Noname manuscript No.
(will be inserted by the editor)

A Virtual World Grammar for Automatic Generation of Virtual Worlds

Tomas Trescak · Marc Esteva · Inmaculada Rodriguez

Received: date / Accepted: date

Abstract Hybrid systems such as those that combine 3D virtual worlds and organization based multiagent systems add new visual and communication features for multi-user applications. The design of such hybrid and dynamic systems is a challenging task. In this paper we propose a system that can automatically generate a 3D virtual world (VW) from an organization based multiagent system (MAS) specification that establishes the activities participants can engage on. Both shape grammar and virtual world paradigms inspired us to propose a Virtual World Grammar (VWG) to support the generation process of a virtual world design. A VWG includes semantic information about both MAS specification and shape grammar elements. This information, along with heuristics and validations, guides the VW generation producing functional designs. To support the definition and execution of a Virtual World Grammar, we have developed a so named Virtual World Builder Toolkit (VWBT). We illustrate this process by generating a 3D visualization of a virtual institution from its specification.

Tomas Trescak
Artificial Intelligence Research Institute
Spanish Council for Scientific Research
Barcelona, Spain
E-mail: ttrescak@iiia.csic.es

Marc Esteva
Artificial Intelligence Research Institute
Spanish Council for Scientific Research
Barcelona, Spain
E-mail: marc@iiia.csic.es

Inmaculada Rodriguez
Applied Mathematics Department
University of Barcelona
Barcelona, Spain
E-mail: inma@maia.ub.es

Keywords Shape Grammars · Virtual Institutions · 3D Virtual Worlds · Multi-Agent Systems · CAD

1 Introduction

Nowadays e-* applications, where * stands either for e-commerce, e-learning or e-government, are mostly web-based applications where stakeholders have no visual clues helping them to carry out their interactions. These applications have limited possibility to deal with user interaction. This lack of multiuser and visual awareness in web-based systems can be handled by the 3D virtual world (VW) technology. Virtual worlds are graphical environments that provide effective communication among participants and let them focus their attention on the who, where, or when of events.

Our belief is that 3D virtual worlds in combination with organization based multiagent systems [5] (i.e a computational version of traditional human organizations or institutions) may provide at least three new possibilities to e-* applications:

– Thanks to the regulation imposed by the multiagent system (MAS), the 3D environment becomes a normative virtual world where norms are enforced at runtime instead of by the terms of service contract.
– A 3D real-time representation of the multiagent system facilitates a better understanding of what is happening at both agent and the entire system levels.
– Virtual world participants can be both humans and software agents facilitating human direct participation in MAS and intelligent agents participation in VW.

The construction of such a 3D virtual world using an authoring system i) is a time consuming task for designers and ii) makes it almost impossible to manage the dynamic update of the virtual world design at runtime. The latter issue is crucial for us as we are interested in the dynamic nature of VWs. Then, we need a powerful method to generate 3D virtual scenes in an automatic way.

In this paper, we propose a system that can automatically generate a 3D virtual world from a MAS specification, that is, from a formal description of activities taking place in the system modeled by the organization based MAS. Both shape grammars and virtual worlds paradigms inspired us to define the concept of Virtual World Grammar (VWG) that supports the generation of such 3D virtual worlds. A shape grammar is a method of generating designs which uses primitive shapes and rules of interaction among them [11][13]. While a shape grammar only conveys geometrical data, a VWG overcomes this lack of semantic data. It includes a shape grammar, semantic information about both MAS specification and shape grammar elements. Moreover, it includes heuristics and validations that guide the VW generation producing functional designs. We have developed a framework named Virtual World Builder Toolkit (VWBT) for the definition and execution of VWGs.

Section 2 provides a motivation example and an overview of our system. Section 3 presents the related work. Section 4 introduces the VWG and formally defines all of its components. Section 5 presents the toolkit developed to exploit VWGs. Section 6 presents results. Finally, Section 7 gives conclusions and ideas for future works.

2 Motivation example and overview of the system

Our motivation example is an auction system which allows both in-house users (bidders present in a real auction room) and internet users to participate in real auctions happening all around the world.

However, how to accomplish the presence in all these places and achieve an effective and comfortable communication between in-house and internet users? Our answer is a hybrid environment which combines 3D virtual worlds and multi-agent system technologies. All these auctions are generated as some virtual space, either as a room in a big auction building or as a separate building in the virtual world. All the users are displayed as avatars. Internet users move around the building and visit different auctions by entering auction rooms. In-house users are tracked either by cameras or some communication device and their act is constantly updated in the 3D representation.

Figure 1 gives an overview of our approach, that facilitates the generation of such type of hybrid environments out of a formal specification. In particular we focus on the generation of a Virtual Institution (VI), from its specification (performative structure). A VI is a 3D virtual world with normative regulation of interactions [2]. A specification of the Auction House virtual institution is depicted in the top rectangle of Figure 1. Rounded rectangles represent *activities* (also called *scenes*). In this performative structure we see the following scenes: Admission, Item Registration, Auction and Auction Info. The initial and final scenes represent the institution entrance and exit points which are mapped to the entry and exit of the generated 3D VW. In our proposed auction house scenario, there are many "Auction" scenes, the number depends on currently active real-world auctions.

Fig. 1: System overview

As indicated by dotted arrows in Figure 1, the definition of objects (from both the specification and shape grammar) along with the list of their properties forms a general vocabulary, that is the *ontology* of the VW grammar. For each object of the specification and shape grammar (SG) we create an instance of the related ontology object. The system also needs to specify *objects' mappings* that define which shape grammar object can represent which specification element. In VI we are focusing on *activities* which are mapped to the spaces, such as stand-alone buildings or rooms in an institution building.

When we have successfully defined our ontology and created all instances of specification and shape grammar objects, we can proceed to the generation of a VW. In what order we process the objects and where are they placed according to the position of previously placed object? This is where *heuristics* take an important role. They decide the next specification element to process and the applicable rule of the shape grammar for the selected specification element.

To make sure that we are generating functional and correct designs, we use *validations* during every step of the generation. We can also *evaluate* the final design. For example we do not want designs where rooms intersect each other or rooms which have no entry or exit. As shown in the bottom of Figure 1, the automatic generation of a virtual institution is done in 2 steps. First, a 2D floor plan of the institution is generated. Then a 3D transformation mechanism transforms this floor plan into a final 3D scene.

3 State-of-the-art

There are several approaches that have worked in the generation of VW designs from conceptual specifications. Bogdanovych [1][2] generated a 2D floor plan of a virtual institution from the conceptual model described in its performative structure. This approach used rectangular dualization of biconnected planar graphs. For this purpose OCoRD software was developed. This approach brings some challenges for the scaling of the sizes of the different rooms. It also does not let the virtual world designer freely create different designs for the institution. In this paper, we provide an alternative to this approach using shape grammars, allowing much more freedom in the VI design and generating many different functional designs with limited additional cost. We do not generate transitions as separate rooms (as in OCoRD) but we map all transitions to a hallway. In this way we can generate floor plans of buildings with much simpler navigation for the user. Our system contemplates not only a generation of the 2D layout but a complete 3D scene.

Our approach is similar to the one of Duarte, where shape grammars are used to generate Siza's Malagueira houses [4] using an online application that rendered such houses depending on user preferences. Duarte introduced the concept of *discursive* grammars that contain a shape grammar, a description grammar and a set of heuristics. We also contemplate semantic data to enrich pure shape grammars. Duarte generated 2D designs from user preferences and we generate virtual worlds from a formal specification taking into account the activities participants can engage on.

VRID (Virtual Reality Interface Design) [12] and VEDS (Virtual Environment Development Structure)[7] have been methodologies that have tried to facilitate the designer task either by dividing the design in high- and low-level phases or guiding him in taking design decisions to get an usable virtual environment. A conceptual model of a virtual environment was presented by Ossa [10], the model considered conceptual graphs and rule based systems that were really complex to be managed by designers. i4D was another methodology based on the representation of conceptual models but this methodology contributed with a thin abstraction layer taking into account only an small space of the domain knowledge [6]. Compared to these approaches, our system provides a high level abstraction layer by means of the virtual world grammar which enclosed both data and processes related to the 2D and 3D generation of designs.

VR-WISE system and Ontoworld tool have focused on the gap between the abstract model and the implementation prototype and have proposed an approach to generate VW from high-level descriptions given by ontologies [14][8]. Objects in the domain and their relationships have been described in a so called domain ontology. The domain ontology is converted into a representable domain ontology which describes how objects in the domain can be represented in the virtual environment. As a main difference with our approach, the domain ontology does not have all the information needed to generate the 3D virtual world whereas the VWG has it. Our approach generates the virtual world layout and situates 3D objects there and VR-WISE system situates objects in an already generated virtual scene.

4 Virtual World Grammar

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

4.1 Ontology

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

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

Definition 1 We define an *ontology* as a tuple $o = (C, \prec)$ where:

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

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

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

4.1.1 Auction House Ontology

From the specification of the Auction House institution, only scenes that define activities are used in the generation (see blue rounded rectangles in Figure 1). Hence, the specification concepts (C_{Spec}) just contain the scene (activity) class. Attributes of this class come from the virtual institution specification. For instance, attributes defining maximum number of participants of an activity. The specification elements for a concrete virtual institution are obtained by searching within the specification document.

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

- **Design wall** is the basic design element which forms higher level objects. It represents an actual separation wall between some virtual spaces. Every wall holds a basic set of geometrical properties such as position, length and more importantly a *wall type.* The wall type defines how the wall is rendered. used, such as wall texture, wall witdh and wall height.
- **Design space** represents an area that will be substituted by a functional or non-functional (see *office-layout* in Figure 8 a)) 3D model. It serves exclusively this issue. Design space holds information about the 3D model, such as path, texture or size and this information is used during the 3D transformation phase.
- **Design block** is a collection of walls and design spaces that creates one "shape" of our shape grammar. We can look at our grammar execution as a lego-like building process, where different blocks are spatially placed together to create the final design. This placement is validated using validation rules.

4.2 Shape Grammar

Shape grammar is a method of generating designs by using primitive *shapes* and the *rules* of interaction among them. One of the shapes is marked as the *starting shape.* Shape grammar rules are composed of left-side shapes and right-side shapes, where right-side shape replaces the left-side shape. Designs are generated from the shape grammar by starting with the initial shape and recursively applying its rules.

Definition 2 A *shape grammar*[11] (SG) is a 4-tuple: SG = (V_T, V_M, R, I) where

1. V_T is a finite set of terminal shapes. V_T^* is a set of shapes formed by the finite arrangement of an element or elements of V_T in which any element of V_T may be used multiple number of times with any scale, rotation or resize operation.
2. V_M is a set of shapes used as *markers*, such that $V_T^* \cap V_M = \emptyset$. Markers permit to control how rules are applied to the left-side shape. Rules with markers are called *labeled rules.*
3. R is a finite set of *rules*, that are ordered pairs (u, v) such that u is a left-side shape consisting of an element of V_T* possibly combined with an element of V_M and v is a right-side shape consisting of an:
   i element of V_T^* contained in u or
   ii element of V_T^* contained in u combined with an element of V_M or
   iii element of V_T^* contained in u combined with an additional element of V_T^* and an element of V_M
4. I is the *starting shape* consisting of elements of V_T^* and V_M.

A rule and the steps of a shape grammar derivation process are displayed in Figure 2. The shape grammar rule is marked with a black square. This rule simply adds a rotated copy of a rectangle to its origin.

Fig. 2: Shape grammar derivation process

4.2.1 Auction House Shape Grammar

We define two different shape grammars to present possibilities of our system. The first grammar, displayed in Figure 7 creates one institution building and for each activity it creates a room within this building. The second grammar, depicted by Figure 9, creates a separate building for each activity. Shapes of a shape grammar represent different blocks (rooms) of the building and placeholders (spaces) for the 3D models. Different activities from specification are associated with these spaces and they are automatically resized to fit to the number of activity participants. Auction House shape grammar uses two different rule types. The first one, an addition rule, positions rooms into different locations within the outline (init rule) and the second one, also an addition rule, distributes the rooms depending on the position of the previous one (distrib rule).

4.3 Validations

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

Definition 3 Given an ontology $o = (C, \prec)$, a set of basic operators Ω and a set of functions Φ, we define the *validation language* $\mathcal{L}_V$ as the language generated by the following grammar with starting symbol E:

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

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

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

Definition 4 *Validation term* $T_{\mathcal{L}_V}$ also called *validator* is a term created using validation language $T_{\mathcal{L}_V} \in \mathcal{L}_V$

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

4.3.1 Auction House Validations

We define a new validator *intersect*, that is executed after each execution step, and checks that (i) design blocks do not *intersect*, but they can *touch* and that (ii) blocks do not touch by walls marked as *outer* (O) or *outer entry* (OE) (this value comes from the *wall type* parameter).

4.4 Heuristics

Heuristics guide the process of world generation. They have two important roles. First, to decide in which order to process the elements from the specification. Second, how to find possible execution nodes in the execution tree for currently selected specification element. The generation process stores information in a tree structure where each node holds specific informations about the state of generation. This tree structure holds the execution states, which are defined either by a shape or a rule. If defined by a shape, it has as many children nodes as there exist rules with this shape on the left side. If defined by a rule, it holds the reference to actual shape and the rule to apply. Figure 3 shows an example of such a tree. Rectangles represent execution states defined by shapes, while ovals represent states defined by rules. The black nodes of this tree have been already expanded. We can see that rule-based nodes have 0 or 1 children, depending if they have been expanded or not. The child of a rule represents the right side shape of the rule.

Definition 5 *We* define heuristic next as a function $h_{next} : term_o^{C_{Spec}} \times 2^{term_o^{C_{Spec}}} \times 2^{term_o^{C_{Spec}}} \to term_o^{C_{Spec}}$, which for any $x \in term_o^{C_{Spec}}$, a set of already processed specification elements and a set of all specification elements returns element $y \in term_o^{C_{Spec}}$, which will be the next processed element. Function $h_{next}(nil, \emptyset, SE)$, returns an initial element.

Definition 6 *We* define heuristic exec as a function $h_{exec} : V_T \times tree_{exec} \to n_{tree}$, that given a shape $x \in V_T$

Fig. 3: An example of execution tree using tree-search protocol

and an execution tree $t \in tree_{exec}$ returns the next node to expand $y \in n_{tree}$.

Definition 7 *W*e define heuristics as a tuple $\mathcal{H} = (h_{next}, h_{exec})$ where h_{next} is a *heuristic next* function and h_{exec} is a *heuristic exec* function.

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

4.4.1 Auction House Heuristics

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

4.5 Virtual World Grammar

After all previous definitions we can now define a Virtual World Grammar. It includes an ontology specifying all the concepts and the definition of the specification instances of the concrete elements that have to be used to generate the virtual world. It also includes a shape grammar, that contains the different shapes and the rules used to generate the final design. Each specification element is mapped to a set of shapes that can represent it in the generated virtual world. During the generation process it is decided which one will represent the element in the generated design. Each terminal shape is associated to a class defining the properties of that shape. VWG also includes a set of heuristics that guide the generation process and validations that bring possibility to control and evaluate this process. At last, it includes function that for each validation term defines its execution time.

Definition 8 We define a *virtual world grammar* (VWG) as a tuple: $VWG = (o, SG, SE, f_{SE}, f_s, \mathcal{V}, f_t, \mathcal{H})$ where

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

5 Virtual World Builder Toolkit

The Virtual World Builder Toolkit (VWBT) provides visual interfaces and mechanisms to define and execute virtual world grammars. The toolkit loads the specification of a multi-agent system and combines it with informations stored in the *Virtual World Grammar* to produce the final output. Furthermore, its graphical user interface provides a friendly way to define all parts of VWG. It is integrated in our Shape Grammar Interpreter (SGI) [13], Figure 4 shows the interface of SGI. An intermediate output of the generation process is a 2D draft of the virtual world (floor plan). Using a 3D transformation engine (jMonkeyEngine), this draft is later transformed into a 3D model. The tool allows to implement different renderers that export the 3D model into different virtual worlds (e.g. Second Life, Project Wonderland). Furthermore this solution allows to:

– dynamically react to changes in the specification and simply regenerate the adapted virtual world
– separate artistic (graphical) design of the institution from the functional implementation
– make generation process transparent to institution designer and 3D virtual world designer
– browse design space and easily explore possible designs

Algorithm 1: Virtual World Builder Algorithm

```
Input: Specification, Virtual World Grammar
Output: 2D draft (floor plan) of the virtual world
begin
    // initialize variables
    $a \leftarrow nil$; $\mathsf{specElems} \leftarrow \emptyset$; $t_{exec} \leftarrow initTree()$; $n \leftarrow \emptyset$
    while $(size(\mathsf{specElems})\ != size(\mathsf{SE}))$ do
        // get element from specification
        $a \leftarrow h_{next}(a, \mathsf{specElems}, \mathsf{SE})$
        // put this element in control set specElems
        $\mathsf{specElems} \leftarrow a$
        // search for valid design
        $valid \leftarrow false$
        // find associated shapes
        $S \leftarrow f_{SE}(a)$
        foreach $(s \in S)$ do
            while $(not(valid) \vee n = \emptyset\ )$ do
                // find unexecuted node in the exec. tree
                $n \leftarrow h_{exec}(s, t_{exec})$
                // execute rule and store right shape
                $c \leftarrow Execute(n, t_{exec})$
                // validate
                $valid \leftarrow$ Validate (c)
            if $valid$ then break
        if $valid$ then $appendChild(t_{exec}, n, c)$
        else return $\emptyset$
    return $t_{exec}$
end
```

5.1 Design Generation Process

Algorithm 1 is used to generate a 2D floor plan and it summarizes the use of all defined parts of VWG. The algorithm first initializes variables. Function *initTree* initializes an execution tree by inserting a starting shape as the root node. Then, using function h_{next} searches for the next specification element to process, assigning it to variable a, and adds it into the list of executed elements $SpecElems$. Using function f_{SE} it finds the set of shapes that can be used to represent this element. It loops over this whole set till it finds a valid design. In this loop it searches for the execution tree node using heuristic function h_{exec} and executes it by calling function *Execute* creating new shape. It validates the result of execution. If the result is valid, it proceeds to the next iteration. The process finishes when it has processed all nodes from the SE (SpecificationElements). The process fails and returns nothing if no valid design was found.

Fig. 4: SGI interface with WVBT extensions

5.2 Workflow for definition and execution of VWG

Virtual World Builder Toolkit brings many creative possibilities into virtual world design process. Designers may explore many different designs based on a shape grammar. Shape grammar elements (SGE) serve as a visual style sheet for a generation process. Trying different values for parameters, or even having prepared multiple sets of instances brings possibilities of *theming* or *skinning* of virtual worlds. Figure 5 describes the workflow process for the definition and execution of a virtual world grammar. Depending on the results of *draft* or *final* generation we can readapt the grammar.

Grammar designer can either browse possible designs or modify existing parts of the shape grammar to obtain satisfiable results. The workflow is divided into three main parts. First, in the *preliminary definition*, he defines the ontology and the shape grammar. Second, in the *instance definition*, he loads the specification, creates and defines all specification and shape grammar elements and specifies mappings between them. Then validations and heuristics are introduced. Finally, in the *execution* part, he browses random designs and modifies instance parameters to produce the 2D draft and at last, transforms this draft to 3D.

Fig. 5: Workflow for definition and execution of VWG

6 Results

In this section we present different results of the generation and we measure generation performance. As an input we take the auction house virtual institution and we vary the number and size of the auction rooms. We also define two different shape grammars. We use simplified display of rules presented in Figure 6 where left-side of the rule is shown in black and right-side in red.

First shape grammar, depicted in Figure 7, generates an institution building and positions all rooms inside this building. The initial shape of this grammar is the outline shape and then it distributes the rooms within this outline. A floor plan and a 3D render for the

(a) Non-simplified (b) Simplified

Fig. 6: Rule display simplification

(a) Shapes (b) Rules

Fig. 7: Shape grammar 1 for the Auction House institution

(a) Floor plan (b) 3D render

Fig. 8: An output of the Virtual World Grammar using shape grammar 1

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

Fig. 9: An excerpt from the shape grammar 2 for the Auction House institution

five auction rooms (we have selected five rooms for a demo example of a small institution) and three remaining rooms (Admission, Item Register, Auction Info) is displayed in Figure 8. This output was produced using three shapes (outline, rectangle room and iso-room) and four rules (two rules place the rooms within this outline and two rules distribute the rooms within this outline). The drawback of this grammar is that it is very simple and the design space it can generate is rather small.

The second shape grammar for the auction house institution is not limited by the initial outline and it generates large design spaces. Figure 9 displays an excerpt from this grammar. We can see four shapes that represent three possible room designs and an initial shape. In the right part we see examples of rules which place rooms according to the position of the previous shape. Rooms in this grammar are generated as standalone buildings. Figure 10 shows two generated floor plans and the corresponding 3D models for five auction rooms. The small rectangles and the rectangles within the shape grammar shapes represent the placeholders (office-layout in Figure 8) for the 3D models that will be substituted during the 3D transformation phase.

Figure 11 shows a graph of the measurements for a given amount of activities. We have scaled the institution up to 30 scenes and in these scenes we have used some complex models to measure the possibilities of the jMonkeyEngine. The generation of the floor plan for a large institution was under one second. The 3D render grew from two seconds for five rooms to 30 seconds for 25 rooms. The reason for increased time is the use of complex models, such as trees, which in total made more that 1.4 million of faces for 30 rooms.

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

Fig. 11: Performance measurements of VWG

7 Conclusions

We have presented a virtual world grammar for the automatic generation of 3D virtual worlds in which inhabitants can be both humans and agents. The VWG holds semantic information about a multiagent system specification, describing activities and relationship between them, a shape grammar, introducing design elements and their characteristics, and a list of validations and heuristics guiding the generation process. The virtual world generation is done in two steps, a first one in which the output is a 2D floor plan, and a second one which generates a 3D representation of the virtual world.

Contributions of our research are i) the introduction of the virtual world grammar concept and its components, ii) the algorithm which defines how to navigate

Fig. 10: Two different outputs of the Virtual World Grammar using shape grammar 2

between the specification and the shape grammar execution tree using heuristics and validations and iii) the Virtual World Builder Toollkit that provides a user-friendly interface allowing a comfortable definition and execution of virtual world grammars.

An important feature of the VWG workflow is that the user can explore many different designs or modify existing parts of the shape grammar to explore new designs. We have demonstrated the VWG applicability in the generation of a 3D visualization of a virtual institution. The definition of the virtual world grammar can be applied generally for any multi-agent system where it has meaning to visualize its activities in a 3D virtual world. Current approach allows to map one activity per one space. Mapping more activities to one space brings challenges to their execution as it is difficult to control the concurrent execution or simply identify which of these action needs to be executed upon arrival to this virtual space.

Until now our main efforts have been concentrated in the design generation step, our next endeavour is to focus on issues happening at run time such as users/agents enrollment and the dynamic update of the hybrid system. We will also study the integration of our previous work on 3D objects' behaviour in virtual environments [9][3]. We also plan to apply our methodology in computer games domain, namely in MMORPG, where electronic institutions control the norm enforcement and VWG takes care of the visualization process.

Acknowledgements: This work is partially funded by EVE (TIN2009-14702-C02-01 / TIN2009-14702-C02-02) and AT (CONSOLIDER CSD2007-0022) projects, EU-FEDER funds, the Catalan Gov. (Grant 2005-SGR-00093) and Marc Esteva's Ramon y Cajal contract.

References

1. M. Ancona, A. Bogdanovytch, S. Drago, and G. Quercini. Rectangular dualization of biconnected plane graphs in linear time and related applications. *VIII Congress of Simai (Società Italiana di Matematica Applicata e Industriale)*, 2006.
2. A. Bogdanovych. *Virtual Institutions.* PhD thesis, University of Technology, Sydney, Australia, 2007.
3. D.Brota, I. Rodriguez, A. Puig, and M. Esteva. A generic framework to exploit virtual worlds as normative and dynamic interactive spaces. In *Computer Graphics and Virtual Reality*, pages 151–157, 2009.
4. J. P. Duarte. *Customizing mass housing : A discursive grammar for Siza's Malagueira houses.* PhD thesis, Cambridge (MA): Massachusetts Institute of Technology, 2001.
5. M. Esteva, B. Rosell, J. A. Rodrguez-Aguilar, and J. L. Arcos. Ameli: An agent-based middleware for electronic institutions. *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 1:236–243, 2004.
6. C. Geiger, V. Paelke, C. Reimann, and W. Rosenbach. A framework for the structured design of vr/ar content. In *VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 75–82, New York, NY, USA, 2000. ACM.
7. W. J.R., E. R.M, and D. M. Structured development of virtual environments. In *Handbook of Virtual Environments: Design, implementation and applications*, pages 353–378. K. Stanney, 2002.
8. H. Mansouri, F. Kleinermann, and O. De Troyer. Detecting inconsistencies in the design of virtual environments over the web using domain specific rules. In *Web3D '09: Proceedings of the 14th International Conference on 3D Web Technology*, pages 101–109, New York, NY, USA, 2009. ACM.
9. I. Rodriguez, A. Puig, M. Esteva, C. Sierra, A. Bogdanovych, and S. Simoff. Intelligent objects to facilitate human participation in virtual institutions. In *Web Intelligence*, pages 196–199, 2008.
10. F. Southey and J. G. Linders. Ossa - a conceptual modelling system for virtual realities. In *ICCS '01: Proceedings of the 9th International Conference on Conceptual Structures*, pages 333–345, London, UK, 2001. Springer-Verlag.
11. G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. In C. V. Friedman, editor, *Information Processing '71*, pages 1460–1465, Amsterdam, 1972.
12. V. Tanriverdi and R. J. K. Jacob. Vrid: A design model and methodology for developing virtual reality interfaces. In *Proc. ACM VRST 2001 Symposium on Virtual Reality Software and Technology, ACM*, pages 175–182. Press, 2001.
13. T. Trescak, I. Rodriguez, and M. Esteva. General shape grammar interpreter for intelligent designs generations. *CGIV'09*, 2009.
14. O. D. Troyer, W. Bille, R. Romero, and P. Stuer. On generating virtual worlds from domain ontologies. In *MMM*, pages 279–294, 2003.