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*.

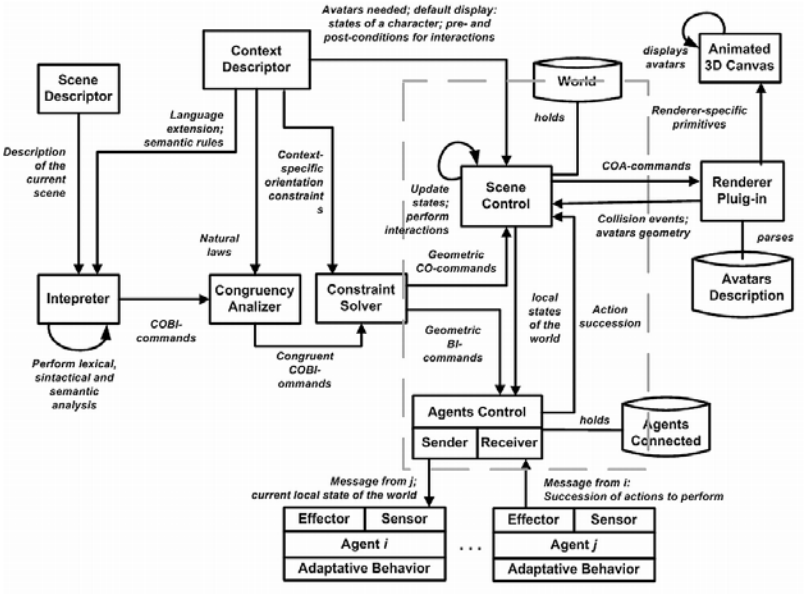

Figure 1. GeDA-3D Agent Architecture

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 $D = (C \times P)$ 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].

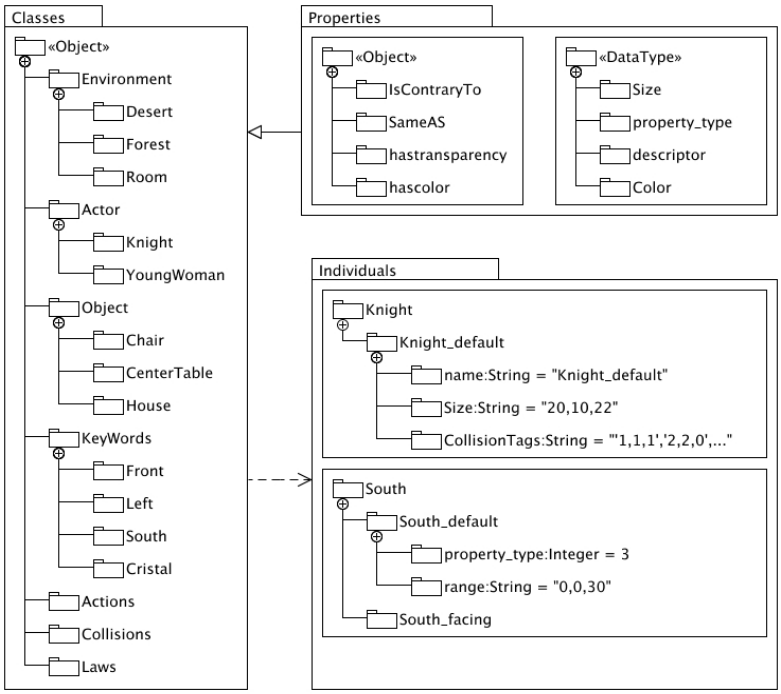

Figure 3. The ontology used by the Virtual Editor

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.

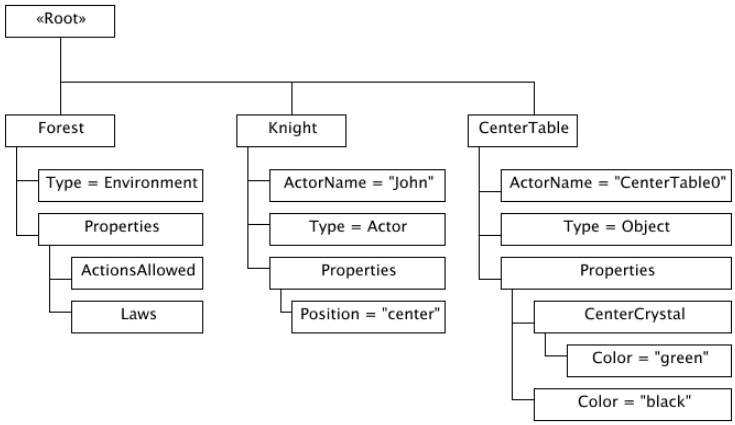

Figure 4. Data structure obtained from a Description

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.

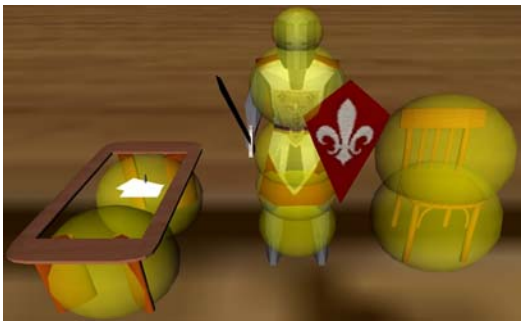

Figure 5. Collision tags examples

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.

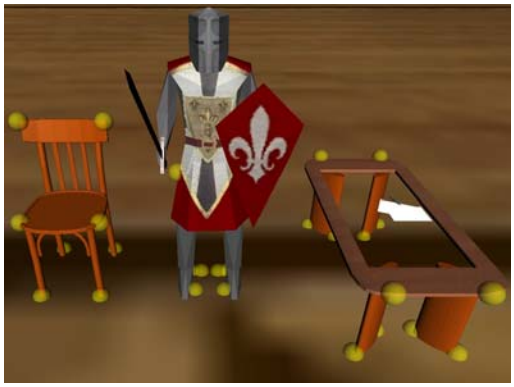

Figure 6. Characteristic points for different entities

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.

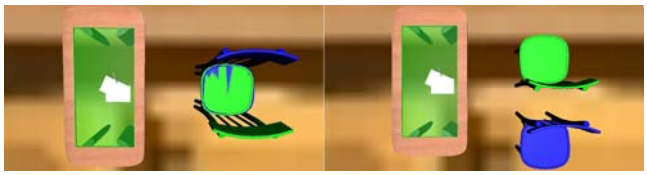

Figure 7. Conflict solving example

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]
```

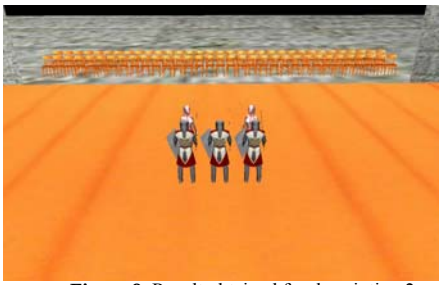

Figure 9. Result obtained for description 2

Description 3.

[ENV]
void.
[/ENV]

[ACTOR]
[/ACTOR]

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

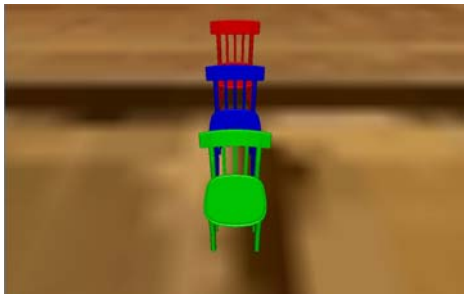

Figure 10. Result obtained for description 3

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.