Domain Ontology-based Management of Virtual Scenes

CRENGUTA BOGDAN  DORIN MIRCEA POPOVICI
Faculty of Mathematics and Computer Science
Ovidius University
124 Mamaia Blvd., Constanta, 900527
ROMANIA
cbogdan@univ-ovidius.ro  dmpopovici@univ-ovidius.ro  www.cerva.ro

Abstract: - In recent years, some people have tried to incorporate semantics into virtual scenes in order to create virtual environments close to the real world as much as possible. Our purpose in writing this paper is to present the OntSceneBuilder object-oriented software system that uses any domain ontology for visualization and management of the virtual scenes. The ontology provides a precise specification of the concepts and their relations of a domain. Each concept is associated with 2D and 3D resources and a virtual artifact. The graph of the virtual artifacts forms a virtual scene of a virtual exposition. This paper therefore presents some models of the system development process, mainly realized during the analysis and design activities. Our aim was to analyze the OntSceneBuilder from the functional and interactional viewpoints and to create its software use case diagram. Furthermore, each software use case was designed from the structural and dynamical viewpoints. At the same time we also constructed the system software architecture. Some classes of the software architecture manage the concepts of the domain ontology associated with the topic chosen by user. The system functioning has been tested to construct virtual scenes of a virtual historic exposition.

Key-Words: - Virtual scene, domain ontology, concept, virtual artifact, software analysis, software architecture, virtual historic exposition

1 Introduction
The present study introduces the OntSceneBuilder software system, the models obtained during its developing process, and it concentrates on the analysis and the design of the system. OntSceneBuilder is an intensive object-oriented software system that uses a domain ontology to construct virtual scenes with the help of the user. For users, the benefits of using a domain ontology are at least three: accuracy, ease of content reuse and management of virtual scenes. The accuracy is ensured by default, since a domain ontology is a precise and formal specification of the knowledge of the domain. The specification contains the concepts and their relations which the user can use to construct the virtual scenes and manage their content. The ontology also acts as a compulsion factor in the process of the scenes content management. For example, the user cannot choose the Color concept if he/she did not add to the scene the concept by which the first concept depends on.

OntSceneBuilder can be used with any domain ontology. We tested the system functioning using the ontology of the Roman artifacts found in the Tomis fortress [1]. The ontology is based on the top-level ontology DOLCE [2] and its modules, such as D&S [3], Temporal Relations and so on.

The main objective was the creation of the virtual scenes of a virtual exposition of the Tomis fortress. The virtual exposition shows virtual artifacts of the Roman exhibits found at the National History and Archaeology Museum of Constanta, Romania. The exhibits have been ontologically classified in categories such as objects, constructions, decorative elements, clothing parts and their accessories, activities, and so on.

And least but not last, OntSceneBuilder permits the virtual reality (VR) expert to easily manage content of a virtual scene using interactive graphical interfaces. The user can also move the virtual artifacts of a scene through drag-and-drop actions. Therefore, the VR expert can construct his/her scenes without having to be an ontology or domain expert.

The paper is organized as follows: sections 2 and 3 present some of the models created during the analysis and design of the OntSceneBuilder system. The models have been created using a standard language called the Unified Modelling Language (UML). This language allows the creation either of the graphic models of the information or of the software system [4]. Section 4 focuses on the implementation and testing aspects. Section 5 covers related work. Finally, section 6 concludes and presents our future work plans.

2 Software Analysis
The objective of the OntSceneBuilder is to provide an easy-to-use content management tool that allows users to add, edit, view and save virtual scenes.

2.1 Software actors
A software actor is a role "played" by one or more individuals (person, team, or organization) or even another software application in its interactions with our system. The role is characterized by a set of properties and actions which each individual in this role can exhibit or "play" in the given context [5].

In the case of the OntSceneBuilder application, we identified two software actors: user and time.

The user interacts with the system for that the later to provide the following functionalities: a) it creates a scene based on the concepts of the domain ontology used by the system; b) it modifies a saved scene, by changing the artifacts properties, or the concepts resources used or deleting virtual artifacts from scene. Modification of the topological properties of the virtual artifacts can also be made from the 3D environment. By default, when a user asks for removing of a virtual artifact, the associated resource and concept are removed by the system from the used lists and the virtual artifact from the scene. However, the system does not delete on the disk the resource associated to the virtual artifact and the concept from the domain ontology by which belongs to; c) it shows and saves an opened scene.

Another software actor of our application is time, since at each five minutes the system saves the current scene.

2.2 Software use case diagram
The software use case diagram belongs to the functional model created during the object-oriented analysis activity of a software system. Besides the software use case diagram, a functional model contains the application requirements, the description of the software use cases and their activity diagrams [5].

The software use case diagram is formed by the software actors and uses cases and their relations. For example, the OntSceneBuilder application has the software use case diagram presented in Fig. 1.

3 Software Design
The object-oriented analysis has focused on learning to "do the right thing": that is, the understanding some of the goals for the OntSceneBuilder application, and related rules and constraints. By contrast, the design work will stress "do the thing right" [5]; that is, skillfully designing a solution to satisfy the system objective. The

Fig. 1: The use case diagram of the OntSceneBuilder system

heart of this solution is creation of the system software architecture, which contains the design classes and their relations.

3.1 Software architecture
The software architecture can be designed in an incrementally manner following the software use cases and applying an architectural pattern. In the case of software architecture creation for our application, we used the Model-View-Controller (MVC) architectural pattern [6].

According to this pattern, the objects are classified in three categories: models, views, and controllers. The classification criterion is given by the responsibilities of the objects from each category. The view objects are objects with which user stakeholders interact directly, such as frames, forms, panels, and so on.

The model objects eventually contain persistent information managed by system. Many such objects come from the business objects of the domain model of the information system where the software system will operate. In our case, the domain model was substituted by the domain ontology used. This is why, we introduced the Concept and Ontology classes into the software architecture.

Another kind of model objects is constituted by the composite objects. These contain model objects created
at the beginning or during the system execution. The idea is that if there are many objects of the same class and they have to be used during the system execution, we temporarily put them in a composite object that is linked using aggregation by the object parts. In addition, a composite object has the responsibilities to create and change the states of objects it contains. In this way, we also fulfill the Creator pattern [7]. Furthermore, the controller objects might send requests to the composite object that solves them. These objects are useful especially when they have to be unique during the system execution. In these cases, we apply the Singleton pattern [6] to the composite objects. For instance, the ConceptsList class contains a collection of objects of the Concept class. We observe in Fig. 2 the ConceptsList class fulfills the Singleton pattern containing three elements:
- a single and private constructor (the operation create());
- a private and static variable named instance of the type ConceptsList, i.e. the class which belongs, and
- a public and static operation called getInstance() that returns (using the variable instance) a reference to the unique object of the ConceptsList class.

There are also other classes that fulfill the Singleton pattern such as Ontology and TopicsList (Fig. 2).

The last category of the model objects is constituted by manager objects. These objects have the responsibility to manage the operations with the ontology or files. The manager objects deal with loading and saving the individuals from/in ontology.

Finally, the controller objects have the responsibility to manage the logical flow and the events produced by users in their interactions with view objects. For instance, the part of the software architecture that contains the design classes used in the “Create scene” use case execution is shown in Fig. 2.

In order to obtain a quality and modular software architecture we applied the design principles of low coupling, high cohesion, and assignment of responsibilities. These principles are fulfilled if we use the general responsibilities assignment patterns (shortly, GRASP) like Information Expert, Creator, Low Coupling, High Cohesion, Controller, and Polymorphism [7].

The behavioral aspect of the software architecture is shown by sequence diagrams. They present the interactions between the user and system in terms of objects and messages sent between them.

Most of the messages are synchronous, but also there are asynchronous messages, such as some messages of the sequence diagram of the “Automatically save scene” use case.

Moreover, OntSceneBuilder integrates a distributed virtual reality toolkit called the ARéViJava [8] (Fig. 3(b)). For example, the ARéViJava and ControlPanel classes of Fig. 2 are part of the ARéViJava toolkit. The first class is a graphical user interface that contains a 3D canvas used to render the virtual scene. The second one permits users to load inside the virtual scene objects and dynamically attach to them specific behaviors.

In essence, ARéViJava is a Java3D based open-source API, used for rendering of the dynamic scene, that is adaptive to different configurations, ranging from desktops to 3D stereoscopic immersion systems and integrating a wide sort of interaction devices from mouse and joystick to space-mouse and WiiMote. It implements a reactive agent-based architecture that permits us to build immersive and interactive virtual spaces as were defined in [9].

4 Implementation

For the implementation of the OntSceneBuilder system, we used the Java programming language (Java Developing Kit 1.6 version).

The ontology was written using the languages Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL) with the assistance of the Protégé tool [10] (version 3.4).

To work with our ontology, we used the Protégé-OWL API, which is an open-source Java library for OWL and RDF [11]. For example, the concepts tree of Fig. 3(a) shows a part of the ontology taxonomy that contains the concepts that belong to the domain ontology, but do not to the imported ontologies, such as DOLCE and D&S. Using the Protégé-OWL API we read all the RDFS/OWL classes of the ontology and filtered them taking only the concepts and their superclasses by which we need them.

Moreover, due to the ARéViJava framework facilities, once the virtual scene is created, the user may adopt different navigation and/or interaction metaphors (e.g. hover, drive, fly, orbit, spacemouse in Fig. 3(b)) in order to obtain better views of the virtual environment and to adjust its topology by means of direct 3D interaction on the virtual artifacts.

The system functioning has been tested to construct virtual scenes of a virtual historic exposition. In this case, the museum custodian “plays” the role of user in our system. With the assistance of the system, the custodian creates his/her own virtual scenes of a historic exposition based on a domain ontology and 2D/3D files incorporated in system (Fig. 3(c)).

5 Related Work

The Intelligent 3D Visualization Platform - I3DVP [12] is a framework for the enhancing virtual scenes with
semantic information and for performing reasoning by inference on content and the semantics of the scenes. The framework uses two kinds of OWL ontologies: a graphic ontology (called OntologyX3D) which describes graphics and virtual reality concepts of 3D models and their animation and is based on VRML and X3D standard; and domain ontologies. The used domain ontology is mapped to OntologyX3D through 15 relationships described by the classes of another intermediate ontology. The virtual scenes created by I3DVP are organized as graphs of OWL individuals (i.e. instances of OWL classes) of the OntologyX3D. Instead, in our system, a scene is organized as a graph of virtual artifacts, where each virtual artifact is associated with a 3D resource which in turn is related to a concept from the domain ontology used. Moreover, I3DVP permits users to construct and manipulate virtual scene through sets of rules that allow inference-based decision making [12].

The HANNAH framework [13] is based on an ontology converted to a database to generate visual graph representations of 3D layouts. The OntSceneBuilder does not use a database, but files to store ontologies, the topics list, and so on. The MUG application [14] allows users to author the structural, behavioral and functional knowledge about a design in a 3D virtual environment of CAD system. Knowledge is described by ontologies written in DAML language [15]. The ontologies are used for synchronous communication and interaction of designers, annotation and saving designs in DAML files. The application is useful for the collaborative creation of conceptual designs for devices.

6 Conclusion and Future Work
This paper presented the OntSceneBuilder system and a part of its development process. The system uses a domain ontology to construct the virtual scenes of a virtual exposition. Therefore, the system software architecture contains classes which manage the domain ontology chosen by a user. The quality of the software architecture is ensured by the MVC architectural pattern and the design patterns applied: Information Expert, Creator, Low Coupling, High Cohesion and so on.

Our work to improve the current OntSceneBuilder system will include reasoning for modifying of the scenes. For the time being, OntSceneBuilder allows users to modify values of the virtual artifacts properties and the resources of the related concepts.

Further work will also consider the ontological description of the processes, activities, and agent states that will be incorporated into the virtual scenes as ARéViJava behaviour objects.

Acknowledgements
This work is partially funded within the "Using virtual reality in 3D Multi-modal reconstruction of historical sites" (TOMIS) project, no: 11-041/2007, by the National Centre of Programs Management, PNCDI-2 – Partnerships program.

References:
[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley Professional, 1994
[11] Protégé-OWL API, link:


Fig. 2: The design class diagram of the “Create scene” software use case
(a) The choice of concepts from the ontology taxonomy and displaying their 2D resources

(b) The Navigation tabbed pane of the ControlPanel frame of the ARéViJava toolkit

(c) Displaying of a virtual scene

Fig. 3: OntSceneBuilder system execution