THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse III - Paul Sabatier
Discipline ou spécialité : Informatique

Présentée et soutenue par Jaime Alberto Zaragoza Rios
Le 16 décembre 2009

Titre : Declarative Modeling Based on Knowledge

JURY

Veronique Gaildrat, Professeur UPS
Félix Francisco Ramos Corchado, Professeur Chercheur titulaire CINVESTAV 3A
Georges Miaoulis, Professeur Technological Educational Institute of Athens et Professeur associé à l'Université de Limoges / Dept Mathématiques et Informatique, **Rapporteur**
Marco Antonio Ramos Corchado, Professeur Chercheur titulaire niveau F de Université de l'état du Mexique, **Rapporteur**
Jean-Luc Koning, Professeur Vice-Président Relations Internationales Institut Polytechnique de Grenoble (Grenoble INP)
José Luis Leyva Montiel, Professeur Directeur du CINVESTAV campus Guadalajara
Luis Ernesto López Mellado, Professeur Chercheur titulaire CINVESTAV 3B
Juan Manuel Ramírez Arredondo Professeur Chercheur titulaire CINVESTAV 3C

Ecole doctorale : MITT
Unité de recherche : IRIT
Directeurs de Thèse : Veronique Gaildrat et Félix Francisco Ramos Corchado

CINVESTAV

Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Declarative Modeling Based on Knowledge

Doctor in Sciences Thesis in Electrical Engineering

Presented by:

M.C. Jaime Alberto Zaragoza Rios

to obtain the degree of:

Doctor in Sciences

in the specialty of:

Electrical Engineering

Guadalajara, Jalisco. Winter of 2010

CINVESTAV

Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Declarative Modeling Based on Knowledge

Doctor in Sciences Thesis in Electrical Engineering

Presented by:
M.C. Jaime Alberto Zaragoza Rios

to obtain the degree of:
Doctor in Sciences

in the specialty of:
Electrical Engineering

Thesis Advisor
Dr. Félix Francisco Ramos Corchado

Thesis Co-Advisor
Prof. Véronique Gaildrat

Guadalajara, Jalisco. January of 2010

Doctor in Sciences Thesis in Electrical Engineering

Presented by:

M.C. Jaime Alberto Zaragoza Rios

to obtain the degree of:

Doctor in Sciences

in the specialty of:

Electrical Engineering

Dr. Félix Francisco Ramos Corchado Thesis Advisor	Prof. Véronique Gaildrat Thesis Co-Advisor
Dr. José Luis Leyva Montiel Sinodal	Dr. Luis Ernesto Lópes Mellado Sinodal
Dr. Juan Manuel Ramirez Arredondo Sinodal	Prof. Jean-Luc Koning Sinodal
Dr. Marco Antonio Ramos Corchado Sinodal	Prof. George Miaoulis Sinodal

December 15, 2009

Acknowledgments

I would like to thank National Council on Science and Technology, CONACyT, for providing PhD Scholarship number 1910965. This research is also partially supported by CoECyT-Jal project No. 2008-05-97094.

I would also like to thank my thesis supervisors, Dr. Félix Francisco Ramos Corchando, and Prof. Véronique Gaildrat, for their valuable guidance in the completion of this research.

Also special thanks to my co-workers from the Distributed Systems group at CINVESTAV, as well as the people from the VORTEX Lab at IRIT, in Toulouse, France.

Dedicated to my parents, Jaime Zaragoza Infante and Margarita Rios Gonzáles.

Resumen

La tecnología moderna ha permitido la creación y representación de *Mundos Virtuales* y criaturas con un alto nivel de detalle, tal que vistos en películas, a veces es difícil distinguir cuales elementos son generados por computadora y cuales no. Así mismo, los juegos de vídeo han alcanzado un nivel cercano al realismo fotográfico.

Sin embargo, tal tecnología está en manos de habilidosos diseñadores, artistas y programadores, para los cuales toma de semanas a años para obtener esos resultados.

Modelado Declarativo es un método que permite crear modelos especificando tan solo algunas propiedades para los componentes del mismo. Aplicado a la cración de Mundos Virtuales, el modelado declarativo puede ser usado para construir el mundo virtual, estableciendo la disposición de los objetos, generando el contexto necesario para incluir animación y diseño de escena, así como generar las salidas usadas por un sistema de visualización/animación.

Este documento presenta una investigación enfocada a explorar el uso del modelado declarativo para crear *Ambientes Virtuales*, usando *Explotación del Conocimiento* como apoyo para el proceso y facilitar la transición del modelo de datos a una arquitectura subyacente, que toma la tarea de animar y evolucionar la escena.

Summary

Modern technology has allowed the creation and presentation of *Virtual Worlds* and creatures with such a high level of detail, that when used in films, sometimes is difficult to tell which elements are computer-generated and which not. Also, videogames had reached a level close to photographic realism.

However, such technology is in the hands of skillful designers, artists, and programmers, for whom it takes from weeks to years to complete these results.

Declarative modeling is a method which allows to create models specifying just a few properties for the model components. Applied to Virtual World creation, declarative modeling can be used to construct the Virtual World, establishing the layout for the objects, generating the necessary context to provide animation and scene design, and generating the outputs used by a visualization/animation system.

This document presents a research devoted to explore the use of declarative modeling for creating *Virtual Environments*, using *Knowledge Exploitation* to support the process and ease the transition from the data model to an underlaying architecture which takes the task of animating and evolving the scene.

Contents

1 Introduction **1**

1.1 Introduction 2

1.2 The Problem 2

1.3 Description of Problem 3

1.4 Research Objectives 5

2 State of the Art **7**

2.1 Technical Introduction 8

2.2 Declarative modeling 9

2.2.1 Description 10

2.2.2 Generation 11

2.2.3 Insight 13

2.3 Knowledge Management 13

2.3.1 Ontolingua 15

2.3.2 Protégé 16

2.3.3 Web Ontology Language 16

2.3.4 WebOnto 17

2.4 Constraint Satisfaction Problems 18

2.4.1 Backtracking 20

2.4.2 Backmarking 20

2.4.3 Backjumping 21

2.4.4 Backjumping based on graphics 21

2.4.5 Forward Checking 22
2.5 Virtual Worlds 22
2.6 Related Works 23
2.6.1 WordsEye: Automatic text-to-scene conversion system 25
2.6.2 DEM^{2}ONS: High Level Isometric Declarative Modeler for 3D Graphic Applications 25
2.6.3 Multiformes: Declarative Modeler as 3D sketch tool 26
2.6.4 CAPS: Constraint-based Automatic Placement System 26
2.6.5 ALICE 27

3 Proposal **28**
3.1 Interaction Language: A Review of VEDEL 29
3.2 Parsing Methodology 31
3.3 Modeler's Architecture 32
3.4 Creating the Model 34
3.4.1 Model Data Structure 34
3.4.2 Modeler procedure 35
3.4.3 Geometrical Validation 39
3.5 Generation of the Outputs 43
3.5.1 Model-View Controller 44
3.6 Modifying the Model 45

4 Research Outcome **46**
4.1 Virtual Environment Editor Prototypes 47
4.1.1 GeDA-3D Virtual Environment Editor Prototype 49
4.1.2 DRAMA Project Module DRAMAScène 53

5 Conclusion **56**
5.1 Conclusions 57
5.1.1 Future Work 60

A Published Works **62**

Bibliography 176

List of Figures

1.1 Project Overview. 4

2.1 Fields of research. 9
2.2 Interactive process of declarative modeling. 10
2.3 Characterization for an object 12
2.4 Ontology example. 18
2.5 Different levels of detail for avatars. 23
2.6 FL-System, City Engine and Instant Architecture 24

3.1 VEDEL examples. 31
3.2 Parsed Entry Structure 32
3.3 Modeler Architecture 33
3.4 Model Structure. 35
3.5 Collision Tags 40
3.6 Characteristic points 41
3.7 Validation volume for equation 1.b 42
3.8 Special case: against 43
3.9 Special case: inside 44

4.1 Virtual Environment Editor GUI 47
4.2 Previous Prototypes: Battle of the Frogs 48
4.3 Previous Prototypes: Earlier version of GeDA-3D 48
4.4 Example 1: Top-Down view 50

4.5 Example 1: General View 50
4.6 Example 2: House environment 51
4.7 Example 2: House environment 52
4.8 Example 3: Detail view 53
4.9 Example 3: Top-Down View 53
4.10 Examples of DRAMAScéne Concepts 55

5.1 GEDA-3D Architecture 60

Chapter 1

Introduction

Abstract

We present the objectives for this research, the motivation that leads us to perform research in the field of declarative modeling, the goals to be fulfilled, and the problems that must be solved in order to reach that objective. We also expose the results of our previous research.

1.1 Introduction

There has always been the need to represent ideas, in order to transmit, preserve, and make them available to others. Oral language was the first method to achieve these goals, followed by painting, and then writing. Any of these methods is enough when the ideas represented are easy to express. However, as these ideas become more and more complex, methods to represent them also become more and more specialized.

Fortunately, these methods can be implemented as tools. However, the complexity of tools useful to handle the representation of complex ideas needs a learning period, going from simply understanding the way to hold the tool properly, such as pencils, to different ways to achieve the desired results. Also, each tool can be composed of different materials, and applied in a number of ways, on different elements.

These tools have evolved through time, reaching rich versions that are implemented through computational systems, which allow a greater flexibility in their usage, even in ways that are not possible in the physical world using manual tools. In these ways, modern tools allow representing almost any kind of idea, from entertainment to education, and from visual aid to formal training. In computer science, *Virtual Reality* (VR) is a discipline which is useful for representing an actual or syntetic world, and can be perceived by the users in a variety of forms: text, sound, visuals, or even sensations.

1.2 The Problem

However, creating *Virtual Worlds* (VW) is a complex task carried out by a complete staff of modelers, programmers and artists. Completing a project can take from a few days to years. Those projects can go from custom presentations for small-business clients to big productions, like movies or video games. The tools needed to create these VW have different degrees of complexity, which forces the users to pass throug training, taking from a few hours to several weeks. In addition, some of these tools require specialized input hardware, that is in some cases costly and difficult to use correctly.

A final user who desires to use a VR may feel intimidated by the cost of having a staff to develop a complex application or discouraged by the learning step necessary to obtain satisfactory results for simple applications. Also, developers must have a certain degree of skill or talent to create results that approach the original idea. Thus, non-expert users can find difficult to create the VWs they intent to use and may prefer to leave the task to experienced creators or use other options. However, final users are the ones who really need VR technology.

To ease the taks of creating a VW we propose designing a tool which will use the necessary

procedures to create such VWs, using as input only a set of properties for the world defined by the user. The VW can later be extended to a *Virtual Environment* (VE), where the entities included in the VW perform actions, display emotions, and are subject to changes made by a set of rules established by the users.

To simplify the problem we divide the problem of using VR technology in several steps: Creating the VW, specifying the scene, that is, the actions to take place in the VE, and visualize them in the scene. We focus just on the first two of these subproblems.

The document is organized as follows:

- Chapter 1, **Introduction**, presents a general view of the research project, and introduces the motivation, goals, and solutions proposed.
- Chapter 2, **State of the Art**, describes previous and current studies on Declarative Modeling, geometric constraint solvers, and knowledge management, as well as some literature on the subject.
- Chapter 3, **Proposed Solution**, states our approach in detail, the methods proposed to solve the problem we are trying to solve, the research conducted to validate such approach, and the selection of the methods that better suit the objectives of our research.
- Chapter 4, **Research Outcome**, presents the outcome of this research, the prototypes developed.
- Chapter 5, **Conclusion**, raises some future objectives to be solved in future researches.

1.3 Description of Problem

This paper proposes, formalizes and implements a method creating VWs, using simple user inputs in the form of descriptions, with a formalization close to natural language, and exploiting knowledge to validate the statements of the input, with the objective of generating a model of VW, and finally presents it to the users by means of a 3D viewer, an underlaying architecture, or any desired platform.

For VW creation we understand the contruction of a simulated space, ruled by a set of laws (friction, gravity, elasticity, etc.) where several animated and unanimated entities can dwell and perform actions that may affect other entities and/or the environment.

Our approach uses *Declarative Modeling* (DM) to create a VW. This is a very powerful technique, allowing the user to describe the scene to be designed in an intuitive manner, by giving only some expected properties for the scenario, and letting the modeler find one or several solutions, if any, that satisfy these properties [1]. Thus in our case, the VW is

created using a natural language as a description provided by final the users. The difference between our approach and those proposed in related researches is that we propose the use of a *Knowledge Database* (KB) needed to validate the input and generate the output during the process.

The DM process is usually formed by three phases [2]:

- *Description*. Defines the interaction language.
- *Generation*. The modeler generates one or more models measured to the user's description.
- *Insight*. Users are presented with the models, then they can choose a solution.

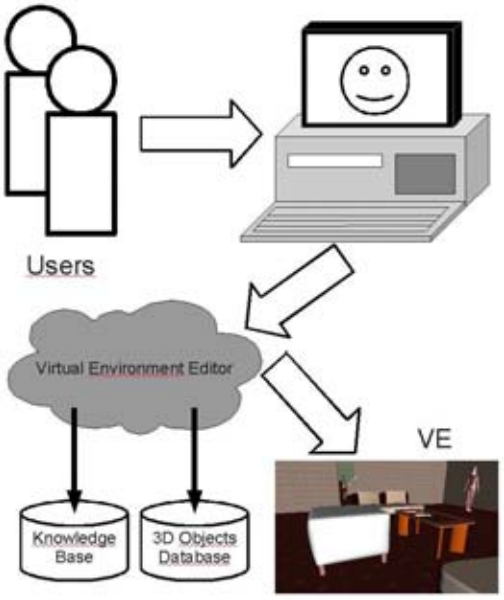

Figure 1.1: Project Overview.

In our previous research we focused on the *Description* phase of the DM technique. We defined a language centered in the creation of *Virtual Scenarios* (VS) and developed a tool that can analyze the descriptions written in such language. To present a visual outcome of the description, we used the rendering machine provided by the GeDA-3D [3] architecture,

which also hosts the tools for evolving the scene. In addition to the language, a KB was constructed, and used in the parser to process the terms in the description, and generates the appropriate outputs.

Our current research is focused on the two last steps of DM. This means first all the possible properties are unique to the environment and to the entities that will dwell must then be validated on its positioning. This will be conducted over the model generated in the previous step, where a tentative positioning is conducted, although it is not validated. The model can be processed using two tools: A *Constraint Solution Problem* (CSP) Solving Algorithm or a Geometric Constraint Solver. The second option involves the construction of a specialized tool or using a commercial product to solve possible model conflicts. The first method implies defining and structuring a method to solve a CSP.

While the latter option involves using widely tested tools, the cost and the necessary changes in the process where a great drawback. The former option allows better adaption of tools to our project, because the KB will also contains constraint informations for validating the spatial relationships between entities and the environment.

A lexical-syntactic parser, a model creator, an inference machine and an output generator form the *Virtual Environment Editor* (VEE), our VW constructor tool. The model creator is formed by a declarative modeler and a CSP solving algorithm, which validates the description and generates the possible models, and allows the modification of the same.

Finally for the context constructor, we need to consider the rest of the GeDA-3D architecture [3], which includes several modules on which the modeler depends, and that are also dependent on the modeler. The VW created through the scene editor contains all the information necessary for the architecture to make a correct representation of the VW.

1.4 Research Objectives

We consider several objectives to be reached during the development of this research. Some of these objectives are oriented to develop new declarative methods for VW generation, while others are focused on solving the problems for constructing our use case, which is a VEE necessary to validate our proposal. We list the objectives for this research next:

- Defining a method for integrating knowledge exploitation into DM.
- Integrating the use of knowledege in a CSP solver algoritm.
- Establishing the necessary information to be included in a KB, in order to create a model for a VW.

- Designing the architecture for a VEE, based on DM, which can receive an input based on a language specifically defined for VW description.
- Including in the VEE architecture the necessary means to access a KB.
- Adding methods in the VEE architecture to allow the generation of different types of outputs, in such a way that those outputs can be added, modified or removed without modifying the modeler itself.
- Implementing the proposed methods for DM and CSP solving into the architecture designed for the VEE.
- Including into the design of the VEE the necessary means to allow the creation of VW in a number of ways, from standard input text, to haptic input devices.
- Integrating the VEE with the rest of the GeDA-3D architecture.

The final result of this research is to propose a friendly, easy to use tool based on DM for final, non-experienced users (Figure 1.1). Thus, it must be possible to use the solution obtained through our method as an input of a 3D viewer, an underlying architecture, or any desired platform.

Chapter 2

State of the Art

Abstract

In this chapter we expose the methodologies used in our research. First, we present in detail the DM method. Next, we explore the different approaches for knowledge representation and exploitation. Later, different methods for solving Constraint Satisfaction Problem are presented. Finally, we take on some concepts for VR, and review current researches in the area.

2.1 Technical Introduction

Before we begin detailing the methodology used and the implementation procedure employed for creating the *Virtual Modeler* (VM), we need to explain some concepts for a better understanding of our project.

A **User Interface** (UI) is the aggregation of means employed by the users to interact with a system. It provides the methods for input, allowing the manipulation of the system, the output, or the presentation of the effects resulting of the users interaction.

The **Backus-Naur Form**, or BNP, is a formal way to describe formal languages. Consists of a context free grammar to define the syntax of a programming language by using two sets of rules: i.e., lexical rules and syntactic rules. The EBNF or Extended Backus-Naur Form is a metasyntax notation used to express context-free grammar, an extension of the basic Backus-Naur Form (BNF) metasyntax notation.

A **Token** is a block of text that can be categorized. This block of text can also be known as a lexeme. Through categorization, a lexical analyzer processes lexemes and provides meaning to them. This is known as tokenization. A token can have any kind of presentation, as long as it is a useful part of the structured text.

An **Application Programming Interface** , or API, is a set of standardized requesta. In essence, it provides the methods for accessing a program services. An API is formed by routines, data structures, object classes and/or protocols provided by libraries and/or operating system services.

A **Model-View Controller**, or MVC, is a paradigm where the user descripts a model of the external world, and the visual feedback to the users is explicitly separated and handled by three types of objects, each of them specialized for its task. The view manages the graphical and/or textual output, the controller interprets the inputs from the user, commanding the model and/or the view to change as appropriate. Finally, the model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change the state (usually from the controller).

Inference is a particular property from KB. It is the action of extrapolating new information from current knowledge, and is a useful characteristic for validating concepts and properties, since the users can begin stating simple characteristics, which can be combined to infer complex knowledge, extending the capabilities of the system which makes use of the KB.

The modeler was coded in the Java language, given its multi-platform capabilities and the need for using the Protégé OWL API, written in the same language. The Standard Development Kit selected was the last one available, Java SE 6, and the coding was conducted

under the Integrated Development Environment (IDE) Eclipse Ganymede.

The ontology was defined on the Protégé Framework. The version chosen was 3.2, since later releases have compatibility issues with ontologies created by previous versions.

2.2 Declarative modeling

Declarative modeling focuses on what users want, instead of the method used to obtain the result, or how to reach that solution. DM can be applied to a variety of problems, and has been used in several fields such as work flow systems [4] or biosystems [5]. It can be characterized as a multidisciplinary method, which involves several research fields, such as virtual reality, knowledge management or artificial intelligence (figure 2.1).

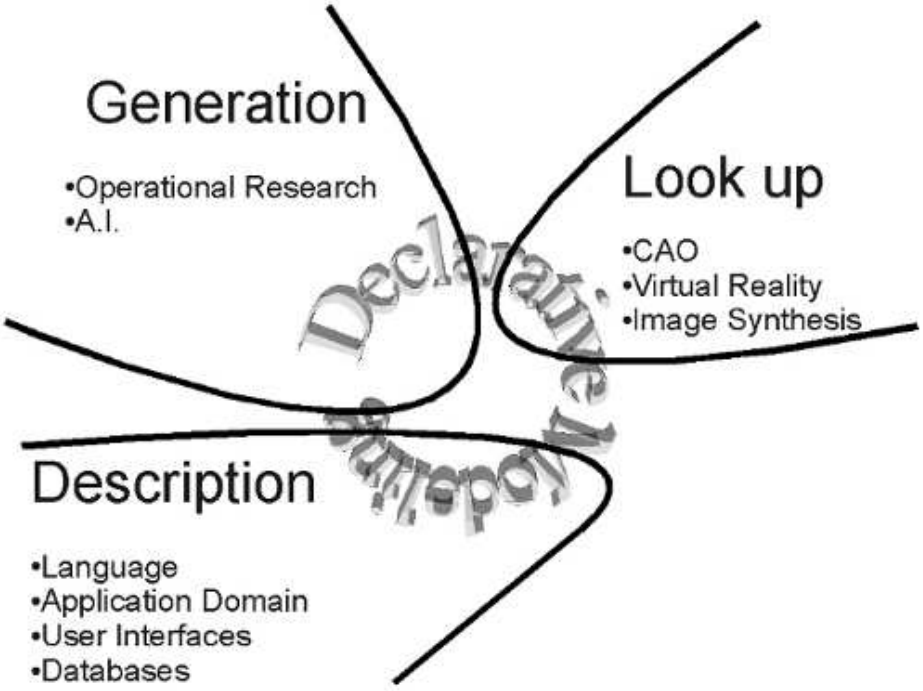

Figure 2.1: Fields of research.

Since our field of interest is the generation of VS using this method, we use the DM definition from Dimetri Plemenos et al[1]:

Definition 2.1 (**Declarative Modeling**)**.** A very powerful technique, allowing to describe the scenario to be designed in an intuitive manner, by only giving some expected properties of the scene, and letting the modeler find solutions, if any, verifying these properties.

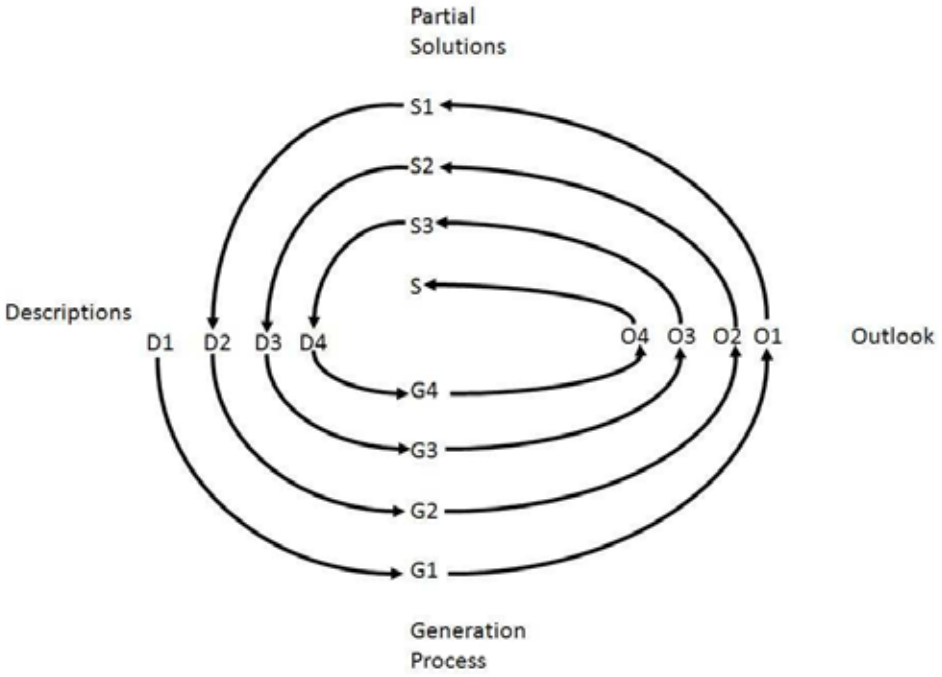

Figure 2.2: Interactive process of declarative modeling.

Following this definition, we aim to provide a system allowing users to create a VW data model, which can then be used by an underlaying architecture to execute a simulation of a scene. We can divide the DM process into three steps:

2.2.1 Description

During this step, the properties and relationships between entities are provided. There are several interaction modalities, such as scripting, gestural or language interaction, as well as multimodal options. We take the language interaction approach, since it is a direct method, and less intimidating than gestural methods (either mouse or haptic inputs), since people learns to express simple ideas from the moment they lern to write. Alos, if it is handled properly, can be close to natural language, allowing an easy interaction between the modeler and the user. An important part of the description step is the semantic knowledge management, with the objective of avoid stating all the concepts explicitly by the user. Any ambiguity must be solved, using the specificities given in the description to fill the gaps in the model (such as placing books on a bookshelf, or orienting the audience in a theater toward the scenario) and obtain the context for the VW. This step defines the interaction language and UI, so the approach must be carefully selected and specified. We present our interaction language and UI in chapter 3.

2.2.2 Generation

Generation is the search of a consistent solution, through the analysis and evaluation of the properties stated. The properties are interpreted and solved according to a set of constraints and are used for obtaining all possible values for the variables in the problem, and exploring the solution space in order to find solutions matching with the user's requests. These solutions can be found by defining and solving a CSP. Several methods have been used to solve CSP, such as search trees [6] or specific procedural approaches. We focus on constraint satisfaction, where methods such Space-CSP [7], Numeric-CSP [8] or Metaheuristics ([9], [10]) have been used. The efficiency of these methods depends on adequacy of the solving method, the representation of constraints, the complexity of the search space, and the application domain. To choose a method we consider four criteria: memory space, accuracy of the representation, efficiency, and simplicity of implementation. We also consider that complete methods are best suited to scenarios with few objects, while metaheuristic methods perform best with numerous elements. Metaheuristics are not able to certify the optimality of the solutions they find, while complete procedures have often proved incapable of finding solutions whose quality is close to that obtained by the leading meta-heuristics particularly for real world problems, which often attain notably high levels of complexity [9].

Objects must be characterized in order to be represented by the constraint set (figure 2.3). In this research, the representation of the objects must include its position, orientation and size, as well as relative and spatial relationship with other objects. The search space model can be represented in several ways:

- Explicit, storing all the possible values. However, the increasing amount of necessary memory is evident.
- Semi-explicit, associating an explicit representation with every variable, but using a projection method.
- Implicit, where the description contains the representation. Memory space is constant, however, it requires a test of consistency.

The constraints can be determined by properties or contextual data, implying that the properties must correspond to constraints. Constraints can also be represented in three levels, depending on the complexity, number of variables, and generation approach: Implicit (constraint is too complex), projective (using projection methods), and explicit (if constraint is simple enough). The model is created using iterative methods, refining the solution and satisfying the constraints at each interaction. First, the values for properties of the entities are solved, then, these values are validated against the rest of the model. If any constraint is violated or not satisfied, the model must be modified. Thus, the process repeats until finding a solution or a stop condition is reached.

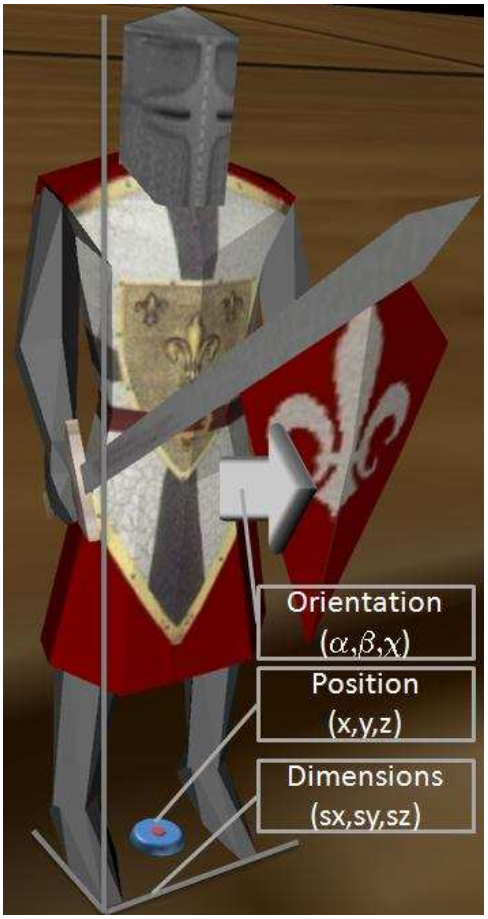

Figure 2.3: Characterization for an object

Properties are used for three main tasks: defining the layout of the objects, partitioning the space and build complex objects. We can also classify them in five sets:

- Basic, when they are used to set the exact object characteristics,
- Fuzzy, which allows partial, imprecise and negative descriptions,
- General, mainly concerning morphological features, positions, numbering, and appearance,
- Specific, assigned to predefined shape models, and
- Spatial, used to define the relative or absolute position of objects in the scenario.

2.2.3 Insight

This allows the users to view all or part of the produced results. It can work using two methods: presenting to the users the solutions, or just presenting the most balanced solution. In both cases the users can decide to modify the solution and adapt it, so it comes closer to the mental image for the VW. There are several methods for the look up, such as freeze or comment [6], classification of solutions [2], presentation tools [11], navigation tools [12] or incremental refinement of the description [13].

It is important to consider two aspects in the design of a DM: first, the interpretation of the properties, where we should consider the translation of the properties into the constraints set. If this is not well defined, the result will be different from the users' requests. A correct interpretation must consider the *normality* of the context, this is, the normal usage for the object, its intrinsic and deictic orientations, and the notion of a *pivot* element, which works as an orientation beacon to all the objects in the scenario.

The second aspect is the look up step, where the selection of the "good" solutions must be carefully directed. The modeler can generate all the solutions, but this would imply the exploration of the whole solution space. It can select a set of representative solutions. Or it can present only one, as long as all constraints are valid in the solution. Also, the modeler should allow the users to modify interactively the solutions, adding new properties as the objects are manipulated.

2.3 Knowledge Management

Knowledge Databases can be used in a variety of areas, from medicine [14] to genetics [15]], and for diverse tasks such as determine non-redundant information system architectures, support information management, enable shared understanding and communication between different entities, or facilitate the inter-interoperability of diverse systems [16] [17].

As we stated in the previous section, an important part during the description step in DM is the semantic management. In other words, how the translation between the properties stated in the description of the modeler data structure is handled. There are many ways to define an entity or a concept, from its physical appearance to the list of its attributes, elements or parts. For example, a virtual human being can be semantically represented as a complete unit, it can be described as the conjunction of several elements (head, torso, arms, legs), or can be defined as a set of philosophical statements (self-aware, conscious, intelligent).

Many methods have been proposed to represent knowledge, such as logic, semantic networks or rules [18]. Those methods are used in formal representation such as ontology, description logic or logic schemes, and in diverse applications such as expert systems or workflow applications [19]. To agree with the concept of knowledge management, we take

the description from Alavi et all [19]:

Definition 2.2 (**Knowledge**)**.** Knowledge is information contained in the mind of individuals (which may or may not be new, unique, useful, or accurate) related to facts, procedures, concepts, interpretations, ideas, observations, and judgments.

From theses concepts, we define the knowledge needed to define an entity as a collection of data, organized in a way that can be related with each other, and that has been formatted in a way that can be modified, corrected and eliminated as new information appears. This is in an analogous way to a Data Base, where the information is stored with a specific method, formated within a layout, and presented according to the users' request. Since knowledge can be casted by many representations, we represent knowledge as a KB, starting from the analogy between knowledge for an entity and a Data Base.

A specific type of KB is called *Ontology*. From a philosophical point of view, ontology is the explicit specification of a conceptualization: a simple and abstract view of the world intended to be represented, with the objective to achieve some goal. For systems based on knowledge, what "*exists*" is exactly what can be represented.

Definition 2.3 (**Speech Universe**)**.** When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called Speech Universe. The set of objects, and the relationships that can be described between them, is reflected in the representation vocabulary used by a knowledge-based program to represent that knowledge [20].

A basic ontology of the real world is the relation existing among all existing things, classified according on how they exist. That is, the way in which something has reached the reality and the way it actually exists [21].

The systems and services based on knowledge are expensive to build, test and maintain. A software methodology based on formal specification of shared resources, re-usable components and complementary services is required. The specifications of shared vocabularies have an important place in the role of such methodology, because different applications require different reasoning services, as well as special purpose language to support them.

Therefore, there are several challenges to overcome for the development of a knowledge-based, shared and reusable software. As conventional applications, knowledge-based systems are based on heterogeneous hardware platforms, programming languages and network protocols. However, knowledge based systems have certain special requirements for interoperability. Such systems operate and communicate using sentences in a formal language. They make requests and send answers, taking "*previous knowledge*" as input. As agents in an AI distributed system, they negotiate and interchange knowledge. Communication at the knowledge level needs conventions at the three levels: representation language format, agents communication protocol, and content specification of the shared knowledge [20].

Since our project deals with the creation of a scenario using only a description written in a natural-like language, it is necessary to design the way to achieve a correct analysis of the sentences, as well as to keep coherence in the scenario. An example of this could be the sentence "The whale is in the living room". If we are not talking about a toy, logically, a whale cannot live a living room; it is too big and is a sea mammal. This can be derived from the ontology and the sentence can be marked as invalid. If the sentence is changed to "The toy whale is in the living room", the analysis must result correct, since a toy shaped like a whale can be in a living room. As before, this conclusion can be derived with the help of the ontology.

The Ontology will help us to conduct the semantic analysis of the language, since this analysis can be reasoned using the knowledge extracted from the ontology. In the previous example, for the word "whale", the ontology will return information that states that a whale is a sea animal, a mammal, and that it is several meters long and is heavy. From there, it can be reasoned that it is not correct for a whale to be placed in a living room. In the second example, the properties for a toy allow it to exist in a living room, and in the shape of a toy whale. Thus, we will use the ontology to exploit knowledge and assure the semantic coherence of the sentences in the description used for generating the VW.

Many applications and standards have been developed to create, modify and access knowledge in knowledge based-form. We present some of these works, completely oriented to ontology building, in the following subsections:

2.3.1 Ontolingua

The system developed at the KSL of the Stanford University [22] consists of a server and a representation language. The server provides an ontology repository, allowing the creation of ontology and its modification. The ontologies in the repository can be joined or included in a new ontology. To interact with the server the user can use any standard web browser. The server was designed for allowing the cooperation in ontology creation, easy generation of new ontologies by including (parts of) existing ontologies from a repository, and the possibility of including primitives from an ontology frame. The ontologies stored in the server can be converted to different formats to be used in other applications. This allows the use of the Ontolingua server for creating a new ontology, export it, and then use it in CLIPS-based application. It is also possible to import definitions from an ontology created on different languages to the Ontolingua language. The Ontolingua server can be accessed by other programs if they can use the ontologies stored in the Ontolingua representation language [20].

2.3.2 Protégé

Protégé is a multi-platform package [23], designed to build domain model ontologies, and has been developed by the Informatics Medic Section of Stanford. It is oriented towards assisting software developers in the creation and support of explicit domain models, and in incorporating those models directly in software code. The Protégé methodology allows the system designer to develop software from modular components, including reusable work frames helping to build domain models and independent problem solving methods that implement procedural strategies to solve tasks [24]. The Protégé framework includes three main sections: a. the *Ontology Editor'*, used to develop the domain ontology by expanding a hierarchical structure and including classes, and concrete or abstract slots; b. Based on the constructed ontology, Protégé is capable of generating a *Knowledge Acquisition tool* to input ontology instances. The KA tool can be adapted to the users' needs by using the "Layout" editor; c. The last part of the program is the *Layout interpreter*, which reads the output of the layout editor and shows the user an input screen with a few buttons. These buttons can be used to create the instances for the classes and sub-classes. The whole tool is graphical, which is friendly for non-experienced user.

2.3.3 Web Ontology Language

The Web Ontology Language is a semantic markup language designed for publishing and sharing ontologies over the World Wide Web. It is a standard [25] that forms part of the W3C Recommendations for the Semantic Web, and was developed by Deborah L. McGuinness and Frank van Harmelen, as a vocabulary extension of RDF (the Resource Description Framework) [26]. It is a language oriented to process information context for applications that need more than just representing information for the users. It provides greater interoperability for web content than similar standards (XML, RDF or RDF-S), due to additional vocabulary and formal semantics. It can be divided in three sub-languages, each more expressive than the previous one: OWL Lite, OWL DL, and OWL Full.

- OWL Lite supports primary needs for classification hierarchy and simple constraints.
- OWL DL provides maximum expressiveness, while retaining computational completeness.
- OWL Full presents the syntactic freedom of RDF and the maximum expressiveness, but does not provide computational guarantees.

OWL Lite uses only some of the OWL language features has more limitations than the others sub-languages. It allows restrictions on the use of properties by instances of a class,

a limited form of cardinality restrictions, a limited intersection constructor, and the RDF mechanism for data values.

OWL DL and OWL Full use the same vocabulary, but OWL DL presents some restrictions. OWL DL requires type separation, i.e., a class can not be an individual or a property and can not be applied to the language elements of OWL itself.

An ontology written with OWL consists of three sets: classes, properties, and individuals. Each element in those classes can hold a superclass-subclass relationship. Classes define the archetype for the concepts in the ontology, and contain a set of restrictions, which are the constrains for the individuals, which are the instantiations of the classes.

The properties are related to the classes, and can be classified in two types: data type or object, and both need a domain, the classes containing the property, and a range, that is, the subset of values allowed for that property. Data type properties are basic level values, such as strings, integer or boolean values. Object properties indicate the relationship between classes, and contain the list of related individuals.

Finally, individuals are instantiations of the classes, and contain the actual values for the specific element in the ontological domain.

In figure 2.4 we present an example of an ontology created with OWL, in a graphical form.

2.3.4 WebOnto

WebOnto [27] is a platform completely accessible from the Internet. It was developed by the Knowledge Media Institute at the Open University and designed to support creation, navigation and collaborative edition of ontologies. In particular, WebOnto was designed to provide an interface allowing direct manipulation and that presents ontological expression using a powerful medium. WebOnto was designed to complement the ontology discussion tool Tadzebao. Thus, it is mainly a graphic tool oriented to construct ontologies. The language used to model the ontologies in WebOnto is OCLM (Operational Conceptual Modeling Language), originally developed in the context of the VITAL project to provide modeling operational capabilities to the work system VITAL [28]. This tool proposes a number of useful characteristics, like saving structure diagrams, relationships views, classes, rules, and so on, all of them individually. Other characteristics include cooperative working in ontologies by drawing, and using broadcast and function reception.

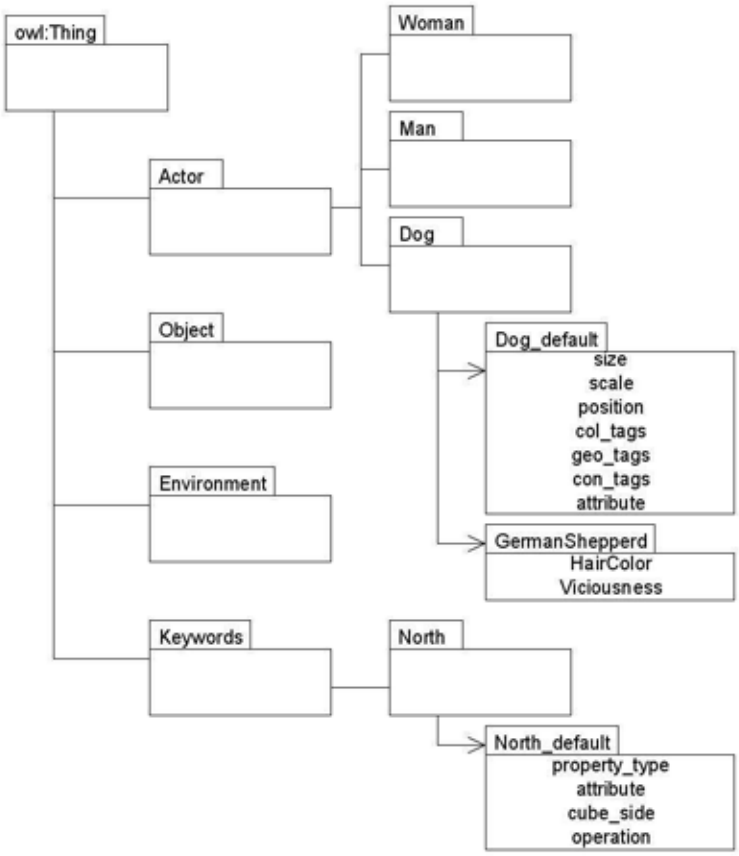

Figure 2.4: Ontology example.

2.4 Constraint Satisfaction Problems

During the generation phase, verifying the location of the different objects in the scenario is an important part of the model creation. The properties provided by the user for the VW are transformed into a set of constraints, which must be verified and transformed to assure the correct configuration of any solution. To solve any conflict between constraints, we employ an algorithm capable of finding a configuration in the solution space for finding a solution satisfying all constraints. An algorithm with those properties is known as Constraint Satisfaction Problem (CSP), and is defined as follows [29]:

Definition 2.4 (**CSP Definition**). A tuple $< X, D, C >$, where:
$X = \{X_0 \dots X_n\}$, is the set of variables for the problem.
D is a domain for each variable X_i.
$C = \{C_0 \dots C_n\}$, the constraints set, where each C_i specifies a subset of X_i and the acceptable

values for that subset.

A variable X_i is considered *instanced* when it has been assigned a value from its domain D_i. Those variables which have not been assigned with any value are called *non instanced*. We also can refer to both variables as *past* variables and *future* variables, respectively. The notation "$X_i = x_j$" means that the variable X_i is assigned with the value x_j. The act of the instantiation is denoted by "$X_i \leftarrow x_j$". The variables in a CPS are instantiated in a given order, denoted by "$\vec{X_i}$".

A variable is in a *dead-end* state, if there is no value in its domain consistent with $\vec{X}_{i-1}$. There are two kinds of dead-ends: *leave*, if there are constraints that forbid each value in its domain, and *interior*, if there are some values making the domain compatible with $\vec{X}_{i-1}$, but the subtree with its root in the variable does not have a solution.

A problem state is the *assignment* of values to one or all of the variables, from the domain value sets of each variable, $\{X_i = v_i, X_j = v_j \ldots\}$. If an assignment does not violate any restriction, is called *consistent* or legal. A solution to a CSP is a value assignment to all variables in such a way that none of the constraints is violated. A problem which presents a solution is considered satisfiable or consistent, otherwise is called *unsatisfiable* or *inconsistent*.

To solve a CSP, there can be used two approximations: search and deduction. Both are based on the idea "*divide and conquer*", that is, transforming a complex problem into a simpler one. Search generally consists of selecting an action to develop, maybe with the aid of an heuristic, which will take us to the closest state for the intended objective. Tracking is an example of searching for CSP. The operation is applied to the value assignation of one or more variables. If a variable can no longer be assigned in such a way that can keep the consistency for the solution, it reaches a dead-end, and tracking is executed.

It is a good idea to visualize a CSP as a *constraint graph*, the nodes corresponding to the problem variables and the arcs to restrictions. Dealing with a CSP allows to generalize the successor function and goal test for adapting to any CSP, as well as to apply efficient and generic heuristics to avoid including additional information or domain expertise. Also, the graph structure can be used to simplify the solving process.

A CSP can be designed following an incremental formulation, as in standard search problems, starting with an empty assignment, this is, no variables assigned yet. A successor function will assign values to variables as long as there are conflicts with previously assigned variables. A test function is designed to find a complete variable assignment, and a constant step cost.

The simplest case of CSP implies discrete variables and finite domains, for example, the Boolean CSP, formed by variables that can be either true or false. For a maximum domain size of d in any CSP, the possible number of complete assignations is $O(d^n)$, where n is the number of variables, in the worst case. If the domain for the discrete variables being handled

by the CSP is infinite, it is not possible to describe restrictions by enumeration of the possible values combinations, but a restriction language can represent it. In the case of continuous domains, the most common in the real world, we find that the most studied cases are linear programming problems, which are solved in polynomial time.

The constraints can be unary, when the constrained values affect only one variable; binary, when the constraint involves two variables; or higher order, implying three or more variables. Constraints can also be absolute, meaning that the violation of any of them excludes a potential solution. Some CSP include preference constraints, which indicate what kind of solution is preferred. Several methods have been proposed to solve CSP; next, we present some of them:

2.4.1 Backtracking

The simplest algorithm for solving CSP is backtracking [30]. This algorithm starts with an empty set of consistently assigned variables, and tries to extend it by adding new variables and values for them. If the inclusion of a new variable does not violate any constraint, the process is repeated until all variables are included. If the newly added variable makes the solution inconsistent, the last variable added is instantiated with a new value. If there are no more possible values for that variable, it is removed from the set and the algorithm starts the backtracking again.

An important element of the backtracking algorithm is the review of consistencies; which is conducted frequently, and makes an important part of the algorithm's work. The time used to conduct this revision is in agreement to the representation of the constraints. Those representations can be a list of the allowed tuples to maintain the constraints free and keep only the incompatible tuples, making use of a Boolean values table, or executing a procedure.

2.4.2 Backmarking

This method reduces the cost of consistency checking while backtracking is conducted [29]. It requires that the consistency review be executed in the same order, as the variables were instantiated. Proceeding in this way, the algorithm avoids previously tested and later rejected combinations, increasing the efficiency. However, this approach is restricted to binary CPSs and static variable ordering.

This method requires two additional tables, $M_{i,v}$ used to register the first variable with a failed consistency check $X_i = x_v$. If $X_i = x_v$ is consistent with the previous variables, $M_{i,v} = i$. L_i records the first variable that has changed its value since $M_{i,v}$ was assigned for X_i with any value v from its domain. If $M_{i,v} < L_i$, the variable pointed by $M_{i,v}$ has not changed and $X_i = x_v$ will fail when being checked against $X_{M_{v,i}}$, therefore the consistency

check is not required and x_v can be rejected. If $M_{i,v} > L_i$, $X_i = x_v$ is consistent with all the variables before X_{L_i} and those consistency checks can be avoided.

2.4.3 Backjumping

This is proposed as a way for reducing the amount of *trashing*, which is the act of finding the same dead-end several times [31]. This algorithm is capable of "jumping " from a dead-end to a previous variable that causes the dead-end with its current instantiation. A variable causes a dead-end, if in conjunction with zero or more variables preceding it in the ordering are instantiated in such way that a restriction will not allow the assignment of values to the variable in conflict.

An array J_i, $i \leq 1 \leq n$ is used to find variables that cause dead-ends. J_i stores the last variable test for consistency with some value from X_i. If X_i does not get into a dead-end, then $J_i = i-1$. If X_i is inconsistent, then each value in D_i was tested for consistency with the past variables until an instantiation failed to pass test, J_i contains the index of the variable inconsistent for some value in D_i. Is important that the consistency review of instantiated variables be in the same order as the instantiation. If X_i is in a dead-end, we can guarantee that conducting the tracking between X_{J_i+1} and X_{i-1} will be unprofitable, since the cause of the dead-end in X_i is not marked. The partial instantiation $\vec{X_{J_i}}$ will cause that any value for X_i generates a dead-end on some restriction, so modifying the values after X_{J_i} will not solve the dead-end.

2.4.4 Backjumping based on graphics

This method is another variation of backtracking. It jumps over variables as a response to a dead-end [32]. Unlike backjumping, which only responds to leave dead-ends. Backjumping based on graphics can respond to previous dead-ends. In order to accomplish this, it reviews the *set of parents* P_i for the dead-end variable X_i, where a parent of X_i is any variable connected to X_i through the constraint graph and precedes X_i in the instantiation order.

If X_i is a dead-end variable, the algorithm jumps to the last variable in the set of parents. If X_i is in a previous dead-end, the new set is formed by the union of the parent set of X_i and those from the dead-end variable found after X_i in the search tree. The algorithm then jumps to the last variable of the inducted set J_i. The algorithm requires to update J_i after every unsuccessful inconsistency review, requiring up to $O(n^2)$ space and $O(ec)$ time, where c is the number of constraints and e is the maximum number of variables for each constraint. The other disadvantage of using the parents is using less refined data of the causes of the dead-end.

2.4.5 Forward Checking

Forward Checking is a variation of backtracking, which acts by instantiating a variable and removing any conflicting value in the domains of future variables. This algorithm rejects any value that can lead to the removal of the last value in the domain of future variables. The values are not removed permanently, but stored in the set D', which contains the narrowed domains. The action of removing values from D' is called filtering [33].

The algorithm works filling D' with all the compatible values from each domain for each variable, and continues looking for at least one compatible value for D'_{cur} with future variables. It is not necessary to review previous variables, since D'_{cur} only contains compatible values with $\vec{X}_{cur-1}$.

Most methods make decisions on how the variables have been instantiated, and then modify values to continue searching for solutions. We know part of the solution space at the beginning of the search, having access to information of possible positions, relations that can be or not be changed, as well as ranges of correct values for the variables. Directed methods such as backjumping based on graphics or forward checking suit our needs better, since the inclusion of Metaheuristics based on knowledge can lead to quicker solution finding, dead-end solving, and corrections in the search direction.

2.5 Virtual Worlds

One of the best definitions for VR we have found is "Virtual Reality is a way for humans to visualize, manipulate and interact with computers with extremely complex data" [34]. Visualization means that the user can perceive the outputs generated by the computer. That is to say, the actions achieved in the world represented inside the computer. The perception can be visual, auditory, sensory or a combination of these. The world being represented can be a CAD model, a scientific simulation, or the view of a database. Interaction means that a world can evolve autonomously, either the objects inside the world or the properties of the world itself. This interaction triggers the evolution (animation), through some process, of either physic simulations or simple animation scripts [35].

In a VW we found entities that dwell inside it, which are named commonly *avatars*. Avatars represent animated entities having complex behavior, for instance animals (persons or other type), or other imaginary animated creatures. The avatars can perform a variety of actions, according to the world they have been put in, as well as represent human emotions ([36],[37]) and perform varied behaviors [38]. Those avatars have different levels of complexity and detail, depending on the overall representation of the world-taking place. We can take as an example a person: in simple simulations, it is not needed great details of the body, for instance the skin, the hair, the face details are represented by very simple models (Figure

2.5). In contrast for movies, a virtual character must have a high level of detail (a detailed face, modeling individual hairs). The level of detail comes at computational cost. Currently several methods are used to provide better levels of representation in real time, such as ray tracing or bump-mapping [39].

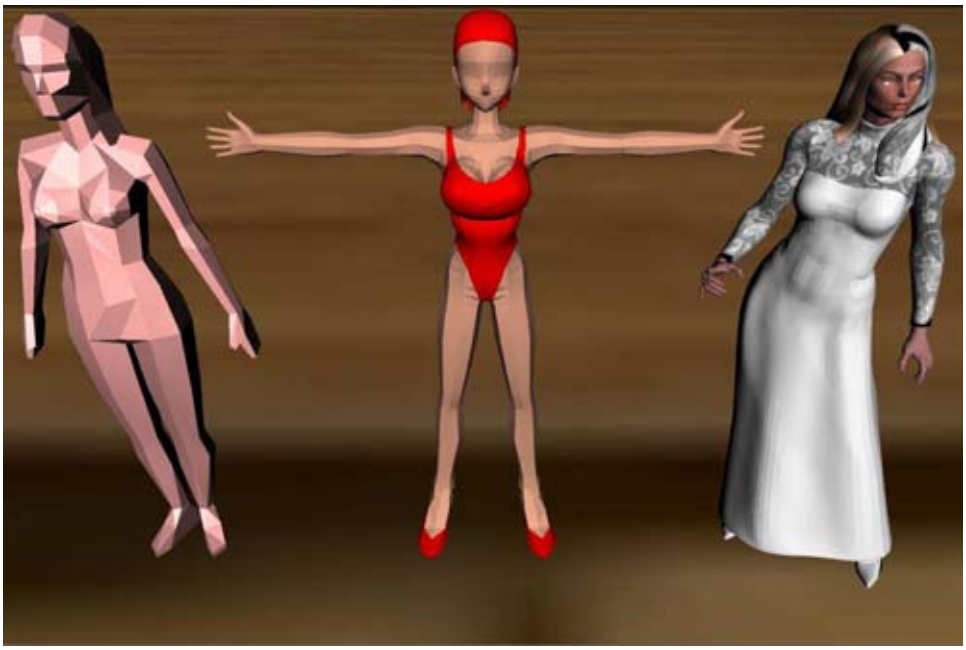

Figure 2.5: Different levels of detail for avatars.

Among some of the applications for VW we can find: the creation of VW for video games, where the player can travel through different environmental settings, interact with diverse characters and perform a set of actions. In movies, VW are used to re-create ancient worlds, fantasy settings or impossible situations. VW are also used in applications such as architecture [40] or city-planning [41][42], as well as for applications such as story telling [43], as a support method in surgery and medical education [44] or as educative tools.

2.6 Related Works

Several works have exploited the methodology of DM. Before commencing the review of some of them, it is necessary to state that since we are focusing on using DM to create VWs, all the works presented in this document are also oriented towards the creation of virtual 3D models. In these works one of the problems solved is validating the disposition of the objects that conform the VW using diverse techniques, whereas other modelers focus on validating other aspects of the model, such as congruency or efficiency (Repast Symphony System [45],

Joseph System [46]). Of the two tasks, the most demanding in computing processing is validating the correct positioning and orientation.

Some of the reviewed works focus on architectural or urban design, such as FL-System [40]. This system is focused on the generation of complex city models, parting from a specialized grammar, and using a variant of the Lindenmayer System [47], called Functional L-System, replacing the generation of terminal symbols by generic objects. It uses VRML97 to visualize the models, and works generating individual buildings and then incrementally working the rest of the city block, and later the entire city.

CityEngine [41] is a system capable of generating a complete city model, using small sets of statistical and geographical data, contained in geographical maps (elevation, land, water maps) and socio-statistical maps (population maps, zoning maps). It works creating a first layer of roads, using L-Systems, and then creating city blocks. The final step is the generation of geometry and visualization, using first a real time render, and then a raytracer.

Wonka et al. in [48] presents a method for automatic architecture modeling, uses a spatial attribute design grammar, or split grammar as input for the user. The input is used to create a 3D layout which is the base for the building in creation. The facade is created next, splitting it into structural elements at the level of individual parts (windows, cornices, etc.). The resulting model is shown to users through a real-time render.

Figure 2.6: FL-System, City Engine and Instant Architecture

None of the previous presented works is oriented toward creating a model of a VW, but just a 3D model of a description, composed in all the cases by a specific grammar, either raw geographical data, architectural designs or three-dimensional design data. These projects do not allow any kind of interaction with the environment, due to completely automated model generation. In those works, once the VW is generated, it is not possible to interact it beyond positioning the camera. If any modification is needed, it is mandatory to modify the description in order to change the output of the system. Other works are oriented to create VS, where different elements are positioned in the VW, and their properties can be modified. In the next subsections we will review some of them.

2.6.1 WordsEye: Automatic text-to-scene conversion system

Bob Coyne and Richard Asproad at the AT&T laboratories developed WordsEye [49]. This system allows the generation of a 3D scenario from a description written on natural language, for instance: "the bird is in the birdcage. The birdcage is on the chair". The text is initially marked and analyzed using part-of-speech taggers and statistical analyzers. The output of this process is an analysis tree, which represents the structure of the sentence. Next, a depictor (low level graphic representation) is assigned to each semantic element. Those depictors are modified to match with the poses and actions described in the text, through inverse kinematics. After that, the implicit and conflicted constraints of the depictors are solved. Each depictor is then applied, while keeping its constraints, to incrementally build the scene. The final step includes adding the background environment, the terrain plane, the lights, and the camera. Then, the scene is rendered and presented to users. If the text includes some abstractions or descriptions that does not contain physical properties or relations, the system employs several techniques, like textualization, emblematization, characterization, lateralization, or personification. This system accomplishes the text-to-scene conversion by using statistical methods and constraints solvers, and also has a variety of techniques to represent certain expressions. However, the scenes are presented in static form, and the user has no interaction with the representation.

2.6.2 DEM^{2}ONS: High Level Isometric Declarative Modeler for 3D Graphic Applications

DEM^{2}ONS_98 has been designed by Ghassan Kwaiter et all [50] offering to the users the possibility to easily construct 3D scenarios in natural way and with a high level of abstraction. Two parts constitute it: a multi-modal interface and, the 3D scene modeler. The multi-modal interface allows the users to communicate with the system. It uses simultaneously several combined methods provided by different input modules (data globes, speech recognition systems, spaceball, mouse). The syntactic analysis and Dedicated Interface modules analyze and control the low-level events to transform them in normalized events. DEM^{2}ONS uses ORANOS as 3D scene modeler, a constraint solver designed with several characteristics allowing the expansion of DM applications, like generality, breakup prevention and dynamic constraint solving. The GUI (Graphic User Interface) is based upon the Open Inventor Toolkit [51] and Motif library [52]. These two modules render the objects, and provide support to present the menus and tabs. DEM^{2}ONS allows the user to interact with the objects in the scene. Also, it solves any constraint problem, but only allows static objects, with no avatar support.

2.6.3 Multiformes: Declarative Modeler as 3D sketch tool

William Ruchaud et al. presents Multiformes [53], a general purpose DM, specially designed for 3D scenario sketches. As any DM, the work over the scenario with MultiFormes is handled essentially through a description (the way the user inputs all of the scenario geometric characteristics of the elements, and the relationships between them). The most important feature in MultiFormes is the ability to automatically explore all the possible variations in a scenario. Unlike most of the existent sketch systems, Multiformes does not present only one interpretation of each imprecise property. Starting with a single description, the designer can obtain several variations of the same scenario as a result. This can lead the users to choose a variation not considered previously. A constraint solver supports this process. The description of the scenario includes two sets: *the geometric objects* set presents in the scenario, and *the set of existent relationships between the geometric sets*. To allow the progressive refinement of the scenario, MultiFormes uses hierarchical approximations for the scenario modeling. Following this approach, a scenario can be incrementally described at different levels of detail. Thanks to its constraint solver, MultiFormes is capable of exploring diverse variations of sketch, satisfying the same description.

The geometric restriction solver is the core of the system and is used to create a hierarchically decomposed scenario. Even when this system obtains its solutions in incremental ways, and is capable of solving the constraints requested by the user, the system requires the list of actions needed to construct the scenario. This requirement makes the use of the system restrictive.

2.6.4 CAPS: Constraint-based Automatic Placement System

Ken Xu, et al. presents CAPS [54], that is a positioning system based on restrictions. It makes possible modeling big and complex scenarios, using a set of intuitive positioning restrictions that allow manipulation of several objects simultaneously. It also employs semantic techniques for the positioning of the objects, using concepts such as fragility, usability or interaction between the objects. The system uses pseudo-physics to assure that the positioning is physically stable. CAPS uses input methods with high levels of freedom, such as Space Ball or Data Glove. The positioning of the object is executed one at the time. Allows direct interaction with the objects, keeping the relationships between them by means of pseudo-physics or grouping. These methods and the tools integrated into this system make it a design tool, mainly oriented towards scenario visualization, with no capabilities for self-evolution.

2.6.5 ALICE

This is a tool for describing the time-based and interactive behavior of 3D objects, developed at the Carnegie Mellon University [55]. Described by its authors as "a 3D interactive, animation, programming environment for building VW, designed for novices", provides a creation environment where users can create, use, and animate 3D objects to generate animations. The user selects an object from a 3D database and then arranges its position in the world. After the object has been positioned, the user can select primitive methods, which send messages to the object. These methods are arranged to form program sentences which are interpreted by a Python Interpreter, which works as a scripting service. The system uses Microsoft 3D Retained Mode (D3DRM) to render the scene. Users can add new content and methods, since the system supports many 3D modeling formats. This project focuses on educational/instructional areas, but still focusing on programming paradigms. The users need to actively create the scenario, and specify the scene using a programming-like tool. Also, the software does not make any context verification.

Chapter 3

Proposal

Abstract

This section is devoted to explain the approach taken to solve problems posed by the inclusion of knowledge in DM. Also, a detailed explanation of the procedure and the methodology is presented.

3.1 Interaction Language: A Review of VEDEL

To allow interaction between the modeler and the user, an interface that allows easy specification of the desired properties for the model intended is necessary. There are different input methods, but we choose to let the user express himself in a natural-like language the characteristics of the VW.

Our objective in defining a declarative method to describe a VW was to provide users a structured, easy to follow method to compose the description of a scenario, in the form of a declarative scripting language, which we called Virtual Environment Description Language, or VEDEL [56] that is a tag language. A description in VEDEL is composed by three sections, each of them devoted to describe one of the three possible types of elements in the VE: the **Environment**, that is, the general settings for the scenario; **Actors or avatars**, which are those entities capable of perform and react to actions, display emotions, represent behaviors, and react to the changes in the environment and other entities; and **Objects**, entities that can be subjected to actions, but cannot act on their own. Each section is delimited by *section tag*, which starts with a keyword for each section (*ENV*, *ACT*, or *OBJ*) respectively enclosed in brackets ([ ]). A slash (/) before the keyword makes the tag close a section.

The sections are composed by sentences, each of them formed by comma-separated statements, and ended by a dot (“**.**”). The first statement must be a *concept word*, this is, a word that explicitly defines the idea that will be represented, followed by a single-word proper name for that entity, which must be unique for every entity (the environment does not need a proper name), which can be of any length and can use special symbols. The rest of the sentence is composed by the entity properties, which must begin with the property name, followed by the values for that specific property. Numeric values must be enclosed in parenthesis, and it is possible to define specific ranges by enclosing the values in brackets ({ }), and separated by commas. This last option is a technical feature of the lexical-syntactic parser, and its use is not intended for final users.

The basic structure of a description composed in VEDEL follows the pattern:

```
‘‘[ENV]’’
  Environment keyword, [ environment settings ], ‘‘.’’
‘‘[/ENV]’’
‘‘[ACTOR]’’
  { Actor Class, [ Actor Identifier ], [ Actor Properties ], ‘‘.’’ }
‘‘[/ACTOR]’’
‘‘[OBJECT]’’
  { Object Class, [ Object Identifier ], [ Object Properties ], ‘‘.’’ }
‘‘[/OBJECT]’’
```

The revised formal Description Structure is defined as follows:

Description ::= environment, actors, objects;
environment ::= “[**ENV**]” environment sentence “[/**ENV**]”;
actors ::= “[**ACT**]” [{ actor sentence }] “[/**ACT**]”;
objects ::= “[**OBJ**]” [{ object sentence }] “[/**OBJ**]”;
environment sentence ::= environment class, [environment properties], dot;
actor sentence ::= entity class, [actor properties], dot;
object sentence ::= entity class, [object properties], dot;
properties ::= { separator, properties };
environment class ::= entity class;
entity class ::= class, [identifier];
environment properties ::= { characteristic };
actor properties ::= { comma, characteristic | property | position };
object properties ::= { comma, characteristic | position };
characteristic ::= class, value;
position ::= class, [[number], identifier];
property ::= class, [value];
class ::= word;
value ::= word | number | range;
identifier ::= letter — digit, [{ letter | digit | special symbol }];
word ::= { letter };
number ::= “**(**” [minus], { digit }, [dot { digit }] “**)**”;
letter ::= “**A**” to “**Z**” | “**a**” to “**z**”;
digit ::= “**0**” to “**9**”;
special symbol ::= “**@**” | minus | “**_**”;
minus ::= “**-**”;
comma ::= “**,**”;
dot := “**.**”;
range := “**{**” number | word [{ comma, number | word }] “**}**”;

[ENV]
House, night, rain.
[/ENV]
[ACTOR]
Man Albert, old, sit FrontCouch, reading todayNewspaper.
Woman Beth, old, sit RockingChair, knitting.
[/ACTOR]
`[OBJECT]`
`FrontCouch, left FirePlace.`
`Chair RockingChair.`
`Paper todayNewspaper.`
`[/OBJECT]`

[ENV]
Farm, day, hot.
[/ENV]
[ACTOR]
Man Albert, close BigTree.
Woman Beth, sit PicknickTable, talking Carrie.
Carrie, sit PicknickTable, next Beth, talking Beth.
[/ACTOR]
`[OBJECT]`
`Table PicknickTable.`
`[/OBJECT]`

Figure 3.1: VEDEL examples.

A description can contain any number of sentences per section; the number of properties allowed per sentence is only restricted to the number of properties associated with the entry of the entity in the KB. The lexical-syntactic parsing method is presented in detail in the next section.

3.2 Parsing Methodology

Descriptions given by the user are analyzed by a lexical-syntactic parser, which is a *state machine* that searches for invalid characters, and verifies if descriptions are composed following the rules established by the language definition. It also formats the entry into a data structure, which the model creator can use.

The first step is searching for *tokens*, individual words which are strings of characters delimited by a space, a comma, a dot, or a carriage return. Then, the *syntactic analyzer* sets the token to a data structure designed to allow easy exploration by the model creator (figure 3.2). This data structure is a hierarchical tree, with branches which start with entity class and the unique name of the entity, if it founded. The leaves are filled with the properties requested for that entity, starting whit the name of the property, and followed by the values requested for that particular attribute. When a dot is found, the newly parsed sentence is stored and the process continues in the next sentence. If the following token is a *section tag*, the parser verifies tag parity, then proceeds the analysis of the next section or reports the corresponding error state. The parser reports any error to the *Error Manager*, and discards the faulty token, statement, sentence, or section. When the closing objects sections tags is

reached, or when the end of the description string is found, the parser ends its task, and sends the parsed entry and the error report to the next module, where the modeling process continues, the necessary actions are taken in order to complete the process, and the user is informed with the errors found so far.

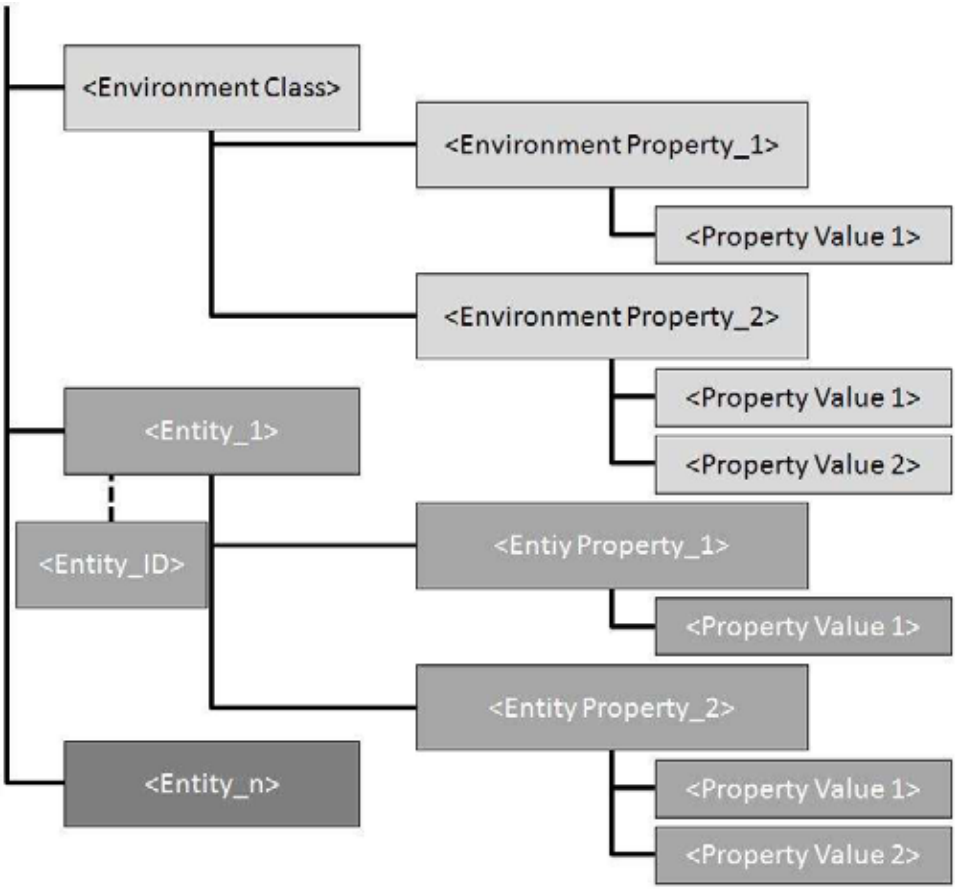

Figure 3.2: Parsed Entry Structure

The parser has been designed, so individual sentences or even individual statements can be sent for verification and conversion, allowing to modify the model previously constructed, by adding, deleting, or changing the attributes for new or existing entities.

3.3 Modeler's Architecture

The modeler is composed by five modules, as shown in figure 3.3. The Lexical-Syntactic Module was explained in the previous section (3.2), while the rest of the modules is presented

in depth in the following sections. This section highlights the functioning of the modeler and resumes briefly each module.

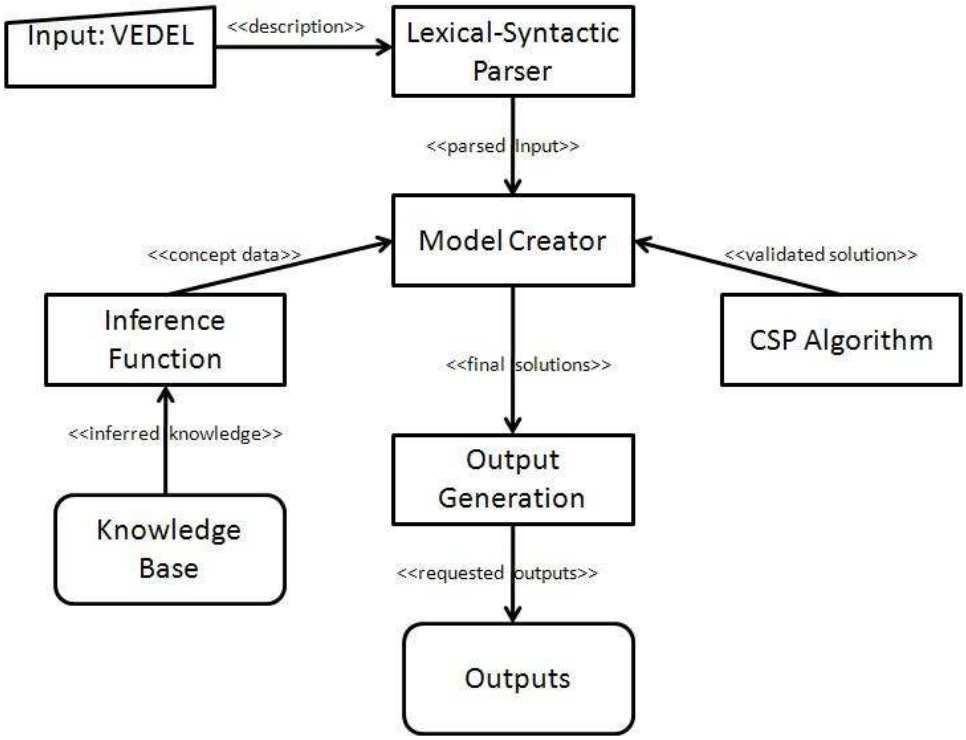

Figure 3.3: Modeler Architecture

Model Creator

The *Model Creator* receives the parsed entry to proceed with the generation of a model satisfying the constraints stated in the description. It makes use of the inference module, and formats the information obtained through it to generate the model. The modeler also creates a list of entities in the environment and their dependencies, so the positioning process can use this information in order to facilitate the task. It also does collision verification in order to solve conflicts involving the geometry of the entities intersecting each other, and positioning validation, to assure the position of the entity corresponds to description constraints.

Inference Module

The inference machine access the KB to gather the necessitated data contained within it. It uses the OWL API to obtain specific knowledge, and formats the data so the modeler can use the values during the model creation. It also validate the semantic values of the constraints, as well as the overall composition of the description.

CSP Algorithm

This module solves the conflicts that arise from the positioning of the elements in the scenario. The CSP Algorithm verifies the model for detecting: collisions, incorrect positioning, or invalid disposition of the elements, to correct any conflict found. It returns a valid model to the *Model Creator*, or the list of unsolved conflicts.

Output generator

The requested outputs are generated by a *Model-View Controller*, which receives the model created by the *Model Creator* to generate the outputs to be used by the underlaying architecture. The templates are formated accordingly to the needs of the architecture or the systems that the model will use.

3.4 Creating the Model

Once the description has been analyzed and the model creator received the output from the lexical-syntactic parser, the actual creation of the model begins. This creation starts with a *default-valued* initial model, which we called the *zero-state model*. This model is refined progressively to finally obtain a model which contains all the values requested by the users and also satisfies the constraints, explicit or implicitly, requested by the users.

3.4.1 Model Data Structure

The obtained model is basically a hierarchical structure. The top elements correspond to the class of the entities, which are obtained from the KB. The leaves on the same level correspond to the unique name, that is, the type of entity (environment, actor or object), and its properties. The structure representing the environment model is presented in figure 3.4.

The classes defining the root branches in the model are Environment, Avatar and Actor. The *Environment* contains the details for the setting of the scenario, as well as the laws that will rule the entities in it. The information contained in the highest level correspond simply to environment size and descriptor, all the rules and properties of the environment are stored in the leaves of the sub-tree with root on the entry Environment. *Avatar* corresponds to the basic information regarding the entity, or the "*default*" entity, which can be viewed as the base model for all the entities of the same type. Example: a "man" avatar contains information about the conformation of a human, i.e., arms, head, legs, as well as the properties for the entity, such as hair, skin color, stamina, mood, with all of them having specific values set in the KB. Also, this class can be used to control the number of avatars of the same type, so the modeler can set the individual names for those entities without a specific identifier. The

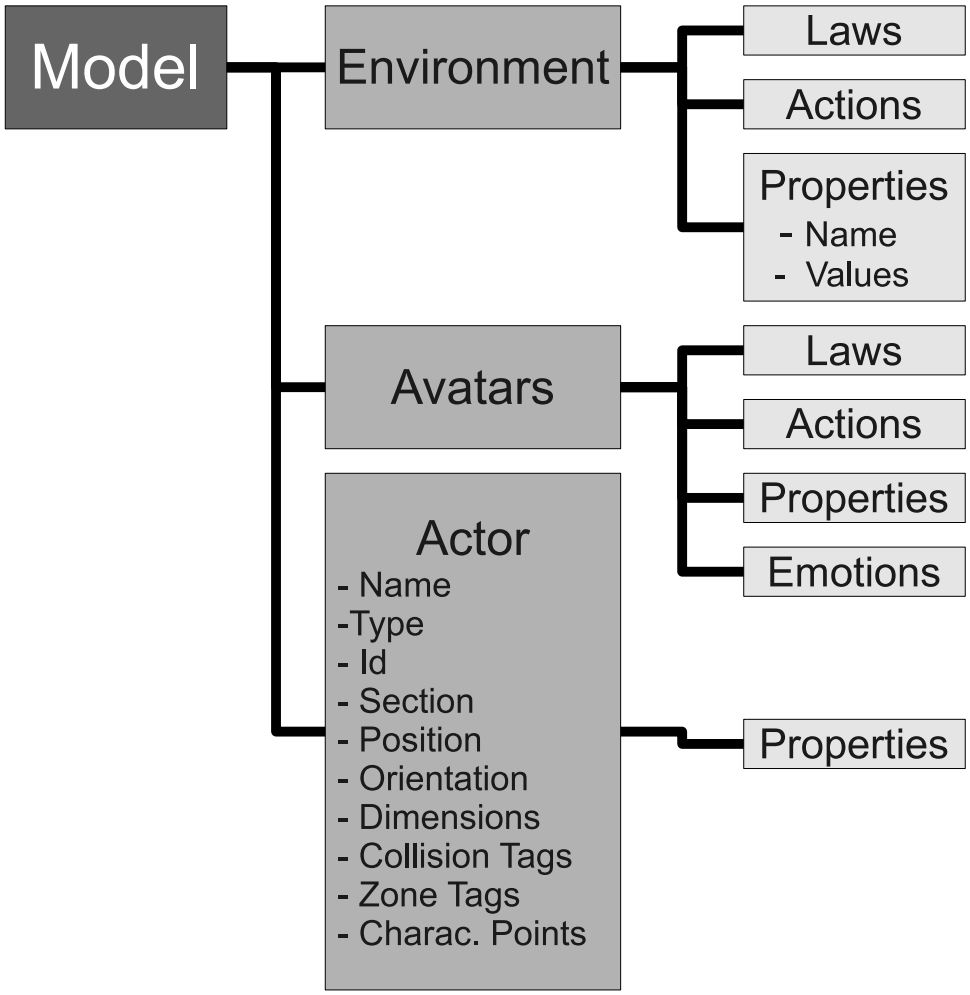

Figure 3.4: Model Structure.

elements of this class are used as a template; this means that all of the properties for new entities are copied from the values stored in this branch. Finally, the last branch contains the avatars instantiations, the *Actor* class instances, where details for each entity are stored. Those details include properties such as size, scale, position, initial action or validation tags. These values are stored as basic data types, since this branch is accessed by other modules of the architecture.

3.4.2 Modeler procedure

The modeler explores the parsed entry, queering the inference machine for every term found, with the exception of individual identifiers or numerical values. The information provided

by the inference machine is processed and stored in the corresponding sections of the model, keeping the linguistic terms for later references or validation procedures during the direct modification of the model. The modeler also conducts the semantic validation using the KB, dropping the values, terms or concepts found to be invalid or out of context.

Knowledge Exploitation

The *Knowledge Base* was define using the Protégé framework, on the OWL Language, and holds four basic classes: **Actors**, **Environment**, **Keywords** and **Objects**. Each of these classes contains all the entities that can be represented, either by the underlying architecture, or by any external software. For each class there must be at least one individual, named as the class, and followed by the "`_default`" suffix. Individuals are the instantiations of the classes, and contain all the data accessed by the inference module.

Each individual contains several mandatory properties such as size, position, entity descriptor, concept ranges, and validation tags. The environment must at least contain properties for size, and descriptor (an embedded description that will represent the environment). Actor and Objects must contain properties for size, initial position, collision tags, characteristic points, and attributes. Actors also must contain the action to be performed by the entity if not action is explicitly or implicitly stated. Actors without a default action are set to idle at the beginning of the scene. Finally, Keywords must include value ranges, property type, and operation type. The modeler to validate properties uses this information.

Knowledge bases can contain two properties types: Object and Data type. *Object* properties express relations among the elements in the KB. This type of property is used by the modeler to extract information about the values allowed for each property, as well as for formatting data for the model data structure. They are also used to determine the semantic validity of statements made in description. *Data type* contains raw values that define concepts. These values can be: character strings, numerical, date, or Boolean values, which are stored at the end of the modeling process in the data structure. The Output Generator or the underlying architecture accesses this information.

The inference machine starts queering the KB for a particular concept. If the information exists, then it subtracts the information regarding the type of concept, and proceeds to explore the values stored for that particular entry. If the values are found to be object type, the inference process continues deeper, subtracting values for objects referenced, and then exploring the values stored for each of these, until a raw data type value is found. To exemplify this process, consider the following request: "`Chair, color white.`". The "`color`" property must be first a valid characteristic for the entity `Chair`, which is verified when the inference function subtracts the values for the avatar. Then the inference machine continues looking for possible values that can be assigned to the property of the entity. If

any of the values correspond to the request made, the inference machine then proceeds to obtain the value assigned to the concept, in this case, "*white*". The inference machine then subtracts the data type value named `RGB` stored in the individual "`white`", and passes the raw value stored as the character string "`1 1 1`" to the modeler, which uses it to create a leave for the entity in the model.

Depending on the concept in process, the inference module conducts conversions between data types. For example, the *individual* (instances of a class) for class "*left*", a child of class Keyword, contains the property `range`, which expresses the values to be used to set the position of an entity. The value is stored as a character string in the KB, for instance, as "`{30,0,0}`". This value cannot be used directly by the modeler, which requires a float-type array. Thus, the inference module conducts the conversion from string to array, and returns it to the modeler.

Since all the modeling process is based on the exploitation of the KB, the data stored can lead to unexpected or invalid values, from the users' perspective expected results, so management of information should be left to a system administrator or an experienced user.

Initial Model

The modeler starts with an empty model, which is updated as the parsed entry is evaluated. The first element to be updated is the environment. The modeler requests the inference machine for the *default* values for the environment where the scenario must be constructed and which are used to create a temporary leave. This leave is updated first with the instructions set for the environment, and later with the user's requests. When the environment has been fully processed, it is assigned to the model.

The scenario can contain several *zones*, which have no visual representation, but are still used for referencing specific areas in the environment as referents for absolute positioning. These areas can be *landmarks*, specific delimitations inside the environment, *walls*, used to define the limits for landmarks, and *doors* which indicate the zones that allow the entities to pass from one zone to another. It is during the initial model generation where these zones are established, thus gives the rest of the entities the referencing points for proposing a positioning.

The procedure for the entities differs in one step. When the modeler finds an actor or object type entity, a search for the corresponding avatar is conducted. If the avatar is already stored in the model, the process to update the values for the properties is started, using a copy of the avatar as template for the current entity. If the avatar is not found, the modeler requests information for that specific entity type, and stores the corresponding avatar in the model, creates a template copy, updates it with the user's request, and adds the new actor to the model.

The next step is to gather information about the position of the entities in the scenario, and the relations between them. The parsed entry is queried to subtract the entities, and store them in a FIFO list. When a new element is added to the list, the relation with other entities is explored, and the entities that hold a relation with the current one are checked. This recursive process allows the modeler to set first those entities which have dependencies on them, which we called *pivots*, and later add the elements that make reference to these pivots.

Any error found during the creation of the model is reported to the *Error Manager*, which stores the corresponding error data and provides the necessary actions needed to solve the error state and continue. If the modeler is set to stop when finding an error, the process is halted and the error presented to the user. Otherwise, the process continues, and the list of errors is presented at the end of the modeling procedure.

Properties Values Assignment

As presented in section 3.4.2, an entry in the KB for any entity contains linguistic terms as well as raw data type terms, and the relations with other elements in the KB. The conversion works over the non-type terms to convert them to basic raw data values. This conversion is made so that the final output can be read by the other modules in the architecture, adapted to their own input languages, or to create output files readable by 3D render machines.

When the Inference Machine is processing and needs a property concept, the KB is queried for the term, and the raw information is in turn processed. If any of these raw data items is a linguistic term, it is processed, and the cycle continues until the low-level data types are obtained.

The values stated for the properties in the description are first validated against the restrictions set in the KB, then converted in the case of linguistic terms. Any inconsistency found during this process is reported to the *Error Manager*, which decides if the property can be removed or must be set to a default value.

Some of these data items, such as the "*furniture*" property in environment classes, or previously generated VWs, could be VEDEL sentences or statements, which are processed by the Lexical-Semantic parser, and the objects that form part of to the environment, or some entities, are included. These items do not disrupt the generation process, since they are processed after the parent entity is validated. In this way, it is not necessary to extend the positioning list to include these new elements, since the parent acts the pivot.

The raw values obtained from linguistic terms are returned to the model creator, which assigns them to the entity in creation, and then continues with the following property. Once all properties have been validated, the modeler adds the entity to the model, and continues with the model generation task if it is needed.

3.4.3 Geometrical Validation

Once all data properties have been processed, the modeler continues with the process for verifying the positioning of all elements. This is handled by a CSP algorithm, which uses the collision and positioning tags to solve conflicts or invalid positions.

Each entity in the VE is represented as a set $X_i = \{P_i, O_i, S_i\}$, where $P_i = \{x_i, y_i, z_i\}, O_i = \{\alpha_i, \beta_i, \gamma_i\}, S_i = \{Sx_i, Sy_i, Sz_i\} \forall X_i \in X$, corresponding to position, orientation and scale for that entity.

Default position corresponds to the VE's center, this is, $\{0, 0, 0\}$. Each entity starts with a position corresponding to the VE center at the first modeling steps. If a specific position is requested, the model creator sends a query the to Inference Function, which returns a vector $V = \{x, y, z\}$, corresponding to the request. V can be either an absolute position, or a point in relation with another entity or environment component. In that case, V is calculated as $Rot(V - (X(P) * X(S)), X(O))$, where $Rot()$ is a rotation function on $X(O)$.

The CSP used by the modeler is defined as follows:

- The set of variables $X = \{X_1, X_2, \ldots X_n\}$ composed by the entities in the scenario, $X_i = \{X_i \bigcup \{Co_i, Ch_i\}\}$, where $Co_i = \{Sp_1, Sp_2, \ldots, Sp_n\}$ representing a set of collision tags assigned to the entity, where $\forall Sp \in Co_i, Sp_i = \{Spx_i, Spy_i, Spz_i, Spr_i\}$, and $Ch_i = \{Pe_1, Pe_2, \ldots, Pe_n\}$ corresponds to a set of characteristic points for the entity, where $\forall Pe \in Ch_i, Pe_i = \{Pex_i, Pey_i, Pez_i\}$.
- The domains for the variables are defined as follows $D(P_i) = [-\infty, \infty]$, $D(O_i) = [0, 2\pi]$, $D(S_i = [0, \infty]$, $D(Co_i) = [-\infty, \infty]$ and $D(Ch_i) = [-\infty, \infty] \forall X_i \in X$
- The constrains set is formed by the following equations:

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 - (r_1 + r_2) \leq t_1 \tag{1.a}$$

$$\left(\frac{x}{p}\right)^2 + \left(\frac{y}{q}\right)^2 - 2z = t_2 \tag{1.b}$$

$$\left(\frac{x}{dx}\right)^2 + \left(\frac{y}{dy}\right)^2 + \left(\frac{z}{dz}\right)^2 - 1 = t_3 \tag{1.c}$$

Thresholds t_1, t_2 and t_3, where $\{t_1, t_2, t_3\} \in \Re$, are values set by the system administrator. These values allow modifying the strictness for the constraints. Each collision tags set is stored in the KB as a vector, which represents the position and radius for the sphere.

The second set of characteristic points is based on the geometry of the entity, selected if they help to represent the contour of the entity, or correspond to a particular feature. These points are used along equations 1.b and 1.c to verify positioning. Both collision and

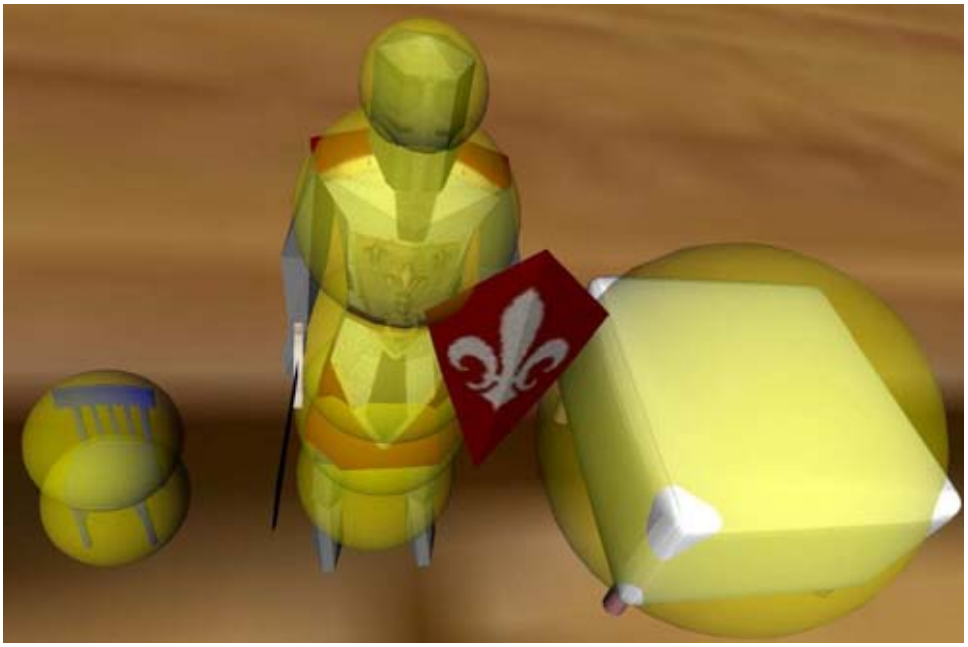

Figure 3.5: Collision Tags

characteristic points set are updated as the entity moves or rotates, using the $Rot()$ function. The radius of collision tags is also modified when the entity is scaled up or down.

The constraints are satisfied if:

1. Be $(X_i, X_j) \in X$, constraint 1.a is satisfied $\leftrightarrow C_1(Sp_m, X_j(Sp_n)) \leq t_1, \forall Sp \in X(Co)_{i,j}$.
2. Be X_i an entity whose position was calculated as $P(X_i) = F(V, X_j)$, constraint 1.b is satisfied $\leftrightarrow C_2(Pe_n, X_j) \leq t_2, \forall Pe \in X(Ch)_i$.
3. Be X_i an entity with a position calculated as $P(X_i) = F(V, X_j)$ and $P(X_i) - P(X_j) < 1$, constraint 1.c is satisfied $\leftrightarrow C_3(Ph, X_j) \leq t_3$, where $Ph \subseteq Ch, \forall Pe \in X(Ch)_i$, and $0 < dn < 1, dn \in \{dx, dy, dz\}$.
4. Be X_i an entity with a position set as $P(X_i) = V$, constraint 1.c is satisfied $\leftrightarrow C_3(Pe_n, Env(S)) \leq t_3, \forall Pe \in X(Ch)_i$, where $Env(S)$ is the environment or area measures.

A verification is taken previously to search for collisions: if the euclidean distance between two entities is less or equal to the sum of the diagonal of a box formed by the dimensions of each entity, collision verification is carried out. This can be defined as follows:

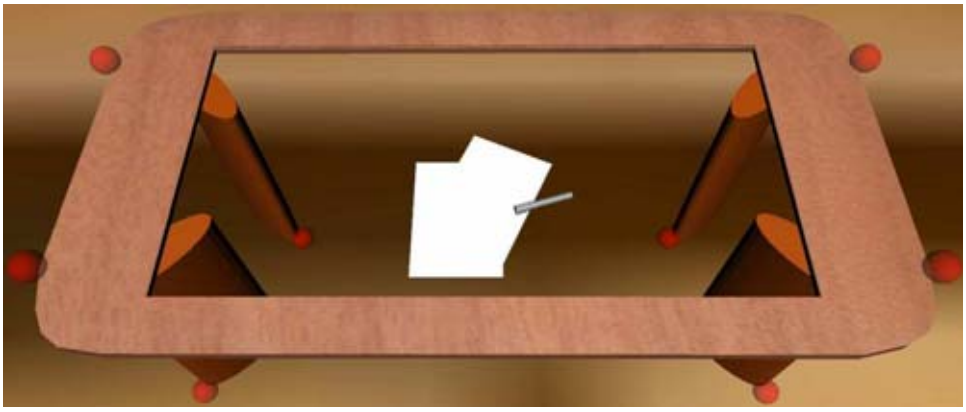

Figure 3.6: Characteristic points

Be $X_i, X_j \in X$, collision verification is executed if

$$\sqrt{(X_i(P_x) - X_j(P_x))^2 + (X_i(P_y) - X_j(P_y))^2} \leq \sqrt{X_i(S_x)^2 + X_i(S_y)^2} + \sqrt{X_j(S_x)^2 + X_j(S_y)^2}$$

When a collision occurs, there exits two actions:

1. If one of the entities is a *pivot*, that is to say, it is set to an absolute position, such as `north`, `south` or `east`, the other entity is moved using a vector which starts at the center of the pivot entity and is directed towards the center of the target entity, and with a magnitude equal to the outcome of equation 1.a.
2. If none of the entities is a pivot, either or both entities are moved using a vector which starts at the center one of the entity and directed towards the center of the other entity, and with a magnitude equal to half of the outcome of equation 1.a.

The second option necessitates carrying some verification before making any change in the position of the entity. First, if both entities in collision make reference to the same entity, an algorithm similar to the one used by the FL-Systems is employed to set the new position for each entity, making use of equation 1.b to validate these new positions. If the entities are unrelated, the one with the smaller volume is moved first, since smaller elements are more likely to be set in valid positions.

In some special cases, the tags assigned to a concept correspond to areas on the entity, for example, the `over` keyword (figure 3.8). When the system deals with those cases, the system administrator can set the number or even the exact characteristic points that should

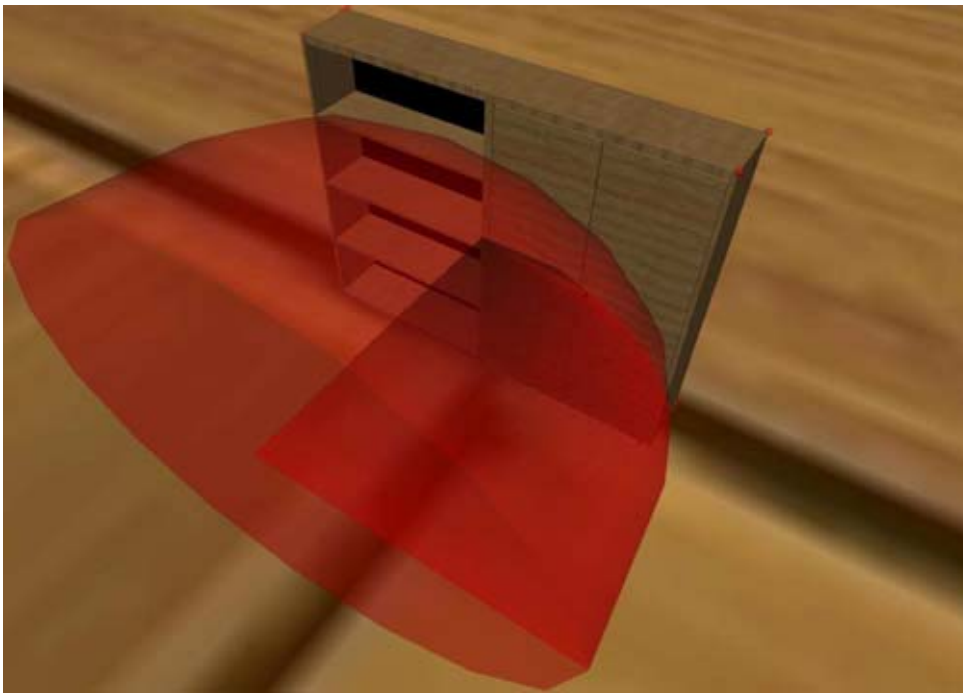

Figure 3.7: Validation volume for equation 1.b

be inside that volume, so the position can be declared valid. The equation 1.c is used in those cases.

When it is required that the entity be positioned inside another, or in a specific geographic location, the volume generated by equation 1.c is used to match most or all of the space. In these cases, the system administrator can also set the characteristic points that must be contained for the position to be valid (figure 3.9).

Previous modifications on the entities that can represent poses are made, using a specialized module in the architecture. This module, called the *Planning Module*, part of the research presented in [37]. This module uses self-conscious entities which modify the position of their limbs in order to represent the desired pose. Sitting, running or holding is computed by means of knowledge which describes the skeleton of the entity, and can be applied to any entity that holds the same structure. This structure can be modified to represent the lack a limb, allowing characteristics such as limping. Modifications are carried out through synergy movements and using the KB to verify the new pose.

After all collisions have been solved, the CSP continues its work to verify if the new position of the entities is valid. If it verified that the entities current position passed test using constraint equations 1.b and 1.c. If true, the process ends and the user is informed. Furthermore, the previous position of the entities is recorded, and the process continues.

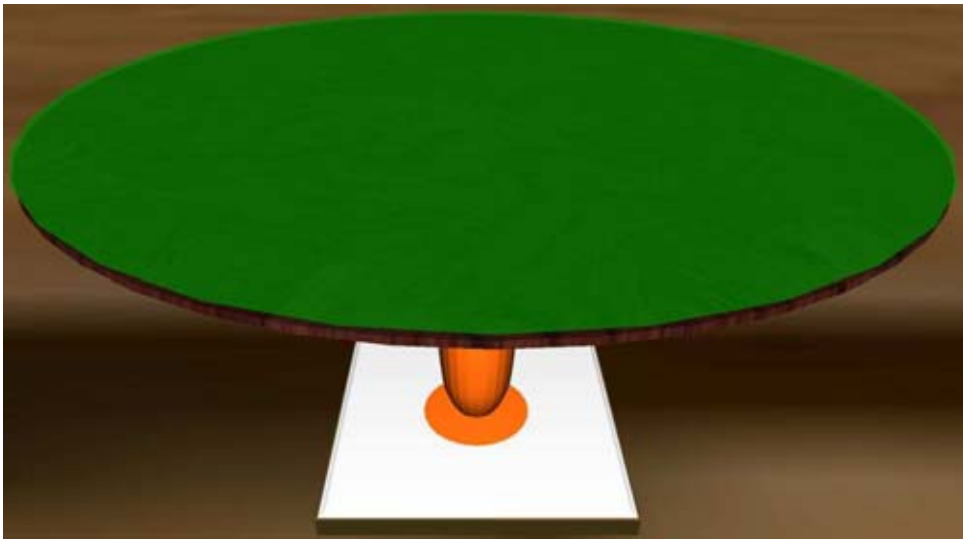

Figure 3.8: Special case: against

When the CSP falls in a dead end, the recorded positions are used to create a new state, and continue the constraint solving process. If the dead end cannot be solved, previous model values are used, moving the pivot entities if needed, and then verifying the new state. The previous positions are also used to determine if the new positions computed are forming a cluster, and then compute a new position far away from it, thus preventing falling into local minima, where apparently there are no solutions, or local maxima, solutions with stiff constraint values, which does not allow further modifications and lead to dead ends for other entities. Other methods used to find solutions are: the total re-arrangement of the entities, rotate an entity that is being referenced by other entity in conflict, or dropping some of the entities.

If the CSP can not find a solution, an error state is raised, and the modeler can either stop the process and present the error status to the user, or can drop the violating entities (to eliminate the conflicts) and continue, presenting an error report at the end of the process.

3.5 Generation of the Outputs

When the *Model Creator* has finished constructing the model, the next step is to create the output that will be sent to the underlying architecture. This step is carried out by a MVC (Model-View Controller), which pre-process a series of templates, and then receives the model to fill the templates with model values.

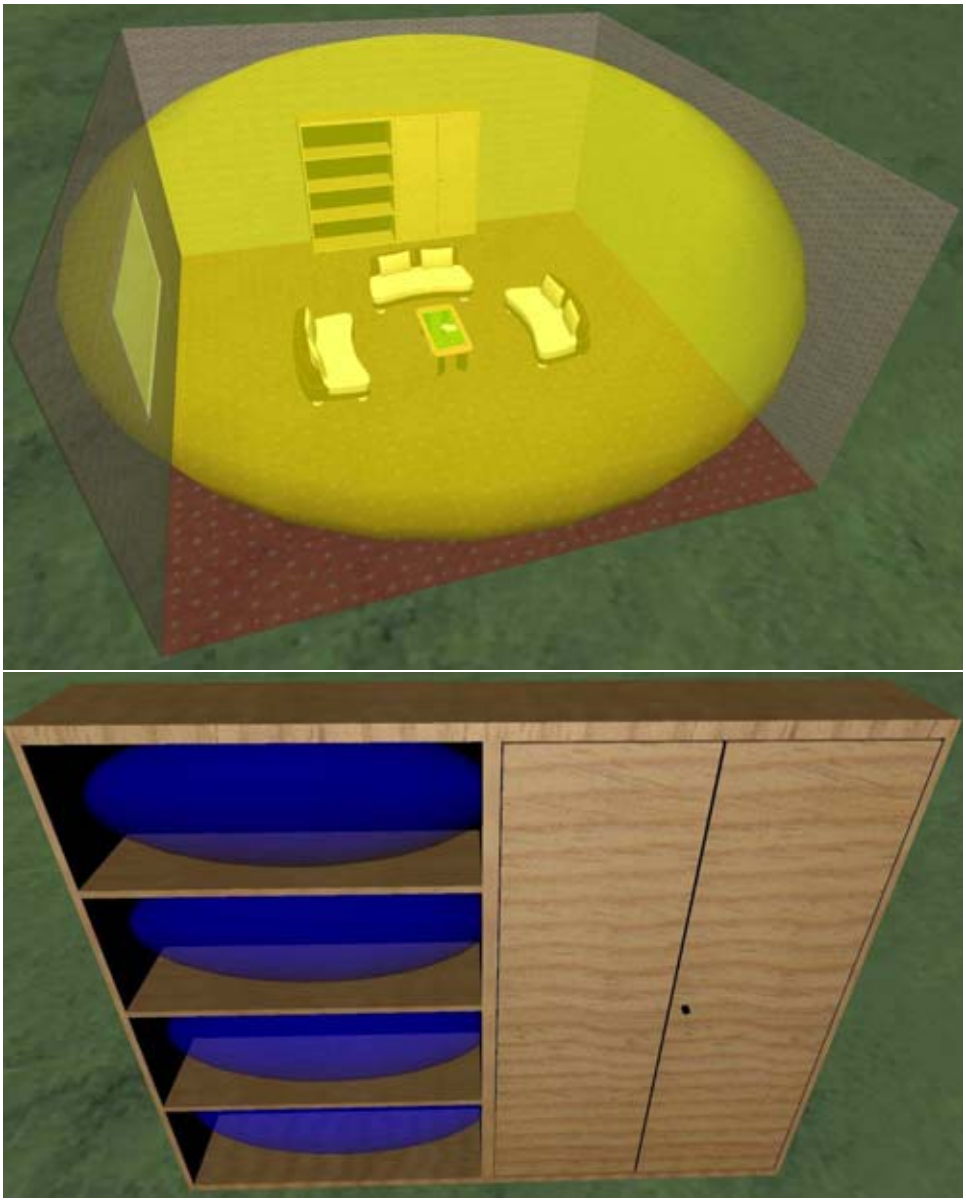

Figure 3.9: Special case: inside

3.5.1 Model-View Controller

A *Model-View Controllers* allows easy translation from data structures to file or character stream outputs. It was chosen to ease communication between the VEE and the rest of the modules in the architecture. Each module requires a specific portion of the information stored in the model, and each of the modules has its own input language. It would have

been difficult if the modeler had created, generated and maintained these inputs by itself. Instead, the MVC uses a serie of templates to create the outputs, using the KB to fill any additional request. This method also allows to modify the type, quantity and destination of each of these outputs.

3.6 Modifying the Model

Once the modeler has shown the users one or several of the possible solutions for the description, direct modifications over the scenario can be made. This is handled through specific commands written in a syntax similar to VEDEL, but with a few modifications. The basic structure of a modification commands is:

<command ><entity identifier ><arguments ... >
arguments ::= [<new position >] | [<modifier >]

A command corresponds to either position (command `M`) or properties (command `C`). The command is sent to the model, which then takes the necessary actions to ensure its execution. If the modified model fails any of the construction validations, the model is returned to its previous state, and the error is reported to the user.

To apply the modifications made to the model, an interface that directly modifies the model was implemented. This is simply a VEDEL translator; the parser processes. The translator receives the command that is translated into a VEDEL-compliant sentence. Then, the parsed command is sent to the *Model Creator* module, which carries out the same verifications made for verifying the model during its creation. If the new model is tested as valid, a new output is created and sent for processing.

Chapter 4

Research Outcome

Abstract

We present the results obtained from two different prototypes, as well as some observations made during the research.

4.1 Virtual Environment Editor Prototypes

During the course of the research, the methodologies proposed where applied in the implementation of a VEE, based on DM. This VEE receives a description written on the VEDEL specification and then proceeds to generate the model, presenting the output, if successful, in a X3D-compliant viewer.

This prototype was developed in the Java language, using the OWL Protégé API and the FreeMarker library, for access to KBs and using MVC methods. So far, our focus has been on modeler functionality, rather than final-user interface. Therefore the GUI for the VEE is still in early development, but is fully functional (figure 4.1).

The selected MVC was Freemaker [57], version 2.3.15, a "*template engine*". This is a tool that creates a text output based on templates, programmed in Java and with a Java API. Although FreeMarker has some programming capabilities, it is not a fully programming language, similar to PHP, but more a data display generator. This first study case uses X3D-formated templates that generate an output that can be viewed using any VRML97 or X3D compliant viewer.

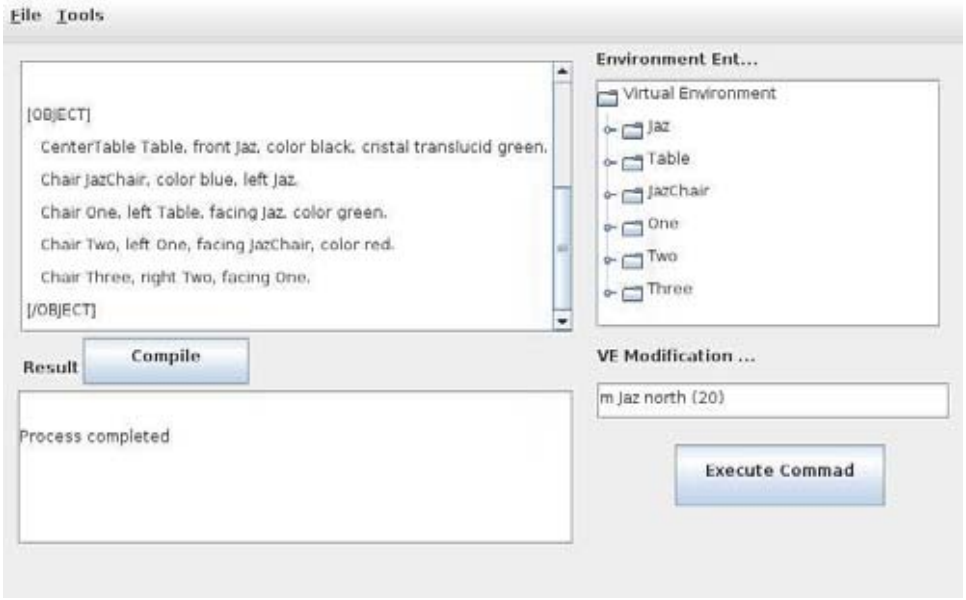

Figure 4.1: Virtual Environment Editor GUI

The KB used by this first prototype contains several entries for environments, actors, objects and keywords, and is an extension of the KB used for the prototype presented in the previous research ([56]), as presented in figure 4.2. This was a work based on the GeDA-3D prototype developed by Gutierrez [58].

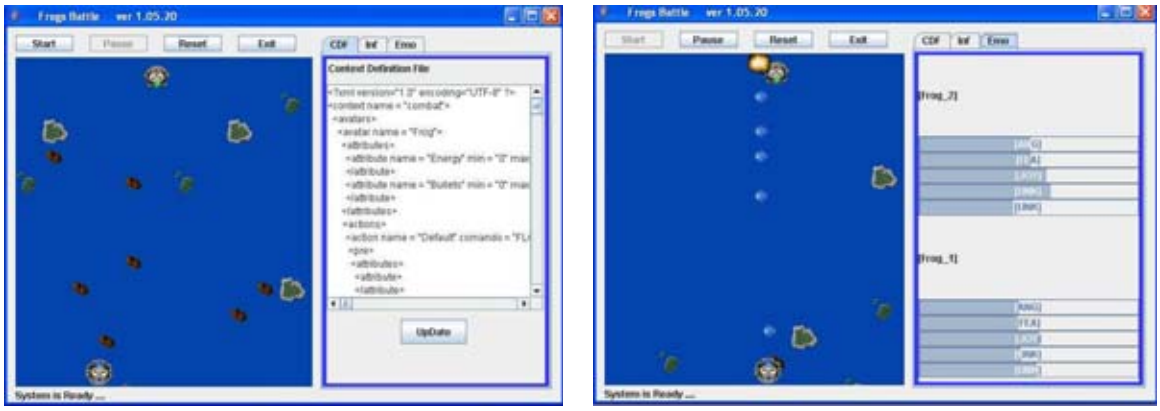

Figure 4.2: Previous Prototypes: Battle of the Frogs

Figure 4.3: Previous Prototypes: Earlier version of GeDA-3D

This first use case was a prototype for the GeDA-3D kernel, which used an earlier version of the modeler to model the initial state of the simulation, which also included the emotional machine by Razo [36]. This earlier basic modeler allowed to initialize the actors values, as well as setting it emotional behavior.

A second prototype was developed, in order to test the functionality of the earlier GeDA-3D kernel, the VEE and the Render Module, a work by Matinez [59]. This prototype sent the final output for both the kernel and the render module, which presented a limited amount of entities, letting us prove also some of the expectations for the architecture (figure 4.3).

This earlier prototypes where the basis for the current efforts, from which we obtained some models presented next.

4.1.1 GeDA-3D Virtual Environment Editor Prototype

Example 1:

```
[ENV]
   room.
[/ENV]

[ACTOR]
   ManSuit, left one, facing Table.
   woman, right one, facing Table.
[/ACTOR]

[OBJECT]
   bookshelf, againts NorthWall.
   CenterTable Table, color black, cristal translucid gray.
   Sofa one, behind Table.
   Sofa 2, left Table, facing Table.
   Sofa tri, right Table, facing Table.
   Chair One, behind (2) ManSuit0, color green, facing ManSuit0.
   Chair Two, behind (0) woman0, color red, facing woman0.
   Puff seat, front Table, color black.
[/OBJECT]
```

In this case, we can see that distance is expressed between parentesis (`(2)` and `(0)`), and that both `ManSuit` and `woman` entities have an unique identifier automatically, *ManSuit0* and *woman0*.

Figure 4.4: Example 1: Top-Down view

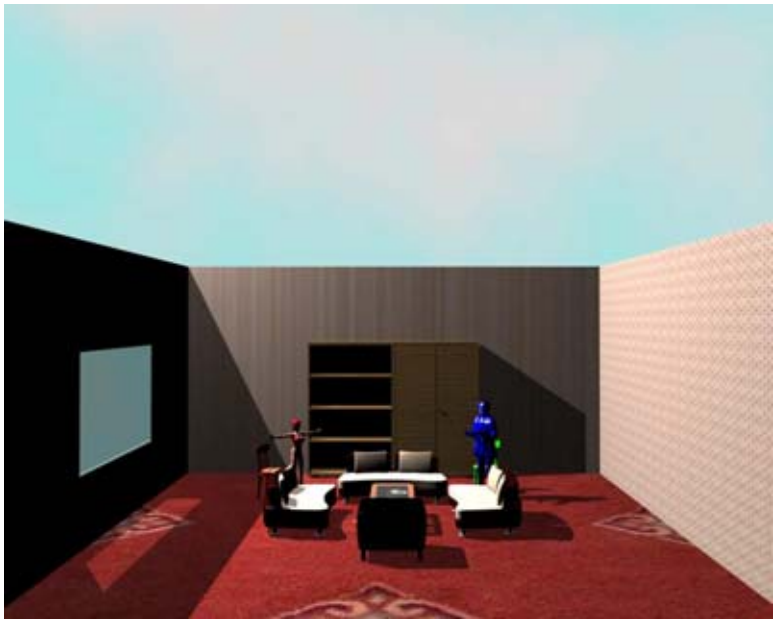

Figure 4.5: Example 1: General View

Example 2:

```
[ENV]
   house.
[/ENV]

[ACTOR]
   ManSuit, anywhere Kitchen.
   woman, anywhere Garden.
[/ACTOR]

[OBJECT]
   CenterTable Table, color black, cristal translucid gray, front (50) bed0.
[/OBJECT]
```

Here, `house` was defined before hand, and its definition stored in the KB. The position for the sink, refrigerator, table, chairs, and trees are set in the KB, as well as the specific areas, such as `Kitchen` and `Garden`.

Figure 4.6: Example 2: House environment

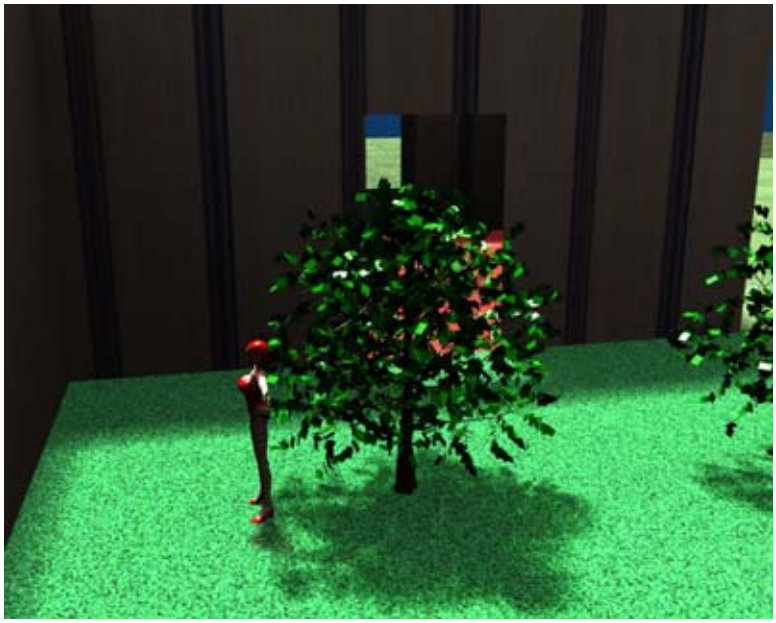

Figure 4.7: Example 2: House environment

Example 3:

```
[ENV]
   forest.
[/ENV]

[ACTOR]
   Knight One, center.
   Knight, near One.
   Knight, near One.
   Knight, near One.
[/ACTOR]

[OBJECT]

[/OBJECT]
```

In this last example `forest` contains the necessary indications to position the trees, and is as well stored in the KB.

Figure 4.8: Example 3: Detail view

Figure 4.9: Example 3: Top-Down View

4.1.2 DRAMA Project Module DRAMAScène

As part of the collaboration with the Institut de Recherche en Infomatique de Toulouse, IRIT, we adapted part of the VEE to form part of the DRAMA Project [60].

DRAMA is a multimodule project, consisting of DRAMATexte, the reader tool for indexing theatrical texts, highlighting the most important elements for the director, and DRAMAScene, a set-in-scene visualization tool, which allows the theater company to work on a possible view for the play.

Both modules work together in the following way: first, the system receives a theatrical piece as input, and then analyzes it and tags all the characteristic elements, such as dialogues or introductions. The user then can add new tags, which will help to make a structured indexation for the non-explicit elements in the play, such as movement, mood or illumination. The indexed text is then used to generate an entry for the DRAMAScène module, in which a DM takes the task of creating a visual representation for the play, and later allowing the user to modify the proposed visualization.

The DM is based on our own VEE, being the main difference the concepts stored in the KB. For this project, strong emphasis was made on the context for the model. Context plays an important role since depending on the type, epoch or style of the play, the model changes significantly. For example, whereas a modern play involves the actors making direct eye contact during dialogues, the style of older plays involve the actor addressing the audience all the time. Technology available in the epoch for the play is also taken into account the, as well as costumes and props.

In figure 4.10, we present some examples of models obtained through this variation of our VEE.

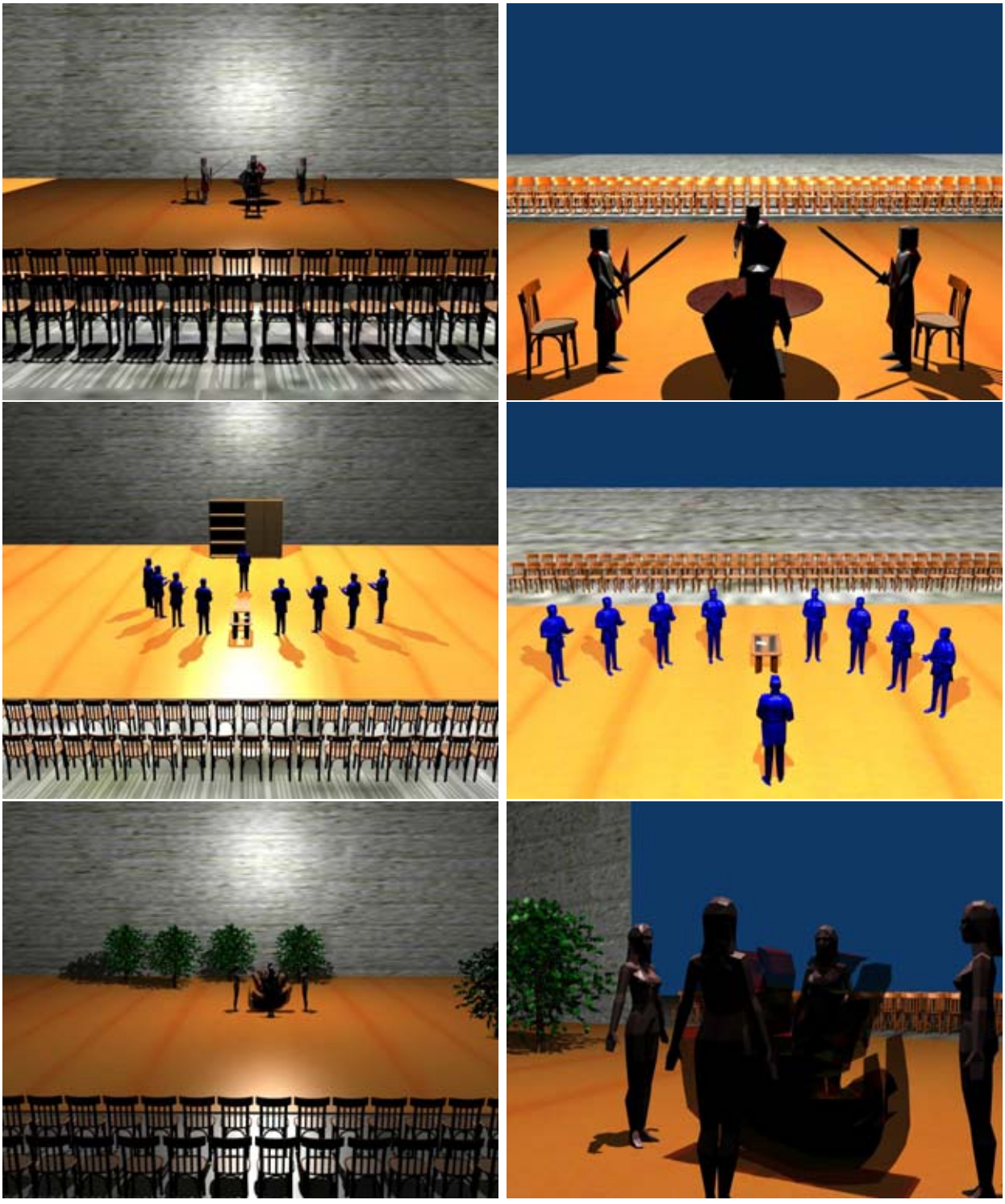

Figure 4.10: Examples of DRAMAScéne Concepts

Chapter 5

Conclusion

Abstract

The last section of this documents presents the conclusion draw form the work and makes a comparison with other works. Finally, the future assets to be covered are presented.

5.1 Conclusions

Through the study of the available literature from related works, both on DM and in knowledge management, we can point several features that distinguish our research from others. First, most of the proposed input methods make use on specialized hardware, which can be intimidating for the non-experienced user, whereas we propose a direct method, which is also structured to aid in the composition of the scenario, supports both complex and simple entries. Also, they are based on the use of the mouse, which is a useful tool in 2D environments, but requires certain learning step to be used on 3D environments. Two researches propose using declarative methods: WordsEye and CAPS. WordsEyes relay on direct human language, using complex parsing methods to obtain the semantic tree used to construct the VW, leading to over-simplistic compositions in order to generate the desired output, although making it more natural to users, the web nature of the system does not allow to extend the existing object database, and even when several techniques are used to provide a visual representation of unknown concepts, the system does not make any semantic verification for the congruency of the scenario, so illogical or non-realistic situations can be created. Finally, the static output does not allow for any further interaction, and the system does not provide a method for further use of the model, that is, does not let the scenario develop into a scene, and does not provide interaction between the user and the entities created.

CAPS uses a specialized constraint declaration to construct the scenario, which is not fully presented in the literature, focusing only on the *placement* of objects, leaving their physical characteristic completely out of any modification. Also, the system uses a direct interaction method with the user: an object being placed highlights the possible surfaces that can be occupied, making the modeling process mechanic and allowing for non-logical positions, if that is the users' desire. It uses direct tags for allowing the placement of objects in cases such as "over" or "inside", but restricted to Boolean tags, which makes the positioning of new objects or incomplete entries difficult. Finally, the modeling process ends when all of the objects have been placed, without providing any methods for interaction between objects or with users.

Other projects, such as CityEngine or FL-System, are completely based on using specialized data or input methods, and provide static visualizations of the scenarios described by the users' input. Again, none of these researches deal with post-modeling tasks such as entity interaction, environment development, or the user's external input.

Our research not only focuses on the generation of a VW, that is, the positioning of elements inside the VW, but it also provides grounds for modifying most of the aspects for the scenario and the entities, either as characteristics visible through graphic representation, as well as implicit properties that can modify the entity behavior during the simulation run, in the form of a context associated with the model.

We do not only crate the visual representation for the users' input. We also verify its congruency, and adapt the non-explicit values, based on a semantic-base (our main contribution of our proposal KB), to find conflicts, solve them, and only after this process has finished, then proceed with the visualization and animation of the world envisioned in the users' mind. None of the researches reviewed during the first phases deal with internal representation for the entities, or the rules of the worlds. The conjunction of these two parts, the visual output and the implicit representation for both the world and the entities, can work to create the simulation of a complex VW.

Our research showed, as in the figures presented in the previous sub-chapter, that a knowledge-based modeler could successfully construct a model that represents a VW, beginning with the description written on a near-natural language. This model can be integrated into the KB to be used further in the construction of more complex worlds, which can be assigned with new or complementary rules. Also, the rules dictating the construction and evolution of the VW can be modified according to the users' needs, by adjusting some values in the KB, or modifying the output-generation templates. This allows recreating almost any possible environment, with the only restriction that the elements, setting, and rules for that particular representation exist in both the KB and a 3D model database, and also, by stating the necessary information in the KB, the modeler can retrieve and even generate the necessary data to allow the evolution of the VE. In fact, that data must include the rules of the world, the entity behaviors, the relationship among entities or the values for the entity internal properties.

Project Features Comparison.

Project	Input Type	Modeling Type	Output Type	Creates Scenes	Editing Tools	Environment Navigation
GeDA-3D VEE	VEDEL Description	Declarative (CSP)	Multiple output MVC Based	Y	Y	Viewer/Render Based
WordsEye	Description in natural language	Declarative (Multiple methods)	Static image	N	N	Visible after render
DEM^{2}ONS	Multiple inputs	Declarative (CSP) + AO + LE	3D Render	N / Y with LE	Y	Yes
CAPS	Specialized Grammar	Semantic Techniques, Pseudo-physics	3D Render	N	Y	Y
ALICE	Graphic GUI	User-Based	3D Render	Y	Y	Y
FL-System	Specialized Grammar	Context Free L-System	VRML-97	N	N	Viewer/Render Based
City Engine	Statistical and geographical data	Extended L-Systems	Rendered Images	N	N	Y
Instant Architecture	"Split Grammar"	L-Systems	3D Render	N	N	N

5.1.1 Future Work

Several aspects must be explored in the interest of extending the reach of this research. For instance, to extend, updating and upgrading the databases, both the KB and the 3D object database must be a main objective for future researches, including a method for finding both characteristic points and the collision tags.

The other important aspect is the fully integration of the VEE with the rest of the GeDA-3D architecture, including updating the modeling process to support future movement and perception sub-modules inside the agents in charge of providing environmental self-evolution, as well as interacting with the parser module and agent community during the animation process.Finally, several arrangements can be made to the GUI for the VEE, to provide the user with a list of all the possible environments and entities available, as well as their must representative properties.

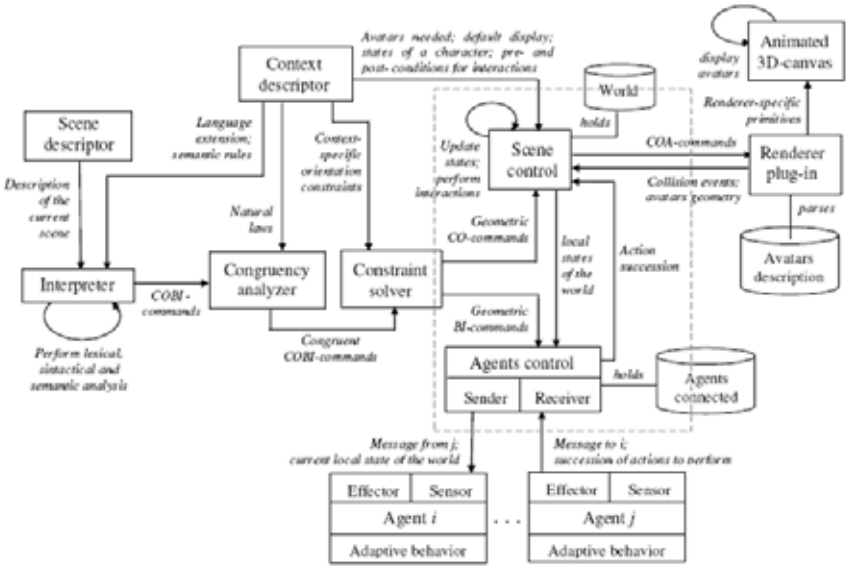

Figure 5.1: GEDA-3D Architecture

This task would help the research by:

- Allowing to design and create new use cases, giving the opportunity to verify novel concepts and the reach of the module.
- Ease the resources needed to create the model, increasing the efficiency of the modeling tool.

- Provide the necessary methods to explore VE self-evolution, and test the context-generation capabilities.
- Cover all possible aspects of DM, providing support for unspecified request and regional or local linguistic accidents.
- Present a refined GUI to final users, easing the interaction and providing end-user oriented features such as VE saving, on-the-fly rendering, on-line interaction and sharing, and end-user technical support.

Being part of the GeDA-3D project, this research is included as a module in the general architecture. This module, named the *Virtual Editor*, also includes a *Scene Editor* and the *Context Descriptor*. The module sends messages through the kernel during modeling time to several other modules (*Planing Module*, *Agents Module*), and sends custom-formated outputs to the rest of the architecture when the model has been approved by the user.

Appendix A

Published Works

Conferences

Jaime Zaragoza, Félix Ramos, Véronique Gaildrat, Generación de Ambientes Virtuales Mediante Modelado Declarativo Basado en conocimiento. Quinta Semana Nacional de Ingeniería Electrónica (SENIE 2009). Centro Universitario de la Ciénega, Universidad de Guadalajara, México.7-9 Octubre 2009.

Virtual World Creation and Visualization by Knowledge Based Modeling. Jaime Zaragoza, Alma Verónica Martínez, Félix Ramos, Mario Siller and Véronique Gaildrat. 5th International North American Conference on Intelligent Games and Simulation, August 26-28 2009. Eurosis-ETI, Atlanta, GA, USA, pages 5-9. 2009.

Creation of Virtual Worlds through Â Knowledge-Assisted Declarative Modeling. Jaime Zaragoza, Félix Ramos and Véronique Gaildrat. 5th International North American Conference on Intelligent Games and Simulation, August 26-28 2009. Eurosis-ETI, Atlanta, GA, USA, pages 20-24. 2009.

Jaime Zaragoza, Félix Ramos, Véronique Gaildrat. Modeling of Vir-tual Environments Through Declarative Modeling Assisted by Knowledge, Workshop on Semantic User Descriptions and their influence on 3D graphics and VR. VRLab, EPFL, Lausanne, Switzerland, November 13, 2008.

J. A. Zaragoza Rios, H. R. Orozco and V. Gaildrat, Modelado Declarativo de Ambientes Virtuales Basado en Explotación del Conocimiento. Cuarta Semana Nacional de Ingeniería Electrónica (SENIE 2008). Universidad Panamericana, Aguascalientes, México, 1-3 Octubre 2008.

H. R. Orozco Aguirre, J. A. Zaragoza Rios and D. Thalmann, Animation of Autonomous Avatars over the GeDA-3D Agent Architecture. Fourth National Week of Electronic Engineering (SENIE 2008). Panamerican University, Aguascalientes, Mexico, October 1-3, 2008.

Book Chapters

J. Zaragoza, F. Ramos, H. R. Orozco and V. Gaildrat, Creation of Virtual Environments Through Knowledge-Aid Declarative Modeling. Frontiers in Artificial Intelligence and Applications (Advances in Technological Applications of Logical and Intelligent Systems). IOS Press, Washington, DC, Volume 186, 144-132, 2009.

H. R. Orozco, F. Ramos, J. Zaragoza and D. Thalmann, Avatars Animation Using Reinforcement Learning in 3D Distributed Dynamic Virtual Environments. Frontiers in Artificial Intelligence and Applications (Advances in Technological Applications of Logical and Intelligent Systems). IOS Press, Washington, DC, Volume 186, 67-84, 2009.

Journals

J.O. Gutiérrez-García, J.A. Zaragoza-Rios, F.F. Ramos-Corchado, J.-L. Koning, M.A. Ramos-Corchado, and M. Siller. "Integration of Agricultural Information Systems Assited by Knowledge". Currently submitted to International Journal of Control, Automation, and Systems. - Selected as one of CTTA 09 Best Papers - acceptance recommended.

GENERACIÓN DE AMBIENTES VIRTUALES MEDIANTE MODELADO DECLARATIVO BASADO EN CONOCIMIENTO

Jaime Zaragoza, Félix Ramos, Véronique Gaildrat*

Grupo de Sistemas Distribuidos
Centro de Investigación y de Estudios Avanzados del Politécnico Nacional, Unidad Guadalajara
Av. Científica 1145 , colonia el Bajío, Zapopan , 45015, Jalisco, México.
Tel. 3337773600, correo electrónico: {jzaragoz,framos}@gdl.cinvestav.mx
*VORTEX Groupe
Institut de Recherche in Informatique de Toulouse
Université Paul Sabatier, 118 Route de Narbonne, F-31062, Toulouse CEDEX 9, Francia
Tel. 33561556765, correo electrónico: veronique.gaildrat@irit.fr

RESUMEN

Crear mundos virtuales usando herramientas dedicadas es una tarea que requiere de tiempo y recursos en cantidades variadas. Aún los usuarios experimentados pueden encontrar difícil el crear representaciones virtuales para diversas necesidades, por ejemplo, simulaciones del mundo real, recreación de mundos fantásticos o antiguos, educación, entre otros. A través del método del modelado declarativo un usuario puede crear un escenario virtual al expresar algunas propiedades deseadas para el ambiente y las entidades en él. Este artículo presenta una aproximación novedosa para el modelado declarativo. La principal contribución es el uso de conocimiento previo sobre el ambiente y las entidades para asistir el proceso de la creación y validación del mundo virtual creado de ésta forma. Éste conocimiento incluye la información necesaria para permitir la evolución del mundo.

Palabras clave: Modelado declarativo, realidad virtual, explotación del conocimiento.

I. INTRODUCCIÓN

La Realidad Virtual se ha utilizado como método para simular entidades en el mundo real, de diferentes campos de la experiencia humana, como medicina, construcción, entretenimiento, y muchos otros. Ésta tecnología permite recrear casi cualquier tipo de escenario o mundo, y puede representar casi cualquier tipo de escena o situación. Se ha propuesto diferentes métodos para diseña y animar esos mundos virtuales.

Sin embargo, la mayoría de las veces esos mundos son creados un equipo multidisciplinario completo, compuesto por muchos artistas, modeladores e ingenieros, siendo necesarias desde algunas semanas hasta años para completar una visualización exitosa del ambiente deseado. Esto incluye el uso de herramientas especializadas, las cuales necesitan entretenimiento especial y muchas horas de práctica para obtener un resultado de apariencia profesional. La problemática de crear mundo virtuales se puede descomponer en dos partes: primero es la creación del ambiente, el segundo, describir la escena, es decir, como los personajes deben de evolucionar en el ambiente. En este trabajo tratamos con el primero de esos subproblemas, al tiempo que se ha conducido trabajo en el segundo subproblema [1].

Un método que permita la creación de escenarios virtuales tanto simples como complejos es un tema actual de investigación. El Modelado Declarativo es una aproximación a éste objetivo.

En éste artículo entendemos Modelado Declarativo como una metodología que permite crear un escenario virtual por medio de una descripción, escrita en un lenguaje lo más cercano al natural, donde se indica el tipo de ambiente, las entidades que lo poblarán, y como estarán distribuidas esas entidades. El Modelador Declarativo debe tomar

como entrada la descripción, y encontrar al menos una solución que satisfaga los deseos del usuario.
La principal deferencia de nuestro trabajo con otros es el uso de conocimiento sobre las propiedades de los objetos y otras entidades para auxiliar el proceso de validación de las posibles soluciones, y que el modelo resultante puede ser utilizado para manejar la evolución del escenario. Este modelo puede ser enviado a cualquier arquitectura subyacente para su visualización y animación, ya que toda información necesaria es provista con el modelo. La arquitectura también puede conducir la simulación del mundo virtual, pues también se incluye en el modelo las reglas para el ambiente y el comportamiento de las entidades. Este trabajo es parte del proyecto GeDA-3D [2], una arquitectura multi-agente distribuida en 3D. El objetivo de GeDA-3D es ofrecer una herramienta completa a cualquier usuario con la necesidad de simular un comportamiento dado.

II. Trabajo Relacionado

WordsEyes [3], desarrollado en los laboratorios AT&T por Bob Coyne y Richard Asproad, es el trabajo más cercano al nuestro. WordsEyes es un convertidor automático texto-a-escena, el cual utiliza marcadores de parte del discurso y analizadores estadísticos para obtener una descomposición jerárquica de las oraciones provistas por el usuario. El sistema utiliza diferentes herramientas para asignar *descriptores*, representaciones gráficas de bajo nivel que corresponden con cada concepto en la oración, y que proveen los características físicas a los elementos en el escenario. Se utiliza cinemática inversa para conseguir las poses de las entidades en la escena, y se emplean algunos métodos para resolver ciertas abstracciones, como textualización, caracterización o personificación. El sistema ésta basada en el Internet, de manera que el usuario solo tiene que entrar al sitio Web del proyecto, escribir una descripción corta, y enviarla para obtener una vista estática del escenario creado.
El proyecto DEM^{2}ONS [4] es un sistema multimodal compuesto por una interfaz modal y un modelador en 3D. La interfaz modal usa diferentes métodos de entrada, desde ratón hasta dispositivos hápticos, permitiendo al usuario crear el escenario de una manera intuitiva. Los módulos en cargo del análisis sintáctico y las interfaces dedicadas manejan las entradas, las cuales se traducen en eventos normalizados. El modelo resultante es validado a través de ORANOS [5], un resolvedor de restricciones diseñado para permitir expansibilidad en aplicaciones de modelado declarativo. El sistema se basa en una Interfaz Gráfica de Usuario o GUI, escrita en el Open Inventor Toolkit [6], y permite interacciones simples, como modificar los elementos no dinámicos presentados en el escenario.
CAPS [7] es un sistema basado en restricciones, orientado a resolver la disposición de objetos dentro de un escenario. El sistema permite el modelado de escenarios grandes y complejos, usando una entrada consistente de restricciones intuitivas de posicionamiento que pueden provenir de distintos métodos de entrada, como texto escrito o dispositivos hápticos. Puede manejar varios objetos a la vez, usando seudo-físicas para asegura la estabilidad de la posición, y emplea conceptos tales como fragilidad o interacción para asegurar la validez de la misma. Permite interacciones y está orientado hacia la manipulación de escenarios, pero no provee método alguno para la auto-evolución de la escena.

III. Modelado Declarativo

El modelado declarativo define un proceso recursivo necesario para crear una solución a las propiedades declaradas por el usuario. Aplicado a la creación de un escenario, el modelado declarativo es un proceso continúo en el cual retroalimentación proveniente de soluciones previamente obtenidas es usada recursivamente, hasta que el usuario está satisfecho con la solución obtenida. El método está compuesto por tres fases: Descripción, Generación y Vista [8].

Reconocimientos:
Esta investigación es parcialmente apoyada por el proyecto CoECyT-Jal No. 2008-05-97094, mientras el autor J.A. Zaragoza agradece el apoyo de lo beca CONACYT no. 190965.

En la fase de generación, el sistema recibe la descripción del usuario, escrita en un lenguaje de definición diseñado para la aplicación de modelado. Este lenguaje puede venir en una variedad de formas, ya sea simples entradas de texto a señales provenientes de dispositivos especializados o procesador del habla. El sistema valida la sintaxis de la entrada, la analiza, y crea una salida en un formato adecuado para el sistema, y entonces transfiere la información obtenida al siguiente paso. A continuación se procede a crear el modelo, normalmente iniciando con valores por defecto para las entidades en el mismo, y procediendo a asignar los valores para las propiedades usando distintos métodos para asegurar la consistencia de la semántica, la lógica y la posición de las entidades. Uno de los métodos más comunes es resolver un Problema de Satisfacción de Restricciones o CSP.

Podemos definir un CSP como un conjunto de variables $X = \{X_1 , X_2, \ldots X_n\}$, un dominio D el cual indica los valores posibles para variable en X, y un conjunto de restricciones $C = \{C_1, C_2, \ldots C_n\}$ el cual especifica un subconjunto de variables y los valores posibles para cada una de ellas.

Una vez que cada variable ha sido asignada con un valor de su dominio, de tal forma que no se viola ninguna restricción, se ha alcanzado una solución. Para un CSP dado puede existir más de una solución, de tal manera que se puede diseñar un algoritmo CSP de tal manera que pueda localizar varias soluciones [9].

El paso de modelado puede crear varios modelos y entonces proceder a validarlos, o puede iniciar con uno y entonces resolver cualquier conflicto hasta que se alcance una solución. Este proceso involucra métodos de rastreo y avance, permitiendo al sistema descubrir si ha alcanzado un mínimo o máximo local. En el primer caso, pueden existir algunas restricciones que en apariencia no pueden ser resueltas, pero al cambiar algunas de las propiedades de las entidades se puede llegar a un nuevo arreglo que las satisfaga todas. En el segundo caso, todas las restricciones se cumplen, pero la solución es rígida y no permite ninguna modificación sin romper alguna restricción

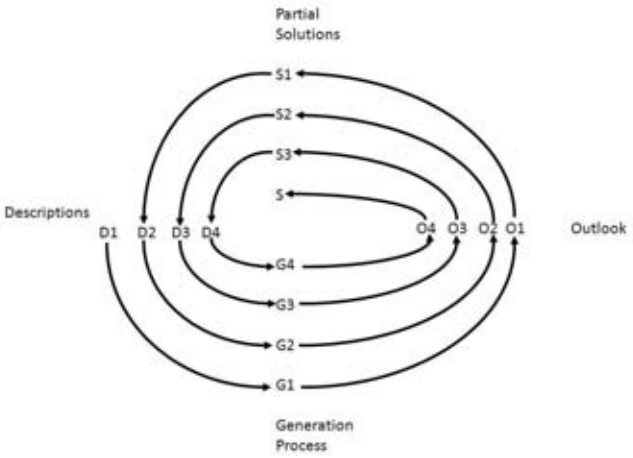

Figura 1. Proceso del Modelado Declarativo.

Nuevamente, algunos cambios pueden llevar a un nuevo conjunto de soluciones con valores más flexibles. Este paso puede incluir la participación del usuario, presentado las soluciones obtenidas y permitiendo al usuario decidir si alguna cumple con la imagen mental empleada para crear la descripción.

La fase de vista consiste en presentar al usuario las soluciones obtenidas. El modelo puede entonces ser modificado para acercarlo a la idea del usuario al iniciar el proceso, pero siempre dentro de las restricciones establecidas en el CSP. El usuario puede rechazar algunas o todas las soluciones encontradas, indicando así al sistema cual espacio de soluciones debe ser explorado para localizar futuras soluciones.

IV. UN MODELADOR DE AMBIENTES VIRTUALES BASADO EN CONOCIMIENTO

Nuestra propuesta tiene una característica principal, la cual no está completamente considerada en otros trabajos. Ya que el usuario expresa que es lo que debe ser posicionado en el escenario virtual, el sistema posee las indicaciones para las características y posición de cada elemento, las cuales pueden aplicadas dentro de ciertos rangos difusos. Por ejemplo, aún cuando la posición exacta de los elementos puede variar en la perspectiva de cada usuario, existe la certeza si es una referencia absoluta o relativa entre los elementos: “derecha” será siempre un área específica definida en relación a una entidad en

referencia, sin importar su orientación, tamaño o entidad en referencia, y puede ser definida dentro de una estructura que provee los parámetros para obtener esa área. Esa estructura de datos pude presentarse en la forma de una base de datos. Sin embargo, la información necesaria debe ser provisto en combinación de las relaciones entre los conceptos en evaluación, por lo cual una solución mejor adaptada son las bases de conocimiento. Ya que una base de conocimientos no solo contiene la información de los elementos en un dominio dado, sino también las relaciones entre conceptos, permite derivar nuevo conocimiento de la información ya presente, expandiendo el alcance de la base de conocimientos.

La acción de extrapolar nueva información de conocimiento actual es llamada *inferencia*, y es una característica útil para validar conceptos y propiedades, ya que el usuario puede comenzar expresando características simples, las cuales pueden ser combinadas para inferir conocimiento complejo.

Con esta idea, diseñamos un modelador declarativo que permite la creación de mundos virtuales a partir de una simple descripción en texto, usando la explotación de bases de conocimiento como base para el proceso de modelado. El modelador está formado por cinco módulos, como se muestra en la figura 2.

V. ANÁLISIS LÉXICO SEMÁNTICO

El Analizador Léxico-Sintáctico recibe la descripción escrita en un lenguaje definido a la medida, llamado VEDEL o Lenguaje para Descripción de Ambientes Virtuales [10]. VEDEL es parecido al lenguaje natural, completamente orientado a conducir el proceso de descripción. Provee una ayuda natural a los usuarios con su estructura para organizar las ideas al describir mundos virtuales. También permite crear escenarios de manera incremental, al añadir o modificar elementos específicos para extender la descripción. Una descripción en VEDEL se compone de tres secciones, Ambiente, Actores y Objetos, delimitados por etiquetas de sección. Cada una de esas secciones está formada por oraciones compuestas por declaraciones separadas por comas. Cada declaración corresponde a una propiedad para le entidad que está siendo descrita, y puede ser declaradas en cualquier orden, con una única restricción, la primera declaración debe corresponder al tipo de entidad, incluyendo un identificador opcional. Cada oración debe terminar con un punto (“.”).

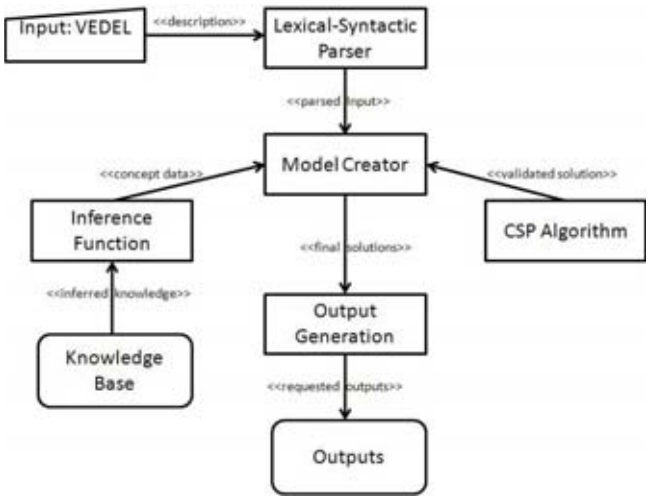

Figura 2. Arquitectura del Modelador

Las razones para definir un lenguaje de interacción como un subconjunto del lenguaje natural son: proveer un método estructurado simple y amigable para escribir y formatear la descripción, así como evitar la necesidad de métodos lógicos difusos y analizadores estadísticos requeridos en el análisis y validación del lenguaje humano diario. La estructura del lenguaje permite mantener un balance en el número de restricciones, evitando descripciones sobre o bajo-restringidas.

El analizador Léxico-Sintáctico es básicamente una máquina de estados que extrae elementos de la descripción, y los utiliza para formar una estructura de datos jerárquica que representa la información expresada en la descripción. La jerarquía está organizada con los tipos de entidad como las ramas superiores y sus propiedades como hojas, permitiendo una extracción de los datos por el módulo de análisis Léxico-Sintáctico, el cual solo verifica la composición correcta de la descripción y busca caracteres no permitidos, dejando la validación semántica para el módulo de Creación del Modelo.

```
[ENV]
house, night.
[/ENV]

[ACTOR]
man Antony, big, old, bald, sit BigCouch.
woman Brittany, young, near CenterTable.
dog Casper, sleep, front FirePlace.
[/ACTOR]

[OBJECT]
Couch BigCouch, against westWall, big,
facing east.
CoffeTable CenterTable, center
LivingRoom.
[/OBJECT]
```

Figura 3. Ejemplo de VEDEL.

VI. CREACIÓN DEL MODELO

El Creador del Modelo recibe la entrada analizada por el Analizador Léxico-Semántico, y procede a generar el modelo.

El modelador trabaja en pasos incrementales, iniciando con un modelo básico instanciado con valores por defecto para los elementos del ambiente virtual, asignando posteriormente los valores solicitados por el usuario.

La creación del modelo termina cuando todas las propiedades han sido instanciadas y éstas no violan ninguna restricción.

Primero, el modelador obtiene toda la información sobre el ambiente. Cualquier petición por información es manejada a través del componente Función de Inferencia, el cual accede directamente a la base de conocimientos. La función de inferencia es un singleton, y es usado a través de todo el proceso de obtención de información, formateando la información para la tarea de modelado y validando las peticiones hechas en la descripción. La información reunida desde la base de conocimientos sobre el ambiente es usada para construir las reglas del mundo virtual y asignar valores a sus propiedades. Si el ambiente posee alguna construcción o elemento especial, el modelador construye las entradas necesarias en el modelo para satisfacer esas indicaciones. Zonas específicas como paredes, puertas o áreas, o elementos propios del ambiente (amueblado, vegetación) son generados de ésta manera. Las zonas específicas corresponden a información implícita sobre el ambiente y no tienen una representación visual explícita, mientras que los elementos propios del ambiente deben ser instanciados, colocados y representados como elementos individuales en el ambiente.

Una vez que el ambiente ha sido creado, el modelador continúa con las entidades que lo poblarán. Cada una de esas entidades es creada usando representaciones básicas instanciadas con valores por defecto almacenados en la base de conocimientos. Tales presentaciones son llamadas Avatares. Cuando se le solicita que cree una entidad en particular, primero verifica si el avatar correspondiente existe. En caso contrario, se hace la solicitud a la base de conocimientos para obtener la información de la entidad, y el avatar se añade al modelo. El avatar es entonces instanciado con los valores de la entidad en proceso. Cada petición es validada primero a través del avatar, para verificar si la propiedad es válida para la entidad. Posteriormente, se verifica a través de la función de inferencia si los valores asignados son correctos o corresponden a la propiedad en proceso. Cualquier conversión es conducida a través de la base de conocimientos. Por ejemplo, si el usuario solicita una silla coloreada en rojo, el modelador primero verifica que “silla” posea la propiedad “color”. Si la propiedad es validad, se procede a verificar que si uno de los posibles valores para la propiedad “color” de “silla” es “rojo”. Si todo resulta positivo, la función de inferencia retorna la información correspondiente al color “rojo” en el formato RGB.

POSICIONADO LAS ENTIDADES

Una vez que todas las entidades han sido creadas y sus propiedades instanciadas con los valores especificados en la descripción, el modelador procede a posicionar cada entidad en la posición correcta, de acuerdo a las declaraciones hechas en la descripción. Las entidades son colocadas primero en una posición por defecto, almacenada en la base de conocimientos. El proceso para obtener los parámetros para posicionar la entidad

es similar al usado para obtener los valores de las propiedades. La declaración es validada directamente a través de la función de inferencia, la cual recibe la información correspondiente a la posición actual de la entidad, su tamaño y su orientación, y, en el caso de posicionamiento relativo, también se envía los valores de la entidad en referencia. Los parámetros correspondientes al concepto son obtenidos de la base de conocimientos y posteriormente instanciados con los valores de la entidad o entidades. Esos nuevos parámetros son enviados al modelador, el cual los asigna a las propiedades de la entidad. La entidad en referencia puede ser cualquier entidad en el ambiente, ya sea el ambiente mismo o una zona específica. Los parámetros del concepto pueden ser definidos con parámetros difusos, de manera que el proceso de modelado presente cierta aleatoriedad, para permitir la generación de diferentes modelos durante el proceso.

Ya que cada entidad ha sido posicionada, el modelador envía el estado actual del modelo al componente CSP. Este componente verifica la posición de cada entidad, esto es, que no existan colisiones entre las entidades y que la posición de cada una de ellas corresponda a las declaraciones hechas en la descripción. Para cumplir esas tareas, cada entidad posee una seria de marcas de colisión, así como marcas de puntos característicos. Esas maracas especializadas están almacenadas en la base de conocimientos, y son instanciadas cuando la entidad es creada. Cuando el modelador cambia la posición de la entidad, esas marcas también son actualizadas.

Las marcas de colisión están definidas como una seria de esferas que envuelven toda la geometría de la entidad (figura 4), y son usadas como la restricción principal a ser satisfecha por el algoritmo CSP. Para ejecutar la verificación, el sistema primero verifica si la distancia euclidiana entre cualquier para de entidades es menor que la diagonal mayor correspondiente a una caja formada por las medidas de la entidad. Si esto es verdadero, se evalúa la ecuación de distancia entre dos esferas para cada par posible de marcas de colisión entre las dos entidades. Esta ecuación debe ser satisfecha para cualquier par de marcas de colisión entre entidades en conflicto, $C_1 = (x_1 , y_1 , z_1 , r_1)$ y $C_2 = (x_2 , y_2 , z_2 , r_2)$, de manera que la relación siguiente se mantenga:

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 - (r_1 - r_2) >= t \quad (1)$$

Si cualquier par falla ésta evaluación, esto es, el resultado de la ecuación (1) es menos que un rango ($t \in \Re$, $1 > t > 0$) establecido por el administrador del sistema, una colisión ha sido detectada y el modelo debe ser modificado. Para realizar los cambios en la posición de las entidades, primero se realizan algunas verificaciones: si una de las entidades es un "*pivote*" (esto es, está asignada a una posición absoluta, como *norte*, *sur*, o *este*), el tamaño de las entidades, y la relación entre ellas.

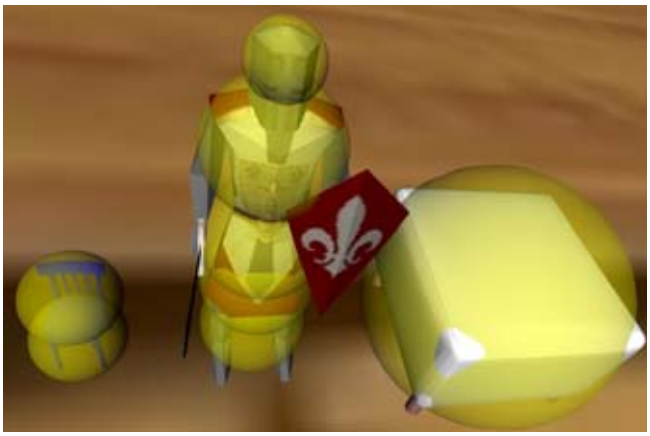

Figura 4. Marcadores de colisión.

Las entidades pivote son modificadas al final, y sólo si el modelo requiere de tales cambios para obtener una solución.

Enseguida, las relaciones entre elementos son exploradas, ya que algunos de los conceptos de posicionamiento requieren que dos entidades entren en contacto, tales como "contra", "dentro" o "sobre", de manera que en esos casos la colisión será desechada y otras validaciones tendrán lugar. En otros casos, las entidades que hagan referencia a otra serán movidas primero. Si la configuración del modelo no puede ser validada, la entidad en referencia será entonces movida. En otros casos, ambas entidades hacen referencia a una tercera entidad, en cuyo caso el algoritmo CSP hace uso de una técnica similar a la usada en los Sistemas-L [11] para posicionar ambas entidades en una nueva

locación que cumpla con las declaraciones hechas en la descripción mientras se resuelven el conflicto. Finalmente, el tamaño es considerado para decir cual entidad debe ser movida primero. Las entidades más pequeñas tienen mejor probabilidad de ser colocadas en una posición válida que los elementos grandes, por lo cual el sistema moverá primero las entidades más pequeñas en conflicto.

La verificación de colisiones es ejecutada hasta que no existen más conflictos. Si el modelo actual cae en un mínimo local, o no se puede encontrar una solución, el sistema restaura una solución parcial previa que contenga menos conflictos y procede a generar nuevas soluciones. Si la solución restaurada es el modelo inicial enviado por el creador de modelo, el sistema verifica si todas las entidades posibles han sido reasignadas, y reinicia el proceso. Si ya no existen más entidades a mover, el sistema informa que no se puede encontrar una solución.

Para posicionar exitosamente una entidad, la siguiente validación debe ser satisfecha: al menos uno de sus puntos característicos debe estar dentro del volumen de un elipsoide o un paraboloide instanciadas con los parámetros correspondientes a la información de la entidad en referencia y a los valores del concepto asociado. Esto es, para cada entidad en el ambiente, al menos uno de sus puntos característicos C = (x, y, z) debe estar en la superficie o dentro de un volumen de validación E = (dx, dy, dz), para elipsoides (3), o P = (p, q), para una paraboloide (2), instanciadas con los valores de la entidad en referencia. Podemos representar esto como:

$$\left(\frac{x}{p}^2\right) + \left(\frac{y}{q}^2\right) - 2z <= 0 \qquad (2)$$

$$\left(\frac{x}{dx}^2\right) + \left(\frac{y}{dy}^2\right) + \left(\frac{z}{dz}^2\right) - 1 <= 0 \qquad (3)$$

Los parámetros para ambas ecuaciones están almacenados en la base de conocimientos, como parte del concepto de posicionamiento o como parte de la entidad en referencia, en cuyo caso también se pueden incluir la cantidad de puntos característicos que necesitan estar dentro del volumen para validar la posición, o incluso la lista de puntos que deben ser evaluados. Las posiciones relativas son validadas siempre a través de volúmenes de paraboloide, para los cuales los parámetros p y q corresponden al tamaño de la entidad en referencia, y z es un valor definido por el administrador del sistema.

Relaciones espaciales tales como “contra” o “sobre” son verificados usando volúmenes elipsoidales, para los cuales unos de sus parámetros es menos a 1, y localizados sobre las superficies de las entidades. Para posiciones que requieren un volumen dentro de una entidad, o una posición en un área dada tal como zonas específicas, el volumen elipsoidal es instanciado con los valores de la entidad o la zona.

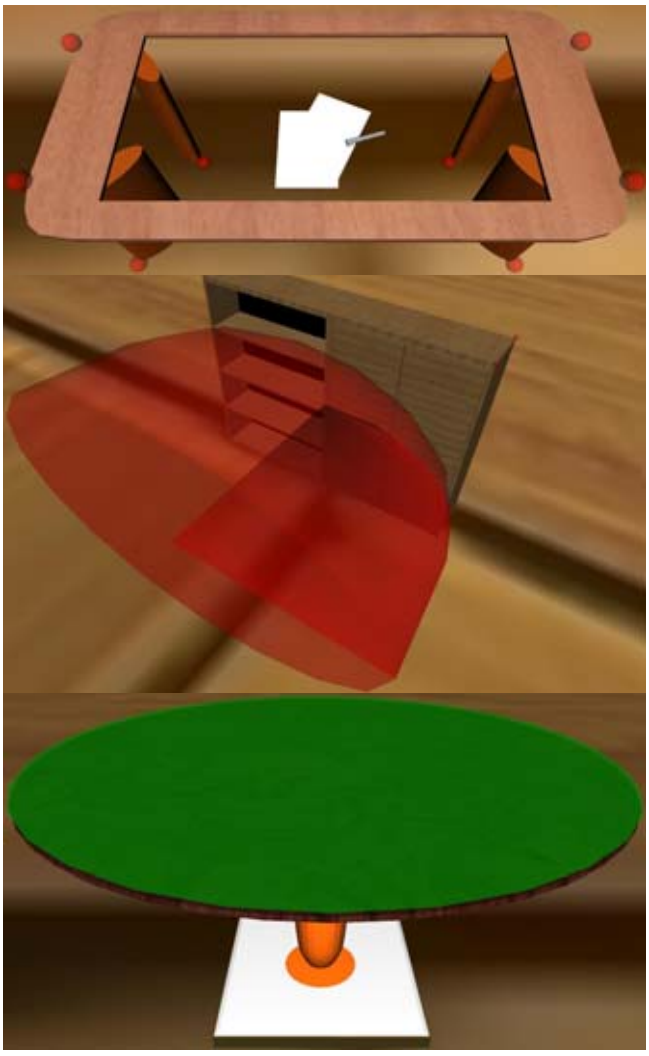

Figura 5. Puntos característicos y volúmenes de posicionamiento

Si la validación de la posición de una entidad falla, se calcula una nueva posición. Antes de asignar ésta nueva posición, se revisa una lista de posiciones previas para la entidad en evaluación. Si el vecindario de esas previas posiciones forma un grupo compacto, entonces una nueva posición fuera de ese grupo es asignada, y la validación en conducida nuevamente. Si la nueva posición no es válida, o no puede ser asignada fuera del grupo, el sistema no puede encontrar una nueva posición para la entidad y regresa a un estado del modelo previo. Al igual que con la verificación de colisiones, si el modelo previo es el estado inicial, la descripción solicitada no puede ser resuelta.

Cada entidad en el modelo almacena la lista de elementos con los cuales comparte alguna relación. De ésta manera, cada entidad puede seguir los movimientos hechos al modelo, y modificar sus propios parámetros en consecuencia, por ejemplo, si una entidad debe orientar en relación a otra, y la entidad en referencia es movida, la primera entidad puede ajustar sus parámetros de orientación para ajustarse a la nueva posición establecida para la entidad en referencia. Lo mismo aplica para posiciones relativas. Ya que cada entidad ha pasado ambas validaciones, el modelo es enviado al componente final, el Generador de Salida.

GENERANDO LAS SALIDAS

El componente de Generación de Salida recibe el modelo final. Este componente utiliza la metodología Controlador Vista-Modelo para generar las estructuras de datos de salida y los archivos necesarios para que la arquitectura subyacente pueda representar las acciones necesarias para crear la visualización del ambiente virtual, y conducir el proceso que activará la simulación. Estas salidas utilizan plantillas que reciben la información del modelo, y a través del MCV son instanciadas con los valores obtenidos a través del proceso de modelado. Este método permite la fácil modificación de los parámetros de salida, de manera que los usuarios pueden agregar o modificar la información enviada a la arquitectura subyacente, que puede ser también un visualizador 3D de cualquier tipo, en tanto las plantillas mantengan los estándares utilizados para la visualización en 3D. La cantidad de salidas puede ser establecida en la base de conocimientos.

VII. TRABAJO PRESENTE

Actualmente contamos con un prototipo del modelador en el lenguaje Java, esto con la finalidad de proporcionarle características multi-plataforma. La base de conocimientos ha sido definida usando el sistema Protégé y fue creada utilizando el estándar OWL [12], seleccionada debido a su expansibilidad sobre la Internet. A continuación se muestran algunos ejemplos obtenidos usando la versión más reciente del modelador, para cuya visualización se utiliza el estándar X3D y un visualizador basado en X3D.

En el primer ejemplo, figura 6, “LivingRoom” es un concepto almacenado en la base de conocimientos que también establece el amueblado del escenario y la posición de cada elemento. El segundo ejemplo, figura 7, muestra diferentes modelos de casa obtenidos de una sola descripción. “House” establece el ambiente completo, incluyendo la mayoría del amueblado y las áreas en referencias para colocar a los actores. El último ejemplo, figura 8, muestra el mecanismo de resolución de conflictos para varios objetos colocados en la misma posición relativa.

VIII. CONCLUSIONES

En este artículo hemos descrito nuestra novedosa aproximación para el modelado declarativo utilizando bases de conocimiento para asistir en el proceso de generación del modelo y vista. Hemos probado nuestra propuesta por medio de un prototipo aún en evolución. Hasta ahora, hemos creado diferentes escenarios con elementos simples, generando incrementalmente entidades complejas. También hemos probado la facilidad para modificar los parámetros y conceptos, utilizando diferentes valores para el mismo modelo, alcanzando modelos significativamente diferentes a través de varias iteraciones del modelo. Otros trabajos tratan con la metodología del modelado declarativo, pero a partir de la

[ENV]
LivingRoom.
[/ENV]
[ACTOR]
ManSuit John.
YoungWoman Sarah.
[/ACTOR]
[OBJECT]
[/OBJECT]

Figura 6. Ejemplo 1.

literatura disponible podemos afirmar que ninguno está orientado a crear mundos dinámicos y controlados por el usuario.

Aún cuando sólo tuvimos acceso a uno de ellos, WordsEye, es aparente que todos auxilian en el proceso de crear mundos complejos, pero la interacción con el usuario es limitada a la creación del escenario, y cambiar la vista con que se presentan los resultados. En el caso de WordsEye, su naturaleza basada en red no permite la modificación directa de los modelos usados para la representación en 3D, y la vista y calidad de la misa está limitada por la capacidad del servidor del sistema. En caso de DEM2ONS y CAPS, la literatura de ambos apunta que los métodos de entrada son muy especializados, y la salida está limitada a la visualización del modelo, sin interacción alguna. El modelador presentado en este artículo permite crear, a través de una descripción escrita en un lenguaje cercano al natural, un escenario en 3D, así como las estructuras de datos necesarias la evolución de una escena en 3D.

[ENV]
house.
[/ENV]
[ACTOR]
Knight, anywhere Kitchen.
YoungWoman, anywhere Garden.
woman, anywhere Livingroom
[/ACTOR]
[OBJECT]
Chair, front woman0.
Puff, front bed0.
[/OBJECT]

Figura 7. Ejemplo 2.

La principal desventaja de esta investigación es que la base de conocimientos debe contener todos los conceptos así como la necesidad de una base de objetos 3D. Sin embargo, una vez que ambos aspectos han sido cumplidos, la creación de escenarios en 3D es posible para usuarios finales. Con esto se alcanza el objetivo principal de ésta investigación. El trabajo futuro incluye una

herramienta amigable para manejo del conocimiento y extensión de la base de objetos 3D, así como tratar con casos específicos que puedan requerir de métodos de modelado especiales.

[ENV]
void.
[/ENV]
[ACTOR]
Knight, center.
[/ACTOR]
[OBJECT]
CenterTable Table, front Knight0.
Chair, left Table.
Chair, left Table, color red
Chair, left Table, color blue
Chair, left Table, color gray
Chair, left Table, color green
facing Knight0.
[/OBJECT]

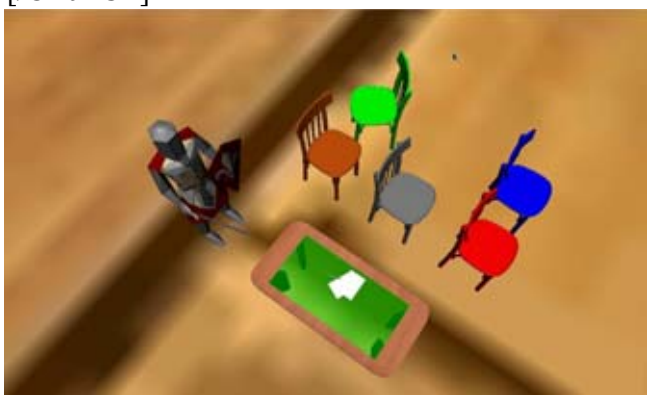

Figura 8. Ejemplo 3.

IX. REFERENCIAS

[1] F. Ramos, F. Zúniga, and H. I. Piza, "*A 3D-space platform for distributed applications management*," International Symposium and School on Advanced Distributed Systems 2002.Guadalajara, Jal., México, Noviembre 2002.

[2] P. Hugo, F. Zúniga, and F. Ramos, "A *platform to design and run dynamic virtual environment*," International Conference on Cyber Worlds, Tokio Japón, pp. 18–20, Noviembre 2004.

[3] B. Coyne and R. Sproat, "*Wordseye: An automatic text-to-scene conversion system*", en SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. AT&T Labs Research, 2001, pp. 487–496.

[4] O. Le Roux, V. Gaildrat, and R. Caubet, "*Design of a new constraints solvers for 3D declarative modeling*" en International Conference on Computer Graphics and Artificial Intelligence (3IA), Limoges, 03/05/200-04/05/200, mayo 2000, pp. 75–87.

[5] G. Kwaiter, V. Gaildrat, and R. Caubet, "*DEM2 ONS: A high level declarative modeler for 3D graphics applications*," en Proceedings of the International Conference on Imaging Science Systems and Technology, CISST'97, 1997, pp. 149–154.

[6] D. Wang, I. Herman, and G. J. Reynolds, "*The open inventor toolkit and the PREMO standard*," Amsterdam, Holanda, Rep. Tec, 1996.

[7] K. Xu, "*Constraint-based automatic placement for scene composition*," en In Graphics Interface, 2002, pp. 25–34.

[8] V. Gaildrat, "*Declarative modeling of virtual environment, overview of issues and applications*," en International Conference on Computer Graphics and Artificial Intelligence (3IA),Atenas, Grecia, vol. 10. Laboratoire XLIM - Université de Limoges, mayo 2007, pp. 5–15.

[9] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach. Pearson Education, 2003.

[10] J. A. Zaragoza Rios, "*Representation and exploitation of knowledge for the description phase of declarative modeling"*, CINVESTAV-PNl, Unidad Guadalajara, Guadalajara, México, Septiembre 2006.

[11] P. Prusinkiewicz and A. Lindenmayer, The algorithmic beauty of plants. Springer-Verlag New York, Inc., 1990.

[12] D. L. McGuinness and F. van Harmelen, "*OWL Web Ontology Language overview*," World Wide Web Consortium, W3C Recommendation, Febrero 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/.

Creation of Virtual Worlds Through Knowledge-Assisted Declarative Modeling

Jaime Zaragoza, Félix Ramos, and Véronique Gaildrat
Centro de Investigación y de Estudios Avanzados, Unidad Guadalajara, México
Institut de Recherche en Informatique de Toulouse, France
email: {jzaragoz, framos}@gdl.cinvestav.mx, veronique.gaildrat@irit.fr

KEYWORDS
Game Design, Game Engine Design, Game Environment Creation, Knowledge Bases, Distributed Systems

ABSTRACT

Creating virtual worlds using dedicated tools is a time and resource consuming task. Even experienced user may find difficult to create virtual representations for different needs e.g. for simulations of real world, for recreations of fantansy or ancient worlds, for teaching, etc. Through the use of the declarative modeling method an user can create a virtual scenario by simply stating some of the properties for the environment and for the entities. This paper presents a novel approach for declarative modeling. Main contribution is to use knowledge about entities conforming the environment to assist the processes of creation and validation of the virtual world created in this way. This knowledge includes the necessary data to aleviate great part of the processing needed to create the environment and the knowledge needed to allow the evolution of the environment in a dynamic scene. The main difference with other works is that the virtual world created using our proposal allows its evolution.

Introduction

Virtual Reality has been used as a method for simulating entities in the real world in different fields of human expertise, such as medicine, construction, entertainment, and many others. This technology allows to created almost any kind of scenario or world, and it can perform almost any kind of scene or situation. Different approaches have been proposed to design and animate such virtual worlds. However, most of the time, these worlds are created by a full multidisciplinary staff composed by many artists, modelers and engineers, taking from a few weeks to years to complete a successful visualization of the desired environment. Even more, the team uses specialized tools, which need special training and many practice hours to create an outcome of professional quality. We think this is the main problem to not use more broadly 3D as output interface of many systems.

The full problem can be split in two subproblems. The first subproblem lies in creating the environment; the second is describing the scene, that is how characters must evolve in the environment. In this work we deal with the first one of these subproblems. Some work was carried out in our team to deal with the second problem (Ramos et al. 2002).

A method allowing the creation of virtual scenarios both simple and complex is currently a topic of research. Declarative Modeling is one approach to reach this objective. In this work Declarative Modeling must be understood as a methodology that allows creating a virtual scenario by means of a declaration in a language (desirably natural) of how we expect entities in the scenario must be arranged. The Declarative Modeler system must take as input this description and find at least one solution satisfying the user needs.

The main differences between our approach and those presented previously are: first, the use of knowledge about properties of objects and other entities provided by the user to help the validation process of possible solutions proposed by the modeling system; second, that the resulting model is useful to manage all kind of animations required by the evolution of a scene. The model created in this way can be sent to any underlying architecture for its visualization and animation, since all the necessary data is provided with the model. The architecture can also perform the simulation of any possible virtual world, because the resulting model includes the environment's rules and possible entities behaviors.

This project is part of the GeDa-3D project (Hugo et al. 2004), a distributed multi-agent architecture for 3D. GeDA-3D objective is to offer a complete tool to any final user with the need to simulate some given situation. The content of this article is: section 2 presents related works; section 3 presents the Declarative Modeling process; section 4 presents our approach to Declarative Modeling; section 5 presents the state of this research; section 6 presents our conclusions about objective and results obtained.

Related Works

WordsEyes (Coyne and Sproat 2001), developed at the AT&T Laboratories by Bob Coyne and Richard Asproad, is the closest project to ours. WordsEyes is an automatic text-to-scene converter, which works us-

ing part-of-the-speech markers and statistical analyzers to obtain a hierarchical decomposition of the sentences provided by the user. The system uses different tools to assign the *depictors*, low-level graphical representations, to each concept in the sentence, and to provide the physical characteristics to the elements in a scene. Inverse kinematics are use to achieve poses for entities in the scene, and some methods are employed to solve certain abstractions, such as textualization, characterization or personification. The system is web-based, so the user just has to enter to the project's website, write a short description, and summit it to obtain a static visualization of the created scenario.

The project DEM^{2}ONS (Le Roux et al. 2000) is multimodal system composed by a modal interface and a 3D modeler. The modal interface uses different input methods, from mouse to haptic devices, allowing the user to create the scenario in an intuitive form. Modules in charge of syntactic analysis and dedicated interfaces handle the inputs, which translate them into normalized events. The resulting model is validated with ORANOS (Kwaiter et al. 1997), a constraint solver designed to allow extensibility in declarative modeling applications. The system is supported by a GUI, written with the Open Inventor Toolkit (Wang et al. 1996), and allows simple interactions, that is, the modification of the non-dynamic elements presented in the scenario.

CAPS (Xu 2002) is a constraint-based system, oriented to solve the placement of objects inside a scenario. The system allows the modeling of big and complex scenarios, using an input consisting of intuitive positioning restrictions that can come from several input methods e.g. writing text to haptic devices. It can handle several objects at once, using pseudo-physics to assure the stability of the position, and employing concepts such as fragility or interaction to assure the validity of the position. Allows interactions and is oriented towards scenario manipulation, but don't provide any method for scene self-evolution.

Declarative Modeling

Declarative modeling defines a recursive process in order to create a solution to the properties stated by the user. Applied to the creation of a scenario, the declarative modeling is a continuous process in which feedback coming from previously obtained solutions is used recursively, until the user is satisfied with the outcome solution.

The method is composed by three steps: Description, Generation and Look Up (Gaildrat 2007). In the generation phase, the system receives the user's description, written in a custom-tailored definition language for the modeling application. This language can come in a variety of forms, from simple text inputs to signal coming from haptic hardware or speech processors. The system validates the input's syntax, parses it, creates a formatted output fitted for the system, and then transfers the information obtained to the next step.

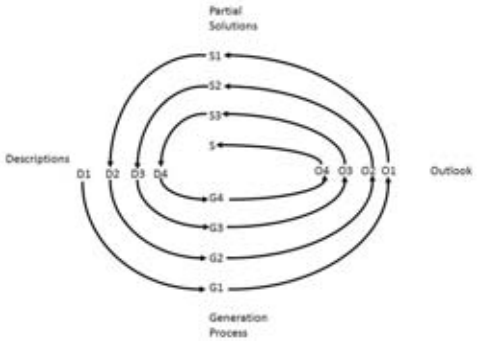

Figure 1: Declarative Modeling Process.

The system then continues to create the model, normally starting with default values for the entities in the model, and then proceeds to assign the properties values using different methods to assure the consistency of the semantic, the logic, and the positioning of the entities. One of the most common methods is solving a Constraint Satisfaction Problem or CSP.

We can define a Constraint Satisfaction Problem as a set of variables $X = \{X_1, X_2, \ldots X_n\}$, a domain D which indicates the possible values for each variable, and a set of constraints $C = \{C_1, C_2, \ldots C_n\}$ which specifies a subset of variables and the possible values for each of these variables. When each variable has been assigned with a value from its domain, such that no constraint is violated, a solution has been found. For a given CSP, several solutions can exist, so the CSP algorithm can be designed so it can provide several solutions (Russell and Norvig 2003).

The modeling step can create several models and then proceed to validate them, or start with one and then solve any conflict until a solution is reached. This process involves backtracking and step forward methods allowing the system to discover if a local minimum or maximum have been reach. In the first case, there can be some restrictions that apparently can not be solve, but changing some of the entities properties can lead to a new arrangement that satisfy all of them. In the second case, all the constraints are fulfilled, but the solution is stiff and doesn't allow any modification without braking some restriction. Again, some changes can lead to a new set of solutions with more flexible settings. This step may include the participation of the user, presenting the solutions obtained and then letting the user to decide if they accomplish with the mental image used to generate the description or not.

The out look step involves presenting the user with the solution or solutions obtained. The model then can be modified in order to make it closer to the idea the user have in mind when the process started, but always within the restrictions established in the constraints set.

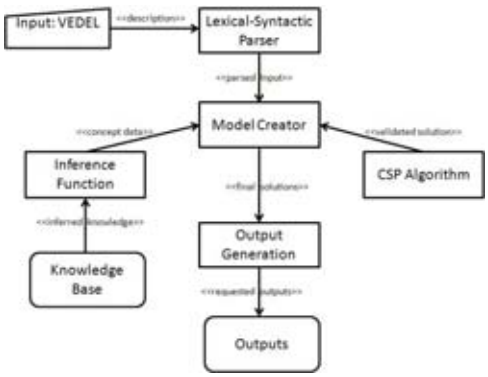

Figure 2: Modeler Architecture.

The user can reject some or all solutions, thus indicating the system which solution space should be explored to find future solutions.

A Virtual Environment Modeler Based on Knowledge Exploitation

Our proposal has a central feature, which is not fully considered in other works. Since the user is expressing what should be placed in the virtual scenario, the system has the indications for the characteristics and position of each element, which can be applied within certain fuzzy thresholds. For example, even when the exact location of the elements can vary in the perspective from user to user, there is certainty when it is referred to absolute or relative positioning between the elements: "`right`" will be always an specific area defined in relation to an entity in reference, not matter its orientation, size, or position, and can be defined within a data structure that provides the parameters for obtaining such area. Such data structure can come in the form of a database. However, the necessary information should be provided with the inclusion of the relationships between the concepts in evaluation, a more suitable solution is the knowledge bases. Since a knowledge base not only contains the information of the elements for any given domain, but also the relationships between these concepts, it allows to derive new knowledge from the data already present, expanding the reach of the knowledge base. The action of extrapolating new information from current knowledge is called inference, and is a useful characteristic for validating concepts and properties, since the user can begin stating simple characteristics, which can be combined to infer complex knowledge.

With this idea, we design a declarative modeler aimed to allow the creation of virtual worlds from simple text descriptions, using knowledge bases exploitation as the base of the modeling process. The modeler is composed by five modules, as presented in figure 2.

[ENV]
house, night.
[/ENV]

[ACTOR]
man Antony, big, old, bald, sit BigCouch.
woman Brittany, young, near CenterTable.
dog Casper, sleep, front FirePlace.
[/ACTOR]

```
[OBJECT]
Couch BigCouch, against westWall, big, facing
east.
CoffeTable CenterTable, center LivingRoom.
[/OBJECT]
```

Figure 3: VEDEL example.

Lexical-Syntactic Parsing

The Lexical-Syntactic Parser receives the description written in a custom-defined language called VEDEL or Virtual Environment Description Language (Zaragoza Rios 2006). VEDEL is like natural language, completely oriented to lead the description process. It provides a natural help to users with its structure to organize the ideas for describing virtual worlds. VEDEL allows creating the scenarios incrementally, by just adding objects or modification in specific sections in order to extend the description. A description in VEDEL is composed by three sections: Environment, Actors and Objects, allowing of which are the main components in the creation of the scenario. Each of these sections is in turn formed by sentences composed by comma-separated statements. Each statement corresponds to a property for the entity being described, and can be stated in any order, with the only limitation that the first statement must be the type of entity, including an optional identifier. Each sentence must end with a dot (" . "). Sections must be surrounded by section marks, as show in figure 3. The reasons for defining an interaction language as a subset of the natural language are to provide a simple, user friendly structured method to write and format the description, but also to avoid the need of fuzzy logic methods and statistical parsers needed for the analysis and validation of every-day human language. The language structure allows keeping the model with balanced constraint amount, avoiding over or under constrained descriptions.

The Lexical-Syntactic Parser is basically a state machine, which extracts tokens from the description, and uses them to form a hierarchical data structure representing the information stated in the description. The hierarchy is organized with entity types as the upper branches, with the properties as their leaves, allowing

an easy and quick extraction of the data by the Lexical-Syntactic Parsing module, which only verifies the correct composition of the description, and searches for non-valid characters, leaving the semantics validation for the Model Creation module.

Model Creation

The Model Creator receives the parsed entry from the Lexical-Syntactic Parser, and proceeds to create the model. The modeler works in incremental steps, starting with a basic model with default values assigned to elements of the virtual environment, and assigning the values requested by the user. The model creation finish when all the properties have been assign and these properties do not violate any constraints.
First, the modeler obtains all the information about the environment. Any request for information is handled through the Inference Function component, which accesses directly to the knowledge base. The inference function is a singleton, and is used through all of the process of obtaining information, formatting the data for the modeling tasks and validating the requests made in the description.
The information gathered from the knowledge base about the environment is used to construct the virtual world's rules and assign data type values to its properties. If the environment has any special constructions or elements, the modeler creates the necessary entries in the model to satisfy these indications. Landmarks such as walls, doors or specific areas, or furniture in the environment are all created this way. Landmarks correspond to implicit information about the environment and have not explicit visual representations, the second sort of descriptions corresponds to objects that must be instantiated, placed, and represented as individual elements in the environment.
Once the environment has been set, the modeler continues with the entities that will dwell the scenario. Each of these entities is created using basic representations instantiated with default values from the knowledge base. Such representations are called *avatars*. When the modeler is request to create a given entity, it verifies that the corresponding avatar exists. If not, the knowledge base is queried for the information on the given entity, and the avatar is added to the model. The avatar is then instantiated with the values requested for the entity in process. Each of the requests is validated first through the avatar, to verify if the entity spots the property requested. Then, the Inference Function is used to validate that the values to be assigned are correct, or correspond to the property in question. Any necessary conversion is carried through the knowledge base. For example, if the user request a chair colored red, the modeler first verifies that "`chair`" has the property "`color`". If true, the Inference Function is queried to validate that one of the possible values for the "`color`" property of "`chair`" is "`red`". If the response is positive, the Inference Function will return the data corresponding to the color "`red`" in RBG format.

Positioning the Entities

Once all the entities have been created and its properties instantiated with the values specified in the description, the modeler proceeds to position all of them in the correct location, according to the description statements.
The entities are first placed using a default position stored in the knowledge base. The process to obtain the parameters to position the entity is similar to the used to obtain property values. The statement is validated directly through the Inference Function, which receives the information corresponding to the entity's current position, size and orientation, and in the case of relative positioning, the referenced entity's values are also sent. The parameters corresponding to the concept are obtained from the knowledge base, and then instantiated with the entity or entities' values. These newly obtained parameters are sent to the modeler, which assigns them to the entity's properties. The referenced entity can be any other entity in the environment, the environment itself, or a landmark. The concept parameters can be defined with fuzzy parameters, so the modeling process achieves certain randomness, to allow the generation of different models through the process.
After every entity has been positioned, the modeler sends the model's current state to the CSP component. This component validates the position of every entity; that is, there are no collisions between the entities and that the position of each of them corresponds to the statements in the description. To accomplish these tasks, each entity has a series of collision tags, as well as characteristic points marks. These specialized tags are stored in the knowledge base, and are instantiated when the entity is created. When the modeler changes the entity's position, these tags are also updated.
The collision tags are defined as a series of spheres that wrap all of the entity's geometry (figure 4), and are used as the primary constraint to be satisfied by the CSP Algorithm. To carry out the collision verification process, the system first verifies if the Euclidean distance between any given pair of entities is smaller than the longest diagonal corresponding to a box formed by the entity's measures. If true, the distance equation for two spheres is evaluated for all the possible pairs of collision tags between the two entities in conflict. This must be satisfied for any pair of collision tags corresponding to different entities, $C_1 = (x_1, y_1, z_1, r_1)$ and $C_2 = (x_2, y_2, z_2, r_2)$, the following relationship must be kept:

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 - (r_1 - r_2) >= t \quad (1)$$

If any pair fails this evaluation, that is, the result of the

Figure 4: Collision tags examples.

evaluation is less than a threshold ($t \in \Re, 1 > t > 0$) set by the system administrator, a collision is detected and the model must be modified. To conduct the changes in the positioning of the entities, previous to any change, some considerations are verified: If one of the entities is a "*pivot*" (this is, is fixed to an absolute position, i.e. *north*, *south*, *east*), the size of the entities, and the relationships between them. Pivot entities are moved at the end, and only if the current model requires such changes to obtain a solution. Next, the relationships between elements is explored, since some of the possible positioning concepts require that two elements come in contact, such as "`against`", "`inside`" or "`over`", so in these cases the collision will be dismissed and other validations will take place. Otherwise, if one of the entities in conflict makes reference to the other, this must be moved first. If the model configuration cannot be validated, the referenced entity will be then moved. In other cases, both entities make reference to a third entity, in those cases the CSP Algorithm makes use of a technique similar to one the used with L-Systems (Prusinkiewicz and Lindenmayer 1990) to position both entities in a new position that fulfills the statements from the description while solving the conflict. Finally the size is considered to decide which entity should be moved first, if there is not relation between the entities. Smaller entities have a better chance to be moved into a valid position than big elements, therefore the system will move first the smaller entities in conflict.

Collision verification conducted until no more conflicts are found. If the current model falls into a local minimum, or cannot find a solution, the system restores a partial previous solution in which less conflicts where found; then proceeds to create new solutions. If the restored solution is the initial model sent by the Model Creator, the system verifies if any possible entity has been repositioned, and restarts. If there are no more entities to move, the system informs that cannot find a solution.

Once it was verified that there are no collisions among entities, positioning validation is carried over. This validation uses the characteristic points marks, which indicate the most prominent points in the entity's geometry, allowing quick verification without resorting to full geometry verification. Positioning validation is conducted over all entities, in order to assure that the current position satisfy the description statements.

In order to successfully locate successfully an entity, the following validation must be satisfied: at least one of its characteristic points must be located inside the volume of an ellipsoid or a paraboloid instantiated with the parameters corresponding to the referenced entity's data and the concept values. That is, for any given entity, at least one of its characteristic point $C = (x, y, z)$ is in the surface or inside a validation volume $E = (dx, dy, dz)$, for ellipsoids (3), or $P = (p, q)$, for a paraboloid (2), instantiated with the values from the referenced entity. We can represent this as:

$$\left(\frac{x}{p}^2\right) + \left(\frac{y}{q}^2\right) - 2z <= 0 \qquad (2)$$

$$\left(\frac{x}{dx}^2\right) + \left(\frac{y}{dy}^2\right) + \left(\frac{z}{dz}^2\right) - 1 <= 0 \qquad (3)$$

The parameters for both equations are stored in the knowledge base, either as part of the positioning concept, or as information for the entity in reference, which can also include the quantity of characteristic points that need to be inside the volume to validate the position, or even the list of points that should be evaluated. Relative positions are always validated through paraboloid volumes, for which the parameters p and q correspond to the referenced entity size values, and z is a user-defined value, set by the system administrator.

Spatial relationships such as "`against`", "`inside`" or "`over`", are validated using ellipsoidal volumes, for which one of its parameters is less than 1, and located over the surfaces of the entities. For positions that require a volume inside an entity, or a location in a given area such as landmarks, the ellipsoidal volume is instantiated with the entity or landmark values.

If the positioning validation for an entity fails, a new position for this entity is calculated. Before the new position is set, the system queries a list of previous positions for that particular entity. If the neighborhood of these previous positions forms a cluster, then a new position outside that cluster is set, and the validation is carried again. If the new position is not valid, or can not be set outside the cluster, the system can not find a new position for the entity and return to a previous model state. As with collision verification, if the previous model is the starting set, the description requested cannot be solved.

Each entity in the model stores the list of elements with which they share some relationship. In that way, each entity can follow the movements made in the model, and modify its own parameters by consequence, for example, is an entity must be facing another, and the referenced entity is moved, the first can adjust its orientation pa-

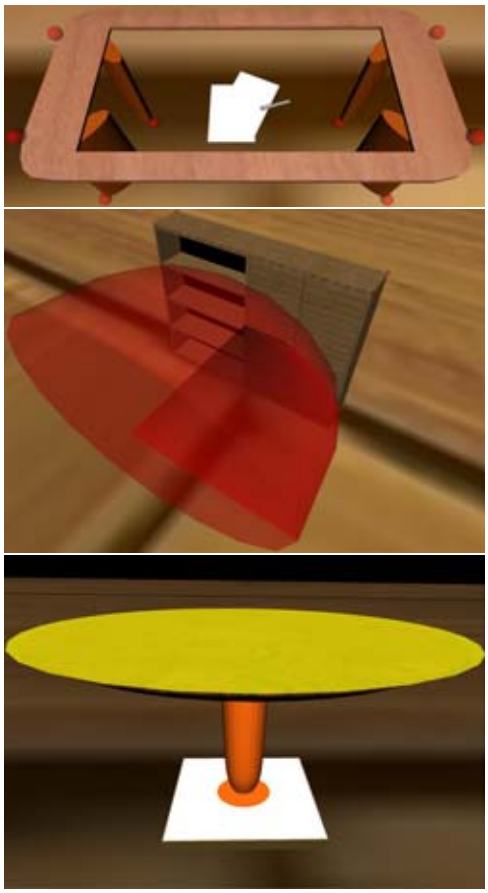

Figure 5: Characteristic points and positioning volumes examples.

rameters to the new position set for the referenced entity. The same applies for relative positions.

Once each entity has passed both validations, the model is sent to the final component, the Output Generator.

Obtaining the Output

The Output Generation Component receives the model. This component uses a Model-View Controller method to generate the necessary output data structures and files, so the underlying architecture can perform the actions necessary to create the visualization of the virtual environment, and conduct the process that will activate the simulation.

This output employs templates that receive the model data structure, and through the MVC are instantiated with the values obtained through the modeling process. This method allows the easy modification of the output parameters, so the users can add or modify the information sent to the underlying architecture, which can be also a 3D viewer of any kind, as long as the templates keep the standards used for the 3D visualization. The amount of outputs and their types are set in the knowledge base.

Current Work

Our fist example shows the output obtained through the GeDA-3D architecture. This example show 4 characters, which can run, hop, dance, and makes changes to its expressions, all this through agents that control each of these characters and their actions (figure 6).

```
[ENV]
  room.
[/ENV]

[ACTOR]
  MeninBlack Albert, center.
  WomaninBlack Beatrice, right Albert.
  MeninGreen Cecil, left Albert.
[/ACTOR]

[OBJECT]
  Grasshopper Insect1, south.
[/OBJECT]
```

Figure 6: Previous Modeler Example

We have developed a prototype of the modeler in the Java language, to allow multi-platform capabilities. The knowledge base has been defined using the Protégé framework, and was created using the OWL Standard (McGuinness and van Harmelen 2004), selected due to its extensibility over the internet. Next, we present some examples obtained using the most recent build of the modeler, for whose visualization we are currently using the X3D standard and a X3D-compliant viewer.

```
[ENV]
 LivingRoom.
[/ENV]
[ACTOR]
 ManSuit John.
 YoungWoman Sarah.
[/ACTOR]
[OBJECT]
[/OBJECT]
```

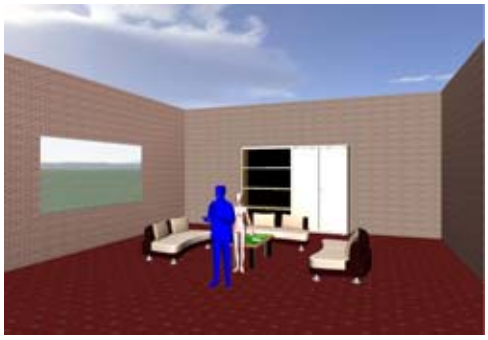

Figure 7: VEE Example 1.

In the first example, figure 7, "`LivingRoom`" is a concept stored in the knowledge base that also sets the furniture for the scenario and the position of each element. The second example, figure 8, shows different models obtained from a single description. "`House`" sets the whole environment, including most of the furniture, and the areas referenced to set the actors' positions. The last example, figure 9, shows the conflict-solving mechanism for several objects placed in the same relative position. The following examples where obtained using the current modeler and its X3D output.

Conclusion

In this article we describe our novel approach for declarative modeling using knowledge bases to assist the processes of modeling generation. We have tested our proposal by in a prototype. So far, we have created different scenarios with low generation times using several objects some are presented in this article. Also we have tested how easy is the modification of the parameters and the concepts to try different values for the same model, achieving significantly differences between solutions obtained through different iterations of the modeler. The result of this test proves the transparency in the modeling process.

The second type of test we have achieved concern our main difference with other similar works and is how useful is our scenarios to represent the evolution of a scene. To test this property of scenarios created using our approach we use the GeDA-3D architecture developed in our laboratory and tested for evolution of a virtual environment. This was carried using our kernel and

```
[ENV]
 house.
[/ENV]
[ACTOR]
 Knight, anywhere Kitchen.
 YoungWoman, anywhere Garden.
 woman, anywhere Livingroom
[/ACTOR]
[OBJECT]
 Chair, front woman0.
 Puff, front bed0.
[/OBJECT]
```

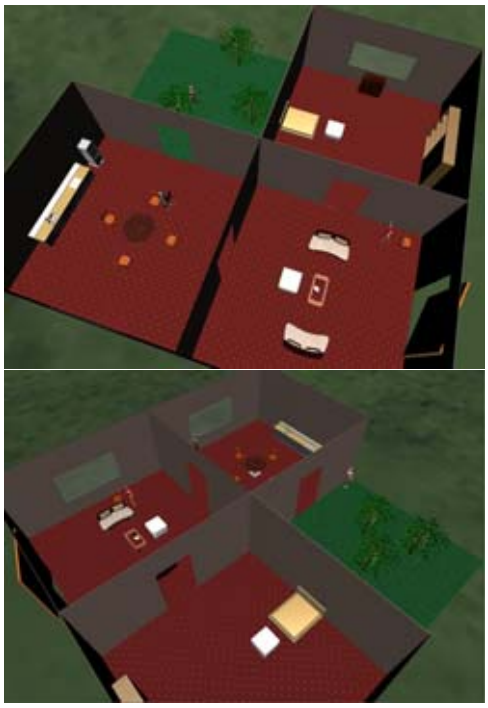

Figure 8: VEE Example 2.

distributed render, which allowed to evolved the scene through the agent-based architecture. These agents follow rules and constraints set by the user.

The main drawback of this research is that knowledge bases must contain all concepts as well as a 3D object database, however once this has been fulfilled the creation of 3D scenarios is available to final users. This achieves the main objective of this research.

Future work includes a friendly tool for knowledge management and 3D object database extension, as well as dealing with specific cases that may need special modeling methods.

Acknowledgment

The authors would like to thank the Mexican Council for Science and Technology, CONACyT, for providing the PhD Scholarship number 1910965.

```
[ENV]
 void.
[/ENV]
[ACTOR]
 Knight, center.
[/ACTOR]
[OBJECT]
 CenterTable Table, front Knight0.
 Chair, left Table.
 Chair, left Table, color red.
 Chair, left Table, color blue.
 Chair, left Table, color gray.
 Chair, left Table, color green,
        facing Knight0.
[/OBJECT]
```

Figure 9: VEE Example 3.

REFERENCES

Coyne B. and Sproat R., 2001. *WordsEye: An Automatic Text-to-Scene Conversion System.* In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* AT&T Labs Research, 487–496.

Gaildrat V., 2007. *Declarative Modelling of Virtual Environment, Overview of issues and applications.* In *International Conference on Computer Graphics and Artificial Intelligence (3IA), Athènes, Grèce.* Laboratoire XLIM - Université de Limoges, vol. 10, 5–15.

Hugo P.; Zúñiga F.; and Ramos F., 2004. *A Platform to Design and Run Dynamic Virtual Environment. International Conference on Cyber Worlds, Tokyo Japan*, 18–20.

Kwaiter G.; Gaildrat V.; and Caubet R., 1997. *DEM^{2}ONS: A High Level Declarative Modeler for 3D Graphics Applications.* In *Proceedings of the International Conference on Imaging Science Systems and Technology, CISST'97.* 149–154.

Le Roux O.; Gaildrat V.; and Caubet R., 2000. *Design of a new constraints solvers for 3D declarative modeling.* In *International Conference on Computer Graphics and Artificial Intelligence (3IA), Limoges, 03/05/200-04/05/200.* 75–87.

McGuinness D.L. and van Harmelen F., 2004. *OWL Web Ontology Language Overview.* W3c recommendation, World Wide Web Consortium. Http://www.w3.org/TR/2004/REC-owl-features-20040210/.

Prusinkiewicz P. and Lindenmayer A., 1990. *The algorithmic beauty of plants.* Springer-Verlag New York, Inc.

Ramos F.; Zúñiga F.; and Piza H.I., 2002. *A 3D-Space Platform for Distributed Applications Management. International Symposium and School on Advanced Distributed Systems 2002 Guadalajara, Jal, México.*

Russell S.J. and Norvig P., 2003. *Artificial Intelligence: A Modern Approach.* Pearson Education.

Wang D.; Herman I.; and Reynolds G.J., 1996. *The open inventor toolkint and the PREMO standard.* Tech. rep., Amsterdam, The Netherlands, The Netherlands.

Xu K., 2002. *Constraint-based automatic placement for scene composition.* In *In Graphics Interface.* 25–34.

Zaragoza Rios J.A., 2006. *Representation and Exploitation of Knowledge for the Description Phase in Declarative Modeling of Virtual Environments.* Master's thesis, Centro de Investigación y de Estudio Avanzados del Intituto Politécnico Nacional, Unidad Guadalajara, Guadalajara, México.

VIRTUAL WORLD CREATION AND VISUALIZATION BY KNOWLEDGE-BASED MODELING

Jaime Zaragoza, Alma Verónica Martínez,
Félix Ramos, Mario Siller and Véronique Gaildrat
Centro de Investigación y de Estudios Avanzados, Unidad Guadalajara, México
Intitute de Recherche en Informatique de Toulouse, France
email: jzaragoz, vmartine, framos, msiller@gdl.cinvestav.mx, veronique.gaildrat@irit.fr

KEYWORDS
Design, Game Engine Design, Game Environment Creation, Knowledge Bases, Distributed Systems

ABSTRACT

Virtual worlds can be used in a variety of areas, from entretainment to education, allowing to us see and interact with all kind of real or fantastic creatures or environments. However, the construction of such worlds, and its correct visualization, is a time and resource consuming task, which also requires expertise in the modeling and engineering of 3D models and render machines. In this paper, we propose a method for the creating and visualization of virtual environments, useful for any kind of simulations.

INTRODUCTION

Today's computational technology allows creating rich, complex, and detailed simulations of virtual environments. From fantasy settings to reconstructions of ancient cities, these virtual worlds can be used for severals ends, from entertainment proposes, such as movies or video games, to teaching, in the form of virtual trips or training.
Creating these virtual worlds and allowing user interaction with the entities dwelling inside these worlds is a task which requires a multidisciplinary staff, from computer engineers who design and create the software that runs the simulations of the virtual environments, to artists who design and create the individual elements that appear in those environments. Along with the staff, a variety of specialized tools are needed in order to created, present, animated, and evolve the virtual environments. Some of which require several training hours to be able of create professional result. In addition, the whole process can take from several hours to years to complete.
Some methods have been proposed to ease the task of creating virtual worlds, one of them being declarative modeling. This is a process which takes an input in which the user expresses the elements to be modeled, by giving some expected properties, and the system then proceeds to find a solution to these properties. In this context, the user could just describe the properties for the virtual world, and then let the modeler find the appropriate elements to be presented and their corresponding behaviors. The model then can be sent to another software system in order to be animated and presented to the user.

RELATED WORKS

There have been some works in declarative modeling of virtual worlds, but most of them focused on architectural design. Some of these works includes:

- FL-System (Marvie et al. 2005), focused in the generation of complex city models, parting from a specialized grammar, and using a variant of the Lindenmayer System (Prusinkiewicz and Lindenmayer 1990), called Functional L-System, in which replace the generation of terminal symbols by generic objects.
- CityEngine (Parish and Müller 2001), which is capable of generating a complete city model, using small set of statistical and geographical data, composed by geographical maps (elevation, land, water maps) and socio-statistical maps (population maps, zoning maps).
- Wonka et al presented a method for automatic architecture modeling that uses a spatial attribute design grammar, or split grammar, as input for the user. From this input, a 3D layout is generated for the building, from where the facade is created, split to structural elements, until the level of individual elements (windows, cornices, etc) (Wonka et al. 2003).

However, some works dealt directly with the generation of virtual environments, using a variety of inputs, from everyday language to speciallized haptic hardware. The first project is the WordsEye, a text-to-scene conversion system, developed by Bob Coyne and Richard Asproad at the AT&T laboratories. Allows any user to generate a 3D scene, from a description written on natural language, The system uses a part-of-speech tagger and a

statistical analyzer to parse the entyr, and then depictors (low level graphic representation) to compose the scene (Coyne and Sproat 2001). Some methods are used to solve some concepts, such astextualization, emblematization, characterization, lateralization or personification.
The second project is DEM^{2}ONS, a High Level Declarative Modeler for 3D Graphic Applications, designed by Ghassan Kwaiter, Véronique Gaildrat and René Caubet. It allows the user to easily construct 3D scenes in natural way and with a high level of abstraction. Composed by two parts: modal interface and 3D scene modeler (Kwaiter et al. 1997), the modal interface user communications using several combined methods (data globes, speech recognition system, spaceball, mouse). The syntactic analysis and Dedicated Interface modules analyze and control the low-level events to transform them in normalized events. The 3D scene is modeled using ORANOS, a constraint solver designed with several characteristics that allows the expansion of declarative modeling applications, like generality, breakup prevention and dynamic constraint solving.
The last work is called CAPS or Constraint-based Automatic Placement System, developed by Ken Xu, Kame Stewart and Eugene Fiume (Xu 2002). It makes possible the modeling of big and complex scenarios, using a set of intuitive positioning restrictions that allow the manipulation of several objects simultaneously, while pseudo-physics are used to assure that the positioning is physically stable. Uses input methods with high levels of freedom, such as Space Ball or Data Glove. It also employs semantic techniques for the positioning of the objects, using concepts such as fragility, usability or interaction between the objects. The object's positioning it conducted one at the time.
From the literalure available for each project, we found that none of them allows the user to make futher modifications beyond reorganizing the layout for the escene, and none of these works include any method for self-evolution or simultions over the 3D environment. Also, the input language, except from WordsEye, uses specialized hardware or data, whereas WordsEyes doesn't allows to include new concepts, due to its web-based nature.
Recently, several approaches have been presented for supporting distributed rendering in cluster systems. In (González Morcillo et al. 2007) are described two architectures for distributed rendering optimization: Yafrid (Yeah! A Free Render grID) and MagArRO (Multi Agent AppRoach to Rendering Optimization). In (Zhu et al. 2003) is presented an study about the research issues (resource allocation, task partition and data management) in constructing a distributed rendering for massive data sets on computational Grids. They also presented an implementation of the Dynamic Pixel Bucket Partition (DPBP) algorithm. This algorithm is used for task allocation of distributed rendering applications on computational Grids and it shows that performance of near real-time rendering can be reached.
The above revised work is not useful for us, because the output of the rendering process is an image. If we will use this result, it will be necessary to generate several images per second to visualize the complete evolution of the 3D scenarios. Another problem in these works is that exists a server in charge of compose the final image, if server fails, it would not be possible to generate the final image and the rendering process would be incomplete. Finally, the features of each element in the process are very specific, limiting its use to a small group of users.
In 2003 Rangel, Aviles and Mould (Rangel-Kuoppa et al. 2003) proposed a 3D rendering system that distributes rendering tasks across a multi-agent platform. The central idea of this system is the rendering of 3D individual objects in different computers. This task is solved using a mechanism based on pulling, where each remote agent is associated with a buffer in which the rendered image is stored. Thus, the agent that generates the 3D visualization takes and merges the information from the different buffers to produce a centralized visualization of the whole 3D VE. Karonis et al. (Karonis Nicholas et al. 2003) implemented a remote rendering system using a collaborative component and a high-resolution remote rendering component. These two components are connected and can either operate independently or as a coupled pair. But this system is for near-real time viewing and later use, and the rendering task are distributed inside one cluster only. However, the geometry data are partitioned into cells to improve the low-resolution rendering performance.
Straßer, Pascucci and Ma (Strasser et al. 2006) presented an interactive visualization system based on progressive refinement and distributed rendering. This system is capable of interactively browsing large multi-resolution datasets through the use of image caching. Thus, progressive refinement is made possible with a hierarchical multi-resolution representation of the volume data. A system for distributed rendering of large and detailed virtual worlds was described in (Chaudhuri et al. 2008). This system is a distributed client-server implementation, where the processing of virtual world is distributed among the available servers. This system can be used for generating virtual worlds with fine detail at planetary scales. The capacity of server limited the amount of client in the system.

VIRTUAL ENVIRONMENT MODELING

Our approach in the creation of virtual environments is through declarative modeling. This methodology is formed by three steps: the first step, *Description*, involve the user giving the setting, entities and properties to be used to generate model, as well as certain restrictions to be solve or satisfied. In this step it is defined

the interaction language. For our project, we defined a language oriented towards the compositions of descriptions, which we called Virtual Environment Description Language or VEDEL (Zaragoza Rios 2006). VEDEL allows for an easy composition and edition of description, focusing the user's attention to the entities and their properties, and providing a structured method for the composition itself. An example can be seen on figure 1.

```
[ENV]
house.
[/ENV]
[ACTOR]
Man Dad, sit Couch, reading.
Woman Mom, sit Chair, writting.
Girl Daughter, near Couch, laying, drawing.
[/ACTOR]
[OBJECT]
Table, center.
Couch Couch, againts EastWall.
Chair Chair, front Table, facing Table.
[/OBJECT]
```

Figure 1: VEDEL Example

The second step in declarative modeling is *Generation*, where the model or models are composed, validated and then presented. This step involves any method which can solve the assignation of the properties stated by the user, as well as any kind of conflict that may appear during the generation. This can be achieve with different methods for solving constrained problems, being the Constrain Satisfaction Problem solving the most well-know. We focused on integrating knowledge exploitation on the solution of CSPs, as well as into the whole generation process, making this two tasks more easy and transparent, as well as avoiding solving implicit meanings for some concepts.

Finally, the *Insight* step allows the user to decide which of the proposed solutions is the best, or to make modifications over the proposed layout, such that the solution matches with the user's ideal.

Using this methodology we designed a modeler, which also implements the concept of knowledge exploitation, in the form or a knowledge base, an ontology, which helps in the model creation process by solving term ambigety and concept value transforming. The architecture for this modeler is presented in figure 2.

VIRTUAL ENVIRONMENT EDITOR

The first part of our proposal corresponds to a Virtual Environment Editor (VEE) module, which will take the task of receiving a user's input written in the VEDEL definition, and the providing a solution for the user's statements. The input is received by a *lexical-syntactic parser*, which transforms the VEDEL entry into a data structure organized according to the syntax rules for

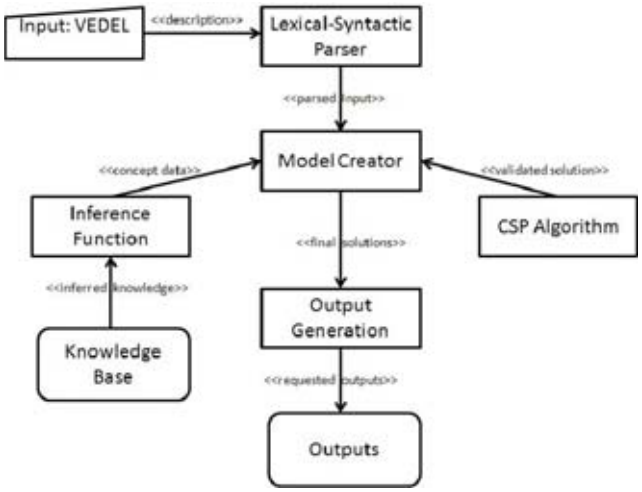

Figure 2: Virtual Environment Editor Architecture

VEDEL, with entity type as the upper branches, and their properties as the lower leafs.

The parsed entry is used then by a *model creator* to generate the model. The process begins with a *zero-state* model, which is a basic skeleton created with default values for the environment and the entities, extracted from the knowledge base by an *inference function*, which makes all the access and queries to the knowledge base, as well as making the necessary data type conversions between the data obtained and the modeler's needs. The model creator then proceeds to retrieve the necessary knowledge to update the model with the values requested by the user for each of the entities' properties. The modeler extracts each of the request from the parsed entry and then uses the inference function to obtain the properties attributes and values, and then proceeds to validate the user's demands. The converted values are set to each of the entities in the model, with the exception of position and orientation, and then the model is sent to the *CSP solving algorithm* to set the layout and solve any spatial conflict that may arise.

The CSP solving algorithm works in two steps: first, it sets all the position values for each of the request made in the description. If there are not any indication for the position of a particular entity, it is set to the center of the virtual environment. Other wise, the corresponding values are calculated using the knowledge base to obtain the ranges for the petition. The firsts entities to be set are those set to an specific place in relation with the environment or any *landmark* (a specially delimited area in the environment), such as `south`, `north`, or `center`. These entities are called *pivots*, and are used to set the rest of the entities' positions. The model generator proceeds to update the remaining entities' positions, using the ranges from the knowledge base and the values from the pivot entities to set the corresponding values to positions and orientations.

The next step is finding any possible conflict and then

solving it. This is conducted through the CSP, which is defined as follows:

- The set of variables $V = \{X_1, X_2, \dots X_n\}$, where $X_1, X_2, \dots X_n \in$ the set of entities in the environment, and $X_i = \{P, O, S\}$ corresponding to Position, Orientation, and Scale $\forall X_i \in V$.
- The domains for each $X_i \in V$ are $D(X_i(P)) = [-\infty, \infty]$, $D(X_i(O)) = [0, 2\pi]$, and $D(X_i(S)) = [0, \infty]$.
- The set of constraints is formed by the following equations:

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 - (r_1 - r_2) >= t_1 \quad (1)$$

$$\left(\frac{x}{p}^2\right) + \left(\frac{y}{q}^2\right) - 2z <= t_2 \quad (2)$$

$$\left(\frac{x}{dx}^2\right) + \left(\frac{y}{dy}^2\right) + \left(\frac{z}{dz}^2\right) <= t_2 \quad (3)$$

Equation 2 is used to solve collisions. The thresholds t_1, t_2 and t_3 are set in the knowledge base, so the strictness of the verification can be modified. This validation is carried using *collision marks* set for each entity. These marks consists of spheres that cover all of the entity's volume, and are retrieved along with the properties for the entity. The marks in an entity are tested against all of the marks in the others entities, and if there are no collision, this is, the result from equation 1 is bigger or equal to threshold t_1, for all the collision marks in all the entities, the collision validation has been passed.
If there is any collision, the CSP finds a new position for the conflicting entity or entities, and then proceeds to verify the new position. This is carried out using equations 2 and 3, in combination with a series of *characteristic points* defined for the entity. As well as the collision marks, the values for equation 2 are stored in the knowledge base, and retrieve with the rest of the entity's properties. The test for a position in which an entity makes reference to another is carried using the characteristic points to evaluate the function. If at least a number n ($n = 1$ by default) of characteristic points passes the test, the result of equation 3 is less or equal to threshold t_2, the validation test has been passed and the position is valid. Equation 3 is used for absolute positioning in landmarks or the environment itself, or in specially cases such as `over` or `inside`, or `against`, this is, where the entities touch each other or are contained inside another. The validation is carried out the sames as with equation 2. Any non-complying test leads to a further modifications in the position for the entity or entities.
The CSP can perform verifications for local minimums or maximums, so finding a solution can be assure. It also records the entity's past positions, in order to locate clusters of invalid or conflicting positions, and find another solution away from the cluster. Other conflict solving procedures are the complete arrangement of the entities in the environment or rotating conflicting or referenced elements.
If the positioning test are passed, the model is marked as valid, and send to the final module in the modeler, the *output generator*. This is a module which works over the Model-View Controller outline, sending any valid model obtained by the model creator, which is transform by the MVC into a suitable output for the visualization and animation process.

VIRTUAL ENVIRONMENT VISUALIZATION
This section describes the way for the creation and visualization of evolution the virtual environment. The description used in the creation is received of the VEE, this is interpreted to identify the entities by the virtual environment (VE).
We considered human avatar who the most complex entity because it can make different animation that include all the part of the avatar. Almost all works used animations were created in a 3D editor, this limits the kind of VE that is possible to use.
Our approach uses skeletal animation based in H-anim (Group 2009) for the human avatar. The purpose is to facilitate real-time management of each part of the skeleton of the avatar.
We take the advantage of the use of the computational grids and characteristics of the peer-to-peer architectures by generating a real-time distributed visualization of 3D VEs. Our architecture has the following components:

- Public knowledge base (KB): Is a set of ontologies based on OWL (Web Ontology Language). These ontologies manage and offer information about the VE components (3D scenarios, avatars and 3D objects).
- Coordinator nodes: These are special users that manage the consistency of VE. These nodes are grouped by areas of interest into the VEs.
- User nodes: They are external entities that can perform actions into the VEs.

With the use of different nodes avoids the dependence on servers, in this manner, it is possible to get a correct and complete evolution of VE in a most easy way, even if there will be a disconnect node (user). Therefore, a VE is organized by various coordinators, which are responsible for a group of users. If the node dont have all the required entities for the creation of VE, the node can make a query to VEE, this way, the node knows the location of the missing elements.
Figure 3 shows that our architecture is divided into different layers, in order to identify and reduce the dependence of upper and lower layers. The local processing

layer is necessary to update the local interface and to maintain a minimum consistency into the VE. That is to say, when a user requests to execute an action, this is evaluated to verify its consistency. For example, all affected entities must exist and the actions must be valid in the entity before to apply changes in the state of them. This in order not to assume that the user's actions are 100% reliable.

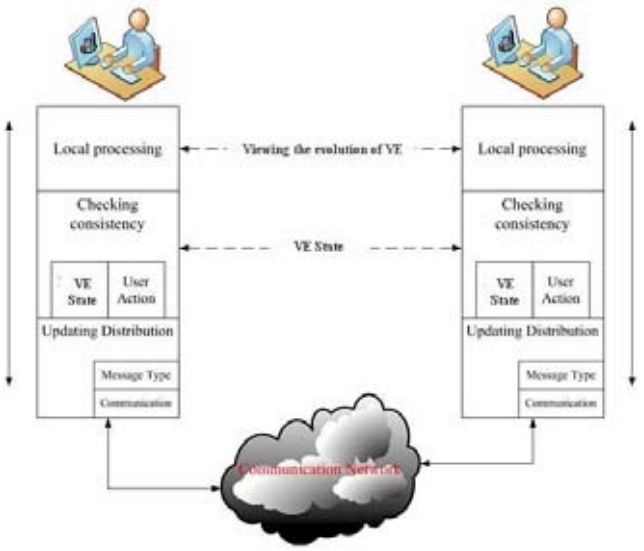

Figure 3: Interaction between the user and the visual port and modeling layered for a P2P communication scheme

When a user wants to perform an action (animation) on an avatar, it is automatically generated a request. This request has the following elements: the required avatar, the action to perform, and the time stamp that indicates the order in which actions must be performed. For each avatar is managed a queue of requests. In this queue are added all actions the avatar must perform. In order to ensure a correct execution order of the actions and a correct sequence of the changes into the VE, these are sorted based on their time stamp. Once the actions have been ordered, these will be executed in a process based on a dynamic time slot (TS). In this slot are placed all action that can be executed in 60 milliseconds (see figure 4).

Figure 5 shows that by using a TS it is possible to manage properly the workload locally, allowing viewing in real-time the required changes into the VE. The TS is handled taking into account an increase in local time that increases or decreases the amount of actions to be executed in a process. In this way, each new state of VE is given by the concurrent execution of processes. Each process returns a result that is sent to the user. This result tells the user if the actions were performed correctly or if there was any anomaly or failure during the execution of these.

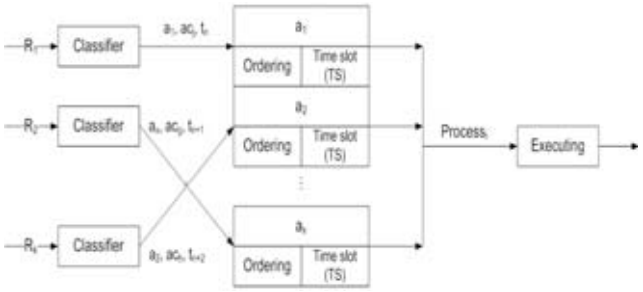

Figure 4: Action classification for process executing into the VE

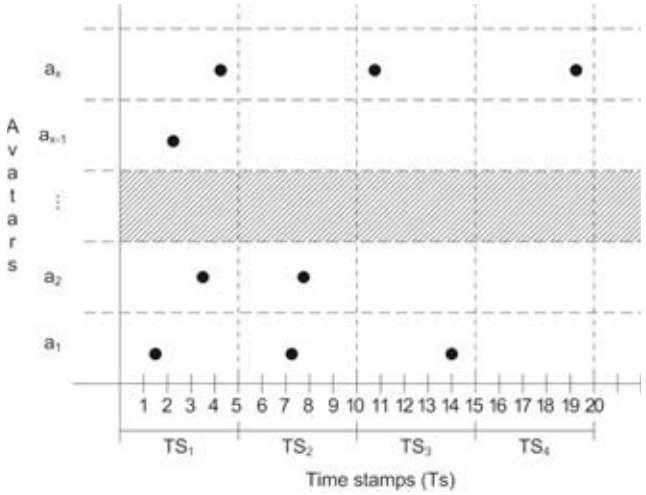

Figure 5: Action selection based on a dynamic TS

In our implementation, the real-time visualization of changes into the VE does not exceed 60 milliseconds, because the human eye perceives a continuous movement when changes are made the transition from one image to another at intervals of 60 milliseconds (Maiche Marini 2002). If these intervals are larger, the movement will be slow and discontinuous.

The requests done by a user are notified to the other users of the area of interest, so that they can take into account these in a timely manner. The following subsection describes the communication between users.

DISTRIBUTION OF THE VIRTUAL ENVIRONMENT

One of the main challenges in a shared VE is to efficiently send update messages in a correct way to provide scalability, minimized the delivery delay of messages, and to obtain a better reliability of VE. When an update message is generated as a result of an change of state of a virtual entity (avatar or 3D object). This message contains the new action of the entity. In our implementation, in order to carry out the distribution of messages among users (nodes in the system), we take

into account the following aspects: the organization of the nodes, the type of communication channel, and the involved communication protocols (see figure 6). These factors affect the delivery of packages, the delivery time is very important to ensure a real-time visualization of VE.

Nodes of system are grouped in a heterogeneous grid, where there are divisions based on the area of interest of each avatar. To identify to which area of interest belongs each avatar, we consider two aspects: the vision area of the avatars and the constraints of vision of VE. The vision area is given by the range of vision of the avatars, that is to say, characteristics and details of VE that are visible for each avatar. Constraints of vision of VE are given by the objects that restrict the range of vision of the avatars.

When in the VE exist several user, is necessary check the consistency among all involved users. To do this, consistency messages are generated. These messages contain a description of the area of interest to which the user belongs. Consistency messages are distributed by the layer distribution of updates. This layer also is responsible for selecting a reliable means to deliver them to a responsible coordinator of an area of interest. All the coordinators assigned to an area of interest are in charge of arrangements for setting the correct state of VE.

Some research are focused on the management of the overload nodes (Ajaltouni et al. 2008), but these depend of use of one server that is in charge of estimating and balancing the load of nodes. In this paper, we present a simple algorithm that not depends on a server or a single node.

The use of areas of interest avoids sending and/or processing messages, where the avatars can not perceive notorious changes or other avatars into the VE. In this way, the areas of interest involved into the VE are formed by nodes belonging to different networks using different communication protocols (see figure 6). Thus, having identified the areas of interest, it is necessary to choose a coordinator node for each area of interest. This choice is done taking into account the average distance between all nodes, and also considering the available bandwidth. This is to reduce the delay in message delivery and minimize loss of them. In a reliable connexion is possible to know the TTL field. This field contains the number of jumps needed for a packet reaches its destination. We define the distance like the number of jumpers that a packet has done.

Figure 6 also shows the calculation of the average distance between all nodes in an area of interest. Based on this calculation, it is possible to choice the location of the best coordinator in the area of interest. For example in figure, node 3 has the shortest distance to all other nodes, so it is taken as the coordinator node.

The new state of VE (see figure 7) is calculated by taking into account two major agreements: the first one among all involved users in an area of interest and its

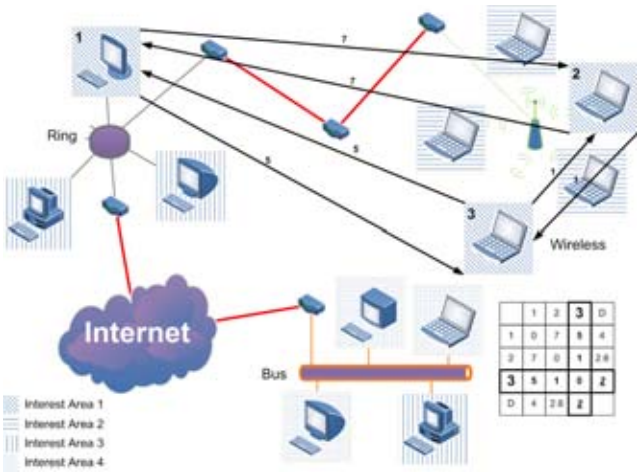

Figure 6: Involved areas of interest into the VE are formed by nodes belonging to different networks using different communication protocols. The election of a coordinator in by area of interest

coordinator (new state of the area of interest) and the second one among all involved coordinators in the VE (final state of VE).

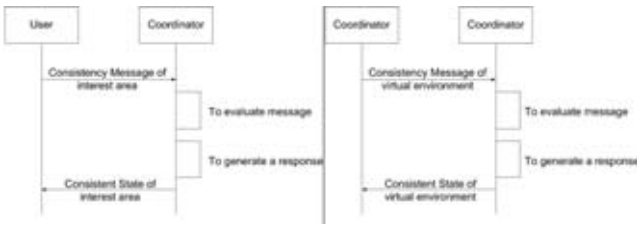

Figure 7: Agreements among all involved nodes and coordinators to choice the new state of VE

When a message is generated it is possible to know the current time of source node. If the coordinator detects different times, an agreement is made between nodes for establish the correct time into the VE.

If a coordinator has a work overload, it is difficult to process the real-time visualization of VE. To solve this problem, a new coordinator is selected. In this way, there may be more of a coordinator per area of interest. These coordinators are selected according to the distances they have to other nodes. This ensures that each coordinator manages a balanced amount of users.

When a user logs off, it is checked whether there needs to be more of a coordinator for the area of interest. In a similar way, when one user changes to other area of interest, it is verified whether there needs to choose a new coordinator for the area of interest.

There are some approaches that reduce the number of messages in the network (Rueda et al. 2007), but at least one node is responsible for processing all the actions of

the group, resulting in a client-server architecture. Our proposal uses only the coordinator node to verify the consistency between users. If at any time the coordinator is not available, the nodes in the area of interest are capable to select a new coordinator among them.

CONCLUSIONS

From the available documentation from ralated works, came to the conclusion that no current project offers the posibility of create a virtual scenario that can also be use to run a simulation, or to let the virtual world to evolve by itself. We also propose a simple method for input the desire settings, entities and properties to be represented, without the need of special hardware, but adaptable enough to be extended to another input methods. Also, this method provides a structured format, allowing the user to focus in the content rather than the format of the description. Finally, the system is accesible enough to let the users add new content, as well as to modify the constraints that will dictate the direction of the search for solutions.

We design our modeler to be extensible enough, by using the knowledge base, which allows to change significatibly the modeling process, as well as the results obtained. The outputs are also fully modellable, thanks to the use of the MVC methodology.

On the downside, there must be an experienced user, which will provide the first entities and environments, and the knowledge to process the request for such elements. Also, the visualization will depend on the underlaying architecture and its rendering engine.

The graphical representation and evolution of the VE, its based on a P2P architecture. The saturation in the user is inherently avoided due to the P2P communication scheme, and management of overload on the coordinator. Thus, a P2P scheme is a convenient architecture to provide a better scalability for large scale VEs.

The animation of the virtual entities is not based on pre-designed animations. When are used pre-designed animations, it is very difficult to manipulate them in diverse situations. By contrast, when micro-animations (modifying the attributes of a join's skeleton) are used to compose macro-animations, it is possible a better manipulation of each entity of VE, giving a major realism to the animations.

The nodes are grouping in different interest areas, this minimize the amount of messages in network. The messages from VE are classified into subsets, a node receives messages just one subset therefore reducing or avoiding the overload in each node.

REFERENCES

Ajaltouni E.E.; Boukerche A.; and Zhang M., 2008. *An Efficient Dynamic Load Balancing Scheme for Distributed Simulations on a Grid Infrastructure.* In *DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications.* IEEE Computer Society, Washington, DC, USA. ISBN 978-0-7695-3425-1, 61–68. doi:http://dx.doi.org/10.1109/DS-RT.2008.38.

Chaudhuri S.; Horn D.; Hanrahan P.; and Koltun V., 2008. *Distributed Rendering of Virtual Worlds.* In *Technical Report CSTR 2008-02.* Computer Science Department, Stanford University.

Coyne B. and Sproat R., 2001. *WordsEye: An Automatic Text-to-Scene Conversion System.* In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* AT&T Labs Research, 487–496.

González Morcillo C.; Weiss G.; Vallejo Fernández D.; Jiménez Linares L.; and Fernández Sorribes J.A., 2007. *3D Distributed Rendering and Optimization using Free Software. FLOSS International Conference.*

Group H.A.W., 2009. *H-Anim.* `http://www.h-anim.org/`.

Karonis Nicholas T.; Papka Michael E.; Binns J.; Bresnahan J.; Insley Joseph A.; Jones D.; and Link Joseph M., 2003. *High-Resolution Remote Rendering of Large Datasets in a Collaborative Environment. Future Gener Comput Syst*, 19, no. 6, 909–917. ISSN 0167-739X.

Kwaiter G.; Gaildrat V.; and Caubet R., 1997. *DEM^{2}ONS: A High Level Declarative Modeler for 3D Graphics Applications.* In *Proceedings of the International Conference on Imaging Science Systems and Technology, CISST'97.* 149–154.

Maiche Marini A., 2002. *Tiempo de reaccion al inicio del moviemiento: Un Estudio sobre la Percepcion de Velocidad.* PhD perception, communication and time, Department of Educational Psychology, Universidad Autonoma de Barcelona, Barcelona.

Marvie J.E.; Perret J.; and Bouatouch K., 2005. *The FL-system: a functional L-system for procedural geometric modeling. The Visual Computer*, 21, no. 5, 329–339.

Parish Y.I.H. and Müller P., 2001. *Procedural modeling of cities.* In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* ACM, New York, NY, USA, 301–308.

Prusinkiewicz P. and Lindenmayer A., 1990. *The algorithmic beauty of plants.* Springer-Verlag New York, Inc.

Rangel-Kuoppa R.; Avilés-Cruz C.; and Mould D., 2003. *Distributed 3D Rendering System in a Multi-agent Platform.* In *ENC '03: Proceedings of the 4th Mexican*

International Conference on Computer Science. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-1915-6, 168.

Rueda S.; Morillo P.; and Orduna J.M., 2007. *A Peer-To-Peer Platform for Simulating Distributed Virtual Environments*. In *ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems - Volume 2 (ICPADS'07)*. IEEE Computer Society, Washington, DC, USA. ISBN 978-1-4244-1889-3, 1–8.

Strasser J.; Pascucci V.; and Kwan-Liu M., 2006. *Multi-Layered Image Caching for Distributed Rendering of Large Multiresolution Data*. In A. Heirich; B. Raffin; ; and L.P. dos Santos (Eds.), *In Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*. 171–177.

Wonka P.; Wimmer M.; Sillion F.; and Ribarsky W., 2003. *Instant Architecture*. *ACM Transactions on Graphics*, 22, no. 4, 669–677.

Xu K., 2002. *Constraint-based automatic placement for scene composition*. In *In Graphics Interface*. 25–34.

Zaragoza Rios J.A., 2006. *Representation and Exploitation of Knowledge for the Description Phase in Declarative Modeling of Virtual Environments*. Master's thesis, Centro de Investigación y de Estudio Avanzados del Intituto Politécnico Nacional, Unidad Guadalajara, Guadalajara, México.

Zhu H.; Wang L.; Yun Chan K.; Cai W.; and See S., 2003. *A Distributed Rendering Environment for Massive Data on Computational Grids*. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2023-5, 176.

Modeling of Virtual Environments Through Declarative Modeling Assisted by Knowledge

Jaime Zaragoza, Félix Ramos, Véronique Gaildrat

Abstract— **The creation of dynamic virtual environments is a task that usually involves several disciplines, from modeling the desired visual representation for the virtual entities to the generation of an architecture allowing handling those entities. This paper focuses in the creation of the model for the virtual environment, to be exploited and animated by any underlying architecture through the use of knowledge. Our approach simplifies the process of creating the environment by focusing the user in the generation step and avoids heavy verification steps by adding only previous logic verification for the context.**

I. Introduction

The creation of dynamic virtual environments is a task that usually involves several disciplines, from modeling the desired visual representation for the virtual entities to the generation of an architecture allowing handling those entities. Those environments are used in the development of video games, simulators and virtual reality tools, as well has been used in movies [1]. Normally, experienced artists, technicians and developers create them with specialized tools. The declarative modeling is an alternative way for generating such environments, or the models behind them. Declarative modeling is a technique that has not been completely explored. In this article we use knowledge to represent not just the concept but the semantic to help the declarative modeling process, this approach has not been exploited before. With both concepts, the aim is to make accessible for final users the creation and animation of such environments. To reach this goal, it is proposed the use of declarative modeling by means of an easy-to-use tool that takes as input with the use of description of the desired result in a near-natural language. The knowledge bases containing the semantic are used for speeding no only the modeling process.

This paper focuses in the creation of the model for the virtual environment, to be exploited and animated by any underlying architecture. The primary goal is the conception and implementation of knowledge database to be used in declarative modeling [3], the second objective is to develop a tool that can be easily used by final users. The tool must allow a basic description of the desired environment and have the sufficient expressiveness for the creation of complex worlds. Although this tool can be used independently, it is considered part of the GeDA-3D project [4]. In this project it must generate the necessary context modules so the virtual world and the entities that dwell inside it can evolve in accord to the rules established by both the user and the semantic stored in the knowledge base.

I'd like to thank Dr. Félix Ramos from CINVESTAV at Guadalajara, México (email: framos@gdl.cinvestav.mx) and Dr. Véronique Gaildrat from IRIT at Toulouse, France (email:gaildrat@irit.fr).

I'd like to thank the Consejo Nacional de Ciencia y Técnologia, from México, for the support with scholarship No. 190965

II. Related Works

There exist several works that deal with generation of virtual models and environments, using information coming from varied sources. Some of these works are focused in declarative modeling of scenarios, others, completely focused on architectural building, and some other toward generation of virtual worlds, or simply as sketching or designing tools.

Among the tools designed for architectural development we can highlight two: System FL, by Jean-Eudes Marvie et al [5], a work base on Lindenmayer Systems, but completely focused on generating complex models of cities. The input for this system is defined in a specialized grammar, and can generated city models of variable complexity. It is complete based on a variant of System-L, and uses VRML97 as model output visualization. CityEngine [6]by Yoav I H Parish y Pascal Müller, is a project that focuses in full city modeling, using statistical data and geographic information as input. It also bases its design mechanism on a System-L, using a specialized grammar to accomplish the modeling.

However our interest, is in those works focused on virtual scenario creation, among those we found more representative:

WordsEye: an automatic text-to-scene conversion system, developed by Bob Coyne and Richard Asprod in the AT&T laboratories allows the user to create a 3D scenario, from a description written in a natural language. Uses text marking and part-of-the-speech and statical analyzers. The graphic representation, if generated from descriptors (low level graphic specifications) assigned to each semantic element, modified to match the postures and positions describe in the text, through inverse kinematics [7]. Uses some techniques such as textualization, emblematization, characterization, literalization o personification in those cases where there are no descriptors defined. DEM2ONS, a high level declarative modeler for 3D graphic applications designed by Ghassan Kwaiter, Veronique Gaildrat and Ren Caubet. Allows to construct 3D scenes in a natural way and with a high level of abstraction. It is composed by two parts: Modal interface and 3D scenario modeler [8]. The modal interface allows the communication with the system, by using several input methods simultaneously (data glove, speech recognition systems, spaceball, mouse). The scenario modeler uses ORANOS, a constraint solver with several characteristics that allows to expand the range of applications in declarative modeling: The

objects are modeled and rendered by the Inventor Tool Kit: This systems allows the interaction with the objects in the scenario and solves any constraint problem, but only allows statics objects, with no support for avatars. Multiformes is an all purpose declarative modeler specially designed for 3D scenario sketching, presented by William Ruchaud and Demitri Plemenos. The work in a scenario with is handled through its description, in other words, the way in which the designer inputs all the characteristics of the geometric elements in a scenario and the relationships between them [9]. The most important characteristic of Multiformes is its ability to automatically explore all the possible variations in one scenario, since it does not oblige to one interpretation of each imprecise property. The description of a scenario includes two set: the set of geometrical elements that are present in that scenario, and the set of relationships between the geometrical objects. Thanks to its constraints solver, Multiformes is capable of explore several variations of a sketch that satisfy the same description. This constraint solver obtains the solutions in a incremental way, and is capable of solving the restrictions requested by the user, but the user must tell the system how to construct the scenario. CAPS is a positioning system base in restrictions [10], developed by Ken Xu, Kame Stewart and Eugene Fiume. It makes possible the modeling of big and complex scenarios, using a set of intuitive positioning restrictions that allow the manipulation of several objects simultaneously, while pseudo-physics are used to assure that the positioning is physically stable. Uses input methods with high levels of freedom, such as Space Ball or Data Glove. It also employs semantical techniques for the positioning of the objects, using concepts such as fragility, usability or interaction between the objects. The object's positioning it conducted one at the time. Allows direct interaction with the objects, keeping the relationships between them by means of pseudo-physics or grouping: The methods and tools integrated in this system make it a design tool, mainly oriented toward scenario visualization, with no capabilities for self-evolution.

III. The GeDA-3D Project

The architecture GeDA-3D [4] is a powerful platform for the creation, design and execution of 3D dynamic virtual environments. GeDA-3D provides a platform useful to integrate and manage distributed applications and facilities to manage the communication among software agents and mobility services used to share other services. Figure 1 shows the architecture of the platform GeDA-3D, this platform has been grouped in four main modules: Virtual-Environments Editor (VEE), Rendering, GeDA-3D and Agents Community (AC). The VEE includes the scene descriptor, interpreter, congruency analyzer and constraint solver. The VEE provides an interface between the platform and the user, specifies the physical laws that governing an environment, and describes a virtual scene taking place in such environment. Rendering addresses all the issues related to 3D graphics, it allows the design of virtual objects and displaying of the scene. The AC is composed by the agents that are in charge of ruling virtual objects behavior. The scene gives to an agent a detailed description about what we want an agent does instead of how we want it does. Furthermore, this scene might involve a set of goals to a single agent, and will not be necessary that these goals must to be reached in a sequential way. Is not necessary to give a set of actions to perform by the avatar, only is necessary give a set of goals in a sequence of primitive actions before reaching them. So, we need agents able to add shared skills into their global behavior. Therefore, behaviors of agents are needed [11].

A user is enabled to construct a scene using a high level language similar to human language, user is not meant to provide the sequence of actions that the avatar must perform instead the user must only specifies the goals that should be achieved by the avatar. Therefore, two similar specifications might produce different simulations.

IV. Virtual Editor

The main objective of this research is the inclusion of knowledge (semantic) in the declarative modeling process, with the final objective of creating a virtual editor. This allows the user to simply state what should be include in the virtual environment, the characteristics of the environment itself and the entities inside, as well as positioning and goals for each of those entities. This approach will make while making easy the inclusion and modification of new entities and concepts of specific syntetic worlds and motivate clearly the re-use.

The input of this modeler is a description written in a natural-like language called VEDEL or *Virtual Environment Definition Language*, defined in [12]. A description written in VEDEL consist in three paragraphs delimited by section tags. Each paragraphs corresponds to the environment, the general setting for the environment, the actors, those entities capable of perform actions, and objects, the entities that cannot perform actions. Each paragraph is formed by sentences, at least one in the case of the environment, which in turn are formed by statements separated by commas, and ended with a dot (figure 1). The first statement must be the entity type, followed by an optional individual identifier in the case of actors and objects. The rest of the sentence must state the characteristics for that particular entity, beginning with the property name, and followed by the values for that property. Numerical values are expressed in parenthesis, and each value must be separated by a space.

```
[ENV]
   Room, big.
[/ENV]

[ACTOR]
   Man Antony, tall, hair dark, old.
[/ACTOR]

[OBJECT]
   Table, left Antony, small.
[/OBJECT]
```

Fig. 1. A description written in VEDEL

The description written in VEDEL is sent to the virtual editor, which then parses and validates it, and the proceed to generated the model, which can be used to generated a 3D view, or to created an input for the underlaying architecture.

A. Declarative Modeling Aided by Knowledge

The virtual modeler is formed by three modules: a lexical-semantic parser, a modeler based in the declarative process, and an inference function. The *lexical-semantic parser* is a state-machine, which verifies the entry description for invalid statements and characters, and creates a list of the statements made in the input. The user can set the behavior of this parser to pass every erroneous entry or to stop all the process. If the first case, the model will be crated using only the correct statements, and the list of errors found will be presented to the user at the end. In the latter case, the user will be presented with the only with the error that make the process stop.

The list of valid statement is used by the *modeler* to start the generation of the virtual environment. This list is used to generate the instances of entities needed for creating the environment, through the exploitation of the knowledge stored in the Knowledge base. Initially, an instance is created, by instantiating the subject, environment and entities, with values set as default in the knowledge base. Then, these properties are set to values established by the user. A process of validation is conducted to avoid duplicated identifiers, and if the modeler finds this case, it can stop the process or continue without the conflicting entity, as stated before. The validation is achieved using the knowledge base that is also used to establish convention between values, from statements to data structures or data type values. As established before, the modeler can be set to stop the process if errors are found, or to continue discarding invalid statements. Positioning is left to the ending step, so all the entities are created before starting with the geometric validation processes.

First a dependency search is made for those entities whose positions depend on other entities. Those entities which are not related to anything but the environment itself, by means of specific position statements such as “north”, “south” or “center”, are positioned first, and the rest of the entities follows. The modeler also can follow the behaviors of the parser, either leaving aside invalid request or unknown concepts, or stopping the process, and presenting the user a list of errors or the error that take the process to a halt.

We define three basic types for positioning: absolute, relative and close used in our system. *Absolute positioning* is executed in relation with the environment, either the whole virtual world, or a *room*, a specific section of the world delimited in the environment section of the input. An absolute position request is formed by a single statement, or the position statement followed by the individual name given to the room, with an optional numeric argument for distance in the cases of cardinal directions, in relation with the center, either from the environment or the room. *Relative positioning* is conducted using two entities, a pivot and a target. Both elements must be either an object or actor. A third argument indicating the distance to be place between the entities is optional. Since this kind of positioning can lead to inconsistencies, such as self-referencing or impossible positioning, basic logic verification is conducted, using statements stored in the knowledge base. Relative positioning can also be carried out between a room and an entity. Finally, *Close Positioning* involves two objects standing one close or even one against the other. We make this differentiation since the method we used to prevent collisions and intersecting entities, explained later, could prevent this request. These requests can also have a second parameter, indicating a specific position to place the entities. As in the relative position, basic logic verification is carried out.

To make the transition from the VEDEL statements to the data required for computing and validating the model, the modeler uses the *inference function* for translating each one of them. The inference function manages all the access to the knowledge base, and subtracts the corresponding information values and structures to be used by the modeler. The organization of the knowledge base is an important aspect to efficient this work, since every data extraction conducts the inference machine from general to abstract concepts and it?s semantics. Data values are stored as vectors, and converted to data structures through parsing functions. This was defined having in mind an easy method for knowledge storage and quick data access, since having a standard vector formatting allows quick conversions, and due to the knowledge tool do not respect the data input order, thus, is forced the use of artificial ordering methods.

The ontology is organized as follows: four main classes, containing the subclasses for all the entities, laws, actions and properties that can be represented by the underlying architecture. Those four classes are: *Environment, Actor, Object, and Keywords*. The user can add new properties to the knowledge base once the architecture or the visual port can represent more complex characteristics, like real hair, transparency, realistic clothes, or can represent new actions. Each class contains the individual entities, which in turn must contain at least one element (one instantiation of the class) named as the class, and ended with the suffix “_default”. This element contains the specific information for the entity, as well as the relationships with others entities or with keywords. Additional classes for laws and actions can be used, but must include the relationships with the entities to be considered into the modeling.

Once all entities have been place accordingly to the request made in the description, the modeler proceeds to verify the consistency of the model in two steps: first, a collision test is carried over on every element in the model, to assure that no entity is in a conflicting space with another. The second step consist in validating the position each entity adopts, since the first step makes the necessary changes to the position of the elements as collision conflicts are found. The functions carrying out this two steps are presented next:

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 - (r_1 + r_2) = 0 \quad \text{(1.a)}$$

(1.b) $\left(\frac{x}{p}\right)^2 + \left(\frac{y}{q}\right)^2 - 2z = 0$

(1.c) $\left(\frac{x}{dx}\right)^2 + \left(\frac{y}{dy}\right)^2 + \left(\frac{z}{dz}\right)^2 - 1 = 0$

To carry out the consistency tests, two types of tags are defined: collision tags and characteristic tags. The first take the form of spheres that are placed so they cover most or all of the entity (figure 2). Several tags are used to assure that all of the entity is covered, even when they cover additional space to the entity's geometry. The collision verification is carried out on every object and to all the other objects, but only once by pair, using sphere collision function 1.a. Once a pair of entities is set, the collision test if carried over all the pairs of collision tags for both entities, again, only once. If a collision is detected, the parameter of the collision function is stored, but if and only if is bigger than the last stored value, which starts at zero.

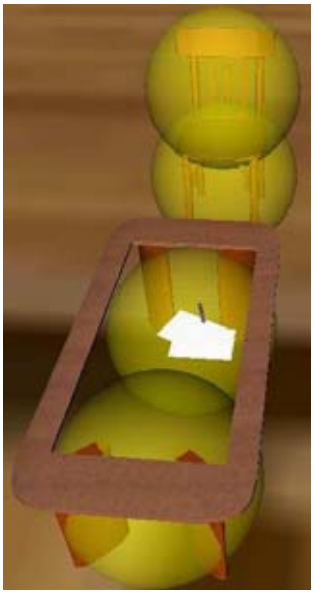

Fig. 2. Example of collision tags.

When all the collision tags have been verified, two methods are executed for solving conflicts: if the objects are not related, both are moved using a *repulsion* vector, which have an r parameter equal to the half of the result of equation 1.a, and a θ parameter equal to the supplementary angle to the arc formed by a line draw between the center of the two conflicting spheres and the x axis. If the entities are related, the reference entity is set as a pivot, and only the referencing entity is move, with parameters r equal to the result of equation 1.a, sphere-to-sphere collision test, and θ obtained as previously mentioned.

If is the case that both entities are related to a third one, we follow a method similar to the L-Systems [5]. In this case, we move both elements in a complementary angle to a line that bisects the consistency function 1.c and r parameter, defined as previously mentioned. In this case, consistency test are carried over, as explained later in this article, and if the function fails, the non-compliant entity is moved straight in the bisecting line, with a r value equal to the other entity's corresponding dimension. The following step is positioning verification. This is carried out on every entity whose position is relative to another, using the characteristic tags and function 1.c a paraboloid (figure 3). If any of the characteristic tags is inside the paraboloid instantiated with the pivot entity's values, the position is set as valid and the verification continue with other entity. On the other hand, if verification fails, the entity is moved to the point it previously was, and a new position is computed from there, using a vector normal to the pivots referenced size. Each entity keeps a list of positions it has previously been put, which is used to find local minimums. In this case, we perform an "step" in the positioning function, and if this new position is non valid, an error condition is arise.Finally, close positions are handle with a method similar to positioning verification. For this requests, we use equation 1.b, an ellipsoid (figure 3), which is instantiated with values stored in the knowledge base. Those values are set in function of the position and the entity's geometry, so we have ellipsoids that cover all of the space *inside* the entity, if needed, or a thin ellipsoid that works as *layers* for positioning another entity in a "close" or "over" position. In this case, the characteristic points of the referring entity are test, and the number and place of the characteristic that must be inside the positioning tag or tags varies depending on the entity and the request (figure 3).

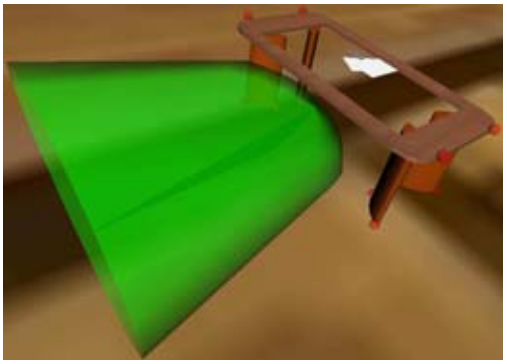

Fig. 3. Example of positioning tag.

The process is repeated until none conflict is found, or the verification methods found a local minimum. In the later case, the conflicting entity can be removed from the model, and the procedure stated again, or the process can be stopped and a list of errors presented to the user.

V. Results

The resulting model resulting from the process described previously can be sent to any formatting function, whose output can then be used by the underlying architecture, since all the necessary information to start the newly created virtual environment is at hand and presented in format allowing quick and simple access to any data. Also, the output can be visualized with any 3D modeling format file, since the position variables are also at hand, and the model can contains all the information needed to created complex renders,

Fig. 4. Example of close positioning

from the position of the camera, to the type and positions of light, as well as the physical properties of each entity.Hay que meter referencias a ests herramientasThe current version was coded in the Java language, using the latest SDK, the Protege API for knowledge base access, and the Freemaker Model-View Controller API for output generation. The output is a X3D-compliant file, which can be visualized in any X3D o VRML viewer, such as Flux Viewer or Octaga Viewer. The entities' geometry files are stored as templates, used by the Freemarker MVC to generated the final composition file, using the model generated by the modeler as input. The resulting output is an X3D file, which can be visualized with any 3D viewer that supports the standard.Next are presented some examples of the output generated by the modeler, as well as their corresponding generation input description.

VI. Results

The resulting model after the process described in the previous section can be sent to any formating function, whose output can then be used by the underlying architecture, since all the necessary information to start the newly created virtual environment is at hand and presented in format that allows quick and simple access to any data. Also, the output can be visualized as any 3D modeling format file, since the position variables are also at hand, and the model can contains all the information needed to created complex renders, from the position of the camera, to the type and positions of light, as well as the physical properties of each entity.

The current build was coded in the Java language, using the latest SDK, the Protege API for knowledge base access, and the Freemaker Model-View Controller API for output generation. The output is a X3D-compliant file, which can be visualized in any X3D o VRML viewer, such as Flux Viewer or Octaga Viewer. The entities' geometry files are stored as templates, used by the Freemarker MVC to generated the final composition file, using the model generated by the modeler as input. The resulting output is X3D file, that can be visualized with any 3D viewer that supports the standard.

We present next some examples of the output generated by the modeler, as well as their corresponding generation input description.

Description 1.

```
[ENV]
   void.
[/ENV]

[ACTOR]
   Knight Jaz, center.
[/ACTOR]

[OBJECT]
   CenterTable Table, front Jaz,
   color black, cristal translucid green.
   Chair JazChair, color blue, left Jaz.
   Chair One, left Table, facing Jaz,
   color green.
   Chair Two, left One, facing JazChair,
   color red.
   Chair Three, right Two, facing One.
[/OBJECT]
```

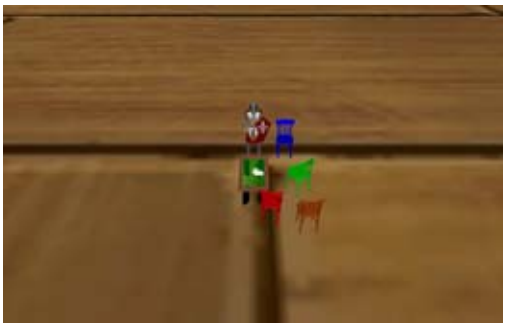

Fig. 5. Example 1.

Description 2.

```
[ENV]
   Theater.
[/ENV]

[ACTOR]
   Knight Antony, center.
   YoungWoman Bertha, left Antony.
   YoungWoman Caroline, right Antony.
   Knight Donald, right Caroline.
   Knight Ernest, left Bertha.
[/ACTOR]

[OBJECT]
[/OBJECT]
```

Description 3.

```
[ENV]
```

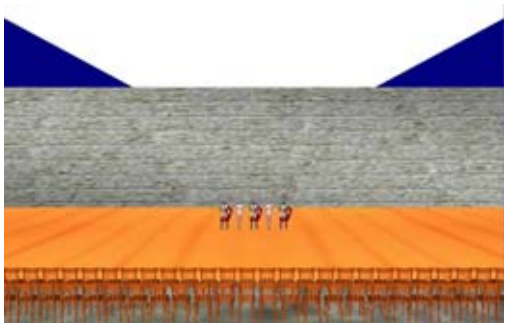

Fig. 6. Example 2.

```
   Forest.
[/ENV]

[ACTOR]
   Knight Antony, front House0.
   YoungWoman Bertha, left Antony.
[/ACTOR]

[OBJECT]
   House.
   Table 1, color black,
   front Jaz,
   cristal translucid green.
[/OBJECT]
```

Fig. 7. Example 3.

VII. Conclusions

Through the use of knowledge we can extend the possibilities of the modeler, thus creating a declarative modeling process that focused completely on the generation step, avoiding all verification steps, and adding only previous logic verification for the context, which then establish the intrinsic properties for the entities that are to be placed in the model.This verification using logic is also aided by the knowledge base, and can adapted to any possible scenario or situation, by adjusting the values stored in the knowledge base, therefore creating a straight-forward, transparent process for the user. The modeler also benefits from this transparency, since the constraint functions are few and easy to solve, and since they were selected having in mind enclosing most of any possible entity. Special cases can be solved by statements made directly on the knowledge base entry, or simply by adjusting the default values for positioning concepts.Future work includes a pre-processor for logical consistency verification, based on the environmental laws and properties of the scenario requested, and also aided by the knowledge base. Also, the modeler will have access to an array of concept-specific constraint for validation, such as material resistance or composition. This feature will allow modifying the constraints for any concept.

References

[1] J. Monzani, A. Caicedo, and D. Thalmann, "Integrating behavioral animation techniques," in *EG 2001 Proceedings*. Blackwell Publishing, 2001, vol. 20(3), pp. 309–318.

[2] D. Plemenos, G. Miaoulis, and N. Vassilas, "Machine learning for a general purpose declarative scene modeller," in *International Conference GraphiCon'2002, Nizhny Novgorod (Russia), September 15-21*, 2002.

[3] J. Zaragoza, F. Ramos, R. Orozco, and V. Gaildrat, "Creation of virtual environments through knowledge-aid declarative modeling," in *Congress of Logic Applied to Technology (LAPTEC 2007), UNISANTA - Santa Cecília University, Santos, Brasil*. IOS Press, Nov. 2007, pp. 1–8.

[4] F. Ramos, F. Zúniga, and H. Piza, "A 3D-space platform for distributed applications management," *International Symposium and School on Advanced Distributed Systems 2002 Guadalajara, Jal., México.*, November 2002.

[5] J.-E. Marvie, J. Perret, and K. Bouatouch, "The fl-system: a functional l-system for procedural geometric modeling," *The Visual Computer*, pp. 329 – 339, Jun. 2005.

[6] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2001, pp. 301–308.

[7] B. Coyne and R. Sproat, "Wordseye: An automatic text-to-scene conversion system," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. AT&T Labs Research, 2001, pp. 487–496.

[8] V. Gaildrat, "Declarative modelling of virtual environment, overview of issues and applications," in *International Conference on Computer Graphics and Artificial Intelligence (3IA), Athènes, Grèce*, vol. 10. Laboratoire XLIM - Université de Limoges, may 2007, pp. 5–15.

[9] G. Kwaiter, V. Gaildrat, and R. Caubet, "Dem^{2}ons: A high level declarative modeler for 3d graphics applications," in *Proceedings of the International Conference on Imaging Science Systems and Technology, CISST'97*, 1997, pp. 149–154.

[10] K. Xu, "Constraint-based automatic placement for scene composition," in *In Graphics Interface*, 2002, pp. 25–34.

[11] F. Zniga, "Agent convenable behavior in dynamic virtual environments," Ph.D. dissertation, CINVESTAV GDL, Mxico, 2007.

[12] J. A. Zaragoza Rios, "Representation and exploitation of knowledge for the description phase in declarative modeling of virtual environments," Master's thesis, Centro de Investigación y de Estudio Avanzados del IPN, Unidad Guadalajara, 2006.

Modelado Declarativo de Ambientes Virtuales Basado en Explotación del Conocimiento.

Jaime Alberto Zaragoza Rios[1], Hector Rafael Orozco Gutierrez[1],
Veronique Gaildrat[2]

Centro de Investigación y de Estudios Avanzados del Politécnico Nacional[1]
Unidad Guadalajara, Institute de Recherche en Informatique de Toulouse[2]

RESUMEN

El modelado de escenarios virtuales, su animación y la interacción con tales modelos son un tema complicado para el usuario no experto. En el presente artículo proponemos una metodología que permite al usuario no experto generar e interactuar con tales modelos, mediante el uso del modelado declarativo y la incursión de la explotación de bases conocimiento en ésta metodología, específicamente, para la resolución de conflictos de carácter geométrico..

Palabras clave: Modelado declarativo, ambientes virtuales, bases de conocimiento, mundos virtuales, animación por computadora.

. I Introducción

La creación y animación de ambientes tridimensionales complejos es una tarea complicada y que requiere de conocimientos y herramientas de complejidad notoria. Los mundos presentados en video juegos y películas toman normalmente varios meses de trabajo y requieren de un personal capacitado para el manejo de las herramientas, así como el talento necesario para la creación de los mundos y personajes.

Nuestro objetivo es el estudio de una metodología que permita la creación de una herramienta de fácil aprendizaje y uso, que permita al usuario común la creación, modificación y animación de ambientes virtuales complejos, a través de una visualización tridimensional y por medio de diversas interfaces.

Con el fin de proveer al usuario con un método sencillo y transparente para la generación del ambiente, se presenta un modelador declarativo, basado en una base de conocimientos, la cual contiene toda la información necesaria para la creación y validación del modelo solicitado por el usuario. El modelado declarativo es una técnica que permite al usuario la descripción de un escenario, diseñándolo de manera intuitiva, al dar solo algunas propiedades esperadas y dejando al sistema de cómputo encontrar una solución, si existe alguna, que satisfaga tales restricciones [1]. Siendo la base para la creación de tales mundos virtuales una base de conocimientos, ésta también contiene la información necesaria para la validación semántica, lógica y geométrica de las restricciones presentadas por el usuario.

La base para la validación geométrica, el tema principal de este articulo, es el uso de algoritmos para la solución de problemas de satisfacción de restricciones, o CSP, los cuales nos permiten encontrar conflictos tanto de posicionamiento como colisiones.

Este trabajo forma parte del proyecto GeDA-3D [2] una plataforma genérica distribuida para la creación y manipulación de mundos virtuales en un ambiente en 3D. También conforma el modelador declarativo del proyecto DRAMA, en desarrollo en el Institute de Recherche en Informatique de Toulouse (IRIT), en Toulouse, Francia.

. IITrabajos en Modelado Declarativo

Existen diversos trabajos enfocados en el modelado declarativo, algunos enfocados completamente en aspectos arquitectónicos, otros dirigidos a la generación de escenarios virtuales, y otros como herramientas de esbozo o diseño.

Entre las herramientas dedicas al desarrollo arquitectónico podemos destacar las siguientes: El sistema FL, por Jean-Eudes Marvie y otros [3], un

trabajo basado en Sistemas Lindenmayer, pero enfocado completamente en la generación de modelos complejos de ciudades. La entrada al sistema esta definido en una gramática especializada, y puede generar modelos de ciudad de complejidad variable. El sistema se basa completamente en una variante de un Sistema-L, utilizando VRML97 para generar la visualización del modelo. CityEngine [4], por Yoav I H Parish y Pascal Müller, es un proyecto enfocado completamente al modelado de ciudades completas, recibiendo como entrada datos estadísticos e información geográfica. También basa su mecanismo de diseño en un Sistema-L, utilizando una gramática especializada para este fin. Sin embargo, nuestro punto de interés son aquellos trabajos enfocados en la creación de escenarios virtuales, donde destacan los siguientes trabajos:

WordsEye, un sistema automático de conversión texto-a-escena, desarrollado por Bob Coyne y Richard Asproad en los laboratorios AT&T. Permite al usuario generar un escenario en 3D, a partir de una descripción en un lenguaje natural. Utiliza marcado de texto y analizadores de parte del discurso y estadístico. La representación gráfica se genera a partir de representadores (especificación gráfica de bajo nivel) asignado a cada elemento semántico, modificados para coincidir con las poses y acciones descritas en el texto, por medio de cinética inversa. [5]. Utiliza algunas técnicas, como textualización, emblematización, caracterización, literalización o personificación en casos donde no existe un representador propiamente establecido.

DEM^{2}ONS, un modelador declarativo de alto nivel para aplicaciones de gráficos en 3D diseñado por Ghassan Kwaiter, Véronique Gaildrat y René Caubet. Permite al usuario construir escenas en 3D de manera natural, con un alto nivel de abstracción. Esta compuesto de dos partes: Interfaz modal y modelador de escenario en 3D [6]. La interfaz modal permite la comunicarse con el sistema, usando simultáneamente múltiples métodos de entrada (guantes de datos, sistema de reconocimiento del habla, spaceball, ratón).El modelador de escenario utiliza ORANOS, un resolvedor de restricciones con varias características que le permiten expandir el rango de aplicaciones de modelado declarativo. Los objetos son modelados y representados por el Conjunto de Herramientas Inventor, que provee la Interfaz de Usuario Gráfica. Este sistema resuelve cualquier problema de restricción, pero solo permite objetos estáticos, sin soporte para avatares.

Multiformes es un modelador declarativo de propósito general especialmente desarrollado para esbozar escenas en 3D, presentado por William Ruchaud y Dimitri Plemenos. El trabajo en un escenario con MultiFormes es manejado esencialmente a través de su descripción, esto es, la forma en que el diseñador introduce todas las características de los elementos geométricos de un escenario y las relaciones entre ellos [7]. La característica mas importante de Multiformes es su habilidad para explorar automáticamente todas las posibles variaciones en de un escenario, pues no fuerza una sola interpretación de cada propiedad imprecisa. La descripción de un escenario incluye dos conjuntos: el conjunto de objetos geométricos presentes en ese escenario y el conjunto de relaciones existentes entre los objetos geométricos. Gracias a su resolvedor de restricciones, Multiformes es capaz de explorar diversas variaciones de un esbozo, satisfaciendo la misma descripción, siendo el resolvedor de restricciones geométricas el corazón del sistema. Este sistema obtiene sus soluciones de manera incremental, y es capaz de resolver las restricciones solicitadas por el usuario, pero éste debe de indicar al sistema como se debe de construir el escenario.

CAPS es un sistema de posicionamiento automático basado en restricciones [8], desarrollado por Ken Xu, James Stewart y Eugene Fiume. Hace posible el modelado de escenarios grandes y complejos, utilizando un conjunto de restricciones de posicionamiento intuitivas que permiten la manipulación de múltiples objetos simultáneamente, mientras se utilizan seudo-físicas para asegurar que el posicionamiento sea físicamente estable. Utiliza métodos de entrada con alto nivel de grado de libertad, como el SpaceBall

o el DataGlove. También emplea técnicas semánticas para el posicionamiento de los objetos, utilizando conceptos como fragilidad, utilidad o interacción con otros objetos. La disposición de los objetos se hace uno a la vez. Permite también la interacción directa con los objetos, manteniendo las relaciones entre ellos, por medio de seduo-físicas o agrupamientos. Los métodos y herramientas integrados a este sistema lo hacen una herramienta de diseño, orientado principalmente a la visualización de un escenario, sin la posibilidad de que éste evolucione por su cuenta.

. IIIGeneric Distributed Architecture 3D

GeDA-3D [2] es un proyecto actualmente en desarrollo, y del cual se desprende la investigación presentada en este articulo. GeDA-3D es una plataforma para crear, diseñar y ejecutar escenas dinámicas virtuales en 3D en una infraestructura distribuida, basada en el paradigma de agentes móviles y está formada por varios módulos: Editor de Ambientes Virtuales (VEE), Visualización, Núcleo de GeDA-3D, Comunidad de Agente (AC) y Descriptor del Contexto, todos representados en la Figura 1. Los componentes principales de GeDA-3D son el control de escena y el control de agentes.

EL VEE incluye el descriptor de escenarios, analizador de congruencia, resolvedor de restricciones y editor de escena. Provee una interfaz entre la plataforma y el usuario, especifica las leyes físicas que gobiernan un ambiente y describe la escena virtual que se lleva a cabo en tal ambiente.

La visualización se encarga de todo los detalles relacionados a los gráficos en 3D, principalmente, representar la evolución en 3D de la escena.

La comunidad de agentes (AC) representa los agentes encargados de dominar el comportamiento de los objetos virtuales y los avatares.

La descripción de escena proporciona a un agente una detallada descripción acerca de los objetivos que el usuario desea que haga, mas no la manera en que deben ser conseguidos.

Una escena puede incluir un conjunto de objetivos para un solo agente, sin que sea necesario que tales objetivos sean alcanzados de una manera secuencial [9].

Figura 1. Arquitectura del proyecto GeDA-3D.

. IVModelador Declarativo para Ambientes Virtuales

La creación del modelo que representa el ambiente virtual solicitado por el usuario es llevada a cabo por un modelador declarativo, parte del Editor Virtual, recibiendo como entrada una descripción escrita en un lenguaje definido explícitamente con el fin de proporcionar al usuario un método sencillo, estructurado y con expresividad suficiente para indicar los elementos a incluir en el escenario así como las propiedades tanto del ambiente como de los elementos que existirán dentro de este. A este lenguaje lo hemos llamado VEDEL, Lenguaje para Descripción de Ambientes Virtuales (Virtual Environments Description Language) [10,11].

En una descripción en VEDEL se pueden identificar 3 secciones o párrafos, delimitados por etiquetas de sección, como se muestra en la figura 2. Cada sección cosiste de varias oraciones, las cuales están formadas por declaraciones separadas por comas, y deben estar finalizadas por un punto. La declaración inicial debe ser el tipo de entidad a representar (ambiente, actor u objeto), seguido de manera opcional de un identificador único, en el caso de actores u objetos. El resto de las declaraciones corresponden a las propiedades deseadas para la entidad y los valores a ser asignados a estas. Tales propiedades pueden incluir posicionamiento, tamaño, propiedades físicas (color, forma, transparencia) o internas (energía, estado emocional). Cada concepto expresado en la descripción es verificado en la base de conocimientos, con la excepción del identificador

individual, de manera que aquellos conceptos no incluidos dentro de la misma serán omitidos o presentados como errores al usuario, según el método de funcionamiento del modelador.

```
[ENV]
  Desert, cloudy, night.
[/ENV]
[ACTOR]
  Knight Arthur, tall, wherever.
  YoungWoman Betty, left Arthur.
[/ACTOR]
[OBJECT]
  House, center.
  Chair, front house, color red, facing House.
[/OBJECT]
```

Figura 2. Ejemplo de descripción escrita en VEDEL.

La descripción es enviada a un analizador sintáctico, el cual analiza y clasifica la entrada para el modelador. El analizador es una máquina de estados, que busca errores de tipo léxico y sintáctico, los señala y resuelve, además envía una estructura de datos organizada de manera jerárquica al modelador. El modelador utiliza esta estructura para generar el modelo del ambiente virtual, utilizando además una base de conocimientos para realizar la verificación semántica de los conceptos solicitados por el usuario. El modelador extrae la información necesaria de la base de conocimientos para su correcta representación en el modelo, primero obteniendo las propiedades internas y visuales de cada entidad, y posteriormente con las propiedades geométricas: tamaño, posición y orientación.

Para este proyecto, la base de conocimientos esta formada por 4 clases base: Environment, que incluye todos la ambientes *base*, Actors, donde se almacena la información pertinente a los actores, Objects, que incluye los datos para las entidades tipo objeto, y Keywords, que son todas aquellas palabras que indican propiedades de cualquier elemento del escenario. Así mismo, se establecen diversas propiedades de tipo objeto y dato, desde relaciones "*Contrario A"* hasta conceptos como *"Color"* o *"Tamaño"*. Estas propiedades se asignan con los valores *por defecto* para la entidad o concepto en al menos un individuo por cada clase. Este individuo debe llevar el nombre de la clase y terminar con el sufijo "_default". Si se desea establecer varios individuos, esto es, varios conceptos cuyo morfología básica sea similar, dentro de una misma clase, estos pueden llevar el nombre propio de la entidad o concepto, pero siempre terminados con el sufijo "_default".

El caso del medio ambiente solicitado, el conocimiento almacenado en la ontología es utilizado para establecer el contexto que la arquitectura subyacente utilizara para desarrollar la escena, incluyendo las leyes que gobernaran sobre el ambiente y las entidades, y las instrucciones que permitirán a los agentes el desarrollo de las entidades que tengan asignadas, tales como las acciones que pueden ejecutar o sufrir, los parámetros que regirán su evolución, o las emociones que puedan representar.

Para las entidades, la información es extraída de los diversos individuos, y las propiedades internas y externas son interpretadas de acuerdo al conocimiento almacenado en la ontología, esto es, conceptos como "color", "cabello", "energía" o "tristeza" son transformados en valores numéricos o estructuras de datos interpretables para el resto de la arquitectura, a través de una función de inferencia que revisa la base de conocimientos para validar, convertir y, de ser necesario, inferir la información necesaria de otros apartados dentro de la misma base.

Una vez que completada la primera fase, el modelador prosigue con el posicionamiento de las entidades dentro del escenario, siempre guiándose por las solicitudes hechas por el usuario. Esta fase es auxiliada por un algoritmo CSP [12], el cual se encarga de verificar y validar que la posición asignada a una entidad sea válida con respecto a la descripción y a la propiedades intrínsecas de la entidad y las entidades a su alrededor.

Nuestro CSP está formado por la tupla siguiente:

- Un conjunto finito V de x de variables, en nuestro caso, las entidades del escenario.
- Un conjunto D de valores posibles para cada variable en V, como se muestra en la figura 3.

- Un conjunto C de restricciones, el cual está compuesto por las ecuaciones listadas en la figura 3., ecuaciones (1.a) a (1.c).

Por tanto, una solución válida a una descripción cualquiera es una asignación de valores de los dominios de D a las variables en V, tales que ninguna restricción en el dominio C sea violada [13].

Las ecuaciones seleccionadas para validar posicionamiento y resolver conflictos espaciales y colisiones fueron seleccionadas debido a que pueden ser resultas en lapso corto, y que sus parámetros permiten moldear el volumen que representan a la forma de las entidades, de manera que al utilizar varios volúmenes se cubra toda la entidad, sin que esto impacte en la complejidad computacional o la velocidad de cálculo en el proceso.

Cada entidad y concepto tiene asignada una o varias funciones en su entrada en la base de conocimientos, llamadas etiquetas, las cuales son instanciadas con los valores asignados a la entidad, y posteriormente validados contra el resto de las entidades. Adicionalmente, se asigna una serie de etiquetas para indicar los puntos característicos de la entidad, mismas que son utilizados para validar la posición de un entidad con respecto a otra (Figura 4).

Conjunto $V = \{x_1, \ldots, x_n\}$ de variables donde $x = \{P, O, S\}$ tal que

- $P = \{x, y, z\}$ (posición de la entidad).
- $O = \{\theta_x, \theta_y, \theta_z\}$ (orientación de la entidad).
- $S = \{S_x, S_y, S_z\}$ (escala de la entidad).

Conjunto D para el conjunto V tal que

- $D(P) = \Re^3$
- $D(O) = [0, 2\pi], \forall \theta \in O$
- $D(S) = \Re^3$.

Conjunto C de restricciones, formado por las siguientes funciones:

(1.a) $(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 - (r_1 + r_2) = 0$

(1.b) $\left(\frac{x}{p}\right)^2 + \left(\frac{y}{q}\right)^2 - 2z = 0$

(1.c) $\left(\frac{x}{dx}\right)^2 + \left(\frac{y}{dy}\right)^2 + \left(\frac{z}{dz}\right)^2 - 1 = 0$

Figura 3. Definición del CSP para el modelador.

La descripción ya analizada es revisada para encontrar dependencias entre las entidades, y a partir de éste análisis, se genera una lista que el modelador utiliza para asignar los valores iniciales a cada entidad, haciendo uso de la función de inferencia para obtener los valores y funciones asignados a cada concepto. Aquí se definen las entidades *pivote*, es decir, aquellas cuyo posición no depende de otras.

El modelo es revisado para verificar que no existan colisiones entre las entidades. Si estas ocurren, se procede a la modificación de los valores de las entidades en conflicto. Una vez realizados los cambios en el posicionamiento, se procede con la verificación de las restricciones, de tal forma que las nuevas posiciones se validen, o se computen nuevas posiciones si la entidad entra en un conflicto de posicionamiento. El proceso se repite hasta encontrar una solución que satisface las restricciones, o un punto cerrado, esto es, una solución que no cumple las restricciones y que no puede ser modificado para cumplirlas. En el caso de un punto cerrado, se procede a lo modificación de los valores de la entidades que fungen como pivotes. Si el proceso no encuentra solución alguna después de estas modificaciones, se procede con un informe de error al usuario, o se excluyen las entidades en conflicto, según la metodología seleccionada para el modelado.

El modelo final es enviado a una función que se encarga de generar las salidas necesarias para que la arquitectura pueda comenzar con la representación del escenario, incluyendo la representación visual y el contexto para el núcleo y los agentes. Esta función utiliza un método Controlador Vista-Modelo, de forma que la salida pueda ser modifica sin necesidad de modificar el código de la aplicación.

. VConclusiones y Resultados

El prototipo más reciente de nuestro modelador ha sido modificado para cumplir con los requerimientos de integración con el proyecto DRAMA [14], actualmente en desarrollo en el IRIT, como una colaboración con el equipo de Sistemas Distribuidos del CINVESTAV, Guadalajara. Fue codificado en lenguaje Java, utilizando el más reciente SDK, versión 6, así

como la API del proyecto Protégé, la cual permite el acceso a la base de conocimientos. Finalmente, el Modelo Vista-Controlador es manejado por la clase FreeMarker, lo cual permite modificar la apariencia de las entidades sin acceder al código fuente de la aplicación.

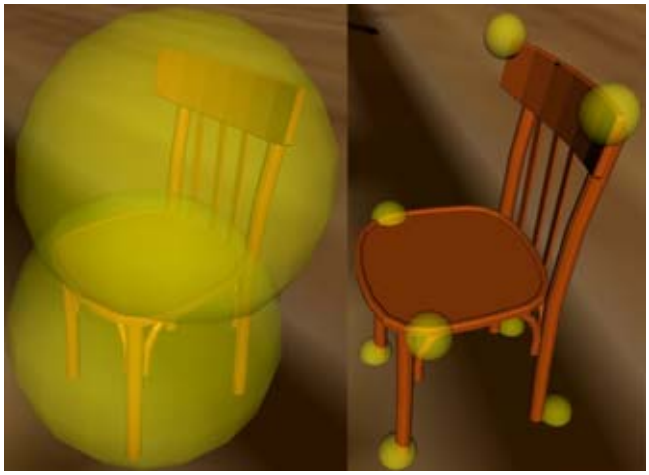

Figura 4. Ejemplos de etiquetas de colisión y de puntos característicos.

La figuras 5, 6 y 7 muestran algunos ejemplos de los escenarios obtenidos a través del modelador, utilizando estándar 3XD para la visualización en 3D. La cantidad de ambientes posibles se puede expandir al ingresar nuevos conceptos a la base de conocimientos, asignando también los modelos que representaran tales conceptos en el caso de entidades físicas. Los conceptos intrínsecos o abstractos son manejados por la arquitectura subyacente, siendo necesario que ésta pueda representarlos para que tomen efecto dentro del ambiente virtual.

A la fecha de redacción del presente artículo, resta conducir la integración del modelador tanto con el proyecto DRAMA como con el resto de la arquitectura GeDA-3D. Así mismo, es necesario el diseño de un verificador de congruencia semántica completo, que permita encontrar inconsistencias dentro del contexto de la descripción. Finalmente, se trabaja en la generación del contexto, encargado de indicar a la arquitectura las reglas que deben regir sobre el ambiente virtual, las propiedades y acciones permitidas dentro del mismo, y las acciones a llevar a cabo para cada suceso posible dentro del contexto del ambiente, en concordancia con los requerimientos del usuario .

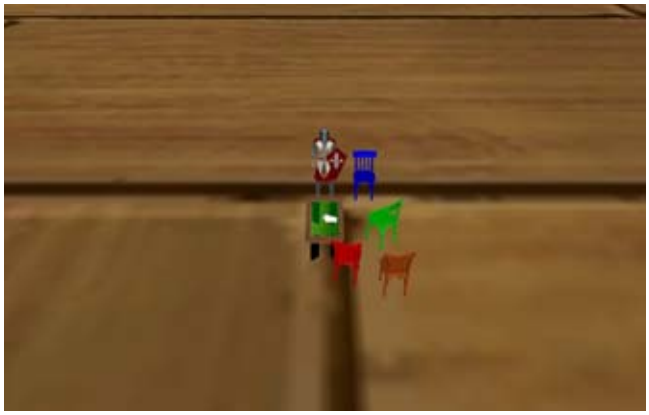

Figura 5. Composición simple.

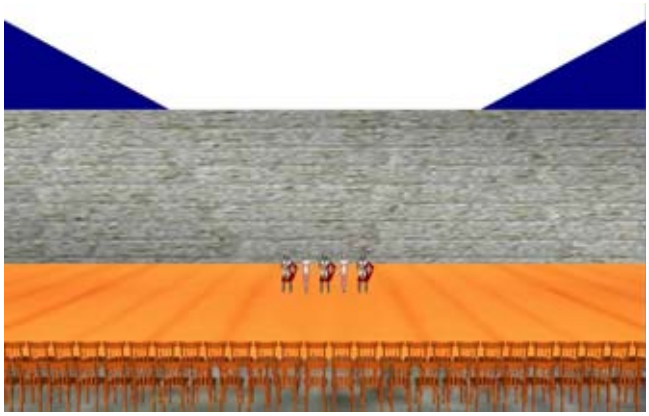

Figura 6. Ejemplo de ambiente "Teatro".

Figura 7. Composición simple en ambiente "Bosque".

. VIReferencias

[1] Demitri Plemenos, Georges Miaoulis, and Nikos Vassilas. Machine learning for a general purpose declarative scene modeller. In *International Conference GraphiCon* ’2002, Nizhny Novgorod (Russia), September 15-21, 2002.

[2] Felix Ramos, Fabiel Zúñiga, and Hugo I. Piza. A 3D-space platform for distributed applications management, International Symposium and School on Advanced Distributed Systems 2002. Guadalajara, Jal., México, November 2002.

[3] Marvie, Jean-Eudes and Perret, Julien and Bouatouch Kadi. The FL-system: a functional L-system for procedural geometric modeling, *The Visual Computer* 5-21, pages 329 -339, june 2005.

[4] Parish, Y. I. and Müller, P. 2001. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and interactive Techniques* SIGGRAPH '01. ACM, New York, NY, 301-308.

[5] Bob Coyne and Richard Sproat. Wordseye: An automatic text-to-scene conversion system. In SIGGRAPH ’01: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 487–496. AT&T Labs Research, 2001.

[6] G. Kwaiter, V. Gaildrat, and R. Caubet. Dem2ons: A high level declarative modeler for 3D graphics applications. In Proceedings of the International Conference on Imaging Science Systems and Technology, CISST’97, pages 149–154, 1997.

[7] William Ruchaud and Dimitri Plemeno. Multiformes: A declarative modeller as a 3D scene sketching tool. In *ICCVG*, 2002.

[8] Ken Xu and James Stewart and Eugene Fiume .Constraint-Based Automatic Placement for Scene Composition, Proc. Graphics Interface, May 2002, Calgary, Alberta

[9] Fabiel Zúniga. Agent Convenable Behavior ñin Dynamic Virtual Environments. PhD thesis, CINVESTAV GDL, México, 2007.

[10] Jaime Alberto Zaragoza Rios. Representation and exploitation of knowledge for the description phase in declarative modeling of virtual environments. Master’s thesis, Centro de Investigaci´on y de Estudio Avanzados del IPN, Unidad Guadalajara, 2006.

[11] Jaime Zaragoza, Félix Ramos, Héctor Rafael Orozco, Véronique Gaildrat. Creation of Virtual Environments Through Knowledge-Aid Declarative Modeling. Dans : Congress of Logic Applied to Technology (LAPTEC 2007), UNISANTA - Santa Cecília University, Santos, Brésil, 21/11/2007-23/11/2007, IOS Press, p. 1-8, novembre 2007.

[12] Daniel Hunter Frost. Algorithms and heuristics for constraint satisfaction problems. PhD thesis, University of California, 1997. Chair-Rina Dechter.

[13] Stuart J. Russell and Peter Norvig.Artificial Intelligence: A Modern Approach. Pearson Education, 2003.

[14] Andriamarozakaniaina T, Pouget M, Zaragoza R, Gaildrat V. DRAMAtexte : indexation et base de connaissances. *Premier colloque international sur la notation informatique du personnage. 16-17 mai 2008*. A paraître.

Animation of Autonomous Avatars over the GeDA-3D Agent Architecture

Orozco Aguirre Héctor Rafael[a], Zaragoza Rios Jaime Alberto[a] and Thalmann Daniel[b].

[a]Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara
Av. Científica 1145, Col. El Bajío 45010 Zapopan, Jal., México
E-mail: {horozco, jzaragoz}@gdl.cinvestav.mx
[b]École Polytechnique Fédérale de Lausanne
EPFL IC ISIM VRLAB Station 14 CH-1015 Lausanne, Switzerland
E-mail: daniel.thalmann@efpl.ch

ABSTRACT

All interactive applications, such as computer games, video games and collaborative virtual environments, are in need of believable virtual entities. But actually, the behavior of the avatars or virtual creatures (VC) in current applications and systems is still very artificial. This work is focused on the definition of the minimum conscience of the avatars, within the GeDA-3D Agent Architecture using Conscious and Affective Personified Emotional (CAPE) Agents. The conscience and cognitive processes of the avatars allow them to solve the animation and behavior problems in a more natural way. The minimum conscience gives the avatars knowledge of how their structure is formed and of their environment. Thus, the avatars can learn movements and compute motion planning activities for acquiring basic skills. We use the CAPE Agents to control the behavior of the avatars in 3D virtual environments and we apply cognitive processes involving emotional information for animating the avatars. These processes include mainly: appraising and expressing emotions and regulating emotion in the self and others, and using emotions in adaptive ways.

Keywords: Avatar, conscience, GeDA-3D, knowledge base, agent, behavioral animation.

. I Introduction

In the last years the graphical representation and animation of VC has been focused on the use and manipulation of predetermined animated forms and sequences. Nevertheless, in order to have autonomous avatars or VC, it is necessary to consider them as 3D semantic entities, with well-defined characteristics and functionalities. In order to make virtual environments more realistic, the avatars should exhibit a complex and believable behavior. A lot of studies about emotions exist and there are several models of emotion proposed in the literature. However, most of designed computational models of emotion only represent specific situations and respond in predetermined way to them. Nowadays, an important trend in the development of 3D dynamic virtual environments is to integrate proper characteristics of human being, such as personality, moods and emotions, into virtual characters or avatars with the aim of making them more believable and conceivable. Avatars need a convenient emotion model in order to synthesize emotions and express them. The emotion model should enable the avatars to distinguish and manage emotions in the same way that human beings do.

The human mind has been studied for many philosophers for a long time. The human consciousness is considered one of the most interesting topics in the philosophy. This topic is called philosophy of mind. An important aspect of human consciousness is the self-knowledge or self-awareness, defined as the ability to perceive and reason about oneself. This aspect is highly developed in the human being in comparison with other animals and it is considered very important

for making agents with an intelligent behavior. A human being unaware of his or her personal characteristics, abilities and skills does not know that he or she can do and cannot do and he or she will have difficulties for interacting with others in a natural way. The conscience in general is defined as the knowledge that the human being has of itself and of its environment. In this work, we define the conscience of an avatar as the notion that it has of its sensations, thoughts and feelings in a given moment in its environment. That is to say, the avatar conscience is the understanding of its environment and its self-knowledge. In other words, it is the notion that it has of its sensations, thoughts and feelings in the determined environment. Therefore, the self-knowledge makes the avatars have self-conscious and an internal representation of themselves. We use CAPE agents to develop the ability of avatar to perceive and reason about itself on the basis of the following: *consciousness* (involve thoughts, sensations, perceptions, personality, moods and emotions), *stimuli and sensorial entrances* (relevant events), *introspection* (ability of avatar to reason about of its perceptions and any conscious mental event), *awareness* (ability of avatar to be conscious and comprises perceptions and cognitive reactions to events, it does not necessarily imply understanding), s*elf-consciousness* (awareness and understanding of avatar, it gives the avatar the knowledge that it exists as an virtual entity separate from other avatars and virtual objects), and *qualia* (subjective properties of the perceptions and sensations of avatar).

In this work we use a knowledge base for animating avatars. We propose an ontology to provide the semantic definition and awareness of the internal structure of avatar (skeleton), its behavior (personality, emotions and moods) and its learned skills. In fact, the avatar uses the knowledge base first as a part of its conscience and second to implement a set of algorithms that constitute its cognitive knowledge. Therefore, an avatar as a human being needs to have conscience to keep equilibrium or achieve successfully complex activities. We argue that consciousness is very important and plays a crucial role in making emotional agents with human abilities and skills. Thus, the avatars can have awareness of how its skeleton is formed (considering its mobility and physical restrictions) and also of the rules that govern its environment. At this moment, we are working at the design of autonomous avatars able to act in 3D virtual environments, maintaining the idea that an avatar is an animated virtual creature, whose motions do not need to be defined previously. But the generation of dynamic autonomous movements with high degree of realism is too complicated. However, it is possible to make models of interactions between avatars and their environment in applications of computer animation and simulation [1]. For example, the virtual humans can be used as virtual presenters, virtual guides, virtual actors or virtual teachers. Thus, the behavior of the virtual creatures such as virtual humans can be controlled using conscious emotional agents to show how humans behave in various situations.

Our main objective is not modeling the complexity of human being's behavior, but simulating conscious emotional agents with a personality and dynamic emotional behavior. Thus, the decision making and action selection of the CAPE Agents must be regulated and controlled not only by external stimuli, but also by their personality, emotions and moods. In the next section we will give an overview of related work. The third and fourth sections are dedicated to the proposal of this work. In the last section we will present the conclusions obtained from this work.

. II RELATED WORK

An avatar (autonomous virtual creature) or virtual character is a virtual entity that lives and interacts in a 3D virtual environment. Avatars are entities composed by well defined features and functionalities. Articulated models are very used for modeling avatars. The animation of such models is often based on motion capture or procedurally generated motions. Despite the availability of such techniques, the manual design of postures and motions is still widespread, because it is possible to have a total control over the results. However, it is a laborious task because

of the high number of degrees of freedom present in the models. In this section we will summarize the most important related topics and we give our opinion about them.

Motion Planning Algorithms

Motion planning has multiples applications. In Robotics is dedicated to endow robots of intelligence (autonomy) so they can plan theirs owns movements. The problem of planning consists in finding a path for the robot since an initial point to a goal point without colliding with the obstacles in the environment [2]. In Artificial Intelligence (AI) the term planning takes a more interesting meaning. In this area the problems of planning are modeled with continuous spaces [3]. The problem of planning seems more natural and consists in defining a finite set of actions that can be applied to a discrete set of states and construct a solution by giving the appropriate sequence of actions. In [4] was presented a motion planner for computing animations for virtual mannequins cooperating to move bulky objects in cluttered environments. In this work were considered two kinds of mannequins: human figures and mobile robot manipulators.

Inverse Kinematics Algorithms

The kinematics algorithms are widely used for the animation of avatars. These algorithms use information like the position of joints angles and limbs lengths. In order to animate the avatars, structures hierarchical are formed by using a parent-child system similar to a tree. Each limb of the structure is a child node in the tree whose parent node provides a reference system from which it is describe. The parents are themselves child nodes of limbs above in the hierarchy and this recursive relationship continues up to a root node and the other end of the tree are leaf nodes which are children that have not descendants.

Reinforcement Learning Algorithms

In [5] two well-known reinforcement learning algorithms are presented. These algorithms are used for exploration, learning and visiting a virtual environment. A different approach for animating humanoids is proposed in [6]. However, this approach has many restrictions in the used models. In this work our approach is different, because we use reinforcement learning as a cognitive process that allows the avatar to learn new skills within a certain context. Our approach is very distinct because it works with conscience that is not just knowledge but cognitive processes, which allow us to animate avatars in a most natural way.

Motion Capture and Skeleton-Based Motion Planning

In recent years the films have been successful exploding the technologies of motion capture and virtual actors. Motion capture is the process of capturing the live motion from a person or animal in order to animate an avatar or virtual character [7]. Motion capture provides an impressive ability to replicate gestures, synthetic reproduction of large and complicated movements, behavior analysis, among others. At the moment, the motion capture systems allow the collection of information for illustrating, studying and analyzing the characteristics of body limbs and joints during various motions, such as walking, running, etc. However, though impressive in the ability to replicate movement, the motion capture process is far away from perfect. Despite the longer time required to visualize the captured motion, the optical motion capture it is often preferred to magnetic technology. The avatars animation design generates libraries of postures and motion sequences using a motion capture system and later combined the obtained data with standard editing tools. In the real-time motion generation, the generation of avatars motions in real-time is based on the combination of pre-recorded sequences or dynamic motion capture allows avoiding the recording stage.

Role of the Emotions in Emotional Agents

Through the time, several models have been proposed in a broad range of scientific areas to describe the functioning of the human mind. Emotions in special have received increasing

attention and interest in several fields related to AI, mainly in Human-Computer Interaction (HCI) and Human-Robot Interaction (HRI), where emotional receptivity (perceiving and interpreting of facial expressions) and emotional expressivity (expressing emotions) are crucial and play an important role. Multi-Agent Systems (MAS) cover problems related to the autonomy, the cooperation and coordination between agents as well as the interaction of believable agents in virtual environments. In MAS, individual agents are assumed to be autonomous. That is to say, they should have the capability to deliberate and decide which actions to take or which tasks to perform in order to reach their goals. Several emotional architectures of agents have been proposed. The researchers have been particularly interested in designing models in order to make realistic and improve believability of the agents, which are applied in artificial situations, such as simulations in virtual environments.

Emotional agents are used to simulate the human being's reasoning by means of the influence and effect of emotions. We argue that it is necessary to add human characteristics, such as personality, mood and emotion, in order to design much more conceivable and believable agents. In [8] is described an architecture based on the Emotional-Belief-Desire-Intention (EBDI) agents by using emotional updating functions into four components: Emotion, Belief, Desire and Intention. This architecture demonstrated that EBDI agents have better performance than rational agents, because they have more flexibility and ability to be adaptable and survive in dynamic environments. Main roles of the emotions in agents are: action selection, motivation, adaptation, social regulation, goal management, attention focus, strategic processing and self-model.

Emotion and Personality Models Applied to Agents

The OCC model [9] is considered as a standard model for emotion synthesis and as the best model of categorization of emotions. The OCC model explains the human emotions and tries to predict under which situations which emotions can be experimented. Emotions are divided into the following groups: reactions to events, actions and objects. This model specified 22 emotion categories based on positive or negative reactions. Thus, the consequences of an event can please or displease the agent (*pleased/displeased*), the agent can accept or reject actions (*approve/disapprove*), and the characteristics of an object can attract or not the attention of agent (*like/dislike*). The OCC model was developed to understand the emotions instead of simulating them.

The model FLAME (Fuzzy Logic Adaptive Model of Emotions) was created to produce emotions and simulate the emotional intelligence process [10]. This model uses fuzzy rules to explore the capability of fuzzy logic for modeling emotional processes, and capturing the fuzzy and complex nature of emotions. Nonetheless, it is very important to consider the personality for determining the consistency of emotional reactions over time.

The OCEAN model or Five Factor Model [11] is a purely descriptive model of personality, it groups personality traits of human being in five dimensions: *Openness, Conscientiousness, Extraversion, Agreeableness* and *Neuroticism*. Although this model is widely accepted, it has many criticisms, because it does not indicate how exactly the personality is affected by obtained stimuli and experienced situations.

Masuch, Hartman and Schuster [12] presented a different model represented by seven personality dimensions: *Suspicion, Curiosity, Sociability, Aggression, Helpfulness, Vividness* and *Conscientiousness*. These seven personality dimensions are closely related to specific aspects of more general dimensions of the OCEAN model. Due to the direct correspondence between emotions and facial expressions, many researchers prefer to employ the Ekman's six basic emotions (joy, fear, sadness, dislike, anger and surprise) for facial expression classification [13] and the OCEAN model or the OCC model in combination with the OCEAN model.

. III GeDA-3D: An Architecture for Developing CAPE Agents

In this work we present an extension of the GeDA-3D Agent Architecture [14, 15]. This architecture provides a platform useful for integrating and managing distributed applications, and it offers facilities to manage the communication between intelligent emotional agents. Figure 1 shows the GeDA-3D Agent Architecture. This architecture has been grouped in different modules: Virtual environments editor (VEE), rendering, GeDA-3D Kernel, agents community (AC) and context descriptor (CD). The striped rectangle encloses the main components of GeDA-3D: scene control, agent control and scene editor. The VEE includes the scene descriptor, interpreter, congruency analyzer and constraint solver.

The VEE provides an interface between the GeDA-3D platform and the user; it specifies the physical laws that govern an environment and it describes a virtual scene taking place in such environment. Rendering addresses all the issues related to 3D graphics representation in a visual medium like the screen or the monitor; it allows the design of virtual objects and displaying of the scene. The EAC is composed by the agents that are in charge of ruling virtual objects behavior. The scene gives to the agents a detailed description about what we want them to do instead of how we want them to do it. Furthermore, this scene might involve a set of goals for a single agent, and it is not necessary that these goals are reached in a sequential way. It is also not necessary to give the avatar a set of actions to perform; we only need to determine a set of goals in a sequence of primitive actions. So, we need agents able to add shared skills into their global behavior. The user is enabled to construct the scene using a high level language similar to human language. The user does not provide the sequence of actions that the avatar must perform; he only specifies the goals that should be achieved by the avatars. That is, the agents behavior is in charge of trying to reach the goals specified in the scene description. The reason of creating this kind of scenes is to visualize a behaviors-based simulation, in other words, the user specifies what the agents must do, instead of how they have to do it. Therefore, two similar specifications might produce different simulations.

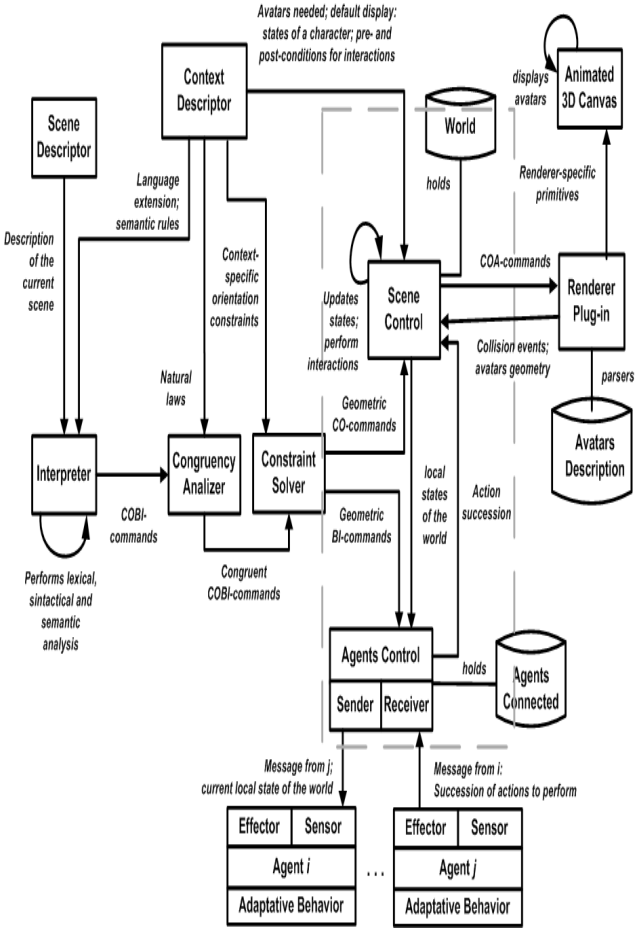

Figure 1. GeDA-3D Agent Architecture

Autonomous Avatars Animation over GeDA-3D

In order to animate avatars it is necessary to use dynamic planning and learning algorithms to compute motions. Avatars should be conscious of their internal structure (skeleton) and know how to combine simple or primitive movements to make complex activities or motions, which allow them to learn several skills and abilities. Figure 2 shows all the necessary elements and considerations to take into account, to animate articulated virtual creatures over the GeDA-3D Agent Architecture. In addition, it is necessary to use sensors in order to obtain stimuli, events and influences of the environment that can alter the behavior and motion of avatars, according to their personality, emotions and moods. For example, if an avatar is happy or

angry, its facial expression indicates its emotion, and its movements and behavior are changed; when it is sad its movements are slower than when it is happy.

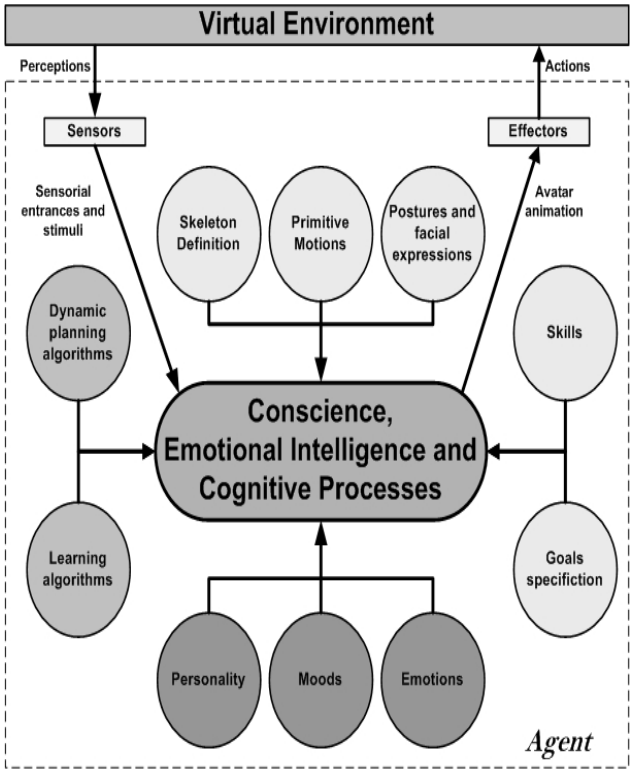

Figure 2. Necessary elements to animate avatars using CAPE Agents

In the scheme shown in figure 3 the posture and the motions of avatar are coordinated. The posture control is performed by references that indicate the direction and the required degree of stability of the avatar body. The sensorial entrances produce readjustments in the position of avatar and contribute to the modification of its corporal scheme. When the avatar makes a motion, advance adjustment of posture is produced (pro-action). The brain of the avatar contains an internal model of its corporal segments, relative sizes, their relations and positions. Therefore, the corporal scheme of avatar is the source of its corporal perception. Thus, the avatar adopts a posture and a facial expression that reflect its emotions and moods. The corporal scheme of avatar is a representation of its skeleton and its possible actions. This scheme is defined using: semantic and lexical information on the parts of the body and the skeleton of avatar, visual-spatial representations of the avatar body and the objects of its environment (for instance, the nose is situated in the middle of the face, the ears one to each side of the head, the mouth under the nose, the eyes above the nose and one to each side of the face), and corporal references and composition of the motions on the basis of the corporal perception of avatar and its influences.

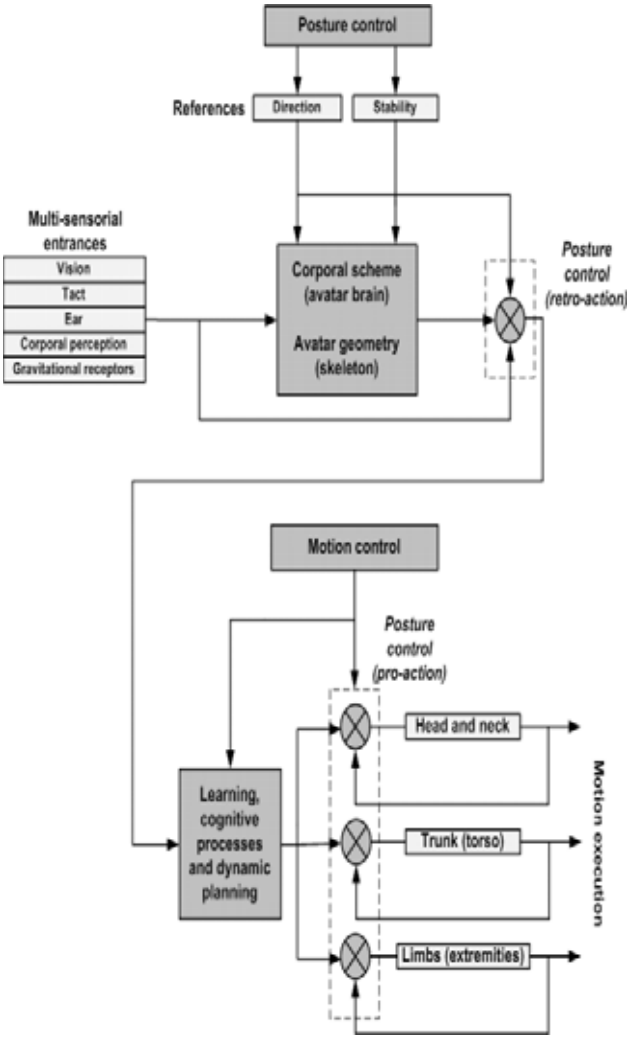

Figure 3. Control and coordination of the posture and motion of avatar

Figures 4 and 5 present the proposed modules of learning and dynamic planning for the GeDA-3D Agent Architecture. The personality, emotions and moods are related to learning and dynamic motion planning of avatar. The avatar should explore its body to know its structure and to learn a set of primitive motions, the basis for generating

complex motions. In addition, the avatar receives a set of goals and plans, and a series of motions necessary to fulfil them.

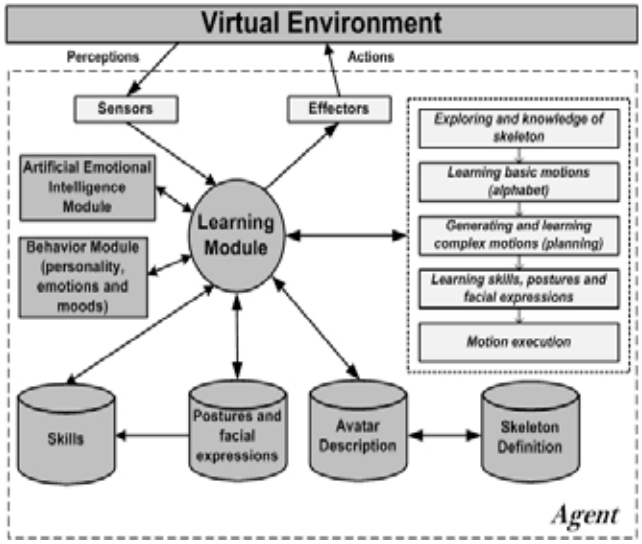

Figure 4. Learning module

The avatar explores its body to know its structure and to learn a set of primitive motions, the basis for generating complex motions. In addition, the avatar receives a set of goals and plans, and a series of motions necessary to fulfil them.

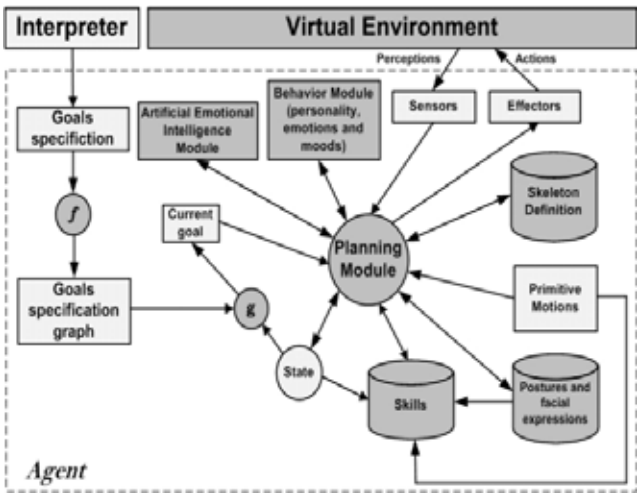

Figure 5. Dynamic motion planning module.

In this work we propose the use of synergies to support the idea that the avatar cannot control all the degrees of freedom of its skeleton. For this reason a set of simple or primitive motions is selected (natural motions) to generate complex motions. Therefore, synergies are the base of the motions of avatars and can be manipulated by means of learning and dynamic planning algorithms.

. IV AN ONTOLOGY FOR CAPE AGENTS

In this work we propose an ontology to define the internal structure of avatar (skeleton), its behavior (personality, emotions and moods) and its learned skills. The main objective is the exploitation and use of the information and knowledge offered by the ontology in order to create autonomous animations of avatars. This ontology provides the definition of the internal structure of avatar, its behavior and its skills, and it permits share semantic information of avatar among CAPE Agents in 3D dynamic virtual environments. Thus, the proposed ontology is used as basis for applications of motion planning and motion learning of avatars. This ontology offers a formal description of CAPE Agents. Figure 6 shows the relationships between the main classes of the proposed ontology. An avatar is defined using a morphology description (qualitative description) that defines its skeleton (geometry of avatar) and the anthropometry description (quantitative description) that offers information about its age, gender, weight and height. In addition, as a part of its behavior, an avatar has a personality, emotions and moods.

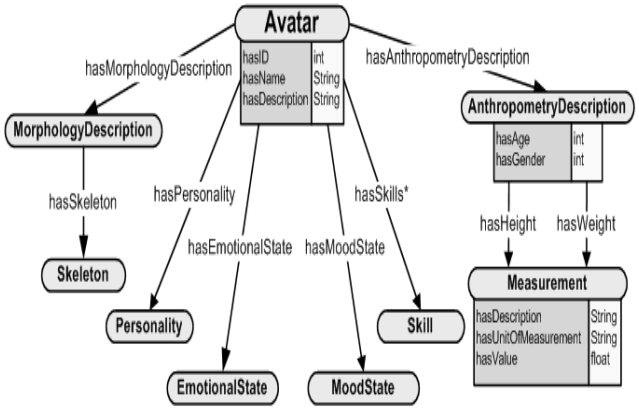

Figure 6. Semantic definition of avatar

The internal skeleton of an avatar (see figure 7) is formed by several parts, bones and joints in specific (skeleton parts). Each joint has a name and can have joints parents and/or joints children.

There are motion constraints defined for each joint and a set of simple motions that define the alphabet of basic movements (micro-animation) that will be used to generate complex motions by means of combination (macro-animation) between them. Also each bone can be united to one or more joints, and each joint has its position in the skeleton of avatar. Each bone of avatar has its measures that can be expressed in a predetermined unit of measurement, for example, in centimeters or decimeters.

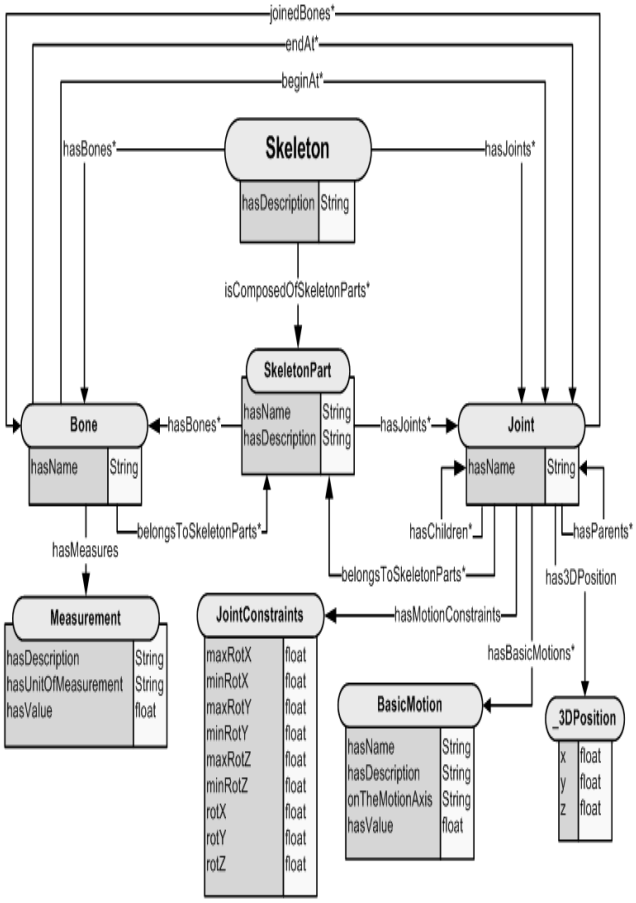

Figure 7. Skeleton definition of avatar

The emotional state of avatar is shown in figure 8. The emotional state is a set of emotions with an emotional history. A set of facial expressions and a set of postures associated with them correspond to every emotion of avatar. The facial expressions are produced with the help of facial markers (similar to joints). Body markers provide each posture with its own set of animation sequences. Emotions can be positive or negative, active or passive, but they always are of short duration. If an avatar is sad, it is able to adopt a facial expression and posture that show its emotional state.

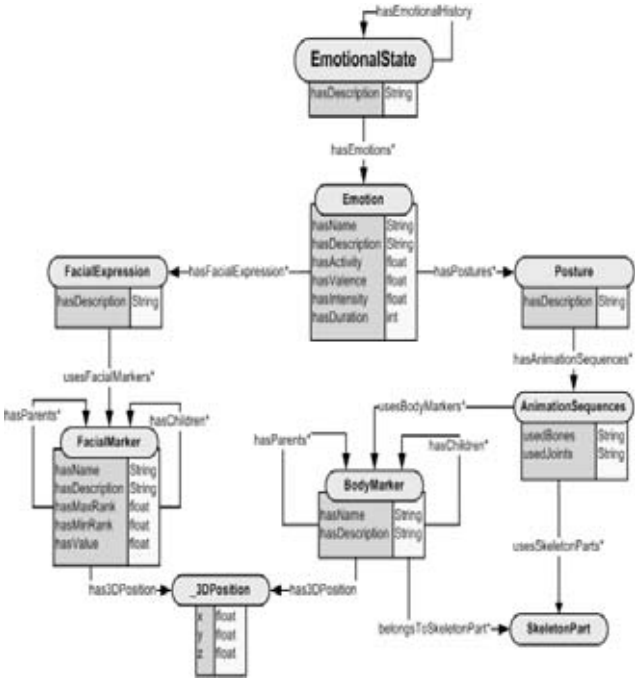

Figure 8. Emotional state of avatar

In the same way, the mood state of avatar is a set of moods with a mood history. Moods can also be positive or negative and active or passive, but also neutral. The personality of avatar is defined with the help of personality traits that can be positive or negative and active or passive. Personality traits can determine the behavior of avatar under the influence of certain events and stimuli. Finally, each skill that avatar learns is defined using a set of animation sequences. These sequences imply skeleton parts of avatar in motion and a set of corresponding facial expressions associated with them.

. V CASE STUDIES

Battle: This is a small example that shows a battle between two agents in a 2D environment (see figure 9). They are represented graphically as frogs. In this battle one frog fights against the other throwing bullets and trying to hit its rival. Both frog agents have the same kind of behavior. The behavior is based on basic actions like *move to the*

left or right and shoot. The mixture of these three simple actions defines the decision making.

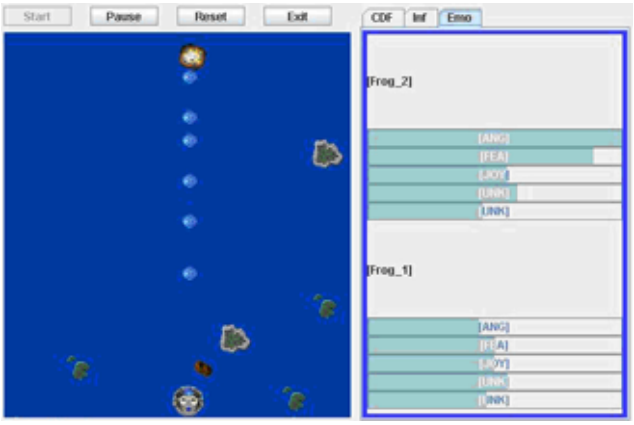

Figure 9. Frogs battle in 2D

Persecution: This case study was implemented using a virtual 3D environment which runs over the GeDA-3D Agent Architecture. Two kinds of agents are used, one is offensive and the other is defensive. The offensive agent persecutes the defensive one. The defensive agent tries to avoid the offensive agent's persecution, evading it when it is near and fleeing when it is closer. Both are emotional agents (see figure 10). Three different emotions have been defined for these agents: anger, joy and fear. The avatars modify their facial expressions and postures showing the emotion predominant in their emotional state. The implemented case studies were made using the PEM model [16] for updating the emotional and mood states of the avatars.

Figure 10. An offensive avatar that persecutes a defensive avatar

. VI Conclusions

In order to provide the avatars with a minimum conscience it is necessary to give them the capacity to understand their emotions. To understand a concept means to be able to express emotions. In this work, we have presented a methodology to determine how the agents must express their emotions. For each emotion we have defined two types of representation: a cognitive representation and a somatic representation. The cognitive representation determines the causes of the emotion. The interpretation of perceived events is based on the cognitive representation of the emotions, determined by the stimuli perceived and by the personality traits of avatar. Thus, emotion is triggered by particular interpretation of an event or stimulus that corresponds to the cognitive representation of the cause of a certain emotion. We have defined the emotional mental state as the cognitive representation of its inductive cause. The somatic representation determines how the avatars express their emotions. These representations allow to identify the mental states correspondent to each emotional state, and to determine the intensity of the emotions taking into account the influence of the avatar's personality.

In this work we presented an extension of the GeDA-3D Agent Architecture, and we have showed how the CAPE Agents must express their emotions using a knowledge base. The emotional agents' behavior and their interactions are influenced by their emotions and moods. Thus, the intelligence, the conscience and the cognitive processes help the avatars to take most accurate decisions in a more natural way. These decisions keep the agents' emotional status balanced and keep them alive in the environment. Sensorial entrances and influences the avatars receive from their environment alter their behavior and motion and also affect their emotions and mood. The GeDA-3D Agent Architecture owns the features defined in MAS theory, but it offers some extra features specific to the problem. This architecture contributes, for example, to the topics of goal-specification, skill based behaviors, collective knowledge bases, posture descriptors and facial animation descriptors.

. VII References

[1] H. Schmidl and M. Lin. *Geometry-Driven Physical Interaction Between Avatars and Virtual Environments*. Computer Animation and Virtual Worlds, Vol. 15, non. 3-4, pp. 229-236, 2004.

[2] F. Schwarzer, M. Saha and J. Latombe. *Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments.* IEEE Transactions On Robotics, Vol. 21, no. 3, pp. 338-353, June 2005.

[3] S. M. LaValle. *Planning Algorithms*, Cambridge University Press, 2006.

[4] C. Esteves, G. Arechavaleta and J. P. *Motion Planning for Human-Robot Interaction in Manipulation Task.* IEEE International Conference on Mechatronics and Automation, Vol. 4, pp. 1766- 1771 Laumond, 2005.

[5] T. CondeW. Tambellini and D. Thalmann, *Behavioral Animation of Autonomous Virtual Agents Helped by Reinforcement Learning*. Lecture Notes in Computer Science, vol. 272, pp. 175-180, Springer-Verlag: Berlin, 2003.

[6] J. Peters, S. Vijayakumar, and S. Schaal, *Reinforcement Learning for Humanoid Robotics*. International Conference on Humanoid Robots, pp. 1-20, Karlsruhe, Germany, September 2003.

[7] L. Herda, P. Fua and D. Thalmann. *Skeleton-Based Motion Capture for Robust Reconstruction of Human Motion.* Computer Animation, pp. 77-83, Philadelphia, PA, USA, 2000.

[8] D. Pereira, E. Oliveira, N. Moreira and L. Sarmento. *Towards an Architecture for Emotional BDI Agents*. Proceedings of 12th Portuguese Conference on Artificial Intelligence, pp. 40–47, 2005.

[9] A. Ortony, G. L. Clore and A. Collins. *The Cognitive Structure of Emotions.* New York: Cambridge Universty Press, 1988.

[10] M. S. El-Nasr, J. Yen and T. R. Ioerger. *FLAME - Fuzzy Logic Adaptive Model of Emotions*. Autonomous Agents and Muti-Agent Systems, Vol 3, no. 3, pp. 219-257, 2000.

[11] R. McCrae and O. John. *An Introduction to the Five-Factor Model and its Application*. *Journal of Personality,* vol. 60, no. 2, pp. 175-215, 1992.

[12] M. Masuch, K. Hartman and G. Schuster. *Emotional Agents for Interactive Environments*. Proceedings of the Fourth International Conference on Creating, Connecting and Collaborating through Computing, pp. 96-102, 2006.

[13] P. Ekman. *Moods, Emotions, and Traits, the Nature of Emotion: Fundamental Questions.* New York: Oxford University Press, 1994.

[14] F. Zúñiga, F. F. Ramos and I. Piza. G*eDA-3D Agent Architecture*. Proceedings of the 11th International Conference on Parallel and Distributed Systems, Fukuoka, Japan, pp. 201–205, 2005.

[15] H. I. Piza, F. Zúñiga and F. F. Ramos. *A Platform to Design and Run Dynamic Virtual Environments*. Proceedings of the 2004 International Conference on Cyberworlds, pp. 78-85, 2004.

[16] A. Egges, S. Kshirsagar, and N. Magnenat-Thalmann. *Generic Personality and Emotion Simulation for Conversational Agents*. Computer Animation and VirtualWorlds, Vol. 15, pp. 1-13, 2004.

Creation of Virtual Environments Through Knowledge-Aid Declarative Modeling

Jaime ZARAGOZA[a], Félix RAMOS[a], Héctor Rafael OROZCO[a] and Véronique GAILDRAT[b]

[a]Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara
Av. Científica 1145, Col. El Bajío, 45010 Zapopan, Jal., México
Email: jzaragoz, framos, horozco@gdl.cinvestav.mx
[b]Universitè Paul Sabatier
IRIT-CNRS UMR,
Toulouse Cedex 9, France
Email: gaildrat@irit.fr

Abstract. This article deals with the problem of Declarative Modeling of scenarios in 3D. The innovation presented in this work consists in the use of a knowledge base to aid the parsing of the declarative language. The example described in this paper shows that this strategy reduces the complexity of semantic analysis, which is rather time- and resource-consuming. The results of our work confirm that this contribution is useful mainly when the proposed language constructions have high complexity.

Keywords. Declarative Modeling, Virtual Environment, Declarative 3D editor, Animated Virtual Creatures, Virtual Agents.

Introduction

The use of computer graphics and technologies in fields such as computer games, film-making, training or even education has increased in the last years. The creation of virtual environments that represent real situations, fantasy worlds or lost places can be achieved with the help of several tools, from basic 3D modelers, like Flux Studio [1] or SketchUP [2], to complete suites for creating 3D worlds, such as 3D Max Studio [3] or Maya [4]. Creating complex and detailed worlds like those of films and video games with these tools requires experience [5]. However, frequently those who design scenarios are not experts; they don't have the experience of those who model the scenarios in 3D.

Our aim is to provide final users with a descriptive language which allows them to set a scenario and a scene and leaves the work of their creation for the computer. This

approach makes the VR available for final users of different areas, for those who create scenarios of plays, games or simulations of film scenes, for instance.

If the user could just describe the scenarios and/or scenes intended to be created and let the computer generate the environment, that is, the virtual scenario and the elements necessary for animating the scene, the creation of virtual environments could be at the reach of any user. So, even the non-expert users could find in the VR a valuable tool for completing their tasks. The objective of the GeDA-3D project is to facilitate the creation, development and management of virtual environments by both expert and non-expert users. The main objective of this article is to expose the method for generating virtual scenarios by means of the declarative modeling technique, supported by the knowledge base specialized in semantic analysis and geometric conflict solving. We adopt the definition from [6] for:

Definition 1: Declarative modeling is "a technique that allows the description of a scenario to be designed in an intuitive manner, by only giving some expected properties of the scenario and letting the software modeler find solutions, if any, which satisfy the restrictions". This process usually is divided in three phases [7]: *Description* (defines the interaction language); *Scene generation* (the result of one or more scenes the modeler generates, that match with the description introduced by the user); *Insight* (corresponds to results presented to the user; if more than one result is found, the user must choose the solution).

The bases for the creation of the virtual scenario are centered in the declarative modeling method, which is supported by constraint satisfaction problem (CSP) solving techniques. These techniques solve any conflict between the geometry of the model's entities. This procedure is also supported by the knowledge base, which contains all the necessary information to both satisfy the user's request, and validate the solutions obtained by the method.

The following parts of this chapter contain the description of the related works, the description of GeDA-3D architecture (the GeDA-3D is a complete project that includes the present work as one of its parts), our proposal to declarative modeling with the use of ontology, our conclusions and future work.

1. Related Works

The creation of virtual worlds and their representation in 3D can be a difficult task, especially for the users with lack of experience in the field of 3D modeling. The reason is that the tools necessary for modeling are often difficult to manage, and even experienced users require some time to create stunning results, such as those seen in video games or movies.

There are several works focused on declarative modeling. Some of them are oriented to architectonic aspects, others to virtual scenarios generation, sketching or design. Among the tools focused on architectonic development we can highlight the following ones:

The FL-System by Jean-Eudes Marvie et al. [8] specializes in the generation of complex city models. The system's input is defined in a specialized grammar and can generate city models of variable complexity. It is completely based on a System-L variant and uses VRML97 for the visualization of the models.

CityEngine [9], by Yoav I. H. Parish and Pascal Müeller, is a project focused on the modeling of full cities, with an entry composed by statistical data and geographic information. It also bases its design method on the System-L, using a specialized grammar.

However, our interest goes to the works focused on the creation of virtual scenarios, such as:

1.1.WordsEye: Automatic text-to-scene conversion system

Bob Coyne and Richard Asproad have presented WordsEye [10] developed at the AT&T laboratories, a system which allows any user to generate a 3D scene on the basis of description written in human language, for instance: "The bird is in the bird cage. The bird cage is on the chair". The text is initially marked and analyzed using a part-of-speech tagger and a statistical analyzer. The output of this process is the analysis tree that represents the structure of the sentence. Next, a *depictor* (low level graphic representation) is assigned to each semantic element. These depictors are modified to match with the poses and actions described in the text by means of the inverse kinematics. After that, the depictors' implicit and conflicted constraints are solved. Each depictor then is applied keeping its constraints, to incrementally build the scene, the background environment, the terrain plane. The lights are added and the camera is positioned to finally represent the scene. In the case the text includes some abstraction or description that does not contain physical properties or relations, other techniques can be used, such as: textualization, emblematization, characterization, lateralization or personification. This system accomplishes the text-to-scene conversion by using statistical methods and constraints solvers, and also has a variety of techniques to represent certain expressions. However, the scenes are presented in static form, and the user has no interaction with the representation.

1.2.DEM^{2}ONS: High Level Declarative Modeler for 3D Graphic Applications

DEM^{2}ONS has been designed by Ghassan Kwaiter, Véronique Gaildrat and René Caubet to offer the user the possibility to construct easily the 3D scenes in a natural way and with high level of abstraction. It is composed of two parts: *modal interface* and *3D scene modeler* [11]. The modal interface of DEM^{2}ONS allows the user to communicate with the system, using simultaneously several combined methods provided by different input modules (data globes, speech recognition system, spaceball and mouse). The syntactic analysis and dedicated interface modules evaluate and control the low-level events to transform them into normalized events. For the 3D scene modeler ORANOS is used. It is a constraint solver designed with several characteristics that allows the expansion of declarative modeling applications, such as: generality, breakup prevention and dynamic constraint solving. Finally, the objects are modeled and rendered by the Inventor Tool Set, which provides the GUI (Graphic User Interface), as well as the menus and tabs. This system solves any constraint problem, but only allows the creation of static objects, with no avatar support and no interaction between the user and the environment, not to mention modifications in the scenario's description.

1.3.Multiformes: Declarative Modeler as 3D sketch tool

William Ruchaud and Dimitri Plemenos have presented Multiformes [12], a general purpose declarative modeler, specially designed for sketching 3D scenarios. As it is by any other declarative modeler, the input of MultiFormes contains description of the scenario to create (geometric elements' characteristics and the relationships between them). The most important feature of MultiFormes is its ability to explore automatically all the possible variations of the scenario description. This means that Multiformes presents several variations of the same scenario. This can lead the user to choose a variation not considered initially. In Multiformes, a scenario's description includes two sets: the set of geometric objects present in the scenario, and the set of relationships existent between these geometric objects. To allow the progressive refinement of the scenario, MultiFormes uses hierarchical approximations for the scenario modeling. This approach allows describing the scenario incrementally to obtain more or less detailed descriptions. The geometric restriction solver is the core of the system and it allows exploring different variations of a sketch. Even when this system obtains its solutions in incremental ways and is capable of solving the constraints requested, the user must tell the system how to construct the scenario using complex mathematical sets, as opposed to our declarative creation, where the user formulates simple states in a natural-like language only describing what should be created.

1.4.CAPS: Constraint-Based Automatic Placement System

CAPS is a positioning system based on restrictions [13], developed by Ken Xu, Kame Stewart and Eugene Fiume. It models big and complex scenarios using a set of intuitive positioning restrictions, which allows to manage several objects simultaneously. Pseudo-physic is used to assure stable positioning. The system also employs semantic techniques to position objects, using concepts such as fragility, usability or interaction between the objects. Finally, it allows using input methods with high levels of freedom, such as Space Ball or Data Glove. The objects can only be positioned one by one. The direct interaction with the objects is possible while maintaining the relationships between them by means of pseudo-physics or grouping. The CAPS system is oriented towards scenario visualization, with no possibilities for self-evolution.

None of the previously described systems allows evolution of the entities or environments created by the user. All these works rely on specialized data input methods, either specialized hardware or input language, with the sole exception of WordEyes. Its output is a static view of the scenario created by means of natural language, but in spite of it, its representation abilities are limited, that is to say, it is not possible just to describe goals. Representing unknown concepts can lead specialized techniques to bizarre interpretations.

1.5.Knowledge Base Creation Tools

There are several ontology creation tools, which vary in standards and languages. Some of them are:

1.5.1.Ontolingua [14].

Developed at the KSL of the Stanford University, consists of the *server* and the *representation language*. The server provides ontology with repository, allowing its creation and modification. Ontologies in the repository can be joined or included in a new ontology. Interaction with the server is conducted by the use of any standard web browser. The server is designed to allow cooperation in ontology creation, that is to say, easy creation of new ontologies by including (parts of) existing ontologies from the repository and the primitives from the ontology frame. Ontologies stored on the server can be converted into different formats and it is possible to transfer definitions from the ontologies in different languages into the Ontolingua language. The Ontolingua server can be accessed by other programs that know how to use the ontology store in the representation language [15].

1.5.2.WebOnto [16].

Completely accessible from the internet, it was developed by the Knowledge Media Institute of the Open University and Design to support creation, navigation and collaborative edition of ontologies. WebOnto was designed to provide a graphic interface; it allows direct manipulation and complements the ontology discussion tool Tadzebao. This tool uses the language OCLM (Operational Conceptual Modeling Language), originally developed for the VITAL project [17], to model ontologies. The tool offers a number of useful characteristics, such as saving structure diagrams, relationships view, classes, rules, etc. Other characteristics include cooperative work on ontologies, also broadcast and function reception.

1.5.3.Protégé [18].

Protégé was designed to build domain model ontologies. Developed by the Informatics Medic Section of Stanford, it assists software developers in the creation and support of explicit domain models and incorporation of such models directly into the software code. The methodology allows the system constructors to develop software from modular components to domain models and independent problem solving methods, which implement procedural strategies to accomplish tasks [19]. It is formed of three main sections: *ontology editor* (allows to develop the domain ontology by expanding the hierarchical structure, including classes and concrete or abstract slots); *KA tool* (can be adjusted to the user's needs by means of the "Layout" editor), and *Layout interpreter* (reads the output of the layout editor and shows the user the input-screen that is necessary to construct the examples of classes and sub-classes). The whole tool is graphic and beginner users oriented.

2. GeDA-3D Agent Architecture

GeDA-3D [20] is a final user oriented platform, which was developed for creating, designing and executing 3D dynamic virtual environments in a distributed manner. The platform offers facilities for managing the communication between software agents and mobility services. GeDA-3D Agents Architecture allows to reuse behaviors necessary to configure agents, which participate in the environments generated by the virtual

environments declarative editor [20, 21]. Such agent architecture contains the following main features:

- *Skeleton animation engine*: This engine provides a completely autonomous animation of virtual creatures. It consists of a set of algorithms, which allow the skeleton to animate autonomously its members, producing more realistic animations, when the avatar achieves its objectives.
- *Goals specification*: The agent is able to receive a goal specification that contains a detailed definition of the way the goals of the agent must be reached. The global behavior of the agent is goal-oriented; the agent tries to accomplish completely its specification.
- *Skills-Based Behaviors*: The agent can conform its global behavior by adding the necessary skills. It means that such skills are shared and implemented as mobile services registered in GeDA-3D.
- *Agent personality and emotion simulation*: Agent personality and emotion simulation make difference in behaviors of agents endowed with the same set of skills and primitive actions. Thus, the agents are entities in constant change, because their behavior is affected by their emotions.

Figure 1 presents the architecture of the platform GeDA-3D. The modules that constitute it are: *Virtual-Environment Editor* (*VEE*), *Rendering*, *GeDA-3D kernel*, *Agents Community* (*AC*) and *Context Descriptor* (*CD*). The striped rectangle encloses the main components of GeDA-3D: *scene-control* and *agents-control*.

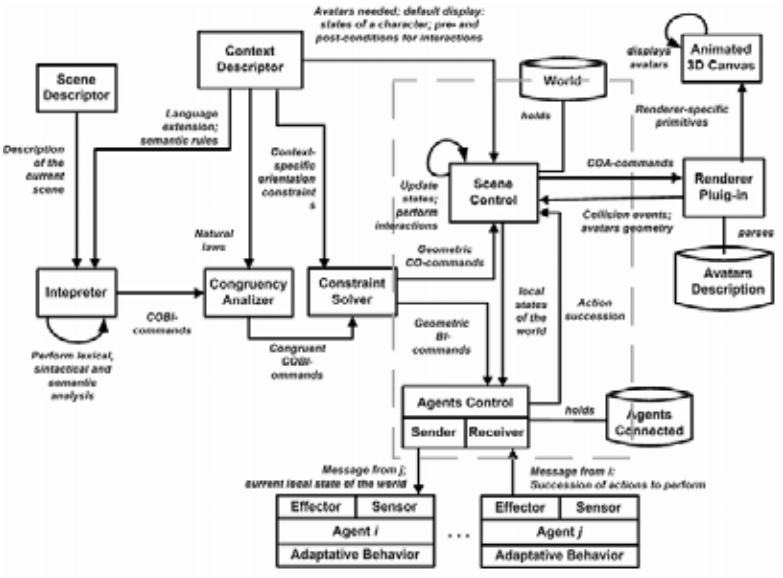

Figure 1. GeDA-3D Agent Architecture

The *VEE* includes the scene descriptor, interpreter, congruency analyzer, constraint solver, and scene editor. In fact, the *VEE* provides the interface between the platform

and the user. It specifies the physical laws that govern the environment and describes a virtual scene taking place in this environment. The *Rendering* module addresses all the issues related to 3D graphics; it provides the design of virtual objects and the scene display. The *AC* is composed by the agents that are in charge of ruling virtual objects behavior.

The scene description provides an agent with detailed information about what task the user wants it to accomplish instead of how this task must be accomplished. Furthermore, the scene might involve a set of goals for a single agent, and it is not necessary that these goals are reached in a sequential way. The user doesn't need to establish the sequence of actions the agent must perform, it is enough to set goals and provide the agent with the set of primitive actions. The agents are able to add shared skills into their global behavior. So, the user describes a scene with the help of a high level language similar to human language. He or she specifies the goals to reach, that is to say, what the agent must do, not the way it must do it. It is important to highlight that one specification can produce different simulations.

3. Proposal

As described previously, the aim of our research is to provide the non-experienced final users with a simple and reliable tool to generate 3D worlds on the basis of description written in a human-like language. This article describes the *Virtual Environment Editor* of the GeDA-3D project. It receives the characteristics of the desired environment as input and validates all the instructions for the most accurate representation of the virtual world. The model includes the environment's context, the agents' instructions and the 3D view render modeling data. The novel approach uses the Knowledge Base [22] to obtain the information necessary to verify first the validity of the environment description, and later the validity of the model in the methodology process. Once this first step has been finished, the ontology is applied to obtain information required for visualizing the environment created in 3D and validating the actions computed by agents, taking in charge the behavior of avatars.

The main concern of this research is to assign the right meaning to each concept of the description introduced as input by the user. Human language is very ambiguous, so the parse of it is time and resource consuming. To avoid this hard work and make the writing of descriptions easy, we have defined a language completely oriented to scenario descriptions. We have called this language ***V***irtual ***E***nvironment ***De***scription ***L***anguage or VEDEL.

3.1. Virtual Environment Description Language (VEDEL)

VEDEL is a declarative scripting language in terms described in [23, 24]: A declarative language encourages focusing more on the problem than on the algorithms of its solution. In order to achieve this objective, it provides tools with high level of abstraction. A scripting language, also known as a *glue language*, assumes that there already exists a collection of components, written in other languages. It is not intended for creating applications from scratch. In our approach, this means that the user specifies what objects should be included in the scenario and describes their location, but he or she does not specify how the scenario must be constructed. It is also assumed

that the required components are defined somewhere else, in our case in the object and avatar database. The definition of VEDEL using the EBNF (Extended Backus Normal Form) is:

Alphabet:
Σ = {[**A** − **Z** | **a** − **z**] , [**0** − **9**] **, , .**}
Grammar:
Description := <environment> <actor> <object>
<environment> := **[ENV]** <sentence> **[/ENV]**
<actor> := **[ACTOR]** <sentence>*** [ACTOR]**
<object> := **[OBJECT]** <sentence>*** [/OBJECT]**
<sentence> := <class>(<**,**> <property>) ***.** >
<class> := <entity> (<identifier>)\
<property> := <propid> (<value>+)
Vocabulary:
<class> := <word>
<entity> := <word>, <entity> ∈ Ontology's Classes
<identifier> := <word>
<propid> := <word>, <propid> ∈ Ontology's Classes
<value> := <word> | <number> | <identifier>
<modifier> := <word>
< identifier >:= ([**A** − **Z** | **a** − **z**] | [**0** − **9**])+
< word >:= [**A** − **Z** | **a** − **z**]+
< number >:= [**0** − **9**] + (**.** [**0** − **9**]+)

The language is supposed to guide the user through the construction and edition of the description, separating the text into three paragraphs: Environment, Actors and Objects. Each paragraph is composed of sentences, at least one for the Environment paragraph, and any number for the Actors and Objects paragraphs. The sentences are composed of comma-separated statements, the first statement always being the entity's class, that is, a concept included in the Knowledge Base and an optional unique identifier. The rest of the sentences correspond to the features characteristic for a particular entity. Each sentence must end with a dot ".".

In the Environment section the user defines the context, the general setting of the scenario. The Actors section contains all the avatars, that is, entities with an autonomous behavior, which interact with other entities and the environment. It is important to notice that the Knowledge Base is the base for assigning behaviors to avatars. For instance, even when the entity "dog" is defined as an actor (autonomous entity), but the Knowledge Base does not define any features of its behavior (walk, bark, etc.), the actor will be managed as an inanimate object.

The description is validated through the lexical and semantic parser. This parser is basically a state machine, which prevents the entry for non-valid characters and bad formatted statements. If the parser finds some error in the description, the statement or the whole sentence is not taken in account for the modeling process, and an error message is produced to notify the user. The parser works with the following rule: given any description X, written under the VEDEL definition $G = \{\Sigma, N, P, S\}$ for the language L, X is a VEDEL compliant description, if and only if $X \subseteq L(G)$ and $L \Rightarrow_i X$.

```
[ENV] //Environment section.
  desert, night,  cloudy.
  //every sentence must end with a dot ".".
[/ENV]
[ACTOR] //Actor section.
  Knight Arthur, centre, facing west.
  YoungWoman Betty, front Arthur, facing south.
  //Class names must be defined in ontology.
[/ACTOR]
[OBJECT] //Object section.
  Chair 0, behind Arthur, facing west.
  //Individual names  can be any long, but only
  // alphanumeric characters.
  House Bettys, south.
  Table, front Betty, crystal blue translucyd.
  //Table individual name will be automatically
  //assigned as Table0.
  //The number of property values varies, accordantly
  //to the knowledge in ontology
[/OBJECT]
```

Figure 2. Example of a description written in VEDEL

The parsed VEDEL entry is formatted as a hierarchical structure, so the modeler can access any sentence at any moment, usually making a single pass from the top entry (the environment) to the bottom entry. Each entry is in its turn formed of the class entry (the optional identifier, which is automatically generated in the incremental way, if the user doesn't explicitly establish one) and the properties of entries. We have decided to use the Protégé system due to its open API, written in Java, and also due to its simple, yet rich GUI, which allows quick creation, modification and maintaining of the Knowledge Base, easy to integrate into our project.

Once we have parsed the entry written in VEDEL, we need to establish the meaning for each concept introduced by the user. To accomplish this task, we rely on the Knowledge Base. This Knowledge Base stores all the information relevant for the concepts to be represented, from physical properties, such as size, weight or color, to internal properties, such as energy, emotions or stamina. The parsed VEDEL entry is revised by the inference function, which makes use of the ontology to validate the entities and their properties, as well as to validate the congruency in the scenario. This function also performs in the parsed entry the operations necessary to transform values from the string entry to primitive data form. The Knowledge Base consists of the following components:

- A finite set C of classes.
- A finite set P of properties.
- A finite set $D = (C \quad P)$ of class-properties assignments.
- If $x \in C$ and $y \in C$ is such that $y \in x$, y is a *subclass* of x.
- If $x \in P$ and $y \in P$ is such that $y \in x$, y is a *subclass* of x.

- Given a certain $D(x)$, $D(x) \supseteq D(y)$, for all y subclass of x.
- An *Object* property is such that $O = (C \lozenge C)$, where $\lozenge$ is the functional relationship (Functional, Inverse Functional, Transitive or Symmetric).
- A *DataType* property is such that $V = \{values\}$, where $\{values\}$ is a finite set of any typeset values.
- An Individual is the instantiation of the class x, in such form that every p in $P(x)$ is assigned with a specific set (empty or otherwise).

In our project, $C = \{Environment, Actor, Object, Keywords\}$, and the following subsets of properties are obligatory for each subclass of C elements:

- For all $x \in Environment$, $Q_e = \{size, attributes\}$, $Q_e(x) \subseteq D(x)$.
- For all $x \in Actor, Objects$, $Q_a = Qe \cup \{geozones, geotags, property\_type\}$, $Q_a(x) \subseteq D(x)$.
- For all $x \in Keywords$, $Q_k = Q_a \cup \{attributes\}$, $Q_k(x) \subseteq D(x)$.

The ontology has been described with respect for naming conventions. However, in addition any class must have at least one individual named as the class and the "_default" suffix at the end. This is the most basic representation for the class. Figure 3 shows the main structure of the ontology, the properties currently added and the individuals view. Any additional class will only be used by the output generation function, but the relationships between the subclasses and the rest of the ontology will be respected by the environment. For instance, Figure 3 shows additional classes "laws", "actions" and "collisions". While these classes are not required directly by the final user by means of calling their members, the modeler will take into account any relationship between the main four classes and these additional classes. This means that the object property "AllowedOn" will be used to validate the actions that can be performed in a given environment not at the modeler's level, but as an explicit rule written for the context and the underlying architecture, since only the actions linked to the environment can be assigned to the architecture.

The process of validating any expression found in the description is carried out by the inference function. This function accesses the Knowledge Base, searching for the individual that matches the pattern "<*expression*>*_default*". We can call it the *definition individual*. This individual can be the obligatory class definition or some other individual. If there is no definition individual for the expression, an error condition will arise, and the statement or the sentence will be excluded from the rest of the modeling process. If the definition individual is found, the inference function precedes to gather all the information stored in the Knowledge Base for that expression, according to the parameter it has received. In the case of entities definitions, it subtracts all the basic information, such as size, specialized tags and the properties specified in the *attribute* property. For the entity's properties, the inference function searches for the corresponding definition individual and gathers the data according to the values stored in the *property type* property. This differentiation is made since there are cases, when the object can be a standalone entity or part of another, bigger entity. Also, the same concept can be used for different requests, i.e.,

the concept “north” can be used to either place an entity in a specific position, or to indicate the orientation of the entity. The rules for the inference function are:

$$\forall x \in \{description\}, \exists y \in C \vee y \in I \Rightarrow P(y) = P(x) \vee P_n(y) = x$$

$$\text{If } P(x) = P(y) \Rightarrow x, y \in C \wedge \exists x_i, y_i \in I$$

$$\text{If } P_n(y) = x \Rightarrow y \in C \wedge ((x \in D \wedge x_i \in I) \vee x \in V)$$

$$\forall x \in C \Rightarrow \exists x_i \in I$$

$$\forall y \in C \Rightarrow \exists y_i \in I$$

The gathered information is formatted as the basic entry for the model data structure, which is a hierarchical structure with the entity entries as the top levels and the properties for each one of the entities as leaves, as depicted in Figure 4. Each entry is instantiated with the basic definition: the feature values for successful representation of the entity in the model, according to the knowledge stored in the Knowledge Base. These basic entries are then modified with the help of values obtained from the parsed entry description, once they have passed the inference process, have been validated and transformed to the format suitable for the underlying architecture.

3.2.Knowledge in algorithms for constraint satisfaction problems

In [25], Hunter defines the Constraint Satisfaction Problems as follows: a CSP is a tuple composed by a finite set of variables *N*, a set of possible values *D* for the variables in *N* and a set of constraints *C*. For each variable N_i, there is a domain D_i, so that to each variable can only be assigned values from its own domain. A restriction is a subset of the variables in N and the possible values that can be assigned them from D. Constraints are named according to number of variables they control. A solution to a CSP is an assignment of values form the set of domains D to each variable in the set N, in such way that none of the constraints is violated. Each problem that presents solution is considered satisfied or consistent; otherwise it is called non-satisfied or inconsistent.

To solve a CSP, two approximations can be used: *search* and *deduction*. The search generally consists in selecting an action to develop, maybe with the aid of a heuristic, which will lead to the desired objective. Tracking is an example of search for CSP; this operation is applied to value assignation for one or more variables. If no value able to keep the consistency of the solution can be assigned to the variable (the dead-end is reached), the tracking is executed.

There are several methods and heuristics for solving CSPs, among them backtracking [26], backmarking [27], backjumping [28], backjumping based on graphics [29] and forward checking [30].

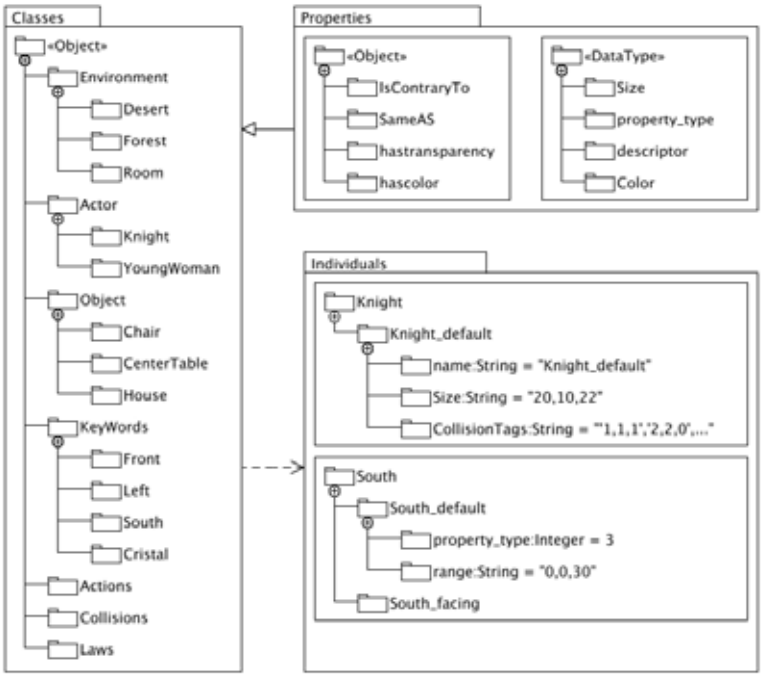

Figure 3. The ontology used by the Virtual Editor

The parsed entry obtained from the description and validated through the inference function is sent to the modeling function, which generates the first model that is called zero-state model. At this stage, all geometrical values are set to default: either to zero or to the entity's default values and the corresponding request is stored in the model for quick access and verification. Objects whose position is not specified are placed in the center of the scenario (geometrical origin) and the specific positions are respected. For those entities whose position is established in relation to others, a previous evaluation is conducted in order to position first the entities referenced. Any specified position is assigned, even if there are possible conflicts with other elements, the environment, the rules of the environment or the properties of the entity.

With the objects in their initial positions and postures, the model is sent to the logical and geometrical congruency analyzer. First it is verified, if no properties are violated with the current requests (objects with postures that cannot be represented, objects positioned over entities that cannot support them, etc.), and then the geometry of the scenario is verified for spatial conflicts. The validation of the current model state, as well as the modifications made to obtain a valid model state that satisfy the user request, is achieved by means of the Constraint Satisfaction Problem (CPS) solving algorithm, whose parameters are detailed as follows:

A set $V = \{x_1, \ldots, x_n\}$ of variables where $x = \{P, O, S\}$ such as

- $P = \{x, y, z\}$ is the entity's position in the environment.
- $O = \{\theta_x, \theta_y, \theta_z\}$ is the entity's orientation.
- $S = \{S_x, S_y, S_z\}$ is the entity's size, including the scale.

A domain D for the set V such as

- $D(P) = \Re^3$
- $D(O) = [0, 2\pi], \forall \theta \in O$
- $D(S) = \Re^3$.

A set C of constraints, formed by the next functions:

(1.a) $(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 - (r_1 + r_2) = 0$

(1.b) $\left(\frac{x}{p}\right)^2 + \left(\frac{y}{q}\right)^2 - 2z = 0$

(1.c) $\left(\frac{x}{dx}\right)^2 + \left(\frac{y}{dy}\right)^2 + \left(\frac{z}{dz}\right)^2 - 1 = 0$

Before validating the request, information for each spatial property is obtained from the Knowledge Base in the form of vectors that indicate positioning, either absolute or relative, the maximum distance to which the objects can be separated from each other still fulfilling the requests, and orientation, relative or absolute.

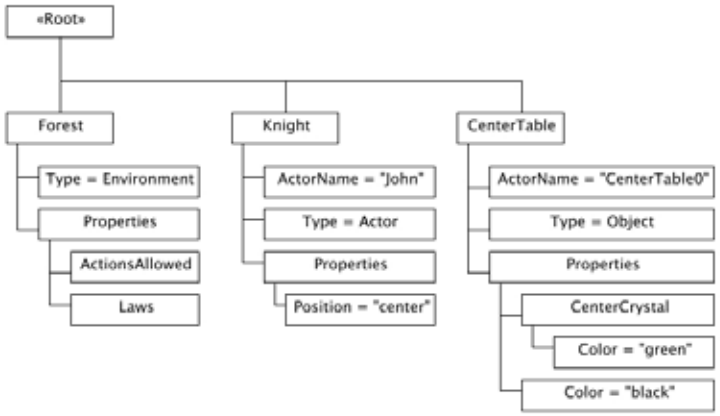

Figure 4. Data structure obtained from a Description

Once position and orientation have been set for each entity, the geometrical analyzer verifies that no collision occurs between objects. This is achieved with the help of several collision tags also stored in the Knowledge Base for each entity. These tags are extracted, instantiated with the entity's parameters, and then compared with the others entities' tags. This is our first constraint, since there must be no collision

between tags to declare the model valid. The only exceptions to this rule are the indications for close proximity between the entities (zero distance or distance against request). The tags are modeled as spheres that cover all or most of the entity's geometry (see figure 5). The equation for verifying the collision between two spheres is simple and quick to solve (see equation 1.a in the CSP parameters definition).

While collision consistency is being conducted, other tests are also conducted to secure position consistency. Each positional statement (left, right, front, behind) has a valid position volume assigned, where the objects that hold a spatial relationship to the entity must be put, once this volume has been instantiated. The test function is a quadratic equation, represented in the model as a parabolic (see equation 1.b in the CSP parameters definition). This representation was chosen, because it is relatively easy to solve, allows the verification of spaces nearly of the same size as the classic collision boxes and can be shaped to the entity's measures. If the entity being positioned does not pass the consistency test, that is, none of its characteristic points (figure 6) is inside the consistency function, a new position is computed, and the collision and positioning revision is executed again. In the specific case where two entities have the same relation to a third one, a new position is computed and the characteristic points tested for each entity. When it is not possible to find a new position that meets both constraints, a model error is generated and the user is informed about it.

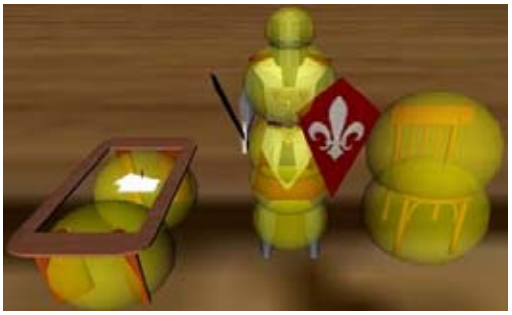

Figure 5. Collision tags examples

For special positioning request there is a third type of consistency function, the ellipsoid (see equation 1.c in the CSP parameters definition). These special positions are the inside, over, under, and against concepts. The reason for the choice of this type of equations is the capacity to shape the volume in different sizes: the volume that covers all or the majority of the space inside or as part of the entity, or a smaller volume set in the corresponding area of the target entity. The same heuristics are applied to the other position request. World boundaries are validated through this function. For the entities in the Actors section, requests are sent to the animation module, which sets the position of each articulation in the entity to obtain the desired posture. This information is used to provide the corresponding collision tags and characteristic points, which in turn allow the modeler to verify consistency for the postures and actions the actor will be performing. If several solutions are found for the current request, the user can choose the one that fits better the requests. Normally, a well constrained problem would find a finite number of solutions, but in the case of under-constrained problems, it could be an infinite number of solutions. In these cases,

the modeler can show only a predefined number of solutions, or can show in iterative form all the possible solutions, until the user chooses one.

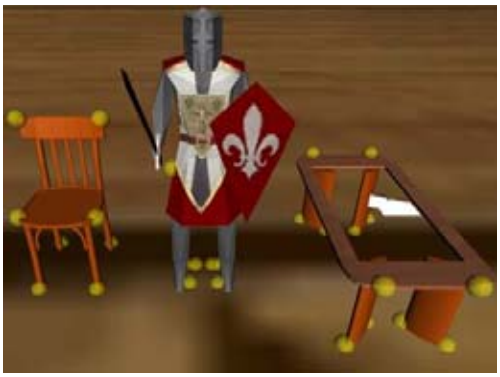

Figure 6. Characteristic points for different entities

The final step in the modeling process is the generation of the necessary outputs in order to allow the created environment to be visualized and modified. These outputs are generated using the MVC or Model View Controller. This method provides any desired modifications in the outputs that will be generated without modifying the code for the modeler, and allows the users to customize the outputs and integrate the modeler in multiple applications.

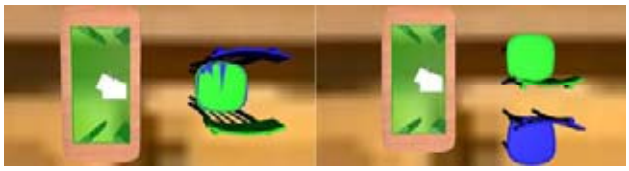

Figure 7. Conflict solving example

4. Results

Until now the parser for the VEDEL language has been implemented and it was functionally working. The necessary links with the GeDa-3D architecture were established in order to obtain a visual representation of the descriptions written in this language.

4.1.The VEDEL Parser and Modeler

The parser was created in Java language to reach multiplatform capabilities and compatibility with the rest of the GeDA-3D architecture, using the JDK 1.5.0_07-b03 [31]. Our parser is basically a state machine: each section of the description corresponds to a state, as well as each sentence and each property. If the state generates

an error output, the process is stopped and an error condition arises. Each word is considered to be a token and validated by the inference machine, with the exception of numbers, particular identifiers and closing/opening constructions. The inference machine and the modeler described in the previous section are used to validate semantically the parsed description, and then the data structure that represents the model is obtained. Finally, the outputs are generated using the MVC (Model View Controller) function. The templates are formatted, so they can be filled with the data stored in the data structure, and the formatted output of each entry in the model forms the complete output.

4.2.The GeDA-3D prototypes

To obtain a visual output of the description written in VEDEL, the prototype of the GeDA-3D architecture was created. The prototype has a kernel [32], a render working upon the AVE (Animation of Virtual Creatures) project [33], and our parser. This prototype works as follows: the kernel received the outputs generated by the parser (one output for the kernel and one in LIA-3D, Language of Interface for Animations in 3D presented in [33], for the parser). Then it generates the necessary agents, and the AVE output is sent to the render module, where the scenario is composed, rendered and presented to the final user. Next, we present some examples obtained by the X3D [34] based output (figures 8 to 10) and their corresponding descriptions.

Description 1.

[ENV]
forest.
[/ENV]

[ACTOR]
Knight A, center.
Knight B, left A.
Knight C, right A.
Knight D, behind B.
Knight E, behind C.
[/ACTOR]

[OBJECT]
House, behind (80) A.
[/OBJECT]

Figure 8. Result obtained for description 1

Description 2.

[ENV]
theater.
[/ENV]

[ACTOR]
Knight A, center.
Knight B, left A.
Knight C, right A.
YoungWoman D, behind B, facing A.
YoungWoman E, behind C, facing A.
[/ACTOR]

[OBJECT]
[/OBJECT]

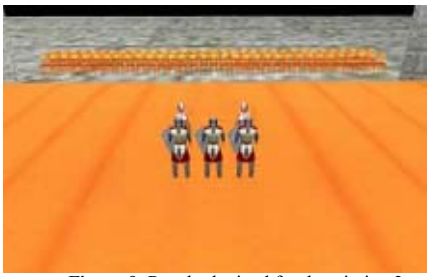

Figure 9. Result obtained for description 2

Description 3.

[ENV]
void.
[/ENV]

[ACTOR]
[/ACTOR]

[OBJECT]
Chair, color blue.
Chair, color red.
Chair, color green.
[/OBJECT]

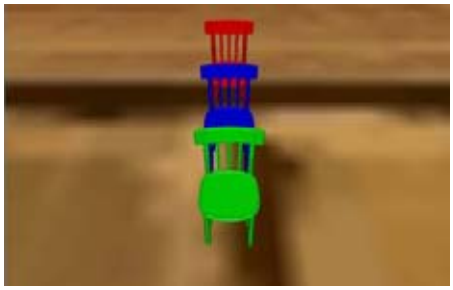

Figure 10. Result obtained for description 3

5. Conclusions

Our contribution to the research topic described in this article concerns declarative modeling for creation of scenarios. This problem is important, because the design of virtual scenarios is time- and labor-consuming even for expert users who have appropriate tools at their disposal. In this article we contribute mainly to the use of knowledge databases to support and accelerate the semantic analysis of sentences that compose the declarative form of a scenario. More specifically, our proposal uses a Knowledge Base in the three phases that constitute the declarative modeling. During the *Description* phase the Knowledge Base helps to get semantic elements necessary for verifying the description, that is, the input to the system. For the *Model Creation* the modeler will use the Knowledge Database to obtain the information necessary for the model creation. To accomplish this task the modeler uses a restriction satisfaction algorithm supported by the Knowledge Base. Finally, the user applies the Knowledge Base in the *Vista* phase to obtain information about the requirements which could not be satisfied, in case there wasn't found any solution, or select one of the possible solutions.

The approach we have proposed shows two very important advantages: The first one concern the fact that the solution obtained in this way can be used in systems able to evolve a scene, as the GeDA-3D project described in this article. The second one concerns the possibility of expanding the available environments and entities just by increasing the knowledge database, leading the declarative editor towards a generic, open architecture.

Some of the results obtained include: a structured method for creating and editing descriptions with the use of tools that validate the inputs, such as lexical and semantic analyzers; the implementation of prototypes useful for visualizing the generated scenario on the basis of the respective description. These results are important because they validate our proposal. This validation proves the possibility for creating more types of scenarios, just increasing the knowledge database with the corresponding information.

Our future work includes the development of more robust CPS algorithms supported by the Knowledge Base, that not only consider the physical properties of the

entities, but also the context properties and the semantic properties; a semantic validation function, which verifies, if every entity inserted in the environment is capable or allowed to exist in such environment, and, in some cases, provides the necessary changes in the entity's properties, so it can get a valid element; and finally, the complete integration with the GeDA-3D architecture.

References

[1] K. Victor. Flux Studio Web 3D Authoring Tool. http://wiki.mediamachines.com/index.php/Flux_Studio, 2007.

[2] Last Software. Google Sketchup. http://sketchup.google.com/, 2008.

[3] Autodesk. 3DS MAX 9 Tutorials. http://usa.autodesk.com/adskservlet/item?siteID=123112&id=8177537, 2007.

[4] Autodesk. Autodesk Maya Help. http://www.autodesk.com/us/maya/docs/Maya85/wwhelp/wwhimpl/js/html/wwhelp.htm, 2007.

[5] J. S. Monzani, A. Caicedo and D. Thalmann. Integrating Behavioral Animation Techniques. In EG 2001 Proceedings, volume 20(3), pages 309–318. Blackwell Publishing, 2001.

[6] D. Plemenos, G. Miaoulis and N. Vassilas. Machine Learning for a General Purpose Declarative Scene Modeller. In International Conference GraphiCon'2002, Nizhny Novgorod (Russia), September 15-21, 2002.

[7] V. Gaildrat. Declarative Modelling of Virtual Environment, Overview of Issues and Applications. In International Conference on Computer Graphics and Artificial Intelligence (3IA), Athenes, Grece, volume 10, pages 5–15. Laboratoire XLIM - Université de Limoges, may 2007.

[8] J.-E. Marvie, J. Perret and K. Bouatouch. The FL-System: A Functional L-System for Procedural Geometric Modeling. The Visual Computer, pages 329 – 339, June 2005.

[9] Y. I. H. Parish and P. Müeller. 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, 2001. ACM Press.

[10] B. Coyne and R. Sproat. Wordseye: An Automatic Text-To-Scene Conversion System. In SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pages 487–496. AT&T Labs Research, 2001.

[11] G. Kwaiter, V. Gaildrat and R. Caubet. Dem2ons: A High Level Declarative Modeler for 3D Graphics Applications. In Proceedings of the International Conference on Imaging Science Systems and Technology, CISST'97, pages 149–154, 1997.

[12] W. Ruchaud and D. Plemeno. Multiformes: A Declarative Modeller as a 3D Scene Sketching Tool. In ICCVG, 2002.

[13] K. Xu, A. J. Stewart and E. Fiume. Constraint-Based Automatic Placement for Scene Composition. In Graphics Interface, pages 25–34, May 2002.

[14] A. Farquhar, R. Fikes and J. Rice. The Ontolingua Server: A Tool for Collaborative Ontology Construction. Technical report, Knowledge Systems Laboratory, Stanford University, 1996.

[15] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2):199–220, June 1993.

[16] J. Domingue. Tadzebao and Webonto: Discussing, Browsing, and Editing Ontologies on the Web. In Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling and Management, KAW'98, Banff, Canada, April 1998

[17] E. Motta. Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1999.

[18] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu and M. A. Musen. Knowledge Modeling at the Millennium (the Design and Evolution of Protégé-2000). Technical Report, Stanford Medical Informatics, 1998.

[19] H. Eriksson, Y. Shahar, S. W. Tu, A. R. Puerta and M. A. Musen. Task Modeling with Reusable Problem-Solving Methods. Artificial Intelligence, 79(2):293–326, 1995.

[20] F. Zúñiga, F. F. Ramos and I. Piza. GeDA-3D Agent Architecture. Proceedings of the 11th International Conference on Parallel and Distributed Systems, pages 201–205, Fukuoka, Japan, 2005.

[21] H. I. Piza, F. Zúñiga and F. F. Ramos. A Platform to Design and Run Dynamic Virtual Environments. Proceedings of the 2004 International Conference on Cyberworlds, pp. 78-85, 2004.

[22] J. A. Zaragoza Rios. Representation and Exploitation of Knowledge for the Description Phase in Declarative Modeling of Virtual Environments. Master's thesis, Centro de Investigación y de Estudios Avanzados del I.P.N., Unidad Guadalajara, 2006.
[23] C. E. Chronaki. Parallelism in Declarative Languages. PhD thesis, Rochester Institute of Technology, 1990.
[24] J. K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. IEEE Computer Magazine, 31(3):23–30, March 1998.
[25] D. H. Frost. Algorithms and Heuristics for Constraint Satisfaction Problems. PhD thesis, University of California, 1997. Chair-Rina Dechter.
[26] S. W. Golomb and L. D. Baumert. Backtrack Programming. J. ACM, 12(4):516–524, 1965.
[27] S. J. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Pearson Education, 2003.
[28] J. G. Gaschnig. Performance Measurement and Analysis of Certain Search Algorithms. PhD thesis, Carnegie-Mellon Univ. Pittsburgh Pa. Dept. Of Computer Science, 1979.
[29] R. Dechter. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cut Set Decomposition. Artificial Intelligence, 41(3):273–312, 1990.
[30] R. M. Haralick and G. L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. Artificial Intelligence, 14(3):263–313, 1980.
[31] SUN Microsystems. The Java SE Development Kit (JDK). http://java.sun.com/javase/downloads/index.jsp, 2007. Last visited 02/01/2007.
[32] A. G. Aguirre. Nucleo GeDA-3D. Master's thesis, Centro de Investigación y de Estudios Avanzados del I.P.N., Unidad Guadalajara, 2007.
[33] A. V. Martínez González. Lenguaje para Animación de Criaturas Virtuales. Master's thesis, Centro de Investigación y de Estudios Avanzados del I.P.N., Unidad Guadalajara, 2005.
[34] X3D. The Virtual Reality Modeling Language - International Standard ISO/IEC. http://www.web3d.org/x3d/specifications/, 2007. Last visited 06/01/2007.

Avatars Animation using Reinforcement Learning in 3D Distributed Dynamic Virtual Environments

Héctor Rafael OROZCO[a], Félix RAMOS[a], Jaime ZARAGOZA[a] and Daniel THALMANN[b]

[a]*Centro de Investigación y de Estudios Avanzados del I.P.N.*
Unidad Guadalajara
Av. Científica 1145, Col. El Bajío, 45010 Zapopan, Jal., México
E-mail: {jzaragoz, framos, horozco}@gdl.cinvestav.mx
[b]*École Polytechnique Fédérale de Lausanne*
EPFL IC ISIM VRLAB Station 14 CH-1015 Lausanne, Switzerland
E-mail: daniel.thalmann@efpl.ch

Abstract. The animation problem of avatars or virtual creatures using learning, involves research areas such as Robotics, Artificial Intelligence, Computer Sciences, Virtual Reality, among others. In this work, we present a Machine Learning approach using Reinforcement Learning and a Knowledge Base for animating avatars. This Knowledge Base (ontology) provides the avatar with semantic definition and necessary awareness of its internal structure (skeleton), its behavior (personality, emotions and moods), its learned skills, and also of the rules that govern its environment. In order to animate and control the behavior of these virtual creatures in 3D Distributed Dynamic Virtual Environments, we use Knowledge-Based Conscious and Affective Personified Emotional Agents as a type of logical agents, within the GeDA-3D Agent Architecture. We focus on the definition of minimum conscience of the avatars. The conscience and cognitive processes of the avatars allow them to solve the animation and behavior problems in a more natural way. An avatar needs to have minimum conscience for computing the autonomous animation. In our approach, the avatar uses the Knowledge Base first as a part of its conscience, and second to implement a set of algorithms that constitute its cognitive knowledge.

Keywords. Reinforcement Learning, Knowledge Base, Conscience, Avatar, Conscious Agent, GeDA-3D.

Introduction

The human mind has been studied for many philosophers for a long time. The human consciousness is considered to be one of the most interesting topics in the philosophy.

This topic is called philosophy of mind. An important aspect of human consciousness is the self-knowledge or self-awareness, defined as the ability to perceive and reason about oneself. This aspect is highly developed in the Human being in comparison with other animals and it is considered very important for making agents with intelligent behavior. A Human being unaware of his or her personal characteristics, abilities and skills does not know what he or she can or cannot do, so he or she will have difficulties for interacting with others in a natural way. The *conscience* in general is defined as the knowledge that the Human being has of itself and of its environment.

Definition 1: The conscience of an avatar is the notion it has of its sensations, thoughts and feelings in a given moment within environment. That is, the avatar's conscience represents the understanding of its environment and its self-knowledge.

In this work, we use Knowledge-Based Conscious and Affective Personified Emotional (CAPE) Agents to develop the ability of avatar to perceive and reason about itself on the basis of the following: *Consciousness* (involve thoughts, sensations, perceptions, personality, moods and emotions), *stimuli and sensorial entrances* (relevant events), *introspection* (ability of avatar to reason about its perceptions and any conscious mental event), *awareness* (ability of avatar to be conscious; comprises perceptions and cognitive reactions to events, does not necessarily imply understanding), s*elf-consciousness* (awareness and understanding of avatar, it gives the avatar knowledge that it exists as a virtual entity separate from other avatars and virtual objects), and *qualia* (subjective properties of the perceptions and sensations of avatar).

In order to implement the conscience in an avatar, we use a Knowledge Base (KB). This KB represents the avatar conscience and it helps us to animate avatars or virtual creatures (VC). Thus, the KB (ontology) provides the semantic definition and necessary awareness of the internal structure of avatar (skeleton), its behavior (personality, emotions and moods), its learned skills, and also of the rules that govern its environment. We argue that consciousness is very important and plays a crucial role in creating intelligent agents with human abilities and skills. Thus, the avatar can be aware of how its skeleton is formed (considering its mobility and physical restrictions) and also of the rules that govern its environment.

The interest of this research is to give the avatars a basic conscience in order to have autonomous avatars able to act in 3D virtual environments. This means that it is not necessary to define previously the movements of the avatars to achieve a task in advance. Avatars must compute their movements themselves. But the generation of dynamic autonomous movements with high degree of realism is too complicated. Nevertheless, it is possible to make models of interactions between avatars and their environment in applications of computer animation and simulation [1]. For example, Virtual Humans (VH) can be used as virtual presenters, virtual guides, virtual actors or virtual teachers. Thus, the behavior and movements of VH can be controlled by using knowledge-based conscious emotional agents in order to show how humans behave in various situations [2]. There are different approaches to deal with the objective of this research. This research is a part of the GeDA-3D Agent Architecture [3, 4]. The following section is devoted to overview the related work.

1. Related Work

Interactive applications such as video games, collaborative virtual environments, simulations of virtual situations, films, among others, need believable virtual entities. But nowadays the behavior of these VC in current applications and systems is still very artificial and limited.

Definition 2: An avatar is a virtual entity with well-defined features and functionalities, able to live and interact in a 3D dynamic virtual environment.

Articulated models are often used for creating avatars. The animation of such models is often based on motion capture or procedurally generated motions. Despite the availability of such techniques, the manual design of postures and motions is still widespread; however, it is a laborious task because of the high number of degrees of freedom present in the models. Kinematic algorithms are also often used in the animation of avatars. Most of these algorithms require information about the position of joints, angles and limbs length. Next we will survey the most important related topics and give our opinion about them.

1.1. Motion Planning

Motion Planning (MP) has multiple applications. In Robotics it is used to endow robots of intelligence (autonomy), so they can plan theirs own movements. The problem of planning consists in finding a path for the robot from an initial point to a goal point without colliding with the obstacles in the environment [5]. In Artificial Intelligence (AI) the term *planning* takes a more interesting meaning. In this area the problems of planning are modeled with continuous spaces [6]. The problem of planning seems more natural and consists in defining a finite set of actions that can be applied to a discrete set of states and construct a solution by giving the appropriate sequence of actions. In [7] is presented a motion planner, which computes animations for virtual mannequins cooperating to move bulky objects in cluttered environments. In this work two kinds of mannequins were considered: human figures and mobile robot manipulators. Incremental Learning (IL) is a novel approach to the motion planning problem. It allows the virtual entities to learn incrementally on every planning query and effectively manage the learned roadmap as the process goes on [8]. This planner is based on previous work, on probabilistic roadmaps, and uses a data structure called Reconfigurable Random Forest (RRF), which extends the Rapidly Exploring Random Tree (RERT) structure proposed in the literature. The planner can take in account the environmental changes while keeping the size of the roadmap small. The planner removes invalid nodes from the roadmap as the obstacle configurations change. It also uses a tree-pruning algorithm to trim RRF into a more concise representation.

1.2. Motion Capture

In recent years the films have been successful exploding the technologies of motion capture. Motion capture is the process of capturing the live motion from a person or animal in order to animate an avatar [9]. Motion capture provides an impressive ability to replicate gestures, synthetic reproduction of large and complicated movements and

behavior analysis, among others. At the moment the motion capture systems allow the collection of information for illustrating, studying and analyzing the characteristics of body limbs and joints during various motions, such as walking, running, etc. However, though impressive in the ability to replicate movement, the motion capture process is far away from perfect. Despite the longer time required to visualize the captured motion, the optical motion capture is often preferred to magnetic technology. The avatars animation design generates libraries of postures and motion sequences using a motion capture system and later combined the obtained data with standard editing tools. In the real-time motion generation, the avatars motions are based on the combination of pre-recorded sequences or dynamic motion captures, avoiding the recording stage.

1.3. Machine Learning

Machine Learning (ML) is a subfield of AI. This subfield covers the design and development of computer algorithms and techniques that improve automatically through experience. These algorithms allow the machines and intelligent agents to learn.

1.3.1. Reinforcement Learning

RL algorithms [10] allow machines and intelligent agents to automatically maximize their performance and learn their behavior, based on feedback from their environment within a specific context.

Definition 3: Reinforcement Learning is a subarea of ML and AI interested and involved in the problem of how an autonomous agent must learn to take optimal actions to achieve its goals in its environment, so as to maximize the notion of reward in long-term.

RL algorithms attempt to find a *policy* that maps *states* of environment to *actions* the agent ought to take in those states. The environment is typically represented using the *Finite Markov Decision Process* (FMDP). Each time the agent performs an action in its environment, a trainer may provide a reward or punishment (penalty) to indicate the desirability of the resulting state. Therefore, the task of agent is to learn from the reward indirectly. So, the agent can choose sequences of actions that produce the greatest cumulative reward.

1.3.2. Uses of Reinforcement Learning

Applications of RL are abundant. In fact, a lot of problems in AI can be mapped to a FMDP. This represents an advantage, since the same methodology can be applied to many problems with little effort. RL has been used in Robotics to control mobile robots and optimize operations in production lines or manufacture systems. An approach to animating humanoids was proposed in [11]. However, this approach has many restrictions in the used models. In [12] two well-known RL algorithms are presented (Q-Learning and TD-Learning). These algorithms are used for exploration, learning and visiting a virtual environment. In [13], RL algorithms are applied for the generation

of Autonomous Intelligent Virtual Robots that can learn and enhance their task performance in assisting humans in housekeeping.

The potential for instructing animated agents through collaborative dialog in a simulated environment is described in [14]. In this work, STEVE, an embodied agent that teaches physical tasks to human students, shares activities with a human instructor by employing verbal and nonverbal communication. This way of work allows the agent to be taught in a natural way by the instructor. STEVE begins learning the task through a process of programming by demonstration. The human instructor tells STEVE how to observe his actions, and then it performs the task by manipulating objects in the simulated world. As the agent watches, it learns necessary information about the environment and procedural knowledge associated with the task.

A new way to simulate an Autonomous Agent's cognitive learning of a task for interactive virtual environment applications is proposed in [15]. This work is focused on the behavioral animation of virtual humans capable of acting independently. The concept of the Learning Unit Architecture that functions as a control unit of the Autonomous Virtual Agent's brain is proposed. The results are illustrated in a domain that requires effective coordination of behaviors, for example driving a car inside a virtual city. In [16], is presented an approach to integration of learning in agents for testing how it is possible to manage coherently a shared virtual environment populated with autonomous agents. Results proved that agents can automatically learn behavioral models to execute difficult tasks.

Learning Classifier Systems (LCS) are a ML paradigm introduced by John Holland in 1978. In LCS, an agent learns to perform a certain task by interacting with a partially unknown environment from which the agent receives feedback in the form of numerical reward. The incoming reward is exploited to guide the evolution of the agent's behavior which is represented by a set of rules, the classifiers. In particular, temporal difference learning is used to estimate the goodness of classifiers in terms of future reward; genetic algorithms are used to favor the reproduction and recombination of better classifiers [17]. In this work our approach is very different, because we use RL as a cognitive process that allows the avatar to learn new skills in its environment within a certain context. This difference is made by working with conscience that is not just knowledge but cognitive processes, which allow us to animate avatars in a more natural way. However, we are more interested in learning than in the exploitation of learning. That is, learning allows us to simulate the behavior and motion of life creatures into the avatars. This application constitutes a new use of RL. The following two sections are dedicated to the proposal of this research work. In these sections we will present the definition of an ontology proposed in order to define the internal skeletons of the avatars and the application of Reinforcement Learning (RL) for animating autonomously a nonhuman virtual arm.

2. Knowledge-Based CAPE Agents

Knowledge and reasoning are two essential elements for making intelligent agents able to achieve successful behaviors and take good actions in complex environments. These elements play a crucial role in dealing with partially observable environments. Knowledge-Based Agents are able to accept new tasks in form of goals. These agents

can adapt to changes in the environment by updating its relevant knowledge about the environment and themselves.

Definition 4: CAPE Agents are Knowledge-Based Agents able to combine general knowledge with current perceptions to infer hidden aspects of the current state prior to taking new actions in their environment. Thus, these agents can increase their knowledge and learn new skills.

CAPE Agents represent a kind of logical agents whose knowledge always is defined. That is to say, each proposition is either true or false in the environment, although the agent may be unbeliever towards some propositions. Even though logic is a good tool to model CAPE Agents in partially observable environments, a considerable part of the reasoning carried out by agents depends on handling the uncertainty. However, logic cannot represent knowledge that is uncertain in a good way.

The main component of a knowledge-based agent is its KB. We use the KB as a set of logical sentences. Each logical sentence represents some assertion about the environment or the avatar. In order to add new sentences and query what is known to the KB, we use the standard sentences *TELL* (telling information to the KB) and *ASK* (asking information to the KB), respectively. Both sentences are used to generate new sentences basing on the old sentences. When we ask something to the KB, the answer depends on what we have told to it previously. In this work we propose a KB (ontology) to store and get knowledge about the internal structure of avatar (skeleton), its behavior (personality, emotions and moods) and its learned skills. The main objective is the exploitation and use of knowledge offered by the ontology in order to make autonomous animations of avatars using RL. This ontology allows sharing semantic information of avatars among CAPE Agents that live and interact in a 3D dynamic virtual environments created by a declarative description over the GeDA-3D Agent Architecture. In fact, the avatar uses the KB first as a part of its conscience and second to implement a set of algorithms that constitute its cognitive knowledge.

Figure 1 shows the relationships between the main classes of the proposed ontology. An avatar is defined using a morphology description (qualitative description) that defines its skeleton (geometry of avatar) and the anthropometry description (quantitative description) that offers information about its age, gender, weight and height. In addition, as a part of its behavior, an avatar has personality, emotions and moods. In this work we will only explain how we have defined the internal skeleton of avatar.

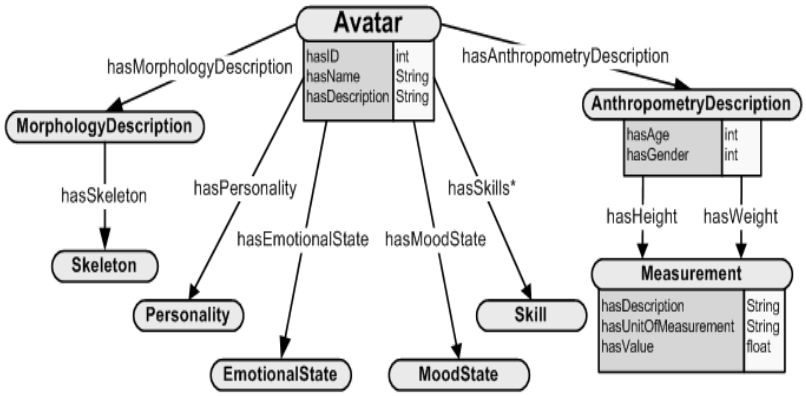

Figure 1. Semantic representation of avatar

The internal skeleton of avatar is composed by several parts. That is to say, bones and joints that form skeleton parts in specific (see figure 2). Each joint has a name and can have joints parents and/or joints children. There are motion constraints defined for each joint and a set of simple motions that define the alphabet of basic movements (micro-animation) that will be used to generate complex motions by means of combination between them (macro-animation). Also each bone can be united to one or more joints, and each joint has its position in the skeleton of avatar. Each bone of avatar has its measures that can be expressed in a predetermined unit of measurement, for example, in centimeters or decimeters. Using the previous definition and applying RL algorithms, we can animate the internal skeleton of avatar as it will be shown in the following section.

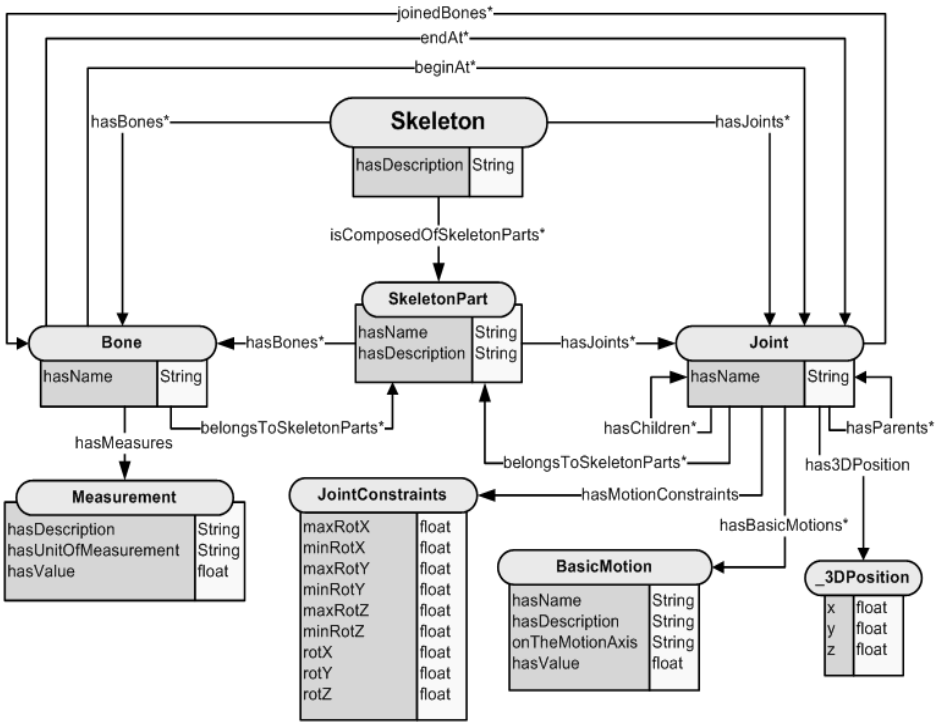

Figure 2. Skeleton definition of avatar

3. Avatars Animation using a Knowledge Base and Reinforcement Learning

In order to animate avatars it is necessary to use RL and MP algorithms to compute their motions. However in this work, we only present the use of RL. Avatars should be conscious of their internal structure (skeleton) and know how to combine simple movements to make complex activities or motions, which allow them to learn several skills and abilities. Using the previous definition and applying RL algorithms we can animate autonomous avatars. Figure 3 shows the proposed module of RL for the GeDA-3D Agent Architecture. The avatar should explore its body to know its structure and to learn a set of primitive motions, the basis for generating complex motions. In this work we propose the use of synergies (simple movements) to support the idea that the avatar cannot control all the degrees of freedom of its skeleton. For this reason a set of simple or primitive motions is selected (natural motions) to generate complex motions. Therefore, synergies are the base of the motions of avatar and can be manipulated by means of RL and MP algorithms.

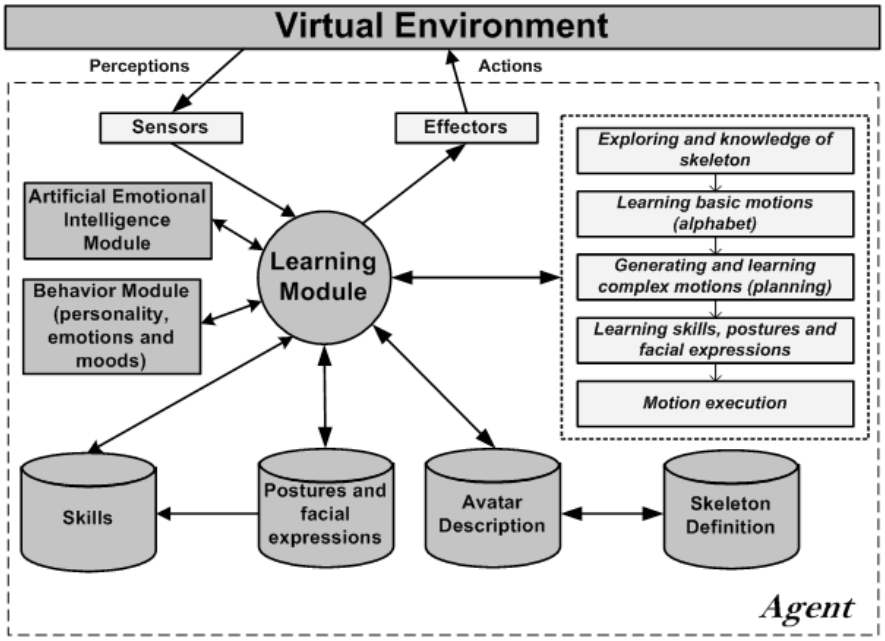

Figure 3. Reinforcement Learning Module of the GeDa-3D Agent Architecture

3.1. Skeleton Parts involved in the Learning Task

We have to identify the used skeleton parts of avatar that we aim to animate on the basis of the following factors:

- Identifying the implied bones and joints of each skeleton part and their relations.
- Establishing the end-effectors and their locations in the skeleton parts.
- Defining a set of basic motions to each joint taking into account their motion constraints.

3.2. Learning Task Definition

It is very important to define clearly the learning task the avatar must perform. That is to say, we have to define the state and action sets used in the RL algorithm.

3.2.1. State Set

We define the state set $S = \prod J_i, \forall_{i=1,2,\cdots,n}$, where:

- State set of each joint J_i indicates each possible angle that can adopt each joint in order to accomplish its motion constraints, and,
- n indicates the number of joints in the skeleton.

3.2.2. Action Set

We translate the set of basic motions of each joint into the action set of RL task. Basic motions are to increase or decrease the angle related to such movement into small values, for example 5 or 10 degrees. Therefore, the action set A indicates the possible motions of each joint of the skeleton of avatar (degrees of freedom) over the three axis x, y and z. Next, we show the functions applied to find the action set and the state set used in the RL algorithm as part of cognitive knowledge of avatar:

```
//Action set of skeleton sk
function ACTIONS_SET (Skeleton sk, increment) returns the action set
begin
    Set A[]; //action set
    Joints joints[];
    //KB is the Knowledge Base
    joints = ASK(KB, JOINTS_SKELETON(sk));
    from i = 0 to i < joints.length do
          A = A + AXIS_ACTIONS(joints[i], x, increment)
                + AXIS_ACTIONS(joints[i], y, increment)
                + AXIS_ACTIONS(joints[i], z, increment);
    return A;
end

//Action set of joint j on the axis a
function AXIS_ACTIONS (Joint j, Axis a, increment) returns the action set
begin
    Set A[]; //action set
    vars min, max;
    //KB is the Knowledge Base
    min = ASK(KB, MINIMUM_ROT(j, a));
    max = ASK(KB, MAXIMUM_ROT(j, a));
    if max - min > 0 then
    begin
          A = new Set[2];
          A[1] = rotating the joint j in positive degrees on the axis a; //Increasing (in increment)
          A[2] = rotating the joint j in negative degrees on the axis a; //Decreasing (in increment)
    end
    return A;
end

//State set of skeleton sk
function ACTIONS_SET (Skeleton sk, increment) returns the state set
begin
    Set S[]; //state set
    Joints joints[];
    //KB is the Knowledge Base
    joints = ASK(KB, JOINTS_SKELETON(sk));
    from i = 0 to i < joints.length do
          S = S x JOINT_STATES(joints[i], increment);
    return S;
end

//State set of joint j
function JOINT_STATES (Joint j, increment) returns the state set
```

```
begin
   Set S[]; //state set
   S = S x AXIS_STATES (j, x, increment) x AXIS_STATES (j, y, increment)
         x AXIS_STATES (j, z, increment);
return S;
end

//State set of joint j on the axis a
function AXIS_STATES (Joint j, Axis a, increment) returns the state set
begin
   Set S[]; //state set
   vars max, min, aux, i;
   //KB is the Knowledge Base
   min = ASK(KB, MINIMUM_ROT(j, a));
   max = ASK(KB, MAXIMUM_ROT(j, a));
   aux = max - min / increment;
   if aux > 0 then
   begin
        S = new Set[aux];
        from i = 0 to aux do
          S[i] = min + i * increment degrees on the axis a;
        end
   return S;
end
```

3.3. Finite Markov Decision Process

In RL, an agent chooses the best action based on its current state. When this step is repeated many times, it turns into the problem that is known as the Markov Decision Process. Therefore, we considerer the representation of learning task to be the *Finite Markov Decision Process* (FMDP) based on the following statements:

- $P(s,a,s'): S \times A \times S \longrightarrow [0,1]$ is the joint probability of making a transition to state s if action a is taken in the state s.
- $R(s,a,s'): S \times A \times S \longrightarrow R$ is an immediate reward for making a transition from s to s' by action a.

Given any state and action, s and a, the probability of each possible following state s' is:

$$P_{ss'}^{a} = P\{s_{t+1} = s' | s_t = s, a_t = a\}$$

Similarity, given any current state and action, s and a, together with any following state s', the expected value E of the following reward is:

$$R_{ss'}^{a} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$$

Basically the agent perceives a set of states S in its environment, a finite set of actions A that it can perform, and a set of obtained rewards in $\Re$. At each discrete time step t, the agent senses the current state $s_t \in S$ and the set of possible actions $A(s_t)$ for that state. When the agent takes an action $a \in A(s_t)$ and executes it, it receives the new state s_{t+1} and a reward or punishment r_{t+1} from the environment. Thus, the agent must learn to develop a policy $\pi : S \rightarrow A$, which maximizes the quantity $R = \sum_{t=0}^{n} r_t$ for a FMDP with a terminal state, or the quantity $R = \sum_{t=0}^{n} \gamma^t r_t$ for a FMDP without terminal states. Where $0 \leq \gamma \leq 1$ represents a discount factor used for future rewards. In fact, the agent does not necessarily know the reinforcement and next-state functions. These functions depend only on the current state and action. Before learning, the agent may not know what will happen when it chooses and executes a action in a particular state. The agent is only aware of its current state. This represents relevant information for the agent and allows it to decide which action to choose and execute. However, the policy is essential, because the agent uses its knowledge to choose an action in a given state.

3.4. Q-Learning Algorithm

There are several ways to implement the RL task. We have chosen the *Q-learning* algorithm, a well-known form of RL in which the agent learns to assign values to state-action pairs. In its simplest form, the *one-step Q-learning* algorithm is defined by the following *action-value function*:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

We have fixed the values of α and γ to 0.5. The value γ is a discount factor used for future rewards. Thus, the agents can learn through experience. We have called a *state* each possible angle that can adopt each joint of the skeleton of avatar. Possible movements of each joint of the avatar's skeleton is called *action*. We can represent the above concepts using a state diagram. In this diagram a state is depicted by a node, while an action is represented by an arrow. We can put the state diagram and the instant reward values into a reward table or matrix R (reward function). The learning algorithm Q-learning is a simplification of RL. We need to put into the brain of agent a similar matrix named Q that will represent the memory of what the agent has learned through many experiences. The row of matrix Q represents current state of agent, the column of matrix Q pointing to the action indicates the following state. At the beginning, we have supposed the agent knows nothing, thus we put Q as a zero matrix. For simplicity in this work, we assume that the number of states is known. But in a better implementation, it is more convenient to start with a zero matrix of single cell and to add more columns and rows to the Q matrix if a new state is found. In general the transition rule of this Q-learning is as follows:

$$Q(state, action) \leftarrow R(state, action) + \gamma \max[Q(next\_state, actions)]$$

In the expression above the entry value in matrix Q (rows are states and columns are actions) is equal to corresponding entry of matrix R added by the multiplication of a learning parameter γ and maximum value of Q for all actions in the following state. Therefore, the agent will explore state after state until it reaches the goal. Each exploration is an *episode*. In one episode the agent will move from the initial state to the goal state. Once the agent has arrived at the goal state, the algorithm passes to the next episode. Each episode is equivalent to one training session. In each training session the agent explores the environment (represented by matrix R), gets the reward (or none) until it has reached the goal state. The purpose of the training is to enhance the brain of agent (represented by the matrix Q). More training will give better matrix Q that can be used by the agent to move in the most optimal way. Parameter γ has range of values form **0** to **1** ($0 \quad \gamma \quad 1$). If γ is closer to zero, the agent tends to consider only immediate reward. If γ is closer to one, the agent will consider that the future reward has greater weight and importance. That it to say, the agent will be willing to delay the reward. In order to use the matrix Q, the agent traces the sequence of states, from the initial state to the goal state.

3.5. Action Selection Rule

An important constraint of RL is the fact that only Q-values for actions that are tried in current states are updated. The agent learns nothing about actions that it does not try. The agent should try a range of actions in order to have an idea about what action is a good decision and what is not. That is to say, at any given moment of time, the agent must only choose an option: it can execute the action with the highest Q-value for the current state (*exploitation*), or it can execute an action randomly (*exploration*). Exploitation is based on what the agent knows about its environment, this probably can give more benefits to the agent. On the other hand, exploration offers the agent the possibility to learn actions that would not be tried otherwise. In order to deal with the exploration vs. exploitation dilemma, we choose an $\varepsilon - greedy$ method. First, we initialize ε to **1.0** and at the beginning of each episode we decrease it. Later, we update the value ε in a way inversely proportional to the number of elapsed episodes in the execution of the learning algorithm in the following way:

$$\varepsilon = 1.0 - \left(\frac{elapsed\_episodes}{total\_episodes} \right)$$

Therefore, the CAPE Agents learn using the following logical principles:

- If an action in a given state causes a bad decision, the agent learns not to execute that action in that situation.
- If an action in a given state causes a good decision, the agent learns to take that action in that situation.

- If all actions in a given state cause a bad decision, the agent learns to avoid that state. That is, the agent does not take actions in other states that would lead it to be in similar states.
- If any action in a given state causes a good decision, the agent learns to prefer that kind of states.

3.6. Reward Function

The *reward function* R is one of the most important components of RL because it defines the purpose or goal of agent in the learning task. We have to define or find a reward function that closely represents the agent's goals in the learning task. Our method consists in minimizing the *Euclidean distance* between $C_{x,y,z}$ (end-effector current position) and $G_{x,y,z}$ (end-effector goal position). That is to say, to minimize the total reward received in the long run. However, there exists a problem: the goal of RL is to maximize the total reward received in the long run, which is exactly the opposite of the RL goal. In order to fix this problem, we simply tag the Euclidean distance as a negative reward. Therefore, we define the reward function $R : S \times A \times S \rightarrow \Re^{-}$ as:

$$R(s,a,s') = -\sqrt{(c_x - g_x)^2 - (c_y - g_y)^2 - (c_z - g_z)^2}$$

Although we have defined correctly the reward function, $C_{x,y,z}$ is unknown and needs to be calculated from s'. That is to say, at each time step, we need to calculate the end-effector's current position $C_{x,y,z}$ given the agent's current state s'. This problem is known as the *Forward Kinematics Problem*. In order to solve this problem, we have used the **Denavit-Hartenberg convention** (D-H convention) [18] to select the frames attached to each joint of the skeleton of avatar in a systematic way. That is, we establish the joint coordinate frames using the D-H convention.

3.7. Results

Using the previous definition we animated a nonhuman virtual arm composed of two bones: *Humerus* and *Forearm* that are connected by two joints: *Shoulder* and *Elbow*. Figure 4 shows the first case study. In this case study we considered two degrees of freedom, one of them assigned to the shoulder and the other assigned to the elbow. Available movements for both shoulder and elbow are pitch (up or down). Possible actions that can be performed are to increase or decrease the angle of each joint (shoulder and/or elbow) by 10 degrees. We represent the state S as a 2-tuple (shoulderPitch, elbowPitch), where:

- shoulderPitch, elbowPitch $\in \{0°, 10°, 20°, \cdots, 180°\}$.

Therefore the state set is:

$$S = \{0°,10°,20°,\cdots,180°\} \times \{0°,10°,20°,\cdots,180°\}$$
$$|S| = 19 \times 19 = 361$$

Angles of each joint are bounded in the rank from 0 to 180 degrees, this rank accomplishes with the motion constraints defined in the ontology (see figure 5). The action set represent all the available movements of the arm considered in this learning task. These motions are pitch up or down the shoulder ten degrees and pitch up or down the elbow ten degrees. Therefore, the action set is:

$$A = \begin{Bmatrix} shoulderPitchUP, shoulderPitchDown, \\ elbowPitchUp, elbowPitchDown \end{Bmatrix}$$
$$|A| = 4$$

In spite of the fact that the animation of this arm was made in a 3D environment, the movements it performs is in a 2D plane, due to the number of degrees of freedom the arm has.

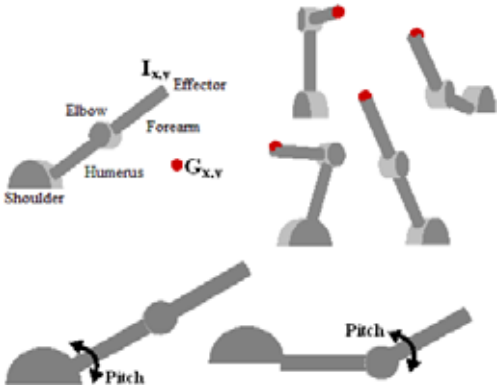

Figure 4. Reinforcement Learning in a nonhuman virtual arm with two degrees of freedom. The arm adopts a final configuration to collocate the end-effector in the goal position

```
<skeleton>
 <joint name="Shoulder">
  <children>
   <joint name="Elbow">....</joint>
  </children>
  <bones>
   <bone name="Humerus">....</bone>
  </bones>
  <rotX minimum="0" maximum="0" angle="0"></rotX>
  <rotY minimum="0" maximum="0" angle="0"></rotY>
  <rotZ minimum="0" maximum="180" angle="0"></rotZ>
  <position x="0" y="0" z="0"></position>
 </joint>
 <joint name="Elbow">
  <parents>
   <joint name="Shoulder">....</joint>
  </parents>
  <bones>
   <bone name="Humerus">....</bone>
   <bone name="Forearm">....</bone>
  </bones>
  <rotX minimum="0" maximum="0" angle="0"></rotX>
  <rotY minimum="0" maximum="0" angle="0"></rotY>
  <rotZ minimum="0" maximum="180" angle="0"></rotZ>
  <position x="30" y="0" z="0"></position>
 </joint>
 <joint name="Effector">
  <parents>
   <joint name="Elbow">....</joint>
  </parents>
  <bones>
   <bone name="Forearm">....</bone>
  </bones>
  <rotX minimum="0" maximum="0" angle="0"></rotX>
  <rotY minimum="0" maximum="0" angle="0"></rotY>
  <rotZ minimum="0" maximum="0" angle="0"></rotZ>
  <position x="80" y="0" z="0"></position>
 </joint>
 <bone name="Humerus" length="30">
  <begin>
   <joint name="Shoulder">....</joint>
  </begin>
  <end>
   <joint name="Elbow">....</joint>
  </end>
 </bone>
 <bone name="Forearm" length="50">
  <begin>
   <joint name="Elbow">....</joint>
  </begin>
  <end>
   <joint name="Effector">....</joint>
  </end>
 </bone>
</skeleton>
```

Figure 5. Definition of an nonhuman virtual arm with two degrees of freedom based on XML using the proposed ontology

Figure 6 shows the last case study. In this case study we animated other nonhuman virtual arm with four degrees of freedom (left arm). Three of them assigned to the shoulder and the last one assigned to the elbow. Available movements for this arm are: for the shoulder: roll (right, left), yaw (right, left) and pitch (up, down), and for the elbow: pitch (up, down). Possible actions that can be performed are to increase or decrease the angle of each joint (shoulder and/or elbow) in 10 degrees. In this case study, we have represented the state set S as a 4-tuple (shoulderRoll, shoulderYaw, shoulderPitch, elbowPitch), where:

- shoulderRoll $\in\{0°,10°,20°,\cdots,90°\}$,
- shoulderYaw $\in\{0°,10°,20°,\cdots,180°\}$,
- shoulderPitch $\in\{0°,10°,20°,\cdots,240°\}$, and,
- elbowPitch $\in\{0°,10°,20°,\cdots,150°\}$.

Therefore, the state set is:

$$S=\{0°,\cdots,90°\}\times\{0°,\cdots,180°\}\times\{0°,\cdots,240°\}\times\{0°,\cdots,150°\}$$
$$|S|=10\times19\times25\times16=76000$$

Angles of each joint accomplishes with the motion constraints defined in the ontology (see figure 7). Possible motions are: roll the shoulder right or left by ten degrees, yaw the shoulder right or left by ten degrees, and pitch the shoulder and elbow up or down by ten degrees. Therefore, the action set is as follows:

$$A = \begin{Bmatrix} shoulderRollRight, shoulderRollLeft, shoulderYawRight, \\ shoulderYawLeft, shoulderPitchUp, shoulderPitchDown, \\ elbowPitchUp, elbowPitchDown \end{Bmatrix}$$

$$|A| = 8$$

In this case study the movements are performed in 3D. These movements are very similar to the movements performed by the left arm of a real Human being.

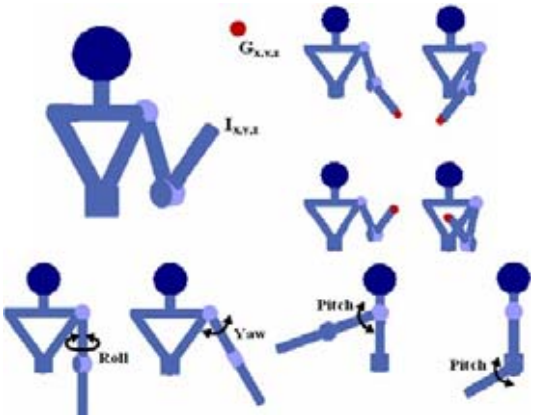

Figure 6. Reinforcement Learning in a nonhuman virtual left arm with four degrees of freedom. The arm adopts the final configuration to collocate the end-effector in the goal position

```
<skeleton>
 <joint name="Shoulder">
  <children>
   <joint name="Elbow">....</joint>
  </children>
  <bones>
   <bone name="Humerus">....</bone>
  </bones>
  <rotX minimum="0" maximum="90" angle="0"></rotX>
  <rotY minimum="0" maximum="180" angle="0"></rotY>
  <rotZ minimum="0" maximum="240" angle="0"></rotZ>
  <position x="0" y="0" z="0"></position>
 </joint>
 <joint name="Elbow">
  <parents>
   <joint name="Shoulder">....</joint>
  </parents>
  <bones>
   <bone name="Humerus">....</bone>
   <bone name="Forearm">....</bone>
  </bones>
  <rotX minimum="0" maximum="0" angle="0"></rotX>
  <rotY minimum="0" maximum="0" angle="0"></rotY>
  <rotZ minimum="0" maximum="150" angle="0"></rotZ>
  <position x="30" y="0" z="0"></position>
 </joint>
 <joint name="Effector">
  <parents>
   <joint name="Elbow">....</joint>
  </parents>
  <bones>
   <bone name="Forearm">....</bone>
  </bones>
  <rotX minimum="0" maximum="0" angle="0"></rotX>
  <rotY minimum="0" maximum="0" angle="0"></rotY>
  <rotZ minimum="0" maximum="0" angle="0"></rotZ>
  <position x="80" y="0" z="0"></position>
 </joint>
 <bone name="Humerus" length="30">
  <begin>
   <joint name="Shoulder">....</joint>
  </begin>
  <end>
   <joint name="Elbow">....</joint>
  </end>
 </bone>
 <bone name="Forearm" length="50">
  <begin>
   <joint name="Elbow">....</joint>
  </begin>
  <end>
   <joint name="Effector">....</joint>
  </end>
 </bone>
</skeleton>
```

Figure 7. Definition of a nonhuman virtual arm with four degrees of freedom based on XML using the proposed ontology

4. Conclusions

Nowadays, one of the great challenges in VR is to create avatars with characteristics proper to real living creatures. That is, it is desirable that these VC are able to reason, learn, feel and react as if they were intelligent creatures with cognitive capability to make decisions. The design of these VC has been a motivating task for researchers of different areas, such as Robotics, Virtual Reality (VR), AI and Computer Sciences (CS). The advance in such research areas is impressive, but there is still much work to be done. In the video game industry the users demand every day more sophisticated video games in which they can enhance their presence in the virtual environment, navigate, perceive elements and interact with the VC. In the film industry is growing the interest to characterize actors in animated movies in a more realistic way. In order to attain this, we have proposed in this work a novel approach to the animation of avatars using RL. Our approach is based on the use of ML algorithms to provide the virtual creature with the capability of learning new skills. Although the presented methodology has proved to work well, its success depends enormously on how well we define the reward function. RL allows the CAPE Agents to learn their behavior on basis of feedback from the environment. This behavior can be learned only once, or adapting in the time. If the problem is modeled in a suitable way, RL algorithm can converge to the global optimum. This represents the ideal behavior to maximize the reward.

Intelligent agents can use the knowledge about their environment and themselves offered by the KB to make new inferences and to take good actions. This knowledge is represented by logical sentences and it is stored in the KB. In this work we have used knowledge-based agents composed of a KB and an inference mechanism. These agents store logical sentences about the world and themselves in the KB, using the inference mechanism to infer new sentences (new knowledge). Agents use these sentences in order to decide which actions to take in a given moment. In this work, we have presented the necessary bases to implement the conscience of an avatar, shown how to define the internal skeletons of avatars, and described some of the main issues to be solved. The current version of our ontology is a work in progress. In this work we have shown some of the main possibilities of use of the ontology in order to animate articulated VC. Our research is focused on advancing the development of the ontology proposed. Our ontology plays a fundamental role in the animation of articulated virtual creatures controlled by conscious and intelligent agents. Knowledge Bases also have very important potential in the motion planning and motion learning in avatars. Our future work includes the generalization of the avatar animation with the use of Knowledge Bases, MP and RL techniques. These results will be applied to the development of semi-autonomous and autonomous avatars that interact in 3D distributed dynamic virtual environments over the GeDA-3D Agent Architecture.

References

[1] H. Schmidl and M. Lin. Geometry-Driven Physical Interaction Between Avatars and Virtual Environments. Computer Animation and Virtual Worlds, Vol. 15, non. 3-4, pages 229-236, 2004.

[2] S. Göbel, A. Feix, and A. Rettig, Virtual Human: Storytelling and Computer Graphics for a Virtual Human Platform. International Conference on Cyber Worlds, 2004.

[3] F. Zúñiga, F. F. Ramos and I. Piza. GeDA-3D Agent Architecture. Proceedings of the 11th International Conference on Parallel and Distributed Systems, pages 201–205, Fukuoka, Japan, 2005.

[4] H. I. Piza, F. Zúñiga and F. F. Ramos. A Platform to Design and Run Dynamic Virtual Environments. Proceedings of the 2004 International Conference on Cyberworlds, pp. 78-85, 2004.
[5] F. Schwarzer, M. Saha and J. Latombe. Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments. IEEE Transactions On Robotics, Vol. 21, no. 3, pages 338-353, June 2005.
[6] S. M. LaValle. Planning Algorithms, Cambridge University Press, 2006.
[7] C. Esteves, G. Arechavaleta and J. P. Motion Planning for Human-Robot Interaction in Manipulation Task. IEEE International Conference on Mechatronics and Automation, Vol. 4, pages 1766- 1771 Laumond, 2005.
[8] T.-Y. Li and Y.-C. Shie. An Incremental Learning Approach to Motion Planning with Roadmap Management. In Proceedings of the 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, pages 3411–3416, 2002.
[9] L. Herda, P. Fua and D. Thalmann. Skeleton-Based Motion Capture for Robust Reconstruction of Human Motion. Computer Animation, pages 77-83, Philadelphia, PA, USA, 2000.
[10] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.
[11] J. Peters, S. Vijayakumar, and S. Schaal, Reinforcement Learning for Humanoid Robotics. International Conference on Humanoid Robots, pages 1-20, Karlsruhe, Germany, September 2003.
[12] T. CondeW. Tambellini and D. Thalmann, Behavioral Animation of Autonomous Virtual Agents Helped by Reinforcement Learning. Lecture Notes in Computer Science, vol. 272, pages 175-180, Springer-Verlag: Berlin, 2003.
[13] T.-Y. Li and Y.-C. Shie. An Incremental Learning Approach to Motion Planning with Roadmap Management. In Proceedings of the 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, pages 3411–3416, 2002.
[14] A. Scholer, R. Angros Jr., J. Rickel, and W. L. Johnson. Teaching Animated Agents in Virtual Worlds. In Smart Graphics: Papers from 2000 AAAI Spring Symposium, pages 46–52, 2000.
[15] T. Conde and D. Thalmann. Autonomous Virtual Agents Learning a Cognitive Model and Evolving. In Proceedings of the 5th International Working Conference on Intelligent Virtual Agents, IVA 2005, pages 88–98, 2005.
[16] T. Conde and D. Thalmann. Learnable Behavioural Model for Autonomous Virtual Agents: Low-Level Learning. In Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2006, pages 89–96, 2006.
[17] J. H. Holland, L. B. Booker, M. Colombetti, M. Dorigo, D. E. Goldberg, S. Forrest, R. L. Riolo, R. E. Smith, P. L. Lanzi, W. Stolzmann, and S. W. Wilson. What is a Learning Classifier System? In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, Learning Classifier Systems. From Foundations to Applications, volume 1813 of LNAI, pages 3–32, Springer-Verlag: Berlin, 2000.
[18] M. W. Spong and M. Vidyasagar. Robot Dynamics and Control. John Wiley & Sons, Inc., 1989.

Étape d'indexation d'un texte théâtral dans le cadre du projet DRAMA

Véronique Gaildrat†, Matthieu Pouget†, Tahiry Andriamarozakaniaina‡, Jaime Zaragoza*, Rabiafaranjato Velonoromanalintantely‡

† Université de Toulouse ‡ Université de Madagascar * Université de Guadalajara

Résumé : Nos travaux de recherche s'inscrivent dans le domaine de la modélisation déclarative d'environnements virtuels en trois dimensions, appliquée à la génération automatique de mises en scènes théâtrales virtuelles. Ces travaux consistent, dans une premier temps, à étudier un texte théâtral d'un point du vue structurel, ce qui permet d'extraire et de classifier ses éléments essentiels, reconnaissables morphologiquement. On peut ainsi repérer les différents éléments textuels qui structurent la dramaturgie textuelle afin d'effectuer une indexation qui pourra ensuite être utilisée pour des interrogations simples ou croisées. Cette indexation s'appuie sur une base de connaissances qui s'enrichit au fur et à mesure de la construction d'un corpus de textes indexés et qui sera utilisée pour générer des mises en scènes virtuelles lors de l'étape suivante de génération.

Mots clés : Indexation, langage de balises, XML, modélisation déclarative, propriétés.

1. Introduction

L'étude que nous présentons dans cet article a été réalisée dans le cadre du projet pluridisciplinaire DRAMA. Notre objectif principal au sein de ce projet est d'étudier l'apport de techniques informatiques pour la mise en scène de pièces de théâtre.
Il se décline en plusieurs supports d'accompagnement à la création théâtrale, parmi lesquels nous pouvons citer :

- DRAMAtexte, outil d'indexation de texte théâtral, qui permet de mettre en évidence les différents indicateurs de scénographie introduits par l'auteur ou le metteur en scène, puis de les visualiser sous une forme adaptée : listes, graphiques, schémas, visualisation 2D ou 3D (réalité virtuelle), afin de servir de cadre à l'élaboration de spectacles.
- DRAMAscène, conçu comme un outil de visualisation de mises en scène, devant permettre à tous les agents du spectacle et au metteur en scène de travailler en réseau, rendant possible une vision globale du travail scénique. Cet outil devra permettre de répertorier et de prendre en compte différents aspects de la création théâtrale (outils spécifiques de notation des mouvements, du dialogue, des didascalies de l'auteur ou du metteur en scène, de la scénographie, des éclairages, de la musique, de la bande sonore, etc.).
- De plus, le projet DRAMA comprendra à terme un volet mémorisation destiné à conserver et à exporter les données scéniques, obtenues grâce à DRAMAtexte et DRAMAscène, pour l'assistance et l'apprentissage, notamment dans le cadre de l'enseignement de la scénographie.

Il existe des carnets de mise en scène en version textuelle (papier ou numérique), des logiciels de scénographie, d'éclairage, ou encore de costumes, mais non spécifiques au théâtre et non adaptés aux besoins. Aucune plateforme n'est actuellement capable de rassembler les différents outils et de structurer la création théâtrale en fonction des différentes composantes du spectacle vivant.

Nous présentons dans cet article la partie de nos travaux, concernant l'indexation de textes de théâtre, développés pour DRAMAtexte, dont le but est d'étudier la reconnaissance logicielle d'œuvres drama-

tiques, permettant de scénariser l'écrit, autrement dit d'extraire toutes les informations nécessaires à la construction de la scène théâtrale. Ainsi, nous avons mis en place un nouveau concept de balisage, permettant de repérer facilement les entités importantes à la création de la scène, ceci afin de pouvoir lancer des requêtes simples ou croisées permettant d'interroger le texte.
Nous abordons également brièvement les principes de génération d'une mise en scène virtuelle, à partir des données issues de la phase d'indexation et d'une base de données stockant les connaissances sur le contexte de la pièce, étant donné que cette étape du projet va faire l'objet des prochains développements.

2. Problématique

Le problème posé par le projet DRAMA, vu sous l'angle de l'étude et du développement d'un outil informatique dédié à la mise en scène virtuelle, est apparenté à un problème de modélisation déclarative d'environnements virtuels complexes.

2.1. Modélisation déclarative

Un modeleur déclaratif est un outil permettant la génération automatique de scènes ou de formes complexes à partir d'une description, énoncée par un concepteur, constituée par un ensemble de propriétés.

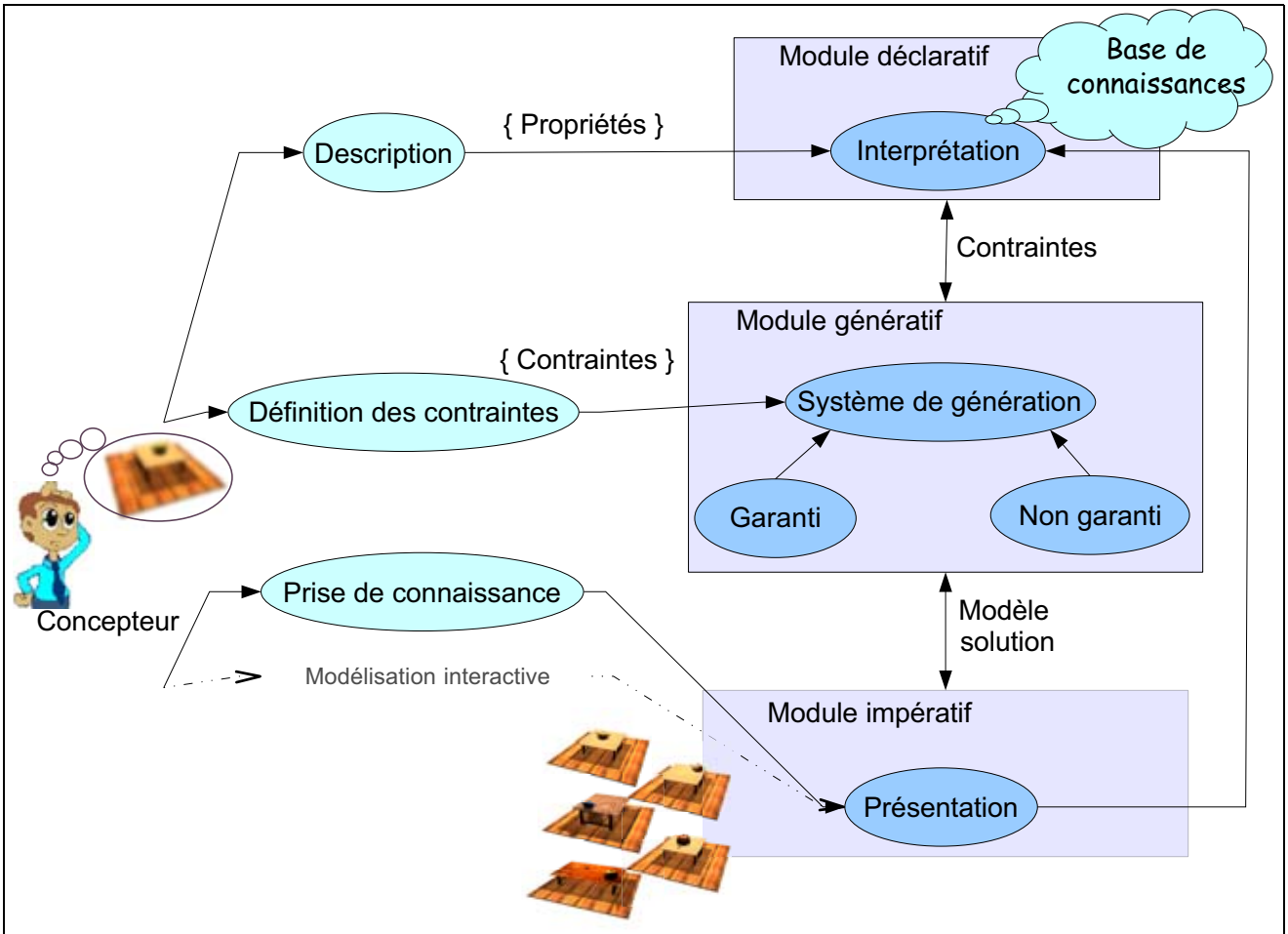

Fig 2.1.1: Les 3 étapes d'un modeleur déclaratif (Description, Génération, et Prise de connaissance) [1]

Cette description, énoncée dans un format accessible à l'utilisateur (utilisation d'un format structuré, du langage (quasi) naturel ou de la sélection au travers d'une interface graphique), est ensuite interprétée pour être traduite en contraintes géométriques, photométriques ou autres.

Fig 2.1.2.: Agetim : Génération de bâtiments à partir de Templates *[2]*

Fig 2.1.3.: CityEngine : Génération d'environnement urbain à partir de données géographiques [3]

Fig 2.1.4.: DEM2ONS_GA : Aménagement d'intérieurs [4]

Fig 2.1.5.: WordsEye : Système « Text to scene » [5]

Ces contraintes sont résolues par un système de génération adapté qui fournit un modèle solution. Ce modèle permet l'exploration de l'espace de solutions qui est généralement de grande taille, afin d'obtenir une ou plusieurs solutions répondant à la description fournie par l'utilisateur, par instanciation du modèle. Ces solutions sont présentées au concepteur afin qu'il puisse valider ses choix, ou bien, s'il n'est pas satisfait des résultats obtenus, il peut modifier la description et relancer le processus d'interprétation et de génération pour obtenir un autre ensemble de solutions. Ce processus de création déclaratif est présenté Fig 2.1.1.

Des projets, basés sur le concept de déclaratif, se développent fortement depuis le début des années 2000, autour de problèmes liés à la génération d'architectures (cf. Fig 2.1.2) d'environnements urbains (Fig 2.1.3), d'aménagement intérieur (cf. Fig 2.1.4), ou extérieur (cf. Fig 2.1.5).

3. DRAMAtexte : indexation et interrogation

L'objectif de cette première étape est de fournir à l'utilisateur, lecteur ou metteur en scène, la possibilité de mettre en évidence les éléments caractéristiques permettant de générer une mise en scène virtuelle. Ces éléments peuvent être issus des indications fournies par l'auteur grâce aux didascalies, ou ajoutés par le metteur en scène pour apporter sa vision et ses choix relatifs à la mise en scène.
L'indexation est effectuée en ajoutant dans le texte initial des balises qui correspondent à des mots clés, associées ou non à des attributs valués.

L'ajout de balises dans un texte permet de mettre en évidence la structure de ce texte et de décrire toutes les informations pertinentes qu'il contient, relativement à l'application qui exploite ces informations. La description de la mise en scène peut donc être obtenue au travers des informations annotées dans le texte. Cette description peut ensuite être interrogée et manipulée par l'intermédiaire de l'outil d'indexation DRAMAtexte.

3.1. Etat de l'art

Les premières formes de balisage ont été introduites manuellement dans les textes par les typographes, par ajout de symboles servant à structurer visuellement le texte. La ponctuation et les espaces utilisés pour délimiter les chapitres, les paragraphes et les phrases correspondent à cette forme initiale de balisage. Appliqué au texte sur support électronique, le balisage permet d'ajouter beaucoup plus d'informations, en rendant explicite pour un outil informatique ce qui est implicite pour le lecteur. Il permet ainsi d'identifier la structure logique d'un texte.
La représentation des informations contenues dans le texte d'un auteur nécessite l'étude d'un langage de description, permettant la représentation informatique des données théâtrales. Ce langage est basé sur le langage XML (*eXtensible Markup Language*) [6], adapté aux besoins de ce contexte particulier. Ce langage permet en effet de baliser le texte et extraire facilement les données, tâche qui ne peut être effectuée que manuellement et laborieusement à partir d'une version imprimée de la pièce de théâtre ou, au mieux, à partir d'une version numérique et d'un traitement de texte.
Cependant, le marquage et l'extraction d'informations peuvent être rendus difficiles en raison de la nature même des textes. Un texte peut contenir plusieurs passages rédigés dans différentes langues, des renvois, des notes, des citations, des descriptions scéniques voire des tableaux. Selon le genre littéraire, la structure du texte est organisée en chapitres, sections, scènes, actes, versets, voire un format particulier et non réellement structuré. Tous ces éléments, souvent implicites, peuvent être nécessaires à certains types d'utilisateurs. Le format d'encodage du texte doit donc tenir compte, selon les besoins particuliers de l'utilisateur, de la structure logique du document.
Pourtant, l'utilisation actuelle des textes, fussent-ils sous un format numérique, est limitée à la lecture ou à la simple recherche d'occurrences de mots ou de chaînes de caractères. Aussi, pour accéder à un texte de façon radicalement dynamique et pour fournir des informations nettement plus significatives, le balisage va se révéler une solution particulièrement intéressante.
A partir de ce constat, différents systèmes de balisage ont été conçus afin de répondre à des besoins particuliers, comme le rapporte Susan-Hockey [7]. On peut citer le système *COCOA* qui a été créé, au début des années soixante à Édimbourg, pour des textes anciens écossais et qui permet d'identifier la structure d'un texte. Il est utilisé par la plupart des outils d'analyse de texte, comme le Oxford Concordance Program. Le *Thesaurus Linguae Graecae* [8] a aussi créé son propre système de balisage nommé *Beta code*. Par la suite, de nouveaux systèmes de balisage performants sont apparus. Notamment TEX (*Tau Epsilon Xi*), un système puissant (dans le domaine public) d'écriture de formules mathématiques qui est très utilisé dans le monde scientifique. Les systèmes de balisage spécifiques (*procedural markup*) utilisés par les traitements de textes comme *MSWord* ou *OpenOffice* sont maintenant également très répandus.
L'utilisation de ces formats, développés indépendamment les uns des autres, a conduit à une véritable anarchie [9][10]. Pour autant, dès qu'il a pu être mis en place, le balisage est déjà apparu comme étant essentiel et nécessaire pour obtenir une analyse des textes quasi exhaustive et surtout bien structurée. Cependant, comme chacun de ces langages a été défini pour répondre à des besoins et des applications spécifiques, cela contribue au manque de flexibilité et d'adaptabilité. De plus, il n'y a pas beaucoup de documentation disponible et les possibilités d'adaptation à de nouvelles situations sont relativement difficiles et parfois impossibles : beaucoup de temps et d'efforts sont nécessaires pour convertir les textes d'un format à un autre. Par conséquent, aucun de ces formats ne peut être adopté comme norme standard, ni même être facilement transposé pour le contexte de notre application.

3.2. Les quatre étapes de DRAMA

Pour obtenir la visualisation en trois dimensions d'une proposition de mise en scène virtuelle à partir du texte théâtral, quatre étapes sont nécessaires (cf. Fig 3.2.1). Ces étapes constituent le noyau des deux premiers outils développés dans le cadre de ce projet : DRAMAtexte et DRAMAscène.
La première étape ((1) Fig 3.2.1) a pour but d'indexer le texte initial, à l'aide d'un langage à balises, basé sur le langage XML, et ceci, afin de caractériser les éléments susceptibles de constituer la description de l'espace scénique souhaité par le metteur en scène.
Cette première étape est elle-même divisée en trois séquences distinctes : l'indexation automatique, suivie de l'indexation manuelle réalisée par un utilisateur lecteur, et enfin l'indexation manuelle effectuée par le metteur en scène afin d'apporter toute modification ou précision qu'il souhaite.
A l'issue de cette première étape, les informations, notifiées grâces à l'ajout de balises dans le texte de l'auteur, sont extraites pour être interprétées en contraintes numériques ((2) Fig 3.2.1), afin d'être traitées par le système de génération ((3) Fig 3.2.1) qui s'appuie sur une base de connaissances construite au fur et à mesure de l'utilisation de l'outil.
La dernière étape consiste en l'instanciation de mises en scènes virtuelles dans l'espace de solutions, afin de présenter à l'utilisateur des possibilités de mise en scène à partir du texte indexé ((4) Fig 3.2.1) .
A la suite de cette présentation, l'utilisateur a la possibilité de revenir au texte pour modifier manuellement l'indexation en ajoutant, modifiant, ou retirant des balises, afin d'obtenir un nouveau résultat.
Enfin, si le résultat lui convient, il peut entériner le balisage qui sera conservé pour une utilisation ultérieure.

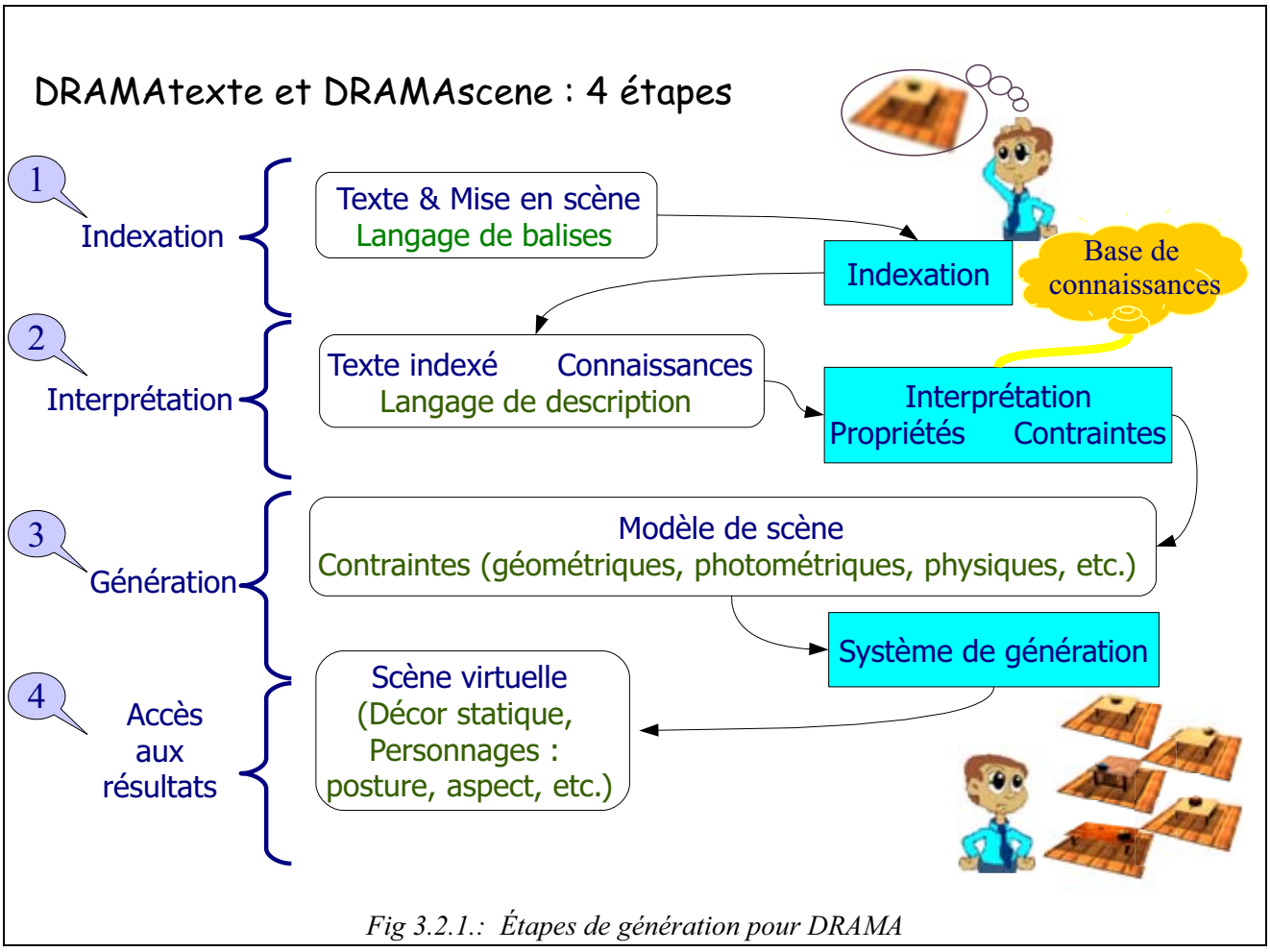

Fig 3.2.1.: Étapes de génération pour DRAMA

3.3. Indexation

Pour DRAMAtexte, nous avons mis en place un nouveau système de balisage conçu spécialement pour la représentation des entités présentes dans un texte théâtral.
Ce système de balisage est disponible au travers d'un outil informatique que nous développons afin de pouvoir étudier et évaluer les apports du système, notamment en ce qui concerne les procédures d'interrogation du texte.
Les textes disponibles au format électronique ne respectent malheureusement pas de normalisation en ce qui concerne la mise en forme du texte. Or, l'importation d'un texte théâtral dans notre outil a nécessité la définition d'une norme permettant l'extraction automatique des principales structures du texte.
L'outil peut ainsi importer des textes au format libre *OpenDocument* (utilisé notamment par la suite bureautique *OpenOffice*) respectant une mise en forme prédéfinie. En effet, certaines entités sont détectées à partir de leur format (titres, parties, etc.) ou relativement à des règles typographiques (nom des personnages commençant par une majuscule et suivis d'un ':' ou d'un '-', titre toujours en gras, taille de caractères respectant la structuration des titres des parties, didascalies toujours en italique, etc.)
Le texte, lors de l'importation, est converti à un format XML vérifiant le schéma DTD (*Document Type Definition*) qui définit la description formelle du système de balisage employé, propre à DRAMAtexte, utilisé pour qualifier les éléments caractéristiques, les attributs, etc.
XML fournit un moyen d'identifier la structuration et les informations implicites contenues dans le texte, indépendamment de l'application qui traite le document enrichi. Le schéma DTD permet à des groupes de personnes d'échanger des données respectant le format pré-établi. L'application peut employer le schéma DTD pour vérifier que les données reçues lors de l'importation sont valides. Un schéma DTD peut être aussi utilisé pour vérifier la validité des données au fur et à mesure de la phase d'indexation manuelle.
Notre outil, en se basant sur un schéma défini de façon externe à l'outil, est ainsi plus facilement extensible.
L'importation du texte initial applique l'opération de balisage automatique, à condition que le texte respecte le format attendu par DRAMAtexte (format qui peut être facilement obtenu grâce à un éditeur de texte tel que *OpenOffice*). L'étape d'indexation se déroule de façon totalement transparente au niveau de l'utilisateur.
Après cette première étape, la structure globale du texte est totalement balisée et interrogeable par l'utilisateur. Les balises automatiques insérées dans le texte sont identifiées grâce à la première lettre de la balise qui est de la forme <A...>.
Suite à cette première opération, l'utilisateur peut effectuer l'indexation manuelle, qui lui permettra de mettre en évidence les informations implicites présentes dans le texte, mais non identifiables lors de l'étape d'indexation automatique. Ces informations concernent en particulier les relations spatiales et temporelles, mais peuvent aussi concerner les descriptions de personnages, les sons, l'éclairage, etc. La liste des informations pouvant être caractérisées dépend du niveau de détail introduit dans le schéma DTD et de sa prise en compte dans l'outil DRAMAtexte.
Pour la mise en place du projet DRAMA global nous avons créé deux types de balises manuelles : les balises de la forme <M...> qui identifient les balises ajoutées manuellement par un lecteur, mais sans modification du texte initial, et celles de la forme <D...> qui identifient les balises correspondant aux indications du metteur en scène (directeur), ceci pour permettre de différencier les indications présentes dans le texte original (de l'auteur) et les ajouts éventuels (du metteur en scène). Ainsi, notre application pourra prendre en compte le texte de l'auteur dramatique, et également les choix du metteur en scène.
Au final, nous nous retrouvons avec trois types de balises. Par exemple, la notion de personnage (CHARACTER) peut être caractérisée dans le texte grâce aux trois types de balises : *automatiques* commençant par 'A' <ACH>, *manuelles* <MCH>, et du *metteur en scène* <DCH>.

Chaque balise ouvrante est toujours associée à une balise fermante. Par exemple <ACH> signifie le début de la définition d'un personnage et </ACH> en marque la fin.
Afin de permettre l'identification de chaque entité dans la scène, nous avons mis en place un ensemble de balises spécifiques à DRAMAtexte, correspondant aux éléments que nous souhaitons mettre en évidence. Nous avons associé à ces balises des attributs décrivant leurs propriétés ainsi que des valeurs permettant de qualifier les attributs. La table 3.1 donne un exemple de balises. On note que la balise relative aux *accessoires* ne peut être ajoutée que manuellement et qu'elle dispose d'un attribut pouvant prendre valeur *fixe* ou *mobile.*

Balises	A=Automatique	M=Manuelle	D=Metteur en scène	Attributs	Valeurs
Personnage	<ACH>	<MCH>	<DCH>		
Groupe de Personnages	<AGR>	<MGR>	<DGR>		
Découpage	<APA>	<MPA>	<DPA>		
Accessoires		<MPR>	<DPR>	Type	Fixe
					Mobile

Table 3.1 : Exemple de balisage

Nous allons illustrer la présentation des résultats sur un exemple de *didascalie*, correspondant à l'extrait d'un texte théâtral[1], décrivant les entités présentes sur la scène :

> *Dans le salon bourgeois de Mrs Peacock.*
> *La vieille dame, Mrs Peacock, est assise dans son fauteuil, un gros carnet et un stylo plume en mains. Elle griffonne, réfléchit. Auprès d'elle, Rose, la bonne, dépoussière vigoureusement les nombreux bibelots alignés sur les étagères, nettoie la pendule à coucou sur laquelle elle peut éventuellement s'attarder. Mrs Peacock s'interrompt dans son écriture et observe la bonne d'un œil sévère.*

La phase d'indexation automatique va déterminer que ce paragraphe est une didascalie, grâce au fait qu'il est écrit en italique (convention d'écriture usuelle dans les textes théâtraux que nous avons retenue pour notre schéma DTD). Le passage indexé va donc être encadré par les balises <ADID> et </ADID> :

```
<ADID>
    Dans le salon bourgeois de Mrs Peacock.
    La vieille dame, Mrs Peacock, est assise dans son fauteuil,
    un gros carnet et un stylo plume en mains. Elle griffonne, réfléchit.
    Auprès d'elle, Rose, la bonne, dépoussière vigoureusement les nombreux
    bibelots alignés sur les étagères, nettoie la pendule à coucou sur
    laquelle elle peut éventuellement s'attarder. Mrs Peacock s'interrompt
    dans son écriture et observe la bonne d'un œil sévère.
    </ADID>
```

Fig 3.3.1: Indexation automatique d'une didascalie

Ce texte est ensuite indexé manuellement pour renseigner les éléments susceptibles d'être intéressants lors de la phase d'interrogation ou lors de la mise en scène. Le texte résultant est celui que nous montre la figure 3.3.2 suivante :

[1] « Sous les masques » Auteurs : Pascale Hedelin, Christophe Merlin. Éditeur : Milan. Collection : Aujourd'hui théâtre

```
<ADID>
    Dans le
    <MSP>salon bourgeois</MSP>
    de
    <ACH>Mrs Peacock</ACH>
    . La
    <MMOR TYPE="Age" CHARACTER="Mrs Peacock">vieille</MMOR>
    <MMOR TYPE="sexe">dame</MMOR>
    ,
    <ACH>Mrs Peacock</ACH>
    , est
    <MPOS TYPE="sit" SOURCE="Mrs Peacock" CIBLE="fauteuil"> assise </MPOS>
    dans son
    <MS TYPE=" Fixed">fauteuil</MS>
    , un
    <MPR TYPE="mobile" CHARACTER="Mrs Peacock">gros carnet</MPR>
    et un
    <MPR TYPE="mobile" CHARACTER="Mrs Peacock">stylo plume</MPR>
    en mains. Elle
    <MAC TYPE="simple" CHARACTER="Mrs Peacock">griffonne</MAC>
    ,
    <MAC TYPE="simple" CHARACTER="Mrs Peacock"> réfléchit </MAC>
    .
    <MPOS TYPE="Near" SOURCE="Mrs Peacock" CIBLE="Rose">Auprès d'elle
    </MPOS>
    ,
    <ACH>Rose</ACH>
    , la
    <MMOR TYPE="OrdCostume" CHARACTER="Rose"> bonne </MMOR>
```

Fig 3.3.2: Texte avec balises après indexation manuelle

Il est à noter que l'utilisateur final de l'outil n'a jamais la nécessité d'avoir accès au texte au format XML tel qu'il est montré ci-dessus. L'introduction de balises manuelles est réalisée dans le texte initial à l'aide d'une interface adaptée de l'outil DRAMAtexte (Fig 3.3.3), qui masque totalement le format interne du balisage.

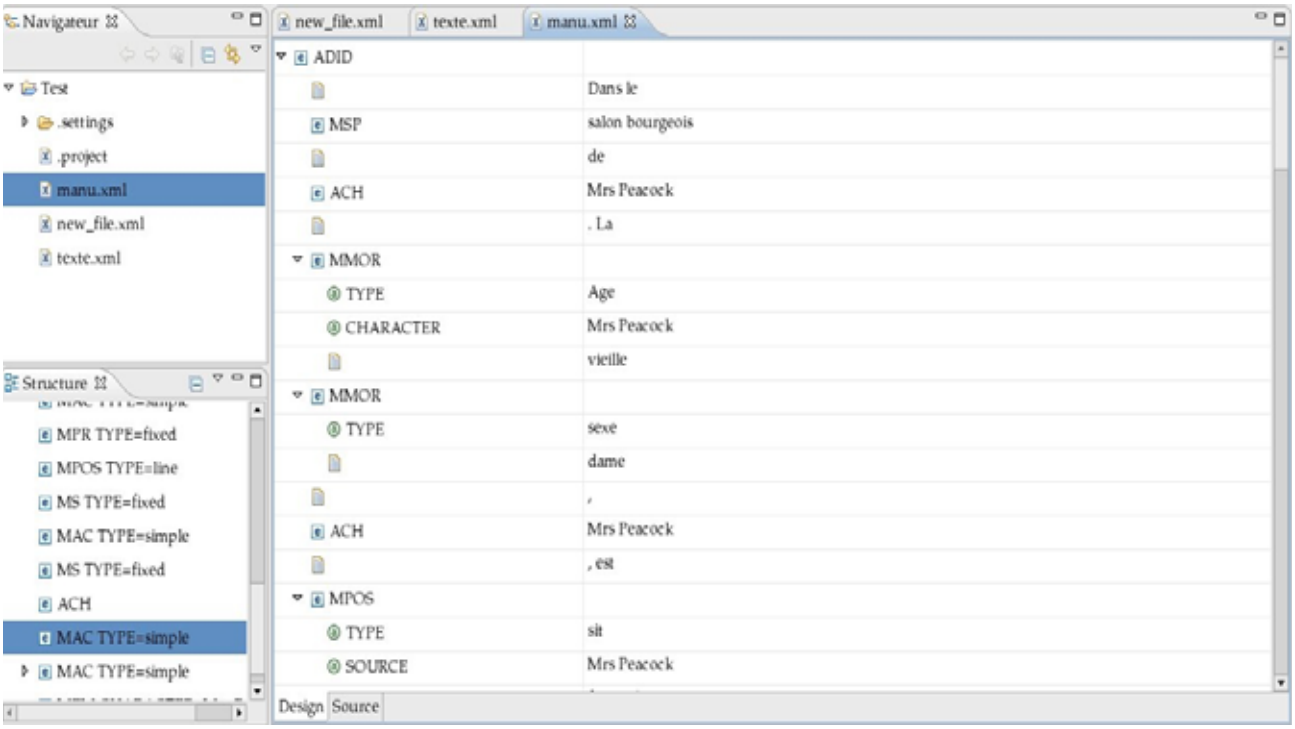

Fig 3.3.3: Affichage structuré du texte

Une fois que l'utilisateur juge le texte correctement indexé, il peut conserver le résultat pour une utilisation ultérieure. Ceci permet de créer une base de données de textes pré-indexés accessibles aux personnes intéressées.

3.4. Interrogation

Le résultat du balisage permet d'interroger le texte à l'aide de requêtes simples ou combinées afin de pouvoir éditer des listes d'éléments qui auront été indexés automatiquement ou manuellement. L'utilisateur peut ensuite croiser certains éléments entre eux (obtenir par exemple, les costumes des personnages dans une scène donnée, compter le nombre d'entrées de personnages dans une partie, etc.) Ce processus d'interrogation se base sur toutes les informations signalées par les balises, tels que les éléments scénographiques, les costumes, l'éclairage, etc.
La possibilité de recouper ainsi des informations va permettre au lecteur ou au metteur en scène d'avoir une lecture plus efficace afin d'avoir une vue synthétique du texte. Mais elle pourra permettre également d'élargir les champs d'études, en mettant en lumière des aspects encore inexplorés de la théâtralogie (notamment tout ce qui a trait au regard du spectateur et à l'expression de l'émotion, encore peu exploré).
L'indexation pourra à terme également permettre :

- d'effectuer aisément une lecture partielle ou intégrale de la scène textuelle ;
- de créer la partition rythmique du texte ;
- d'analyser l'histoire conversationnelle ;
- de créer une base de données de textes, indexés suivant des entrées différentes.

Ainsi, à partir d'un texte indexé, l'utilisateur pourra effectuer des interrogations complexes et dynamiques, afin d'extraire les informations nécessaires à l'étude d'un texte théâtral. Ces deux possibilités complémentaires d'indexation et d'interrogation vont être très utiles et sont très attendues par les personnes étudiant le domaine théâtral. Elles faciliteront le travail, qui peut être fastidieux, de recherche exhaustive d'informations dans un corpus de textes en vue d'en établir une synthèse ou une analyse.
Enfin, dans un but plus pratique, le processus d'interrogation pourra permettre d'obtenir immédiatement des informations sur les éléments de décors ou les accessoires, les éclairages, les costumes ou encore le maquillage, et ceci dans le but de faciliter la tâche de tous les corps de métiers impliqués dans la réalisation d'une mise en scène.

3.5. Mise en oeuvre

Pour des raisons de cohérence entre les différentes partie du projet et pour garantir la portabilité et la robustesse de l'application, nous avons opté pour un développement avec le langage de programmation orienté objet JAVA [13].
Vu la portabilité offerte par le langage JAVA, cet outil pourra s'installer et s'exécuter facilement sur tout système d'exploitation, des ordinateurs portables voire des ordinateurs de poche PDA (Personal Digital Assistant).
Afin de simplifier la tâche des utilisateurs de DRAMA, nous avons conçu et développé un assistant facilitant la détermination et la correction des erreurs éventuelles pouvant advenir lors de la phase d'indexation.
Nous avons automatisé le plus de tâches possibles. En effet, l'insertion d'une balise dans le texte se fait simplement grâce à des menus contextuels. L'utilisateur a la possibilité de créer ses propres balises. Chaque attribut est automatiquement associé à une valeur par défaut, de façon à éviter de devoir renseigner tous les champs de façon systématique. Cependant, l'utilisateur peut modifier les choix par défaut chaque fois que ça lui paraît nécessaire. La figure suivante (Fig 3.5.1) donne un aperçu de la structure finale de DRAMAtexte.

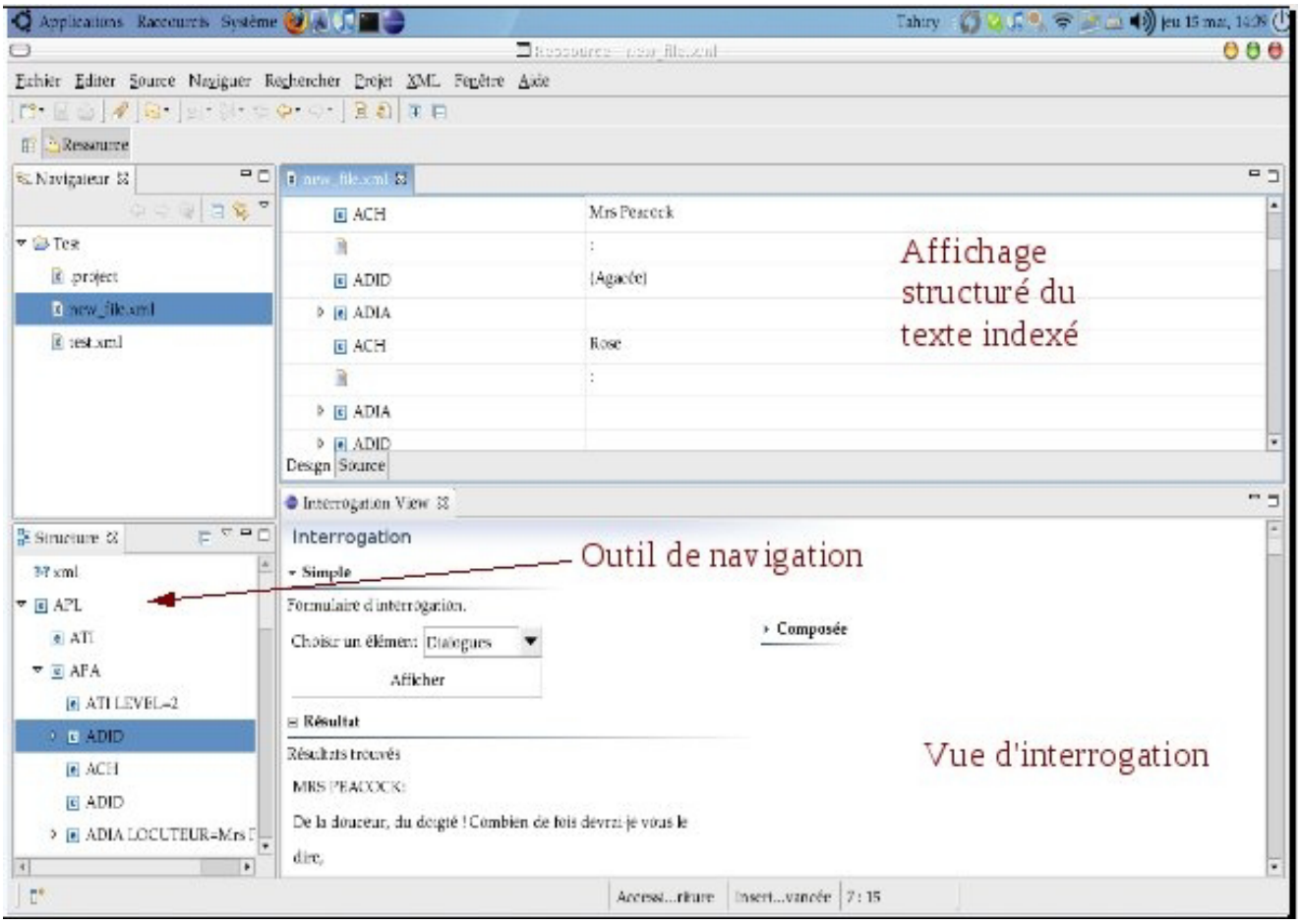

Fig 3.5.1: Interface globale de DRAMAtexte

4. Base de connaissances

Le texte indexé obtenu à partir de DRAMAtexte est utilisé pour déterminer les propriétés (au sens de la modélisation déclarative) qui vont être interprétées en contraintes pour être ensuite utilisées par un système de génération (cf. §2.2).
Or, le texte de théâtre étant un texte laissant place à l'interprétation du lecteur, il devient nécessaire de compléter un certain nombre d'informations non renseignées, mais pourtant nécessaires, pour pouvoir lancer la génération d'un environnement scénique et obtenir un résultat pouvant être jugé satisfaisant par l'utilisateur.

Pour cela, nous avons introduit un langage de description de scénario déclaratif appelé :*Virtual Environment Description Language (VEDEL)* [14], étudié dans le cadre du projet Geda-3D [15] et développé au *Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV – IPN),* à Guadalajara au Mexique, par le *Distributed Systems Group*.

Une description fournie dans le langage VEDEL, est constituée de trois grandes parties :

- les éléments descriptifs de l'environnement virtuel ;
- les acteurs (entités qui peuvent effectuer des actions et sont soumises à des changements dus à la modification de l'environnement ou aux actions d'autres entités) ;
- les objets (entités qui ne peuvent effectuer d'actions, mais sont également soumis aux effets de l'environnement et à des autres entités).

La description est composée de « phases » qui indiquent pour chaque élément descriptif et chaque entité quel est son type, ses attributs et les valeurs des attributs.
Cette description, fournie en langage VEDEL, correspond à une formalisation des propriétés, telles qu'elles ont été définies grâce à l'indexation. Elle est destinée à être interprétée en contraintes, devant être résolues par un système de génération.

Mais dans le cas où la description est très incomplète, l'espace des solutions possibles peut être de taille quasiment infinie. Quand le système de génération doit faire face à un système de contraintes fortement sous-contraint où le nombre de solutions se révèle trop important pour être exploré, même très partiellement.
Par exemple, si on a juste la connaissance du fait qu'une table est sur la scène, il y a une infinité de positions et d'orientations possibles pour cette table, sans aucune indication pouvant permettre de guider le choix du système de génération.

C'est pour cela que nous avons introduit une base de connaissances destinée à renseigner le système de génération, afin de préciser les choix possibles en fonction de données disponibles.
Ces connaissances peuvent concerner, par exemple, les mises en scène correspondant à l'époque ou au genre de la pièce, ou encore les habitudes de l'auteur, voire du metteur en scène.
Ainsi, dans le cas où des propriétés n'ont pas été suffisamment renseignées, le système de génération va pouvoir rechercher les attributs et leurs valeurs, tels qu'ils ont été stockés dans la base de connaissances.
La base de connaissances joue donc le rôle d'une ontologie utilisée lors de l'interprétation des propriétés descriptives, afin de compléter et valider les choix effectués lors de la phase d'indexation.

On trouvera également dans cette base et pour chaque type d'objet pouvant être placé sur une scène, des caractéristiques propres qui vont permettre au système de génération de résoudre les contraintes d'une façon pertinente. Dans l'exemple (Fig 4.1), on peut voir une mise en évidence des marqueurs logiques qui vont désigner les points caractéristiques et les zones de placement relatifs aux objets de type *Chaise*.

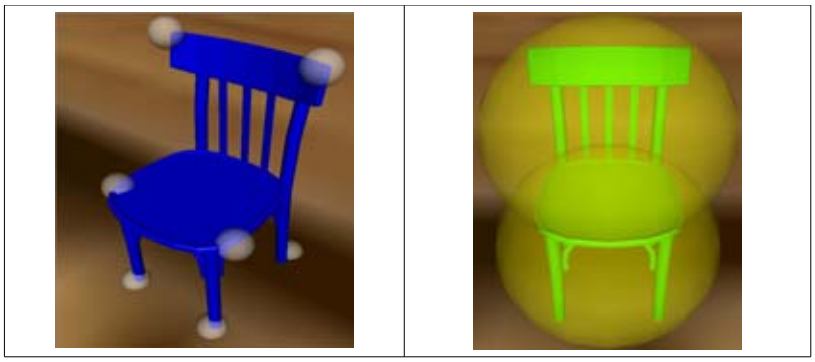

Fig 4.1 : Exemple de marqueurs utilisés lors de la phase de génération [16]

A partir de toutes les données disponibles, obtenues grâce à l'indexation ou issues de la base de connaissances, le système de génération pourra calculer et présenter à l'utilisateur un sous-ensemble d'instanciations de l'espace de solutions, parmi lesquelles il pourra faire un choix.
Parmi ces instanciations pourra peut-être émerger une configuration scénique à laquelle le metteur en scène n'avait pas songé a priori. L'outil de génération pourra ainsi également devenir un outil de proposition, laissant toute la liberté à l'utilisateur de retenir ou non les solutions issues de la description qu'il aura fournie.

5. Conclusion et perspectives

DRAMAtexte est un outil qui permet à un lecteur ou à un metteur en scène d'indexer un texte théâtral. L'indexation est un moyen d'enrichir le texte en ajoutant des informations implicites, autorisant par la suite des interrogations pouvant fournir des résultats dont l'obtention grâce à des moyens classiques se révélerait fastidieuse.
Les opérations de balisage automatiques et manuelles vont pouvoir avoir des applications dans des domaines qui vont au delà de ceux présentés dans cet article. Concernant la recherche fondamentale cet outil va pouvoir faciliter, voire rendre enfin possible :

- l'analyse synchronique des différents éléments (impossible dans le cas d'une lecture diachronique) ;
- la visualisation de l'espace et de son occupation par les personnages, grâce à l'apport de l'imagerie 3D (plus difficile dans l'espace imaginaire de la lecture, surtout pour des personnes se formant à la mise en scène) ;
- la globalisation de la construction du personnage (difficile à réaliser en amont dans la lecture).

Pour la recherche appliquée aux arts de la scène l'outil va permettre de fournir aux praticiens du théâtre (scénographes, costumiers, acteurs, techniciens, etc.) : des listes, des graphiques, et des schémas, permettant une visualisation de ces différents éléments qui serviront de cadre à l'élaboration de la dramaturgie scénique.
Tout ceci va permettre aux *généticiens du théâtre* de comparer en direct la partition originale et celle des différentes mises en scènes.

Cet enrichissement du texte se traduit également par l'ajout de propriétés (principalement des relations spatiales entre objets ou personnages présents sur la scène), utilisées afin de calculer et proposer une visualisation scénique virtuelle en trois dimensions. Pour cela, il va falloir étudier un système de génération robuste et rapide permettant de déterminer, à partir des propriétés et de la base de connaissances, un ensemble d'instanciations de mises en scènes possibles, respectant la description et fournissant des solutions valides.
Ce système de génération qui va être le fondement de DRAMAscène, est basé sur le principe de la modélisation déclarative présenté à la Fig 2.1.1. Il a pour but de fournir à l'utilisateur des exemples schématiques de ce que pourrait être la mise en scène, en fonction des contraintes issues du texte, de la volonté du metteur en scène, ou bien encore de la configuration de la scène.
Si aucune des solutions présentées ne conviennent, l'utilisateur pourra revenir sur la phase d'indexation afin de préciser ou corriger ses choix. Ensuite il pourra relancer le système de génération qui lui proposera d'autres solutions.

Cette étape nécessitera également l'étude et le développement d'un système d'interprétation dont le rôle est de traduire les propriétés obtenues grâce à l'indexation des didascalies en contraintes, géométriques ou non, prises en charge et résolues par le système de génération.

Ensuite, le projet DRAMA se poursuivra par l'introduction d'aspects dynamiques, tels que la prise en compte des déplacements sur la scène, les éclairages ou la synchronisation avec une bande sonore. Au delà des applications pour le théâtre, et de façon plus générale les arts du spectacle, ce projet implique des problématiques de recherche et la mise en oeuvre de technologies pouvant être directement transposées dans d'autres domaines.
Par exemple, on pourra transposer cette étude aux environnements virtuels distribués, liés à des systèmes d'informations multimédia incluant des textes automatiquement indexés à partir de documents techniques, et des vidéos. Ces applications sont particulièrement appropriées à l'ingénierie collaborative, à la surveillance de sites industriels, à la construction, ainsi qu'aux tâches de maintenance.

Dans cette volonté permanente de confronter théorie et pratique, le module DRAMAtexte est la première étape d'un projet en gestation depuis 12 ans, mais qui commence tout juste depuis l'année 2008 à connaître une véritable mise en oeuvre.
Ce projet permet de faire se rencontrer praticiens et théoriciens autour d'un même objectif : conserver la mémoire du processus de création théâtrale. En effet, il n'existe aujourd'hui aucun support permettant, non seulement de conserver efficacement cette mémoire, mais également de la transmettre aux futures générations de metteurs en scènes.
Les outils que nous étudions n'ont surtout pas pour but de proposer ou établir un schéma de création, mais uniquement de simplifier et de conserver certaines étapes du processus de création.

Ce projet est également un lieu d'échange important entre chercheurs issus d'horizons très différents puisque certains sont membres de l'IRIT (Institut de Recherche en Informatique de Toulouse) et proviennent de l'univers des sciences de l'information et d'autres sont membres du laboratoire Españ@31 (groupe de recherche sur l'Espagne contemporaine de l'Université de Toulouse II).

6. Bibliographie

[1] GAILDRAT, V., *Modélisation déclarative d'environnements virtuels : Création de scènes et de formes complexes par l'énoncé de propriétés et l'emploi d'interactions gestuelles*, Habilitation à diriger des recherches, Université Paul Sabatier, janvier 2003.

[2] LARIVE, M., GAILDRAT, V., *Wall Grammar for Building Generation*, Graphite 2006, 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and South-East Asia, ACM SIGGRAPH Conference, ACM Press, Kuala Lumpur, Malaysia, pp. 429-438, 29 novembre-02 décembre 2006.

[3] PARISH, Y. I.H., MULLER, P., *Procedural modeling of cities*, SIGGRAPH'01, Los Angeles, pp 301-308, août 2001.

[4] SANCHEZ, S., LE ROUX, O., LUGA, H., GAILDRAT, V., *Constraint-Based 3D-Object Layout using a Genetic Algorithm*, 3IA'2003, The Sixth International Conference on Computer Graphics and Artificial Intelligence, Limoges, 14 - 15 mai 2003.

[5] COYNE, R., SPROAT, R., *WordsEye : An automatic text-to-scene conversion system*, SIGGRAPH'01, Los Angeles, août 2001.

[6] BRAY, T., PAOLI, J., SPERBERG-McQUEEN, C. M., *Extensible Markup Language (XML) 1.0 (2nd Edition),* http://www.w3.org/TR/REC-xml.

[7] HOCKEY, Susan, *Evaluating Electronic Texts in the Humanities*, Library Trends 42, no 4, Spring 1994.

[8] PANTELIA, M., PEEVERS, R., *Thesaurus Linguae Graecae*, University of California, Irvine, California, 2004.

[9] BURNARD, L., *Report of Workshop on Text Encoding Guidelines*, Literary and Linguistic Computing 3: 131–3, 1988.

[10] PRICE-WILKIN, J., *Text Files in Libraries : Present Foundations and Future Directions*, Library hitech 9, no 3, pp 7- 44, 1991.

[11] BEECHAM, S., HOWARD, G., *Making opera sing: OpenDrama and the Magic Flute*, International Cultural Heritage Informatics Meeting: Ichim'03, 2003.

[12] HART, Michael, *Gutenberg: The History and Philosophy of Project Gutenberg*, http://www.gutenberg.org/wiki/Main_Page, 2006.

[13] Sun Microsystems, Java 2 Micro Edition, http://www.java.sun.com/j2me/, 2001.

[14] ZARAGOZA, J. A. Rios, *Representation and exploitation of knowledge for the description phase in declarative modeling of virtual environments,* Master's thesis, Centro de Investigación y de Estudio Avanzados del IPN, Unidad Guadalajara, 2006.

[15] RAMOS, F., ZUNIGA, F., PIZA, H., *A 3D-space platform for distributed applications management,* International Symposium and School on Advanced Distributed Systems 2002 Guadalajara, Jal., Mexico, November 2002.

[16] LE ROUX, O., *Modélisation déclarative d'environnements virtuels : contribution à l'étude des techniques de génération par contraintes,* Thèse de doctorat, Université Paul Sabatier, 2003.
[17] FROST, D. H., *Algorithms and heuristics for constraint satisfaction problems,* PhD thesis, University of California,1997.
[18] BRASSARD, G., BRATLEY, P., *Fundamentals of algorithmic,* Prentice-Hall, Inc., 1996.
[19] RUSSEL, S. J., NORVIG, P., *Artificial Intelligence: A Modern Approach,* Pearson Education, 2003.
[20] GASCHNIG, J. G., *Performance measurement and analysis of certain search algorithms,* PhD thesis, Carnegie-Mellon Univ. Pittsburgh Pa. Dept. Of Computer Science, 1979.
[21] DECHTER, Rina, *Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition.* Artif. Intell., 41(3), pp 273-312, 1990.
[22] HARALICK, R. M., ELLIOTT, G. L., *Increasing tree search efficiency for constraint satisfaction problems,* Artif. Intell., 14(3), pp 263-313, 1980.

Bibliography

[1] Demitri Plemenos, Georges Miaoulis, and Nikos Vassilas. Machine learning for a general purpose declarative scene modeller. In *International Conference GraphiCon'2002, Nizhny Novgorod (Russia), September 15-21*, 2002.

[2] Véronique Gaildrat. Declarative modelling of virtual environment, overview of issues and applications. In *International Conference on Computer Graphics and Artificial Intelligence (3IA), Athènes, Grèce*, volume 10, pages 5–15. Laboratoire XLIM - Université de Limoges, may 2007.

[3] Felix Ramos, Fabiel Zúniga, and Hugo I. Piza. A 3D-space platform for distributed applications management. *International Symposium and School on Advanced Distributed Systems 2002. Guadalajara, Jal., México*, November 2002.

[4] Dirk Fahland. Towards analyzing declarative workflows. In *Autonomous and Adaptive Web Services*, number 07061 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[5] Antoine Spicher and Olivier Michel. Declarative modeling of a neural-like process. *Biosystems*, 87(2-3):281 – 288, 2007.

[6] Dimitri Plemenos. Using artificial intelligence techniques in computer graphics. In *GraphiCon*, 2000.

[7] Olivier Le Roux, Véronique Gaildrat, and René Caubet. Design of a new constraints solvers for 3d declarative modeling. In *International Conference on Computer Graphics and Artificial Intelligence (3IA), Limoges, 03/05/200-04/05/200*, pages 75–87, may 2000.

[8] Olivier Le Roux, Véronique Gaildrat, and René Caubet. Using constraint satisfaction techniques in declarative modeling. In *Geometric Modeling Techniques, Applications, Systems and Tools*, pages 1–20. Kluwer Academic Publishers, 2004.

[9] Mathieu Larive, Olivier Le Roux, and Véronique Gaildrat. Using Meta-Heuristics for Constraint-Based 3D Objects Layout. In *International Conference on Computer Graphics and Artificial Intelligence (3IA), Limoges, France, 12/05/04-13/05/04*, pages 11–23, maY 2004.

[10] Stephane Sanchez, Olivier Le Roux, H. Luga, and Véronique Gaildrat. Constraint-Based 3D-Object Layout using a Genetic Algorithm. In *International Conference on Computer Graphics and Artificial Intelligence (3IA), Limoges, 14/05/2003-15/05/2003*, may 2003.

[11] Ghassan Kwaiter, Véronique Gaildrat, and René Caubet. Controlling object natural behaviors with a 3d declarative modeler. In *Computer Graphics International*, pages 248–, 1998.

[12] Pierre Barral, Guillaume Dorme, and Dimitri Plemenos. Visual understanding of a scene by automatic movement of a camera. In *GraphiCon*, 1999.

[13] Olivier Le Roux, Véronique Gaildrat, and Réne Caubet. Using constraint propagation and domain reduction for the generation phase in declarative modeling. In *IV '01: Proceedings of the Fifth International Conference on Information Visualisation*, page 117. IEEE Computer Society, 2001.

[14] A. Winter, A. Strübing, L. Ißler, B. Brigl, and R. Haux. Ontology-based assessment of functional redundancy in health information systems. *Lecture Notes in Computer Science*, 5421:213 – 226, 2009.

[15] Veronique Giudicelli and Marie-Paule Lefranc. Ontology for immunogenetics: The IMGT-ONTOLOGY. *Bioinformatics*, 15(12):1047–1054, 1999.

[16] Mike Uschold, Mike Uschold, Michael Grüninger, and Michael Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136, 1996.

[17] Gutiérrez-García J. Octavio, Koning Jean-Luc, and Ramos-Corchado Félix F. An obligation approach for exception handling in interaction protocols. In *Workshop on Logics for Intelligent Agents and Multi-Agent Systems (WLIAMAS 2009) at IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - WI-IAT'09*, Milan, Italy, September 2009. IEEE CS Press.

[18] Stephan Grimmp, Pascal Hitzler, and Andreas Abecker. Knowledge representation and ontologies logic, ontologies and semantic web languages. draft, 2006.

[19] Maryam Alavi and Dorothy E. Leidner. Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly*, 25(1):107–136, 2001.

[20] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis*, 5(2):199–220, June 1993.

[21] Phillip Breay. The social ontology of virtual environments - criticisms and reconstructions. *The American Journal of Economics and Sociology*, 62(1):269–282, January 2003.

[22] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: A tool for collaborative ontology construction. Technical report, Knowledge Systems Laboratory, Stanford University, 1996.

[23] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge modeling at the millennium (the design and evolution of protege-2000). Technical report, Stanford Medical Informatics, 1998.

[24] Henrik Eriksson, Yuval Shahar, Samson W. Tu, Angel R. Puerta, and Mark A. Musen. Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79(2):293–326, 1995.

[25] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C recommendation, World Wide Web Consortium, February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[26] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C recommendation, World Wide Web Consortium, February 2004. http://www.w3.org/TR/2004/REC-rfd-concepts-20040210/.

[27] John Dominguez. Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. In *In Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling and Management, KAW'98, Banff, Canada*, April 1998.

[28] E. Motta. *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1999.

[29] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[30] Solomon W. Golomb and Leonard D. Baumert. Backtrack programming. *J. ACM*, 12(4):516–524, 1965.

[31] John Gary Gaschnig. *Performance measurement and analysis of certain search algorithms.* PhD thesis, Carnegie-Mellon Univ. Pittsburgh Pa. Dept. Of Computer Science, 1979.

[32] Rina Dechter. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41(3):273–312, 1990.

[33] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3):263–313, 1980.

[34] Steve Aukstakalnism and David Blatner. *Silicon Mirage, The Art and Science of Virtual Reality*. Peachpit Press, Berkeley, CA, USA, 1992.

[35] Jerry Isadle. What is virtual reality?, a web-based introduction. WebPage, 1998. http://vr.isdale.com/WhatIsVR/frames/WhatIsVR4.1.html, Last visited 06/14/2007.

[36] Luis Alfonso Razo Ruvalcaba. Algoritmos de comportamiento y personalidad para agentes emocionales. Master's thesis, Centro de Investigación y de Estudios Avanzados del IPN, Unidad Guadalajara, 2007.

[37] Orozco H. R., Ramos F., Zaragoza J., and D. Thalmann. *Frontiers in Artificial Intelligence and Applications (Advances in Technological Applications of Logical and Intelligent Systems)*, volume 186, chapter Avatars Animation Using Reinforcement Learning in 3D Distributed Dynamic Virtual Environments, pages 67–84. IOS Press, Washington, DC, 2009.

[38] Philippe Codognet. Declarative behaviors for virtual creatures. In *SIGGRAPH '99: ACM SIGGRAPH 99 Conference abstracts and applications*, page 237. ACM, 1999.

[39] Samuel R. Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, 2003.

[40] Jean-Eudes Marvie, Julien Perret, and Kadi Bouatouch. The fl-system: a functional l-system for procedural geometric modeling. *The Visual Computer*, 21(5):329–339, jun 2005.

[41] Yoav I. H. Parish and Pascal Müller. 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, 2001. ACM.

[42] Mathieu Larive, Yann Dupuy, and Véronique Gaildrat. Automatic generation of urban zones. In *WSCG (Short Papers)*, pages 9–12, 2005.

[43] Stefan Göbel, Oliver Schneider, Ido Iurgel, Axel Feix, Christian Knöpfle, and Alexander Rettig. Virtual human: Storytelling and computer graphics for a virtual human platform. *Lecture Notes In Computer Science*, pages 79–88, 2004.

[44] Riva G. Application of virtual reality in medicine. *Methods of information in medicine*, 5(5):524–534, October 2003.

[45] M. J. North, T. R. Howe, N. T. Collier, and J. R. Vos. *A Declarative Model Assembly Infrastructure for Verification and Validation*, pages 129–140. Springer Japan, 2007.

[46] R. Raymond Lang. A declarative model for simple narratives. In *Proceedings of the AAAI Fall Symposium on Narrative Intelligence*, pages 134–141. AAAI Press, 1999.

[47] P. Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., 1990.

[48] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. *ACM Transactions on Graphics*, 22(4):669–677, july 2003.

[49] Bob Coyne and Richard Sproat. Wordseye: An automatic text-to-scene conversion system. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 487–496. AT&T Labs Research, 2001.

[50] G. Kwaiter, V. Gaildrat, and R. Caubet. Dem^{2}ons: A high level declarative modeler for 3D graphics applications. In *Proceedings of the International Conference on Imaging Science Systems and Technology, CISST'97*, pages 149–154, 1997.

[51] Issn x, D. Wang, D. Wang, I. Herman, I. Herman, G. J. Reynolds, and G. J. Reynolds. The open inventor toolkit and the premo standard, 1997.

[52] "The Open Group". Motif 2.1-programmer's guide, 1997.

[53] William Ruchaud and Dimitri Plemeno. Multiformes: A declarative modeller as a 3D scene sketching tool. In *ICCVG*, 2002.

[54] Ken Xu. Constraint-based automatic placement for scene composition. In *In Graphics Interface*, pages 25–34, 2002.

[55] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 191–195, New York, NY, USA, 2003. ACM.

[56] Jaime Alberto Zaragoza Rios. Representation and exploitation of knowledge for the description phase in declarative modeling of virtual environments. Master's thesis, Centro de Investigación y de Estudios Avanzados del Intituto Politécnico Nacional, Unidad Guadalajara, Guadalajara, México, 2006.

[57] Mike Bayer Benjamin Geer. Freemarker: Java template engine libray. webpage, december 2008.

[58] Alonso Gutierrez Aguirre. Núcleo geda-3D. Master's thesis, Centro de Investigación y de Estudios Avanzados del IPN, Unidad Guadalajara, 2007.

[59] Alma Verónica Martínez González. Lenguaje para animación de creaturas virtuales. Master's thesis, Centro de Investigación y de Estudios Avanzados del IPN, Unidad Guadalajara, 2005.

[60] Andriamarozakaniaina T., Pouget M., Zaragoza J., and Gaildrat V. Dramatexte: indexation et base de connnaissances. Premier Colloque international sur la notation informatique du personnage, may 16-17, 2008, Toulouse, France. Publishing pending.