subcultures, working and praying is specific for some of these them. Fisherman work by catching fish, Potmakers make pots and Farmers collect fruit from the fruit trees. Fishing requires a spear made of reed, while to make a pot, Potmakers need water and clay. Fruit trees grow only when a goddess Ningursag allows it; therefore, Farmers pray to her directly in their fields. Fishermen and Potmakers worship Enki by the temple.

In this simulation, we generate several members of each culture, representing the population of Uruk. Every individual has a unique physical appearance, automatically plans and executes its actions according to the list of goals, navigates to locations of its desire, interacts with the environment and manipulates the objects.

Thus, our goal is to create a large virtual city with hundreds of avatars controlled by artificial intelligence, who understand their environment, interact with it and also interact with other agents and humans. The problem is how to accomplish something like this. Do we design every agent manually? Do we specify every agent's behavior from scratch? How do we handle complex interactions of hundreds of agents that include team work, rituals, trade, wars, etc.? How do we ensure culturally authentic behavior? How to make the environment dynamically evolve and the agents being able to sense the changes and adjust their plans accordingly?

Classical approaches offer two popular solutions to the previous questions. The first solution is to manually design and program every individual agent. While this will eventually achieve the desired effect, the practicality of this method is low as this is an extremely time-consuming and error-prone process. Another popular approach is to employ crowd simulation techniques. However, classical crowd simulation techniques are (i) not suitable for generating ethnic crowds with a large degree of variation in agent appearance and behavior; (ii) they do not normally feature agents playing different social roles; (iii) they do not handle complex agent interactions with objects in a virtual environment; (iv) they have to rely on manual programming to integrate culturally specific behavior; (v) they do not normally allow human controlled avatars to actively participate in the agent crowd; (vi) they do not help agents to achieve common goals in a collaborative fashion.

Thus, in our approach, we advance the state of the art and define a sophisticated general-purpose model of an individual agent capable of generating its own goals based on physiological needs, featuring dynamic planning for pursuing those goals based on the current state of the agent, its culture, the state of the environment and states of other agents. Then, we offer the possibility of generating a large ethnic crowd of agents, where each agent is designed based on a small sample of manually created avatars, which define the ethnicity of the Ururk population. The generated crowd is integrated into a Virtual Institution

that ensures complex, culturally specific and normative behavior of each agent. Moreover, Virtual Institution supports agents with different social roles and allows them to engage into complex interactions with other agents and humans, while providing a high-level method of interpreting their own behavior and the behavior of other actors. Furthermore, using our model of interactive objects, agents have the possibility to react to dynamic changes of the environment.

In the following section, we describe the workflow used to define and execute Virtual Institution for the cultural and social simulation of Uruk.

8.2 Workflow

The definition and execution of a Virtual Institution, using techniques and mechanisms presented in this thesis, requires the following steps:

1. Definition of Electronic Institution specification. In this step, we define the dialogic framework with ontology and roles, performative structure with activities/scenes and transitions and all the scene protocols with a definition of interactions[1]. In this definition, we also include information on Virtual Cultures, which specify culture-specific actions, objects, locations and rituals.

2. Definition of a virtual environment. We can use Virtual World Grammar to either generate the whole virtual environment or only modify its parts (as in our case).

3. Defintion of interactive Virtual World Objects and annotating environment with their instances.

4. Definition of goals for all roles that participate in the system.

5. Definition and implementation of custom actions, which react to specific virtual world actions or AMELI events. These actions, for example, modify parameters of messages sent to AMELI or modify a virtual world model. We use the Movie Script mechanism to map such virtual world actions/ AMELI events to these Movie Script actions.

6. Definition of an initial population, which is used to generate the appearance of avatars entering the application. The initial population models the ethnicity of the population.

7. Definition of runtime configuration, which assigns what agent having what role, using which avatar participates in the simulation.

[1] Currently, Electronic Institutions do not support dynamic update of a specification at institution run-time. This feature is in active research and will provide us the possibility to dynamically update agent knowledge and interactions.

Further we show how this workflow was followed in development of the Uruk case study. In the first step, we define the specification of the underlying Electronic Institution. This specification defines roles and their properties, a common ontology, activities/scenes and scene protocols that structure agent interactions. It defines the whole organizational structure, and interaction protocols of the simulation. Information stored in this specification serves as core for agent reasoning process when creating plans for its goals. However, agents can represent different cultures, where each culture performs specific actions at specific locations with a culture-specific object, what also affects the reasoning process. While traditional approaches required rigid programming of culture information into agent behavior, using a Virtual Culture concept [Bogdanovych et al., 2010a], we can declaratively specify culture-specific information into EI and define high level goals for participant roles, letting Virtual Culture information decide a specific action that shall be taken. An example is the goal "work", where for the Fisherman culture, this means to go fishing, while for PotMakers' culture work corresponds to making clay pots.

Next, we design the virtual environment. We can either employ Virtual World Grammar (VWG) and generate the whole environment from scratch, or, like in the Uruk case, create the initial design manually and let VWG manipulate only specific parts of a virtual world model. VWG allows us to generate the content at various locations, using specified shape grammar rules. We use VWG to place two different types of fruit trees on the field (see Figure 8.1), where fruits are harvested by farmers.

After the virtual world has been created, using the Virtual World Objects (VWO), we define which objects of the environment are interactive, and how they can be included in agent planning. Plans represent a sequence of institutional actions (i.e. movement between scenes and transitions, and illocutions). In the virtual world, these actions are performed only by interacting with a VWO, or by communicating with other participants. In traditional approaches, locations of objects along with a specification of actions, is specified directly in the plan (e.g. decision tree). This produces systems with static knowledge, where agents are able to operate only fixed set of objects at predefined locations. Solution using VWO is more dynamic and relies on keeping the knowledge about an object in the object itself. Such object provides information on actions that can be performed with it, how these actions are activated and who is permitted to do it (i.e. owner of the object, group of owners or anybody). Thus, an agent first has to find a VWO that allows it to execute this action. Then, the action is validated with an EI, and in case of positive validation, the object is activated using a specified virtual world action. Agent considers locations of VWO in its planning, and if it does not know a location of a desired VWO, it can try to search for such an object, or try a different approach or a different goal. This creates a tight integration between an agent and an environment.

In the next step, we define goals for each role participating in the system (goals are inherited from parents as well). Virtual Culture concept allows to specify high-level goals, which are transformed into an action. In our case, this

is an illocutionary action from a specific scene protocol of a performative structure. Moreover, to simulate the daily routine and let agents follow a predefined schedule, we limit the goal validity by a time interval.

Next, we define virtual world Movie Script and AMELI Movie Script and implement related Movie Script actions. This mechanism allows us to react to changes in both layers, virtual world and AMELI, and causally update the other layer. Such changes are provoked by agent participation in the system.

What remains is to populate the virtual world with agents, where the population crowd has to be believable; thus, each agent has a unique appearance and behavior. Traditional approaches include manual design of every avatar, distribution of random attachments or variation of predefined appearances. In our approach, we employ genetic algorithm approach, where using reproduction we generate child carrying features of parents. In the current implementation, we deal only with an appearance as no psychological model has been implemented. Compared to previously mentioned approaches, this approach allows to maintain parent ethnic properties. The requirement for this approach is a definition of an initial population using which, along with already generated agents, we generate the virtual world population.

In the last step, we define the execution configuration. This configuration defines: which agents, having which roles are launched within the simulation.

8.3 Definition of Virtual Institution Components

In this section, we provide a thorough description of all parts of the Virtual Institution, implemented in our example scenario[2] of the city of Uruk.

8.3.1 Virtual World

The presented scenario follows the work done by [Bogdanovych et al., 2010b], and reuses its virtual world environment of the City of Uruk portrayed in Figure 8.1. As mentioned earlier, this environment includes the archeological reconstruction of the main city buildings such as the temple and several common houses with typical roof openings for the roof access where Uruk inhabitants reposed after work.

In this figure, we point to locations where different roles perform their activities, that is the temple, fishing spot for Fisherman and field, where farmers harvest fruit from fruit trees. Each agent participating in the simulation owns a house where it sleeps after work.

[2]We apologise that due to tight schedule and unavailability of history consultants at the time of designing this case study - the presented scenario may not be completely historically or theologically correct. So this study should not be treated for historical references, but instead should only be used as an illustration of the techniques developed in this dissertation.

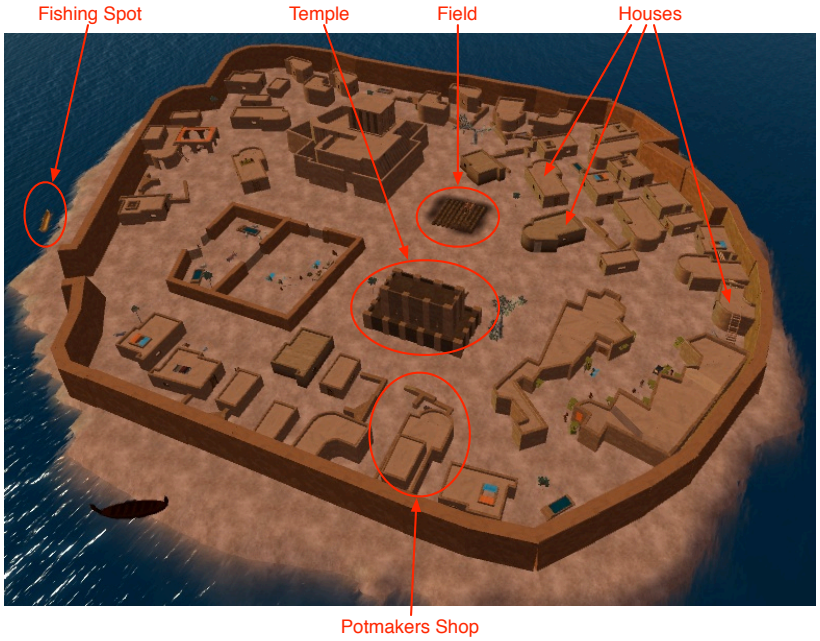


Figure 8.1: Uruk city overview

8.3.2 Electronic Institution

An Electronic Institution (EI) specifies roles and the common language for its participants. It also structures interactions between participants and enforces institutional norms and constraints during such interactions. An EI is defined by a *dialogic framework*, *performative structure*, *scenes* and *norms* (see Section 2.3). Agents create their plans depending on the current goal and the information stored in the EI definition. In this section, we present all parts of the Uruk Electronic Institution.

Dialogic Framework

Dialogic framework defines roles, their hierarchy and common language (i.e. ontology) used during interactions between these roles. Figure 8.2 shows a role hierarchy for Uruk, separating roles of men and women. Please note that citizens of Uruk can never be gods (due to the ssd[3] relationship) and vice-versa.

Each role contains a list of properties hierarchically inherited from its parent. Role Citizen has the following properties: fishes, fruits, pots, cookedFishes. These properties specify the quantity of different agent belongings. Role Fisherman has following properties: hasBoat, hasSpear, hasReed. These properties

[3]Static Separation of Duty

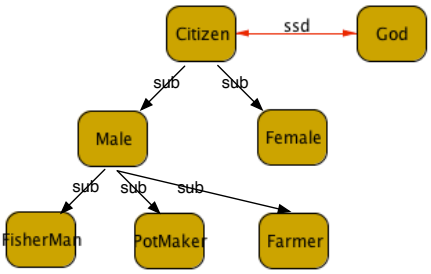


Figure 8.2: Role structure in Uruk

define the existence of belongings. These properties are modified during participation in institution.

Considering the Uruk ontology, we define propositional contents of illocutions uttered during conversations. The propositional content of an illocution is a function with n parameters. Example functions are (i) eatFruit(), uttered before an agent eats a fruit and removes one piece of fruit from its belongings (ii) getClay() uttered before an agent obtains clay from a clay pit, (iii) growFruitTree(), a divine action which triggers the addition of a fruit to the virtual world, (iv) initCitizen(fishes, fruits, pots) uttered to initialize a Citizen's properties and (v) harvestFruit() uttered before an agent starts harvesting fruit.

Performative Structure

A performative structure establishes networks of scenes, which define how agents can legally move among different scenes (that is from activity to activity) depending on their role. Figure 8.3 contemplates a performative structure of Uruk with eight scenes, one initial (Birth) and one final (Death) scene. Agents, enter the institution through the *Birth* scene. Gods continue to the *GodActions* scene, where they can perform their divine actions. Citizens continue to *InitRoles* scene where their parameters are initialized (e.g. number of fish and fruit). Then they join the *Idle* scene, which serves as a center point of agent decisions of its actions. Through the *Idle* scene, agent can enter all the other scenes *Pray*, *Eat*, *MakePot*, *Farm* and *Fish*, each of them representing a specific action. Scenes *MakePot*, *Farm* and *Fish* are only accessible to specific roles.

Figure 8.4 shows a scene protocol of the scene/action Eat. This protocol defines the eating action, in which agents can either eat fruits if they possess it, or eat a cooked fish, which has to be prepared on the fireplace with the fire lit. All this information is stored in constraints and actions of this scene protocol. Please recall that states are annotated with squares having "+" and "−", where"+" means that agent with a specific role can enter this state and "−" means that it can exit while the scene is in this state.

Figure 8.5 shows a scene protocol of the scene Fish. This protocol defines

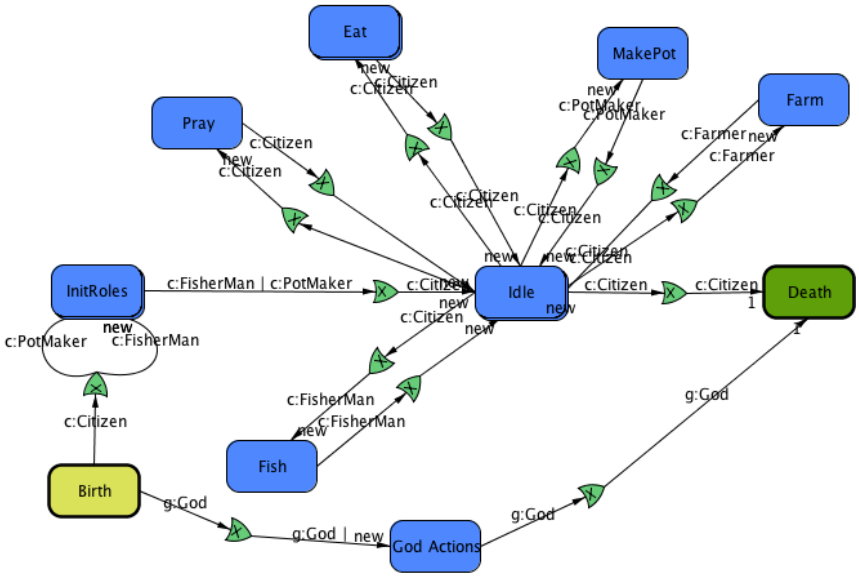


Figure 8.3: Uruk - Performative Structure

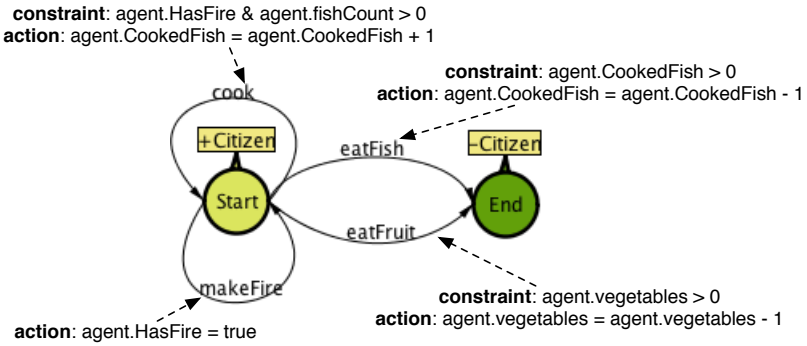


Figure 8.4: Uruk - Eat Scene

the fishing action, in which agents can fish only if they have a spear. If they do not have it, they need to find reed and create a spear out of it.

Figure 8.6 shows a scene protocol of the scene *MakePot*. This protocol defines the action of making pot, in which agents can make a pot only if they have a water and clay. If they do not have it, they need to find water and clay before they start.

Remaining scenes have a very simple protocol, where agents can perform

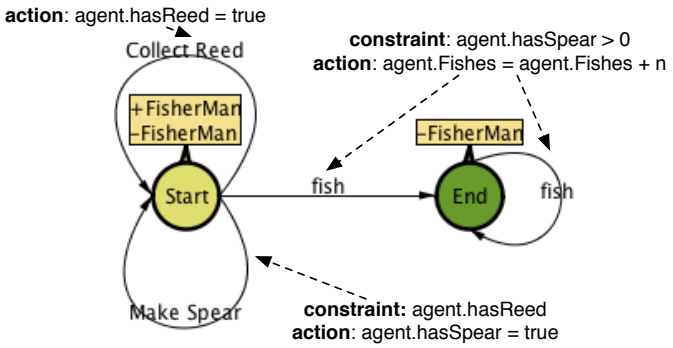


Figure 8.5: Uruk - Fish Scene

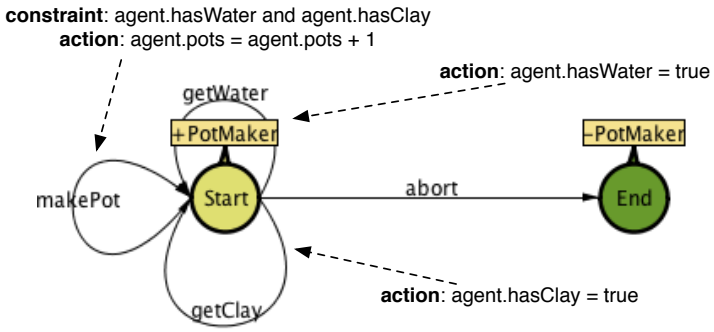


Figure 8.6: Uruk - MakePot Scene

only one specific action, e.g., sleep, after which they exit the scene.

8.3.3 Virtual World Objects

After we have defined all EI parts, we annotate the environment with Virtual World Objects (VWO). VWO annotate interactive objects with information that allows to perform specific institutional actions, along with parameters of such actions. As mentioned in Chapter 6, VI Agents can only perform illocutionary actions by interacting with a VWO.

Table 8.1 describes the temple VWO. It is a religious place, where agents come to pray. We use the *SceneName.action* notation for actions description. Please note that table columns O, G, U (see Table 1) describe if a given action is permitted for **O**wner, **G**roup of owners or any **U**ser. Currently the group ownership is not supported by VIXEE, yet it is part of VWO definition. Temple VWO cannot be owned by anyone. Praying action is triggered by clicking on the temple. In the temple, we also initialize all roles. Citizens are initialized with random amount of pots, fishes and fruits and Potmakers with no water or clay.

Name:	Temple					
Description:	Religious place for Uruk citizens.					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
Pray.pray	-			x	Click	20m
InitRoles.initCitizen	(rnd, rnd, rnd)			x	None	100m
InitRoles.initFarmer	()			x	None	100m
InitRoles.initPotmaker	(false, false)			x	None	100m

Table 8.1: Temple VWO

Table 8.2 contemplates the Fireplace VWO, which is a place where food can be prepared but only when the fire is lit. Fireplace can be used by anyone and it is activated by clicking on it, from within proximity of one meter. When cooking on the fireplace, agents can decide to cook more than one piece of fish, specified by the "amount" parameter.

Name:	Fireplace					
Description:	Place where to cook food					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
Cook.makeFire	-			x	Click	1m
Cook.cook	amount			x	Click	1m

Table 8.2: Fireplace VWO

Table 8.3 contemplates the Table VWO, where its owner can eat a cooked fish or fruit. It is activated by sitting on it.

Name:	Table					
Description:	Place to eat					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
Eat.eat	-	x			Sit	1m

Table 8.3: Table VWO

Table 8.4 contemplates the PotteryRing VWO, where its owner can create a clay pot. It is activated by sitting on it.

Name:	PotteryRing					
Description:	Pottery ring tool to make pots					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
MakePot.makePot	-	x			Sit	2m

Table 8.4: PotteryRing VWO

Table 8.5 contemplates the Well VWO, where anyone can obtain water. Potmakers use well to get water for pots. It is activated by clicking on it.

Name:	Well					
Description:	Place that provides water					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
MakePot.getWater	-			x	Click	2m

Table 8.5: Well VWO

Table 8.6 contemplates the ClayPit VWO, where anyone can obtain clay. Potmakers obtain clay for their pots. It is activated by clicking on it.

Name:	ClayPit					
Description:	Clay pit from which potters extract clay					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
MakePot.getClay	-			x	Click	2m

Table 8.6: ClayPit VWO

Table 8.7 contemplates the Field VWO, where gods plant fruit trees. Farmers also come here to pray. Gods add a fruit tree by clicking on the field.

Name:	Field					
Description:	Place where farmers harvest crops					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
GodActions.growFruitTree	-			x	None	100m
Pray.pray	-			x	Sit	100m

Table 8.7: Field VWO

In the Uruk simulation, Farmers only collect fruit from the trees after the divine intervention from their god. This is also the reason why farmers worship a different god than other Uruk citizens. Table 8.8 contemplates the FruitTree VWO, collected by Farmers on Fields. It is activated by clicking on it, and it adds one piece of fruit to the farmer's belongings.

Name:	FruitTree					
Description:	Fruit tree that provides fruit to Uruk citizens					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
Farm.harvest	-			x	Click	2m

Table 8.8: FruitTree VWO

Table 8.9 contemplates the Boat VWO, is used to go fishing. It is activated by sitting in the boat, what triggers a fishing animation.

Name:	Boat					
Description:	Boat is used for fishing					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
Fish.fish	(caught)	x			Sit	2m

Table 8.9: Boat VWO

Table 8.10 contemplates the Bed VWO, where their owners sleep. It is activated by sitting in the bed, what triggers sleeping animation.

Name:	Bed					
Description:	Place to sleep					
Action	**Parameters**	**O**	**G**	**U**	**Activation**	**Distance**
Idle.sleep	-	x			Sit	1m

Table 8.10: Bed VWO

8.3.4 Goals

Agents' goals are set in VIXEE. Temporal goals define what activity agent should be doing at a particular time. Goals define an activity which is mapped to an illocutionary action by the agent's culture. Goals are defined for every role, and they are inherited from a parent's role. For our simulation, we define two different sets of goals. One, for the role *Citizen* (see Table 8.11) and one for the role of *God* (see Table 8.12). The highest priority goal has priority 1, the lowest has priority 10. The goal type defines what should an agent do after the goal has been achieved. Either it will repeat the execution of actions conducting to the goal (type *repeat*), or it will execute another goal, if defined (type *single*).

Time	**Activity**	**Priority**	**Type**
6:00 - 6:15	Pray	2	Repeat
6:15 - 7:00	Eat	2	Single
7:00 - 12:30	Work	2	Repeat
12:30 - 12:45	Pray	2	Repeat
12:45 - 13:30	Eat	2	Single
13:30 - 19:30	Work	2	Repeat
19:30 - 19:45	Pray	2	Repeat
19:45 - 20:30	Eat	2	Single
20:30 - 6:00	Sleep	2	Repeat
00:00 - 24:00	Work	3	Repeat

Table 8.11: Goals for the role of Citizen

As we do not know the exact time schedule of Uruk citizens, we have assumed

some possible times. Please note the goal Work, with priority 3 in the Table 8.11. This goal defines that if agents have any possible free time, they should start to work. Gods in the Uruk simulation do not need to sleep or eat; thus, they work the entire time, where their "work" corresponds to looking after the correct flow of things and ensuring the stability of the given eco-system.

Time	**Activity**	**Priority**	**Type**
00:00 - 24:00	Work	2	Repeat

Table 8.12: Goals for the role of God

8.3.5 Culture

Virtual culture for VI Agents consists of culture-specific *gestures*, *actions*, *objects*, *locations* and *rituals*. Each role can have defined its own culture, and it inherits cultural information from its parent role. Culture defines exact actions and their locations that will be performed depending on the current goal. In the Uruk simulation, we define five different cultures presented in following tables. Please note that we use only the parts of virtual culture related to the simulation. In following tables rituals represent activities/scenes from the performative structure of Uruk. Table 8.13 contemplates the culture of gods, where the only action is GodActions and their work is to grow fruit trees.

Actions	Work = GodActions.growFruitTree
Rituals	GodActions

Table 8.13: Culture: God

Table 8.14 contemplates the general culture of Uruk citizens, which perform several rituals including eating and praying and use Fireplace, Table and Bed objects.

Actions	Eat = Eat.eat, Sleep = Sleep.sleep, Pray = Pray.pray
Objects	Fireplace, Table, Bed
Rituals	Eat, Idle, Pray

Table 8.14: Culture: Citizen

Table 8.15 contemplates the subculture of Fisherman, whose work is to fish on their boats, and to pray in the temple.

Actions	Work = Fish.fish
Objects	Boat, Pray = temple
Rituals	Fish

Table 8.15: Culture: Fisherman

Table 8.16 contemplates the subculture of Potmakers, whose work is to make clay pots on PotteryRings, using clay from ClayPits, water from the Well and to pray in the temple.

Actions	Work = MakePot.makePot
Objects	PotteryRing, Well, ClayPit, Pray = Temple
Rituals	MakePot

Table 8.16: Culture: Potmaker

Table 8.17 contemplates the subculture of Farmers, whose work is to work on the fields by harvesting fruits. Farmers also pray in the fields.

Actions	Work = Farm.harvest
Objects	Field, Pray = Field
Rituals	Farm

Table 8.17: Culture: Farmer

8.3.6 Virtual World Grammar

Virtual World Grammar is our mechanism for automatic generation of the virtual world content. In the case of Uruk, we are not generating the model of the whole city, but only some of its parts. In this particular case, we are generating fruit trees on the virtual field, which provide fruits to be harvested by farmers. Next, we define all part of the Virtual World Grammar.

Ontology

VWG ontology is the formal definition of the relevant concepts of the domain. In this case, we are generating fruit trees that grow on Uruk fields. Therefore, we define two ontology concepts: *fruitTree* and *field*. *Field* serves as a shape that defines the boundary in which fruit trees grow. *FruitTree* is generated as a 2D object on the virtual world floor plan. This object is then transformed using the 3D transformation engine to the virtual world model representing a specific fruit tree. Thus, ontology item *fruitTree* holds one property defining the 3D model that this item represents.

Instances

For the purposes of Uruk simulation, we define two instances of ontology type *FruitTree*. First instance shows a *pomegranate* fruit tree, thus showing a 3D model of a *pomegranate* tree. The second instance represents a *fig* fruit tree.

Shape Grammar

Shape grammar defines shapes and rules that are used to generate a 2D floor plan of a virtual world design. This floor plan is later transformed to 3D using a transformation engine. In Uruk case, we are not generating a full virtual world model, rather dynamically manipulating its content during runtime. Our task is to add fruit trees to the virtual field, where farmers can harvest them. This field is positioned in the virtual space of the Uruk city. Figure 8.7 contemplates the shape grammar that we use to add fruit trees to the field. Position of the field in the Uruk floor plan is shown in the top part of this figure. In the bottom part, we see 14 different rules. The two left rules place two different shapes to the rectangle shape representing a field. The rest of the rules positions a new shape depending on the position of a previous one.

Mappings

Mappings decide which ontology types are substituted by which shape grammar shape. We map *pomegranate* instances to rectangles and *fig* instances to squares. Thus, when we require a new *pomegranate* to be generated, we have to search for rules that have a rectangle on the right side of the shape and execute it.

Validations

Validations take care of the correct execution of a shape grammar. While adding fruit trees to the field, we do not want them to intersect, producing unappealing results. Therefore, we define only one validation rule $intersect(s, all)$, which checks if shape s added to the design is intersecting with previously added all shapes. Another test we need to perform is that all the fruit trees need to grow inside the field. Thus, we define one more validation rule $checkBoundary(s, b)$, which checks if the shape added to the design s is inside the boundary b.

Heuristics

Heuristics are responsible for finding a correct shape grammar rule to apply depending on the current input. In this example, heuristics use an input parameter which tells the grammar what ontology type has to be used. Then, heuristics find the appropriate rule which has the shape mapped to this ontology type on its right side, finds this rule in the shape grammar execution tree and executes it.

3D Transformation Mechanism

3D Transformation Mechanism uses information stored in generated 2D grammar along with information from ontology instances to generate a 3D virtual world from this floor plan. In our case, it takes the position of a generated fruit tree, and it creates a specific model, pomegranate or fig, on this position. We are using OpenMetaverse library to manipulate the 3D content of Open Simulator.

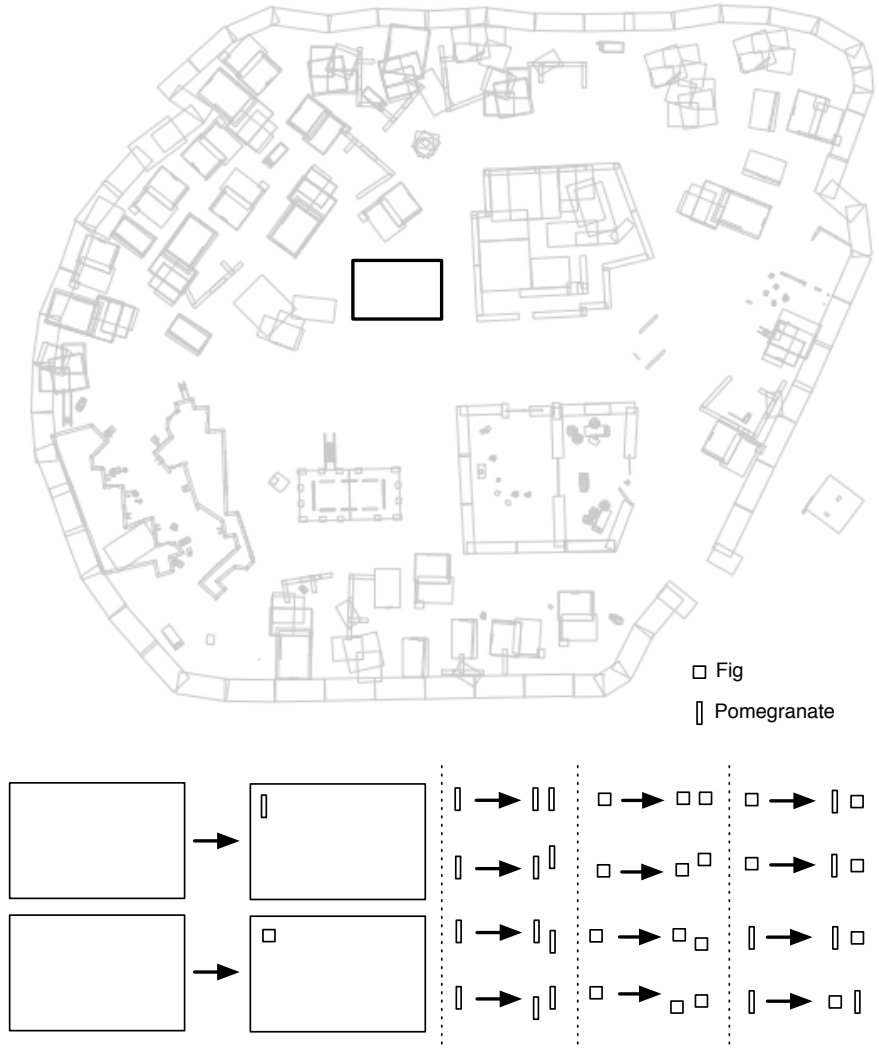


Figure 8.7: Uruk shape grammar

8.3.7 Movie Script

Movie Script is our proposed mechanism, which uses Movie Script actions to map virtual world actions to AMELI messages and also AMELI events to virtual world updates. Recall that AMELI is the execution infrastructure of Electronic Institutions. Using this mechanism, we create the mapping between the context of the message/event and a specific Movie Script action, which is an executable code running in VIXEE.

We define single AMELI Movie Script line, which maps AMELI event of role *God* saying the message *GodActions.growFruitTree* to the Movie Script action *PlantFruitTree*. This action, using Virtual World Grammar, plants a new fruit tree to the virtual field. In the following table, we list the AMELI Movie Script with a single line:

Institution	**Role**	**Event**	**Action**
Uruk	God	GodActions.growFruitTree	PlantFruitTree

In order to simulate fishing with different success, we have defined the illocutionary message *Fish.fish(count)* with the parameter *count* which tells the institution how many fishes a fisher man has caught. The value of this parameter is decided by the Movie Script action *DecideFishCaught*, which is executed prior to the calling of AMELI message *Fish.fish(count)*. For this purpose, we create the following Virtual World Movie Script:

World	**Location**	**Role**	**Message**	**Action**
Uruk	any(*)	Fisherman	Fish.fish	DecideFishCaught

8.4 Results

In this section, we present results on the social and cultural simulation of the city of Uruk. We divide these results in two different aspects, the generation of the Uruk crowd and simulation of life in Uruk.

Regarding the generation of Uruk inhabitants, we are interested in generating a crowd of Farmers who work on their fields. Generated crowd was evaluated on the diversity of agent appearance. First, we defined a base population formed by six agents of different race shown in Figure 8.8. We use the same base population as we used in Chapter 7 to evaluate the efficiency of our genetic algorithm. This population is formed by two avatars of asian origin, two of african origin, one european and one arab. Although this does not reflect actual Uruk demographic breakdown, it serves us to present the variance of generated avatars.

Using the base population, we have generated 14 new agents as their descendants, using our genetic based approach. Figure 8.9 and Figure 8.10 show distant and close-up views of the crowd. Analyzing the results we see, that each generated avatar is unique, concerning the body properties, having a different body structure, facial features, even a skin color. In this case, we have generated different physical properties while clothing and attachments remained the same.

Regarding the simulation of life in the city of Uruk, we focus on Fisherman and Potmaker roles and actions these roles perform during their daily schedule. Figure 8.11 contemplates an agent sleeping in his bed. In Uruk, beds were situated on rooftops, where due to the warm night provided them a comfortable place to rest. Please note that this figure and all the remaining figures were

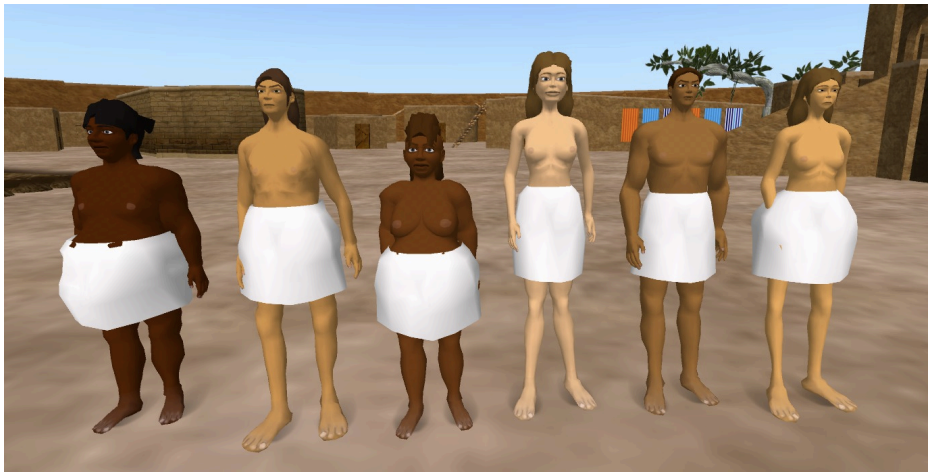


Figure 8.8: Base population of farmers

Figure 8.9: Generated crowd from distance

taken from Second Life as a separate experiment from crowd generation; thus, avatars differ in their appearance.

After Uruk citizens wake up, they all head to their place of prayer. Fisherman and Potmakers pray by the temple, while Farmers pray in fields. Figure 8.12 shows a group of Uruk citizens performing praying rituals.

After prayer, agents return to their homes and gather to eat. Agents prepare their food in common fireplaces, where fire has to be lit. Our Uruk Citizens con-

Figure 8.10: Base population of farmers

Figure 8.11: Uruk citizen sleeping in his bed

sume mainly fish and fruit, although in Figure 8.13 we display Citizens enjoying a baked lamb (it was a nice model to use).

After meal, Uruk citizens head out to work and move into the location where their job is performed. Fishermen find their boat, Farmers go to their field, and Potmakers sit by their Pot Rings. Agents can automatically navigate in the virtual space, calculating routes using a potential fields algorithm. Figure 8.14 shows two Fishermen walking to their job.

Figure 8.12: Uruk citizens praying by the temple

Figure 8.13: Citizens preparing their meal

Figure 8.15 shows a Potmaker creating a new pot. Potmakers first gather clay and water, so they can proceed with their work. After they create the pot, we remove the pot from the world model and add it to the Potmaker's belongings.

Figure 8.16 contemplates farmers harvesting fruits from the pomegranate tree. In this scenario, god Ninhursag places at random intervals a new fruit tree on the field where farmers can harvest it. To place a new fruit tree on

Figure 8.14: Fisherman walking to the work

Figure 8.15: Potmaker creating a new clay pot

the field, god Ninhursag announces to all agents in the GodActions scene the *growFruitTree* message. This is an example of dynamic manipulation of a virtual world content. This message is captured by VIXEE as an AMELI event. A Movie Script defines that if God tells this message a Movie Script action using the Virtual World Grammar places a new fruit tree in the virtual world model. Thus, we call a Virtual World Grammar that generates new geometry, and then using a 3D transformation engine, operated by Open Metaverse library, we add

Figure 8.16: Farmers harvesting fruit from the pomegranate tree.

a new fruit tree to the virtual world.

Another important feature of the VIXEE infrastructure is that the results of Ninhursag sending *growFruitTree* and *growFruit* messages can be sensed by all the agents with the "Farmer" role through the institution. So, when the new tree or fruit is planted the farmers can update their beliefs about the environment and include the new location of the corresponding fruit tree as a potential source of food and use this knowledge in their future planning routines.

8.5 Summary

In this chapter:

* We have presented a complex example in which VI Agents populate the city of Uruk 3000 BC, the first city on earth. In this example we have used all the techniques and mechanisms presented in this thesis,

With this chapter we finish our explanation of our work done for the automatic generation and control of interactive virtual worlds. In the next chapter we present our conclusions and state possible directions for future work.

Chapter 9

Conclusions and Future Work

In this research, we have presented different mechanisms and techniques for intelligent generation of interactive 3D virtual worlds. We have also presented a tool chain that supports and implements the presented techniques (Shape Grammar Interpreter, Virtual World Builder Toolkit, Virtual Institution eXEcution Environment and Genetic Mixer). In this section, we conclude our work in the order it was introduced; we present our future work and open discussion in related areas.

First, we introduced our algorithm for efficient subshape detection and its implementation in the shape grammar interpreter for rectilinear forms named SGI. SGI is not tied to any particular shape grammar. It supports interactive definition and manipulation of shape grammar shapes and rules. Several mechanisms (i.e execution protocols) have been implemented to select a candidate shape and rule to proceed in the design generation process. Our contributions, mainly to the shape grammar research field, are:

1. Definition of a new shape grammar framework and implementation of Shape Grammar Interpreter (SGI)

2. Definition of an efficient algorithm for subshapes detection.

3. Implementation of the subshape detection algorithm in SGI.

4. SGI architecture is extendible with plug-ins, that can be used by future researchers to include new SGI functionality (e.g. subshape detection for curved shapes).

5. Intuitive SGI user interface allows its easy deployment in class rooms as a learning tool for shape grammars. It can be also used by industry and architecture designers in order to better explore the possible design space.

Shape Grammar Interpreter is distributed with a GNU license from Sourceforge[1] website. The success of SGI have superseded our expectations. So far more there have been more than 1200 downloads and it is used in classrooms around the world (e.g. by Dr. Krishnamurti in Carnegie Mellon University[2]).

We have evaluated SGI tool from the perspective of performance of our provided algorithms. We showed, that in the general case our algorithm outperforms the original algorithm, thus optimizes it. This algorithm was thoroughly explained in text and on a provided example.

In our implementation markers have only limited functionality, i.e to control how rules are applied to the left-side shape. In future works, we plan to extend this functionality. We can generate designs using shapes with descriptive markers/labels. These markers can represent way-points useful for pathfinding and path-planning algorithms (e.g. every hole representing a door have inside and outside marks) which would allow autonomous movement through generated spaces. Moreover, we plan to collaborate with the shape grammar community and to integrate more features into SGI. We plan to incorporate the use of curves in the generation process of tree-search algorithm, as well as to include the subshape detection algorithm for curvilinear shapes from [Jowers, 2006]. We also plan to include the evaluation form directly in our tool, so that users can give us their opinions on our tool online and allow us to assess some aspects of the usability of our tool. Another possibility we would like to look at is to apply results of [Prats et al., 2004] and allow shape grammar designers to draw their grammars by hand and then analyze the drawings and synthesize new designs. Moreover, we would like to explore the possibility of using 3D shape grammars and skip the 2D to 3D transformation process.

Second, we have presented a Virtual World Grammar (VWG) for the automatic generation of 3D virtual worlds. The Virtual World Grammar holds semantic information about a system specification describing activities and relationship between them, a shape grammar, introducing design elements and their characteristics, and a list of validations and heuristics guiding the generation process. The virtual world generation is done in two steps, a first one in which the output is a 2D floor plan, and a second one which generates a 3D representation of the virtual world. Contributions of our research are:

1. The definition of the Virtual World Grammar and its components allows the automatic generation and manipulation of different virtual worlds from a formal specification of activities performing in the virtual environment.

2. The algorithm which defines how to navigate between the specification and the shape grammar execution tree using heuristics and validations.

3. The Virtual World Builder Toolkit allows an interactive definition and execution of Virtual World Grammars.

[1] `https://sourceforge.net/projects/sginterpreter/`
[2] `http://www.andrew.cmu.edu/course/48-747/subFrames/schedule.html` (05/2005)

An important feature of the VWG workflow is that the user can explore many different designs or modify existing parts of the shape grammar to explore new designs. We have demonstrated the VWG applicability in the generation of a 3D visualization of a Virtual Institution. Generation of of virtual world using Virtual World Grammar can be applied to any system where it has meaning to visualize its activities in a 3D virtual world. Current approach allows to map one activity per one space. Mapping more activities to one space brings challenges to their execution as it is difficult to control the concurrent execution or simply identify which of these actions need to be executed upon arrival to this virtual space.

Virtual World Builder Toolkit (VWBT) facilitates the functionality defined by the Virtual World Grammar. The architecture of SGI tool provides the possibility of extending it with new functionalities. The Virtual World Builder toolkit (VWBT) is an example of such an extension.

In the future we want to study the integration of previous work on 3D objects' behavior in virtual environments [Rodriguez et al., 2008] [D.Brota et al., 2009]. We also plan to apply our methodology in computer games domain, namely in MMORPG, where Electronic Institutions control the norm enforcement and VWG takes care of the visualization process.

Third, we have presented VIXEE, the Virtual Institution Execution Environment, which is an innovative communication infrastructure for Virtual Institutions where participants are human and software agents. Virtual Institutions are normative 3D virtual worlds where participants interact to achieve their common or individual goals. The main contributions of our research are:

- Our design of a middleware layer, which provides causal connection of several virtual worlds with an Electronic Institution which uses our Movie Script mechanism, improving previous version of an Action/Message approach. This allows users from different virtual worlds to participate in the same institution.

- Combination of our middleware with Virtual World Grammar allows dynamic manipulation of an environment content in different environments.

This thesis described the architecture and communication processes of VIXEE and explained what changes were made in comparison to previous approaches [Bogdanovych, 2007] [Bogdanovych et al., 2008] [Seidel, 2010]. VIXEE provides *architecturally neutral* and *domain independent*, and *scalable* solution for causal connection in Virtual Institutions. Architectural-neutrality from the agent point of view is given by AMELI, allowing to execute heterogeneous agents with any architecture. From the Virtual World point of view, a Virtual World Manager allows to connect different Virtual Worlds with any architecture. Domain independence is supported by Virtual Institutions concept and the Movie Script mechanism, where VIXEE uses movie script to handle communication in Virtual Institutions from many domains. We have presented the e-auction house institution example to contemplate the dynamic update of the 3D model of this institution.

We evaluated the performance of VIXEE in a high load scenario, with 500 agents executed in 25 threads. We have measured average VIXEE response time, while increasing the number of connected and communicating agents. We conclude that VIXEE does not introduce any limitations on the scalability of the system, and it is only limited by the complexity of the implementation of the movie script actions and by scalability limits of selected virtual worlds.

Currently, we are evaluating VIXEE in the e-learning scenario, which is a cultural simulation of the city of Uruk. As future work we plan to evaluate our VIXEE in other scenarios from e-* (e-commerce, e-government) applications. We also plan to evaluate the usability of the system with human users using different virtual worlds.

Fourth, we have presented a general purpose model for intelligent virtual agents. In our case we apply this model to the domain of Virtual Institutions; thus, we named this model VI Agents. VI Agents have the capability to participate in a Electronic Institution as well as visualize their actions in a 3D virtual world. Agents participate in the 3D virtual worlds through their avatars. They can interact with other agents as well as with interactive objects, named Virtual World Objects (VWO). Such objects are annotated to provide to an agent complete information on what are the possibilities, constraints and outcomes of such interaction.

Agent architecture consists of several modules which interact between them in order to accomplish agent goals. An agent plays a specific role in the system. Each role has assigned a list of temporal goals. Temporal goals are valid only in specific time interval. General goals are valid 24 hours. Agent reasons about its goals every specific time interval, or upon external events.

The main contributions of VI Agents are:

- Advanced believability features, such as agent psychology and physiology, allows to specify high level goals.
- The agent model contains information on its virtual culture. This information can be further disseminated. Virtual culture is responsible for making the general purpose agent model behave in culturally specific manner.
- Agents encode their defining properties into genetic structures, which can be operated by genetic operators to generate agents with unique appearance and behavior.
- Design of Virtual World Objects (VWO) creates tight bonding between agent and an environment by letting agents reason about these objects and base their plans on them. VWO model allows world designer to dynamically introduce new objects during system run-time.
- Designed to execute automatically in Virtual Institutions with no, or limited effort in programming.

In the future, we would like to add new features for VI Agents that would increase believability of their behavior. This includes implementation of an exist-

ing psychological and psychosomatical model. In other work we seek to include agent's non-verbal behavior.

Also, current implementation considers plan creation with no possibility of agent collaboration, using the brute force search in a performative structure. In our future works we would like to employ A-* algorithm to speed up reasoning and also create such representation of performative structure in a graph structure, that would allow us to plan collaborative agent actions with different strategies. Another possibility is to employ existing models on supply-chain problems or constraint satisfaction problems, to create collaborative plans for multiple agents and study how such plan creation is affected by the spatial and temporal factors.

And last, we have presented an algorithm for the generation of unique 3D avatars. We showed the importance of using genetic algorithms approach for generating large crowds of virtual agents and presented the Genetic Mixer application and its evaluation.

In our approach, first, parent chromosomes are combined using a crossover operator. We have provided the definition of several crossover operators and presented ideas on how to define new ones. We proposed the *fuzzy* crossover operator that allows fuzzy inheritance of visual features from parents. Second, we simulated the deep inheritance (situations when recessive genes become active generations later) with a process called gene skipping. Third, to introduce novelty in the generated population we use the mutation operator, which mutates specified amount of genes in the child chromosome. Finally, to respect the specifics population group and let the population designer fine tune the generation process, we define the genotype rules. Although, our process is not as biologically accurate as the previously mentioned [Vieira et al., 2010], it brings more transparent execution, better control and more diverse results. The main contributions of our research are:

- Gene skipping allows inheritance of features from our deeper ancestors.
- Genotype rules allow preservation and variance of defining visual feature. These rules can also encode relations between features maintaining ratios during their modification.
- Our fuzzy operator, together with mutation, gene skipping and genotype rules can simulate biological evolution.

The algorithm was evaluated for the variety of avatars it produces and preservation of visual features from parents, as well as for the speed of generation, and showed very good results on all those occasions.

Our system is modelling a genetical inheritance of agent's attributes. These attributes do not have to be visual. There exist many implementations [Egges et al., 2003] [Su et al., 2007] of agents with personalities based on OCEAN model [Ortony et al., 1988]. This personality model is used to produce emotions of agents, creating emotional agents. We can extend the chromosome

of agents to incorporate the personality values of the OCEAN model. In this way children would inherit personalities of their parents.

As our future work, we want to let the population designer carefully select exact individuals from the generation process, thus we seek to employ Interactive Genetic Algorithm, IGA [Caldwell and Johnston, 1991]. IGA belongs to a more general category of Interactive Evolutionary Computation and uses human evaluation to select the next population to be generated. In other work, we plan to use this approach with different virtual worlds and game engines and employ the genetic mechanism to create unique agent personalities.

9.1 Publications

This section contains the list of main publications we have produced while working on this dissertation:

Journals

- Trescak, T., Esteva, M., and Rodriguez, I. (2010b). A virtual world grammar for automatic generation of virtual worlds. *The Visual Computer Journal*, 26:521–531. Springer
- Aranda, G., Trescak, T., Esteva, M., and Carrascosa, C. (2011). Building quests for online games with virtual institutions. *Journal of Agents for games and simulations II*, 2:192–206. Springer-Verlag
- Trescak, T., Esteva, M., and Rodriguez, I. (2012a). A shape grammar interpreter for rectilinear forms. *Journal of Computer-Aided Design*, 44:657 – 670. Elsevier
- Aranda, G., Trescak, T., Esteva, M., Rodriguez, I., and Carrascosa, C. (2012). Massively multiplayer online games developed with agents. *Journal of Transactions on Edutainment VII*, 7145:129–138. Springer Berlin / Heidelberg
- Trescak, T., Esteva, M., Rodriguez, I., Sanchez, M. L., and Almajano, P. (to appear in 2012b). Execution infrastructure for normative virtual environments. *International Scientific Journal Engineering Applications of Artificial Intelligence*. Elsevier

Conferences

- Trescak, T., Rodriguez, I., and Esteva, M. (2009). General shape grammar interpreter for intelligent designs generations. In Werner, B., editor, *Proceedings of the Computer Graphics, Imaging and Visualization conference (CGIV'09)*, volume 6, pages 235–240, Tianjin, China. IEEE Computer Society, IEEE Computer Society

- Trescak, T., Esteva, M., Rodriguez, I., and Morales, J. (2010d). A virtual world builder toolkit (extended abstract). In *Proceedings of The 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pages 1627–1628, Toronto, Canada. IFAAMAS

- Trescak, T., Esteva, M., Rodriguez, I., and Morales, J. (2010c). A virtual world builder toolkit. In *Proceedings of The 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pages 1627–1628, Toronto, Canada. IFAAMAS

- Trescak, T., Esteva, M., and Rodriguez, I. (2011). Vixee an innovative communication infrastructure for virtual institutions. In *Proceedings of The 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11)*, volume 3 of *AAMAS '11*, pages 1131–1132, Richland, SC. IFAAMAS

- Trescak, T., Esteva, M., and Rodriguez, I. (2010a). Generating 3d virtual environments using the virtual world builder toolkit. In *Computer Graphics International 2010 (CGI'10) - Demo*, Singapore

- Almajano, P., Trescak, T., Esteva, M., Rodriguez, I., and Lopez-Sanchez, M. (to appear in 2012a). v-mwater: a 3d virtual market for water rights (demonstration). In *AAMAS '12*

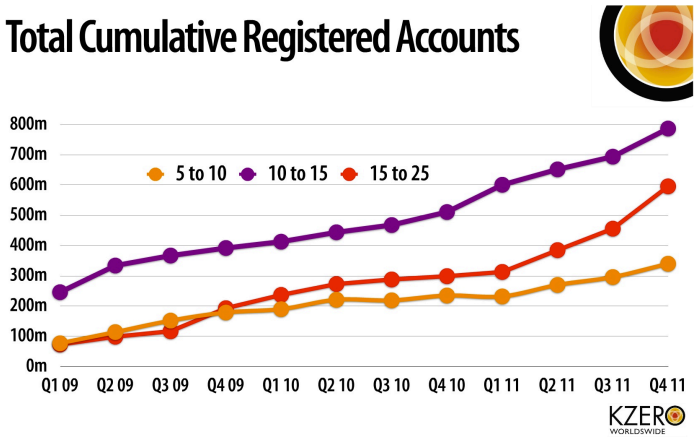
- Almajano, P., Trescak, T., Lopez-Sanchez, M., Esteva, M., and Rodriguez, I. (to appear in 2012b). An infrastructure for human inclusion in mas. In *ECAI '12*

Book Chapter

- Almajano, P., Trescak, T., Lopez-Sanchez, M., Esteva, M., and Rodriguez, I. (to appear in 2012c). v-mwater: an e-government application forwater rights agreements. In *Agreement Technologies Handbook*. Agreement Technologies

Appendix A

KZero research results



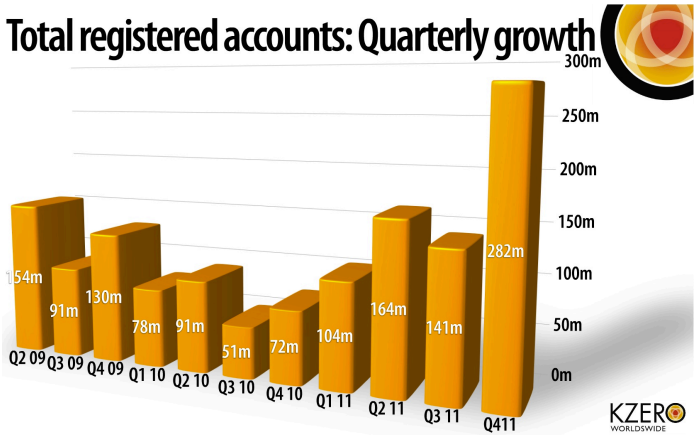


Figure A.1: Quarterly growth of Virtual World accounts

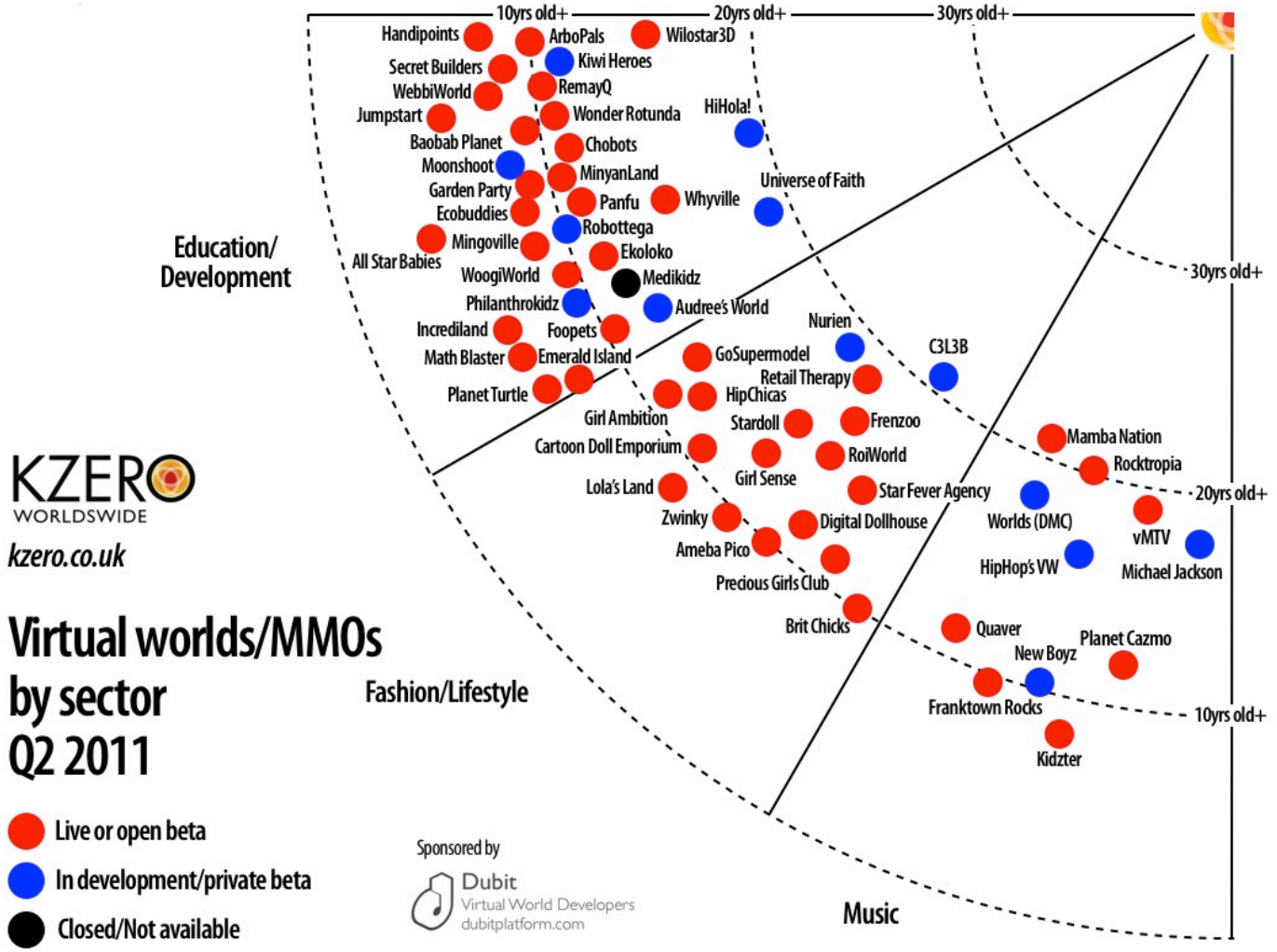


Figure A.2: Virtual Worlds by sector

Appendix B

XML Definition of a Shape Grammar

This is the source XML file of the shape grammar that contains a simple shape grammar with one square shape and an addition rule, that adds another copy of the square to its top right corner.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<shapeGrammar>
  <shapes>
    <shape id="0" main="true">
      <name>Square</name>
      <description />
      <content>
        <polyLine closed="true" id="1">
          <point x="120" y="60" />
          <point x="180" y="60" />
          <point x="180" y="120" />
          <point x="120" y="120" />
        </polyLine>
      </content>
    </shape>
  </shapes>
  <rules>
    <rule id="2">
      <name>AddSquare</name>
      <description />
      <type>Addition</type>
      <left id="0">
        <transformation>
          <transform>1.0,0.0,-22.0,0.0,1.0,10.0</transform>
        </transformation>
```

```
      </left>
      <right id="0">
        <transformation>
          <transform>1.0,0.0,8.0,0.0,1.0,-20.0</transform>
        </transformation>
      </right>
      <visual grid="false" rulers="true" snapGeometry="false">
        <leftGuides />
        <topGuides />
      </visual>
    </rule>
  </rules>
</shapeGrammar>
```

Bibliography

[Adami, 1998] Adami, C. (1998). *Introduction to artificial life*, volume 1. Telos Pr.

[Adobbati et al., 2001] Adobbati, R., Marshall, A., Scholer, A., Tejada, S., Kaminka, G., Schaffer, S., and Sollitto, C. (2001). Gamebots: A 3d virtual world test-bed for multi-agent research. *Proceedings of the second international workshop on Infrastructure for Agents, MAS, and Scalable MAS*, pages 47–52.

[Agarwal and Cagan, 1998] Agarwal, M. and Cagan, J. (1998). A blend of different tastes: the language of coffeemakers. *Environment and Planning B*, 25:205–226.

[Allen et al., 2003] Allen, B., Curless, B., and Popović, Z. (2003). The space of human body shapes: reconstruction and parameterization from range scans. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 587–594, New York, NY, USA. ACM.

[Almajano et al., 2012a] Almajano, P., Trescak, T., Esteva, M., Rodriguez, I., and Lopez-Sanchez, M. (to appear in 2012a). v-mwater: a 3d virtual market for water rights (demonstration). In *AAMAS '12*.

[Almajano et al., 2012b] Almajano, P., Trescak, T., Lopez-Sanchez, M., Esteva, M., and Rodriguez, I. (to appear in 2012b). An infrastructure for human inclusion in mas. In *ECAI '12*.

[Almajano et al., 2012c] Almajano, P., Trescak, T., Lopez-Sanchez, M., Esteva, M., and Rodriguez, I. (to appear in 2012c). v-mwater: an e-government application forwater rights agreements. In *Agreement Technologies Handbook*. Agreement Technologies.

[Ancona et al., 2008] Ancona, M., Drago, S., Quercini, G., and Bogdanovych, A. (2008). Rectangular dualization of biconnected planar graphs in linear time and related applications. *Applied and Industrial Mathematics in Italy II*, 75:37–48.

[André et al., 2000] André, E., Klesen, M., Gebhard, P., Allen, S., and Rist, T. (2000). Integrating models of personality and emotions into lifelike characters. *Affective interactions*, pages 150–165.

[Aranda et al., 2011] Aranda, G., Trescak, T., Esteva, M., and Carrascosa, C. (2011). Building quests for online games with virtual institutions. *Journal of Agents for games and simulations II*, 2:192–206. Springer-Verlag.

[Aranda et al., 2012] Aranda, G., Trescak, T., Esteva, M., Rodriguez, I., and Carrascosa, C. (2012). Massively multiplayer online games developed with agents. *Journal of Transactions on Edutainment VII*, 7145:129–138. Springer Berlin / Heidelberg.

[Aranda et al., 2010] Aranda, G. B., Trescak, T., Esteva, M., and Carrascosa, C. (2010). Building quests for online games with virtual institutions. In Dignum, F., editor, *AGS*, volume 6525 of *Lecture Notes in Computer Science*, pages 192–206. Springer.

[Badler, 1997] Badler, N. (1997). Real-time virtual humans. In *Computer Graphics and Applications, 1997. Proceedings., The Fifth Pacific Conference on*, pages 4–13. IEEE.

[Badler et al., 2002] Badler, N. I., Magenat-Thalmann, N., McCulloch, L., Hirsch, E. M., and LoPiccolo, P. (2002). Digital humans: what roles will they play? In *SIGGRAPH '02: SIGGRAPH 2002 conference abstracts and applications*. ACM Request Permissions.

[Bainbridge, 2007] Bainbridge, W. (2007). The scientific research potential of virtual worlds. *Science*, 317(5837):472–476.

[Barbara, 1998] Barbara, R. (1998). Crowd control. *Comput. Graph. World*, 21(2):30–32.

[Bartle, 2003] Bartle, R. (2003). *Designing Virtual Worlds*. New Riders Games.

[Bartneck, 2002] Bartneck, C. (2002). Integrating the occ model of emotions in embodied characters. *Workshop on Virtual Conversational Characters*.

[Beals, 2010] Beals, L. (2010). Content creation in virtual worlds to support adolescent identity development. *New Directions for Youth Development*, 2010(128):45–53.

[Bertram, 2000] Bertram, J. (2000). The molecular biology of cancer. *Molecular aspects of Medicine*, 21(6):167–223.

[Bickmore and Cassell, 2005] Bickmore, T. and Cassell, J. (2005). Social dialongue with embodied conversational agents. *Advances in natural multimodal dialogue systems*, pages 23–54.

[Blanz and Vetter, 1999] Blanz, V. and Vetter, T. (1999). A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 187–194, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Bogdanovych, 2007] Bogdanovych, A. (2007). *Virtual Institutions*. PhD thesis, University of Technology, Sydney, Australia.

[Bogdanovych et al., 2005] Bogdanovych, A., Berger, H., Sierra, C., and Simoff, S. (2005). Humans and agents in 3d electronic institutions. In Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M. P., and Wooldridge, M., editors, *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*. ACM, ACM.

[Bogdanovych et al., 2011] Bogdanovych, A., Rodríguez, J. A., Simoff, S., Cohen, A., and Sierra, C. (2011). Developing virtual heritage applications as normative multiagent systems. In *Proceedings of the 10th international conference on Agent-oriented software engineering*, AOSE'10, pages 140–154, Berlin, Heidelberg. Springer-Verlag.

[Bogdanovych et al., 2010a] Bogdanovych, A., Rodríguez-Aguilar, J. A., Simoff, S., and Cohen, A. (2010a). Authentic interactive re-enactment of cultural heritage with 3d virtual worlds and artificial intelligence. *Applied Artificial Intelligence*, 24:617–647.

[Bogdanovych et al., 2010b] Bogdanovych, A., Rodríguez-Aguilar, J. A., Simoff, S., and Cohen, A. (2010b). Authentic interactive re-enactment of cultural heritage with 3d virtual worlds and artificial intelligence. *Applied Artificial Intelligence*, 24:617–647.

[Bogdanovych et al., 2008] Bogdanovych, A., Simoff, S., and Esteva, M. (2008). Normative virtual environments: Integrating physical and virtual under the one umbrella. In *Third International Conference on Software and Data Technologies (IC-Soft 2008)*, pages 233–236. INSTICC.

[Bogdanovych et al., 2009] Bogdanovych, A., Simoff, S., and Esteva, M. (2009). Virtual institutions prototype. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1373–1374, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Bonabeau, 2002] Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences of the United States of America*, 99(Suppl 3):7280.

[Bordini et al., 2007] Bordini, R. H., Wooldridge, M., and Hubner, J. F. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons.

[Bouras et al., 2001] Bouras, C., Philopoulos, A., and Tsiatsos, T. (2001). e-learning through distributed virtual environments. *Journal of Network and Computer Applications*, 24(3):175–199.

[Bouras and Tsiatsos, 2006] Bouras, C. and Tsiatsos, T. (2006). Educational virtual environments: design rationale and architecture. *Multimedia tools and applications*, 29(2):153–173.

[Brom et al., 2009] Brom, C., Korenko, T., and Lukavský, J. (2009). How Do Place and Objects Combine¿'What-Where" Memory for Human-Like Agents. *Intelligent Virtual Agents*, pages 42–48.

[Brom et al., 2008] Brom, C., Lukavský, J., and Kadlec, R. (2008). Episodic memory for human-like agents and human-like agents for episodic memory. In *AAAI Fall Symposium on Biologically Inspired Cognitive Architectures (BICA)*.

[Caldwell and Johnston, 1991] Caldwell, C. and Johnston, V. S. (1991). Tracking a Criminal Suspect through "Face-Space" with a Genetic Algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithm*, pages 416–421. Morgan Kaufmann Publisher.

[Cassell et al., 1994] Cassell, J., Pelachaud, C., Badler, N., Steedman, M., Achorn, B., Becket, T., Douville, B., Prevost, S., and Stone, M. (1994). Animated conversation: rule-based generation of facial expression, gesture & spoken intonation for multiple conversational agents. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 413–420. ACM.

[Cavazza, 2007] Cavazza, F. (2007). Virtual universes landscape. `http://www.fredcavazza.net/2007/10/04/virtual-universes-landscape/`, last accessed 04/2012.

[Chase, 1989] Chase, S. C. (1989). Shapes and shape grammars: from mathematical model to computer implementation. *Environment and Planning B: Planning and Design*, 16:215–242.

[Chau et al., 2004] Chau, H. H., Chen, X., McKay, A., and de Pennington, A. (2004). Evaluation of a 3D shape grammar implementation. In Gero, J. S., editor, *Design Computing and Cognition '04*, pages 357–376, Dordrecht. Kluwer.

[Chesney et al., 2009] Chesney, T., Chuah, S.-H., and Hoffmann, R. (2009). Virtual world experimentation: An exploratory study. *Journal of Economic Behavior and Organization*, 72(1):618 – 635.

[Comings et al., 1996] Comings, D., Rosenthal, R., Lesieur, H., Rugle, L., Muhleman, D., Chiu, C., Dietz, G., and Gade, R. (1996). A study of the dopamine d2 receptor gene in pathological gambling. *Pharmacogenetics and Genomics*, 6(3):223.

[Costa and McCrae, 1992] Costa, P. and McCrae, R. (1992). Neo pi-r professional manual. *Odessa, FL: Psychological Assessment Resources*, 396:653–659.

[Cranefield and Li, 2009] Cranefield, S. and Li, G. (2009). Monitoring social expectations in second life. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[D.Brota et al., 2009] D.Brota, Rodriguez, I., Puig, A., and Esteva, M. (2009). A generic framework to exploit virtual worlds as normative and dynamic interactive spaces. In *Computer Graphics and Virtual Reality*, pages 151–157.

[DeCarlo et al., 1998] DeCarlo, D., Metaxas, D., and Stone, M. (1998). An anthropometric face model using variational techniques. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 67–74, New York, NY, USA. ACM.

[Dillenbourg, 1999] Dillenbourg, P. (1999). *Collaborative Learning: Cognitive and Computational Approaches. Advances in Learning and Instruction Series.* Elsevier Science, Inc., Madison Square Station, New York.

[Doce et al., 2010] Doce, T., Dias, J., Prada, R., and Paiva, A. (2010). Creating individual agents through personality traits. *Intelligent Virtual Agents*, pages 257–264.

[Duarte, 2001] Duarte, J. P. (2001). *Customizing mass housing : A discursive grammar for Siza's Malagueira houses.* PhD thesis, Cambridge (MA): Massachusetts Institute of Technology.

[Egges et al., 2003] Egges, A., Kshirsagar, S., and Magnenat-Thalmann, N. (2003). A model for personality and emotion simulation. *Knowledge-Based Intelligent Information and Engineering Systems*, pages 453–461.

[Esteva, 2003] Esteva, M. (2003). *Electronic Institutions: From Specification to Development.* PhD thesis, Artificial Intelligence Research Institute (IIIA-CSIC), Spain.

[Esteva et al., 2002] Esteva, M., de la Cruz, D., and Sierra, C. (2002). Islander: an electronic institutions editor. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1045–1052, New York, NY, USA. ACM.

[Esteva et al., 2008] Esteva, M., Rodriguez-Aguilar, J. A., Arcos, J. L., Sierra, C., Noriega, P., Rosell, B., and de la Cruz, D. (2008). Electronic institutions development environment. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1657–1658, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Esteva et al., 2004] Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., and Arcos, J. L. (2004). Ameli: An agent-based middleware for electronic institutions. In Jennings, N. e. a., editor, *AAMAS 2004, Third international joint conference on autonomous agents and multiagent systems*, pages 236–243. ACM, ACM.

[Farmer and Foley, 2009] Farmer, J. and Foley, D. (2009). The economy needs agent-based modelling. *Nature*, 460(7256):685–686.

[Ferber et al., 2004] Ferber, J., Gutknecht, O., and Michel, F. (2004). From agents to organizations: an organizational view of multi-agent systems. *Agent-Oriented Software Engineering IV*, pages 443–459.

[Flemming, 1987] Flemming, U. (1987). More than the sum of parts: the grammar of queen anne houses. *Environment and Planning B: Planning and Design*, 14(3):323–350.

[Fogel, 1995] Fogel, D. (1995). *Evolutionary Computation*. IEEE Press, New York.

[Footprint, 2007] Footprint, T. V. B. (2007). The marketing opportunity in second life. Technical report, Diversified Media Design, Combined Story, Market Truths.

[Geiger et al., 2000] Geiger, C., Paelke, V., Reimann, C., and Rosenbach, W. (2000). A framework for the structured design of vr/ar content. In *VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 75–82, New York, NY, USA. ACM.

[Gemrot et al., 2009] Gemrot, J., Kadlec, R., Bída, M., Burker, O., Píbil, R., Havlíček, J., Zemčák, L., Šimlovič, J., Vansa, R., Štolba, M., Plch, T., and Brom, C. (2009). Pogamut 3 can assist developers in building AI (not only) for their videogame agents. *Agents for Games and Simulations*, pages 1–15.

[Gillis, 2000] Gillis, P. (2000). *Gillis: Cognitive behaviors for computer generated... - Google Scholar*. US Army Research Institute Technical Report.

[Goldberg, 1989] Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley.

[Gratch and Marsella, 2004] Gratch, J. and Marsella, S. (2004). A domain-independent framework for modeling emotion. *Cognitive Systems Research*, 5(4):269–306.

[Gratch and Marsella, 2005] Gratch, J. and Marsella, S. (2005). Evaluating a computational model of emotion. In *Autonomous Agents and Multi-Agent Systems*, pages 23–43. Univ So Calif, Inst Creat Technol, Marina Del Rey, CA 90292 USA.

[Gratch et al., 2002] Gratch, J., Rickel, J., André, E., Cassell, J., Petajan, E., and Badler, N. (2002). Creating interactive virtual humans: Some assembly required. *Intelligent Systems, IEEE*, 17(4):54–63.

[Gu and Maher, 2003] Gu, N. and Maher, M. (2003). A grammar for the dynamic design of virtual architecture using rational agents. *International Journal of Architectural Computing*, 1(4):489–501.

[Gu and Maher, 2004] Gu, N. and Maher, M. (2004). Generating virtual architecture with style. In *Proceedings of the Design Computing and Cognition'04 Workshop on Design and Research Issues in Virtual Worlds*.

[Hoisl and Shea, 2011] Hoisl, F. and Shea, K. (2011). Interactive, visual 3D spatial grammars. In Gero, J. S., editor, *Design Computing and Cognition 10*, pages 643–662. Springer Netherlands.

[Holland, 1975] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA.

[Hudlicka, 2005] Hudlicka, E. (2005). A computational model of emotion and personality: Applications to psychotherapy research and practice. *Proceedings of the 10th Annual CyberTherapy Conference: A Decade of Virtual Reality*.

[Jan et al., 2009] Jan, D., Roque, A., Leuski, A., Morie, J., and Traum, D. (2009). A Virtual Tour Guide for Virtual Worlds. In *IVA '09: Proceedings of the 9th International Conference on Intelligent Virtual Agents*. Springer-Verlag.

[Janis and Mann, 1977] Janis, I. L. and Mann, L. (1977). *Decision making: A psychological analysis of conflict, choice, and commitment.* Free Press.

[Jowers, 2006] Jowers, I. (2006). *Computation with Curved Shapes: Towards Freeform Shape Generation in Design*. PhD thesis, The Open University.

[Joyce et al., 1980] Joyce, B., Weil, M., and Calhoun, E. (1980). *Models of teaching*, volume 499. Prentice-Hall Englewood Cliffs, NJ.

[J.R. et al., 2002] J.R., W., R.M, E., and M., D. (2002). Structured development of virtual environments. In *Handbook of Virtual Environments: Design, implementation and applications*, pages 353–378. K. Stanney.

[Kadlec, 2008] Kadlec, R. (2008). *Evolution of intelligent agent behaviour in computer games*. PhD thesis, Charles University in Prague.

[Kasap and Magnenat-Thalmann, 2008] Kasap, Z. and Magnenat-Thalmann, N. (2008). Intelligent virtual humans with autonomy and personality: State-of-the-art. *New Advances in Virtual Humans*, pages 43–84.

[Kipp et al., 2010] Kipp, M., Heloir, A., Schroder, M., and Gebhard, P. (2010). Realizing multimodal behavior: closing the gap between behavior planning and embodied agent presentation. *Intelligent Virtual Agents*, pages 57–63.

[Kolb et al., 1974] Kolb, D., Rubin, I., and McIntyre, J. (1974). *Organizational psychology: an experiential approach*. Prentice-Hall behavioral science in business series. Prentice-Hall.

[Koning and Eizenberg, 1981] Koning, H. and Eizenberg, J. (1981). The language of the prairie: Frank lloyd wright's prairie houses. *Environment and Planning B*, 8(3):295–323.

[Kopp et al., 2005] Kopp, S., Gesellensetter, L., Kramer, N. C., and Wachsmuth, I. (2005). A conversational agent as museum guide: design and evaluation of a real-world application. *Lecture Notes in Computer Science*.

[Koza, 1992] Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

[Krishnamurti, 1981] Krishnamurti, R. (1981). The construction of shapes. *Environment and Planning B: Planning and Design*, 8:5–40.

[Kshirsagar and Magnenat-Thalmann, 2002] Kshirsagar, S. and Magnenat-Thalmann, N. (2002). Virtual humans personified. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, AAMAS '02, pages 356–357, New York, NY, USA. ACM.

[Laister and Kober, 2002] Laister, J. and Kober, S. (2002). Social aspects of collaborative learning in virtual learning environments. In *Proceedings of the Networked Learning Conference Sheffield, March*.

[Layne and Lee, 2001] Layne, K. and Lee, J. (2001). Developing fully functional e-government: A four stage model. *Government information quarterly*, 18(2):122–136.

[Leduc, 1911] Leduc, S. (1911). *The Mechanism Of Life*. William Heinemann.

[Lee et al., 2010] Lee, J., Wang, Z., and Marsella, S. (2010). Evaluating models of speaker head nods for virtual agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, pages 1257–1264, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Lewis, 2000] Lewis, M. (2000). Evolving human figure geometry. Technical Report OSU-ACCAD-5/00-TR1, ACCAD, Ohio State University.

[Lewis and Parent, 2000] Lewis, M. and Parent, R. (2000). An implicit surface prototype for evolving human figure geometry. Technical Report OSU-ACCAD-11/00-TR2, ACCAD, Ohio State University.

[Li et al., 2008] Li, X., Mao, W., Zeng, D., and Wang, F. (2008). Agent-based social simulation and modeling in social computing. *Intelligence and Security Informatics*, pages 401–412.

[Liao, 2008] Liao, C. L. (2008). Avatars, second life®, and new media art: The challenge for contemporary art education. In *Art Education*. National Art Education Association.

[Lim et al., 2008] Lim, S., Prats, M., Jowers, I., Chase, S., Garner, S., and McKay, A. (2008). Shape exploration in design: formalising and supporting a transformational process. *International Journal of Architectural Computing*, 6(4):415–433.

[Lipp et al., 2008] Lipp, M., Wonka, P., and Wimmer, M. (2008). Interactive visual editing of grammars for procedural architecture. *ACM Trans. Graph.*, 27(3):1–10.

[Liu et al., 2009] Liu, Y. L. Y., Yang, C. Y. C., and Yu, J. Y. J. (2009). Research on a model of emotion for virtual agent. *Audio, Transactions of the IRE Professional Group on*, 2:96–99.

[Lou Maher and Gu, 2003] Lou Maher, M. and Gu, N. (2003). Situated design of virtual worlds using rational agents. In *ICEC '03: Proceedings of the second international conference on Entertainment computing*. Carnegie Mellon University.

[Lou Maher et al., 2005] Lou Maher, M., Liew, P., Gu, N., and Ding, L. (2005). An agent approach to supporting collaborative design in 3d virtual worlds. *Automation in Construction*, 14(2):189–195.

[Loyall, 1997] Loyall, A. (1997). *Believable agents: building interactive personalities*. PhD thesis, Carnegie Mellon University.

[Maes and Nardi, 1988] Maes, P. and Nardi, D., editors (1988). *Meta-Level Architectures and Reflection*. Elsevier Science Inc., New York, NY, USA.

[Magnenat-Thalmann et al., 2004a] Magnenat-Thalmann, N., Seo, H., and Cordier, F. (2004a). Automatic modeling of virtual humans and body clothing. *Journal of Computer Science and Technology*, 19(5):575–584.

[Magnenat-Thalmann et al., 2004b] Magnenat-Thalmann, N., Seo, H., and Cordier, F. (2004b). Automatic modeling of virtual humans and body clothing. *Journal of Computer Science and Technology*, 19(5):575–584.

[Maher et al., 2000] Maher, M., Simoff, S., Gu, N., and Lau, K. (2000). Designing virtual architecture. In *CAADRIA 2000: The Fifth Conference on Computer-Aided Architectural Design Research in Asia*, pages 481–490.

[Maim et al., 2009] Maim, J., Yersin, B., and Thalmann, D. (2009). Unique character instances for crowds. *Computer Graphics and Applications, IEEE*, 29(6):82–90.

[Mansouri et al., 2009] Mansouri, H., Kleinermann, F., and De Troyer, O. (2009). Detecting inconsistencies in the design of virtual environments over the web using domain specific rules. In *Web3D '09: Proceedings of the 14th International Conference on 3D Web Technology*, pages 101–109, New York, NY, USA. ACM.

[Marsella and Badler, 2011] Marsella, S. and Badler, N. (2011). Special section: Intelligent virtual agents guest editors' introduction. *PRESENCE: Teleoperators and Virtual Environments*, 20(5).

[Marsella et al., 2010] Marsella, S., Gratch, J., and Petta, P. (2010). Computational Models of Emotion. In Scherrer, K., Banziger, T., and Roesch, E., editors, *Blueprint for Affective Computing*. Oxford University Press.

[McCormack and Cagan, 2002] McCormack, J. P. and Cagan, J. (2002). Supporting designers and hierarchies through parametric shape recognition. *Environment and Planning B: Planning and Design*, 29(6):913–931.

[McCormack and Cagan, 2003] McCormack, J. P. and Cagan, J. (2003). Extending the representation capabilities of shape grammars: A parametric matching technique for shapes defined by curved lines. Technical report, AAAI SS-03-02.

[McInnerney and Roberts, 2004] McInnerney, J. and Roberts, T. (2004). Online learning: Social interaction and the creation of a sense of community. *Educational Technology & Society*, 7(3):73–81.

[McNeill, 1996] McNeill, D. (1996). *Hand and mind: What gestures reveal about thought*. University of Chicago Press.

[Messinger et al., 2008] Messinger, P., Stroulia, E., and Lyons, K. (2008). A typology of virtual worlds: Historical overview and future directions. *Journal of Virtual Worlds Research*, 1(1).

[Messinger et al., 2009] Messinger, P. R., Stroulia, E., Lyons, K., Bone, M., Niu, R. H., Smirnov, K., and Perelgut, S. (2009). Virtual worlds - past, present, and future: New directions in social computing. *Decision Support Systems*, 47(3):204–228.

[Min, 2004] Min, P. (2004). *A 3D Model Search Engine*. PhD thesis, The Princeton University.

[Mitham, 2010] Mitham, N. (2010). Virtual goods: good for business? *Journal of Virtual Worlds Research*, 2(4).

[Monahan et al., 2008] Monahan, T., McArdle, G., and Bertolotto, M. (2008). Virtual reality for collaborative e-learning. *Computers & Education*, 50(4):1339–1353.

[Muller et al., 2006] Muller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. (2006). Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623.

[Napagao et al., 2010] Napagao, S. A., Koch, F., Sebastia, I. G., and Vazquez, J. S. (2010). Making games alive: an organisational approach. In *Proceedings of AAMAS 2010 Workshop on Agents for Games and Simulations*, pages 112–124.

[Noriega, 1999] Noriega, P. (1999). *Agent Mediated Auctions: The Fishmarket Metaphor*. Monografies de l'Institut d'Investigació en Intel.ligència Artificial. Institut d'Investigació en Intel.ligència Artificial.

[Nuxoll and Laird, 2007] Nuxoll, A. M. and Laird, J. E. (2007). Extending cognitive architecture with episodic memory. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1560–1565. AAAI Press.

[Ofria and Wilke, 2003] Ofria, C. and Wilke, C. O. (2003). Avida: a software platform for research in computational evolutionary biology. *Artificial Life*, 10(2):191–229.

[Orsborn and Cagan, 2009] Orsborn, S. and Cagan, J. (2009). Multiagent shape grammar implementation: automatically generating form concepts according to a preference function. *Journal of Mechanical Design*, 131.

[Ortony et al., 1988] Ortony, A., Clore, G. L., and Collins, A. (1988). *The Cognitive Structure of Emotions*. Cambridge University Press.

[Parish and Muller, 2001] Parish, Y. I. H. and Muller, P. (2001). Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA. ACM Press.

[Prats et al., 2004] Prats, M., Garner, S., Jowers, I., and Earl, C. (2004). Improving product design via a shape grammar tool. In D., M., editor, *Proceedings of the 8th International Design Conference, DESIGN 2004, Dubrovnik, Croatia*, pages 477–482.

[Prats et al., 2009] Prats, M., Lim, S., Jowers, I., Garner, S., and Chase, S. (2009). Transforming shape in design: observations from studies of sketching. *Design Studies*, 30(5):503–520.

[Prusinkiewicz et al., 2001] Prusinkiewicz, P., Muendermann, L., Karwowski, R., and Lane, B. (2001). The use of positional information in the modeling of plants. In *Proceedings of ACM SIGGRAPH 2001*, pages 289–300.

[Ranathunga et al., 2010] Ranathunga, S., Cranefield, S., and Purvis, M. (2010). Interfacing a cognitive agent platform with a virtual world: a case study using second life surangika ranathunga (extended abstract). In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1182. International Foundation for Autonomous Agents and Multiagent Systems.

[Rasmussen et al., 1990] Rasmussen, S., Knudsen, C., Feldberg, R., and Hindsholm, M. (1990). The Coreworld - Emergence and Evolution of Cooperative Structures in a Computational Chemistry. *Physica D*, 42:111–134.

[Ray, 1991] Ray, T. (1991). An approach to the synthesis of life. *Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity*, 9:371–408.

[Ray, 1993] Ray, T. (1993). An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artificial Life*, 1(1_2):179–209.

[Rechenberg, 1973] Rechenberg, I. (1973). *Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann Holzboog.

[Rodriguez et al., 2008] Rodriguez, I., Puig, A., Esteva, M., Sierra, C., Bogdanovych, A., and Simoff, S. (2008). Intelligent objects to facilitate human participation in virtual institutions. In *Web Intelligence*, pages 196–199.

[Rodriguez-Aguilar, 2001] Rodriguez-Aguilar, J. A. (2001). *On the Design and Construction of Agent-Mediated Electronic Institutions.* PhD thesis, Universitat Autonoma de Barcelona, Spain.

[Rolls et al., 2002] Rolls, E., Stringer, S., and Trappenberg, T. (2002). A unified model of spatial and episodic memory. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 269(1496):1087–1093.

[Russell et al., 1995] Russell, S., Norvig, P., and Artificial Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs.*

[San Chee, 2001] San Chee, Y. (2001). Networked virtual environments for collaborative learning. In *Proceedings of ICCE/SchoolNet 2001Ninth International Conference on Computers in Education, Seoul, S. Korea*, page 311. Citeseer.

[Schill and Zetzsche, 1995] Schill, K. and Zetzsche, C. (1995). A model of visual spatio-temporal memory: The icon revisited. *Psychological Research*, 57(2):88–102.

[Searle, 1969] Searle, J. R. (1969). *Speech Acts.* Cambridge University Press, Cambridge, UK.

[Seidel, 2010] Seidel, I. (2010). *Engineering 3D Virtual World Applications Design, Realization and Evaluation of a 3D e-Tourism Environment.* PhD thesis, Technischen Universitat Wien Fakultat fur Informatik.

[Seo and Magnenat-Thalmann, 2003] Seo, H. and Magnenat-Thalmann, N. (2003). An automatic modeling of human bodies from sizing parameters. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics.* ACM Request Permissions.

[Silverman et al., 2006a] Silverman, B., Bharathy, G., O'Brien, K., and Cornwell, J. (2006a). Human behavior models for agents in simulators and games: part II: gamebot engineering with PMFserv. *Presence: Teleoperators & Virtual Environments*, 15(2):163–185.

[Silverman et al., 2006b] Silverman, B., Johns, M., Cornwell, J., and O'Brien, K. (2006b). Human behavior models for agents in simulators and games: part I: enabling science with PMFserv. *Presence: Teleoperators & Virtual Environments*, 15(2):139–162.

[Silverman et al., 2011] Silverman, B., Pietrocola, D., Weyer, N., Osin, O., Johnson, D., Weaver, R., and Nye, B. (2011). Rich socio-cognitive agents for immersive training environments: case of NonKin Village. *Autonomous Agents and Multi-Agent Systems*, pages 1–32.

[Sims, 1994] Sims, K. (1994). Evolving virtual creatures. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM Request Permissions.

[Singhal and Zyda, 1999] Singhal, S. and Zyda, M. (1999). Networked virtual environments: design and implementation. *Recherche*, 67:02.

[Sipper, 1995] Sipper, M. (1995). An introduction to artificial life. *Explorations in Artificial Life (special issue of AI Expert)*, pages 4–8.

[Sivanandam and Deepa, 2007] Sivanandam, S. and Deepa, S. (2007). *Introduction to genetic algorithms*. Springer Verlag.

[Smith et al., 2007] Smith, G., Maher, M., and Gu, N. (2007). Designing Virtual Worlds for 3D Electronic Institutions. *Computer-Aided Architectural Design Futures (CAADFutures) 2007*, pages 397–400.

[Soni et al., 2010] Soni, S., Khanna, P., and Tandon, P. (2010). Multi-agent feature based shape grammar implementation for concept generation of industrial product design. *Computer-Aided Design and Applications*, 7(6):797–807.

[Southey and Linders, 2001] Southey, F. and Linders, J. G. (2001). Ossa - a conceptual modelling system for virtual realities. In *ICCS '01: Proceedings of the 9th International Conference on Conceptual Structures*, pages 333–345, London, UK. Springer-Verlag.

[Steinkuehler, 2004] Steinkuehler, C. (2004). Learning in massively multiplayer online games. In *Proceedings of the 6th international conference on Learning sciences*, pages 521–528. International Society of the Learning Sciences.

[Stephenson, 2000] Stephenson, N. (2000). *Snow Crash*. Bantam spectra book. Bantam Books.

[Steunebrink et al., 2009] Steunebrink, B., Dastani, M., and Meyer, J. (2009). The occ model revisited. In *Proceedings of the 4th Workshop on Emotion and Computing*.

[Stiny and Gips, 1972] Stiny, G. and Gips, J. (1972). Shape grammars and the generative specification of painting and sculpture. In Friedman, C. V., editor, *Information Processing '71*, pages 1460–1465, Amsterdam.

[Stiny and Mitchell, 1978] Stiny, G. and Mitchell, W. J. (1978). The palladian grammar. *Environment and Planning B*, 5(1):5–18.

[Stiny and Mitchell, 1980] Stiny, G. and Mitchell, W. J. (1980). The grammar of paradise: on the generation of mughul gardens. *Environment and Planning B*, 7(2):209–226.

[Su et al., 2007] Su, W.-P., Pham, B., and Wardhani, A. (2007). Personality and emotion-based high-level control of affective story characters. *IEEE Transactions on Visualization and Computer Graphics*, pages 281–293.

[Swartout et al., 2006] Swartout, W., Gratch, J., Hill Jr, R., Hovy, E., Marsella, S., Rickel, J., Traum, D., et al. (2006). Toward virtual humans. *AI Magazine*, 27(2):96.

[T. Speller, 2007] T. Speller, D. Whitney, E. C. (2007). Using shape grammar to derive cellular automata rule patterns. *Complex Systems*, 17:79102.

[Takahashi et al., 2007] Takahashi, S., Sallach, D., and Rouchier, J. (2007). *Advancing Social Simulation: The First World Congress*. Springer Series on Agent Based Social Systems Series. Springer.

[Tanriverdi and Jacob, 2001] Tanriverdi, V. and Jacob, R. J. K. (2001). Vrid: A design model and methodology for developing virtual reality interfaces. In *Proc. ACM VRST 2001 Symposium on Virtual Reality Software and Technology, ACM*, pages 175–182. Press.

[Tapia, 1992] Tapia, M. (1992). Chinese lattice designs and parametric shape grammars. *The Visual Computer*, 9:47–56.

[Tapia, 1999] Tapia, M. (1999). A visual implementation of a shape grammar system. *Environment and Planning B: Planning and Design*, 26(1):59–73.

[Tapscott, 1998] Tapscott, D. (1998). *Growing up digital*, volume 302. McGraw-Hill New York.

[Thalmann, 2007] Thalmann, D. (2007). *Crowd simulation*. Wiley Online Library.

[Thalmann et al., 2009] Thalmann, D., Grillon, H., Maim, J., and Yersin, B. (2009). Challenges in crowd simulation. In *CyberWorlds, 2009. CW'09. International Conference on*, pages 1–12. IEEE.

[TheSun, 2010] TheSun (2010). Black parents... white baby. `http://www.thesun.co.uk/sol/homepage/news/3060907/Black-parents-give-birth-to-white-baby.html`.

[Trescak et al., 2010a] Trescak, T., Esteva, M., and Rodriguez, I. (2010a). Generating 3d virtual environments using the virtual world builder toolkit. In *Computer Graphics International 2010 (CGI'10) - Demo*, Singapore.

[Trescak et al., 2010b] Trescak, T., Esteva, M., and Rodriguez, I. (2010b). A virtual world grammar for automatic generation of virtual worlds. *The Visual Computer Journal*, 26:521–531. Springer.

[Trescak et al., 2011] Trescak, T., Esteva, M., and Rodriguez, I. (2011). Vixee an innovative communication infrastructure for virtual institutions. In *Proceedings of The 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11)*, volume 3 of *AAMAS '11*, pages 1131–1132, Richland, SC. IFAAMAS.

[Trescak et al., 2012a] Trescak, T., Esteva, M., and Rodriguez, I. (2012a). A shape grammar interpreter for rectilinear forms. *Journal of Computer-Aided Design*, 44:657 – 670. Elsevier.

[Trescak et al., 2010c] Trescak, T., Esteva, M., Rodriguez, I., and Morales, J. (2010c). A virtual world builder toolkit. In *Proceedings of The 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pages 1627–1628, Toronto, Canada. IFAAMAS.

[Trescak et al., 2010d] Trescak, T., Esteva, M., Rodriguez, I., and Morales, J. (2010d). A virtual world builder toolkit (extended abstract). In *Proceedings of The 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pages 1627–1628, Toronto, Canada. IFAAMAS.

[Trescak et al., 2012b] Trescak, T., Esteva, M., Rodriguez, I., Sanchez, M. L., and Almajano, P. (to appear in 2012b). Execution infrastructure for normative virtual environments. *International Scientific Journal Engineering Applications of Artificial Intelligence*. Elsevier.

[Trescak et al., 2009] Trescak, T., Rodriguez, I., and Esteva, M. (2009). General shape grammar interpreter for intelligent designs generations. In Werner, B., editor, *Proceedings of the Computer Graphics, Imaging and Visualization conference (CGIV'09)*, volume 6, pages 235–240, Tianjin, China. IEEE Computer Society, IEEE Computer Society.

[Troyer et al., 2003] Troyer, O. D., Bille, W., Romero, R., and Stuer, P. (2003). On generating virtual worlds from domain ontologies. In *Proceedings of the 9th International Conference on Multi-Media Modeling*, pages 279–294.

[Van Dyke Parunak et al., 2006] Van Dyke Parunak, H., Bisson, R., Brueckner, S., Matthews, R., and Sauter, J. (2006). A model of emotions for situated agents. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 993–995.

[Ventrella, 2000] Ventrella, J. (2000). Avatar Physics and Genetics. In *VW '00: Proceedings of the Second International Conference on Virtual Worlds*. Springer-Verlag.

[Vieira et al., 2008] Vieira, R. C. C., Vidal, C. A., and Cavalcante-Neto, J. B. (2008). A biologically inspired generation of virtual characters. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 1218–1224, New York, NY, USA. ACM.

[Vieira et al., 2010] Vieira, R. C. C., Vidal, C. A., and Neto, J. B. C. (2010). Simulation of genetic inheritance in the generation of virtual characters. In Lok, B., Klinker, G., and Nakatsu, R., editors, *VR*, pages 119–126. IEEE.

[Wang and Duarte, 2002] Wang, Y. and Duarte, J. P. (2002). Automatic generation and fabrication of designs. *Automation in Construction*, 11(3):291 – 302.

[Weidlich, 2000] Weidlich, W. (2000). *Sociodynamics: A systematic approach to mathematical modelling in the social sciences*. Harwood Academic.

[Weiss, 1999] Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT press.

[Weitnauer et al., 2008] Weitnauer, E., Thomas, N. M., Rabe, F., and Kopp, S. (2008). Intelligent Agents Living in Social Virtual Environments — Bringing Max into Second Life. In *IVA '08: Proceedings of the 8th international conference on Intelligent Virtual Agents*. Springer-Verlag.

[Witmer et al., 2002] Witmer, B. G., Jerome, C., and Goldberg, S. L. (2002). Modeling Human Performance: Effects of Personal Traits and Transitory States. Technical report, Army research institute for the behavioral and social sciences.

[Yaeger, 1993] Yaeger, L. (1993). Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or PolyWorld: Life in a new context. In *Artificial Life III, Vol. XVII of SFI Studies in the Sciences of Complexity, Santa Fe Institute*, pages 263–298. Addison-Wesley.

[Yaeger et al., 2010] Yaeger, L., Sporns, O., Williams, S., and Shuai, X. (2010). Evolutionary selection of network structure and function. *Artificial Life*, 12.

[Yee, 2007] Yee, N. (2007). *The Proteus Effect*. PhD thesis, Stanford University.

[Zimmer, 2005] Zimmer, C. (2005). Testing darwin. *Discover*.