AN ARCHITECTURAL PATTERN FOR X3D-BASED VIRTUAL ENVIRONMENTS
An Object-oriented Approach

George Anastassakis and Themis Panayiotopoulos
Knowledge Engineering Lab, Department of Informatics, University of Piraeus, Piraeus, Greece

Keywords: X3D, Virtual Environment.

Abstract: X3D is an open, royalty-free, scene-graph-based standard for 3D virtual worlds that has been used by virtual environment developers many times until today. As a virtual environment development tool, it has numerous appealing features; however, it also has certain characteristics that may, under certain circumstances, create significant problems to developers. With a motivation to communicate our own experience to, and contribute to the efforts of, other researchers who are using or consider using it, we evaluate X3D in a virtual environment developer's perspective and identify potential problems with its usage. Also, we present an architectural pattern aimed at addressing those problems effectively, with a focus on transparency, standards-compliance, reusability and extendibility. In conclusion, we present a case study of the application of the proposed pattern in a fully-implemented intelligent virtual environment system.

1 INTRODUCTION

Today's virtual worlds are typically three-dimensional and consist of virtual objects of various kinds, each with its own geometry, structure, location and orientation in space, presentation, functionality and behaviour. Host virtual environment systems render virtual worlds in real time and enable users to experience and interact with them multimodally, while autonomous behaviour exhibited by synthetic actors – *virtual agents* – is an essential element of applications aimed at believability, realism and an increased sense of user presence. To provide such functionality, virtual environments systems employ world representation technologies which, in most (if not all) cases, are based upon the concept of the *scene graph*.

As its name implies, a scene graph is a graph structured so as to represent a scene which, in most cases, is three-dimensional, dynamic and interactive. A typical scene graph contains nodes of various types, each with a specific purpose, such as to define geometry and apply transformations. Scene graphs are the preferred method for the representation of complex scenes mostly because they (a) can naturally capture the conceptual hierarchy of the represented scene, and (b) are highly independent, in principle, of the underlying hardware and software implementation.

X3D is a scene-graph specification for virtual worlds (Web3D Consortium, 2011). It is the successor of *VRML* version 2.0, also known as *VRML97*. It is an ISO standard currently steered by the Web3D Consortium.

X3D has several features that make it an appealing candidate for employment in virtual environment systems. It is for those features that we have adopted it for our own research which has yielded, among other results, a virtual environment system titled *REVE Worlds* (Anastassakis, 2011; REVEnet, 2011) that uses X3D both as a format for external source data and to maintain an internal virtual world representation at run-time.

Nonetheless, X3D provides no original facilities to accomodate a range of needs inherent to virtual environments. In addition, certain parts of the X3D specification, while facilitating the processing of the kind of online content X3D was originally meant to support, may lead to problems when fine-grained and accurately-timed control over internal virtual world representation data is needed. The latter is no less than a non-negotiable requirement especially when it comes to *intelligent virtual environments* – that is, virtual environments in which the element of

Anastassakis G. and Panayiotopoulos T..
AN ARCHITECTURAL PATTERN FOR X3D-BASED VIRTUAL ENVIRONMENTS - An Object-oriented Approach.
DOI: 10.5220/0003863404660471
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2012), pages 466-471
ISBN: 978-989-8565-02-0

intelligent behaviour exhibited by *intelligent virtual agents* is dominant – where multiple actors, both artificial and human, simultaneously exhibit complex behaviour (Aylett and Luck, 1999).

In this paper, we evaluate X3D from a virtual environment developer's standpoint as a component of virtual environment systems, and identify what we consider to be its key benefits and drawbacks as such. In addition, we propose an architectural pattern aimed at effectively addressing those drawbacks with a focus on transparency, standards-compliance, reusability and extendibility. We also show how we have applied the proposed pattern in practice.

2 RELATED WORK

The potential of X3D – either in its contemporary form or in the earlier form of VRML97 – as a tool for developing virtual environments has been investigated and exploited on numerous occasions and in various domains until today, for the purposes of both research and applications.

Cabral et al. (2007) discuss their use of X3D as a means to define virtual worlds for virtual heritage applications, as well as their implementation of a X3D browser based on the Ogre3D graphics engine. They conclude that X3D has helped them improve the design review process, save time and enable heritage specialists with no 3D graphics experience to benefit from virtual reality technologies.

Bouras, Panagopoulos and Tsiatsos (2005) discuss how an existing VRML-based research platform for networked virtual environments titled "EVE" was modified using the Xj3D toolkit to rely on X3D. The authors also offer useful information about problems they faced during their effort. An application in the form of a tool for collaborative design and spatial arrangement of multi-grade virtual classrooms is presented by Bouras, Tegos, Triglianos and Tsiatsos (2007).

In the context of annotation-based approaches for the definition of virtual world semantics, the use of VRML's WorldInfo and PROTO nodes as well as GeoVRML's GeoMetadata nodes is discussed by Ibanez-Martinez and Delgado-Mata (2006).

Behr, Dähne and Roth (2004) discuss techniques for using X3D in virtual reality applications with a specific focus on user immersion, and propose extensions to the X3D specification to that end.

Ieronutti and Chittaro (2007) present an architecture for virtual humans in educational virtual environments based on Web3D technologies.

Along similar lines, a Web-based teaching software system relying on VRML is presented by Ong and Mannan (2004). The authors examine – to great technical detail, which is rather an exception to the rule – the use of VRML's *external authoring interface* (*EAI*) (that is, an interface for communication between VRML scenes and external applications and a predecessor to a similar interface specified by X3D) to enable control over the VRML scene by external processes.

In a wide spectrum of virtual environment systems and applications such as those mentioned above, X3D plays a central role, not only as a world representation means but also as a facilitator of diverse interactions among virtual objects, virtual agents and user avatars, and a component deeply involved with virtual world dynamics in general. Nonetheless, in spite of the numerous and highly-technical implications in incorporating X3D into complex software, very little detail is available – as a general rule – on how those implications were dealt-with, what kinds of obstacles were encountered and how they were overcome in each case.

3 X3D IN A VIRTUAL ENVIRONMENT DEVELOPER'S PERSPECTIVE

Because virtual environments are complex systems involving multiple interoperating components, the following presentation of X3D and all further discussion in this paper is based on X3D's *external scene access interface* (*external SAI*), an interface for interaction between external processes and X3D worlds (as opposed to the *internal scene access interface*, which can be used to interact with X3D worlds from within).

3.1 Desirable Features

X3D can benefit virtual environment developers on several levels: it is an abstract specification enabling virtual world definitions largely disentangled from technical details and the underlying implementation; it can be used both for the purposes of run-time world representation and as a format for 3D scene source data; it specifies a number of powerful application programming interfaces (APIs) for the interaction between application code and X3D scene graphs; in contrast to many de facto standards, it is an ISO standard; it is an open and highly accessible standard; it is backed by a large and active online community.

3.2 Potentially Problematic Features

X3D has specific features that may create non-negligible obstacles under certain circumstances. Based on our own relevant experience, we feel that virtual environment system developers must be aware of those characteristics and the problems they may create, and have thoroughly considered them before investing in X3D.

3.2.1 Asynchronous Execution Model

According to the X3D specification, a scene graph can be modified by application code on an exclusively asynchronous basis. In other words, scene graph modification requests are not processed when issued by application code but, as the specification clearly dictates, when the *browser*, that is, the component whose responsibility is to render subsequent instances of the scene graph, decides to do so. In cases where application code must make multiple changes to a scene graph in a single burst, modification requests can be grouped and processed as a single *event cascade*, again when the browser decides to do so.

This is a useful specification-level feature that results in increased browser performance and safe concurrent access to the same scene graph by different threads, among other benefits. However, it raises an issue when scene graph contents must be accessed immediately, or very shortly, after requests for modification are issued. Consider the following example of readily-compilable Java code meant to write a certain value to a certain field of some node in a X3D scene graph:

```
float before = initial;
((SFFloat) foo).setValue(before);
float after = ((SFFloat) foo).getValue();
boolean test = before == after;
```

In the above code snippet, a call to *foo* field's *setValue* method – a request to modify the field's value – with the value *initial* (which shall not be equal to the field's value at the time) is immediately followed by a call to the field's *getValue* method – a request to access the field's value. Because the time difference between the two calls is extremely small, the browser will have most likely not processed all pending event cascades when the *getValue* method is called. As a result, the value that will be read will not be equal to the value just written, and the *test* variable will have a value of *false*.

Where similar patterns of rapidly successive scene graph access and modification operations are inevitable (as it is the case with virtual environments due to multiple virtual agents interacting with the same virtual world at the same time), X3D's asynchronous execution model may become a considerable obstacle. One way to circumvent this is to register an interest for notifications of fully-realized modifications to specific fields and only access those fields when corresponding notifications are received, something that is indeed supported by X3D; however, when the ability to strictly order and schedule complex interactions with the scene graph is of the essence, such an approach is inapplicable.

3.2.2 Inaccessible Geometry Dimensions

The X3D specification provides that several fields of primitive geometry nodes, such as *Box*, *Sphere* and *Cone*, are only accessible as *initializeOnly*, which essentially means that their values cannot be read at run-time. This is a major drawback as it renders bounding-shape calculation by application code in an automated fashion virtually impossible, while bounding shape information is essential to several aspects of a virtual environment system's operation, such as collision detection and response, visual annotation of virtual objects, automated arrangement of virtual objects in space, and many more.

A possible solution is to supply bounding-shape information calculated at design-time in the *metadata* field which all X3D nodes have; however, such an approach requires additional pre-processing and, most importantly, only applies to non-deformable virtual objects (which virtual bodies and user avatars rarely are, if ever).

Luckily, it is possible to access the coordinates of vertices in *IndexedFaceSet* and *IndexedLineSet* nodes: their *coord* field as well as the *Coordinate* node's *point* field are both accessible as *inputOutput*. This enables calculation of local geometric extents, which, in turn, enables calculation of bounding-shapes based on a number of methods.

3.2.3 Lack of Per-frame Processing Support

The term *per-frame processing* (or *per-frame behaviour*) is used to refer to functionality engaged on every frame rendered. Support for per-frame processing is crucial to any virtual environment system: it is required by mechanisms such as those enforcing laws like gravity and friction, and performing collision detection and response. X3D only supports per-frame processing as part of its internal SAI by means of *X3DPerFrameObserverScript*, a dedicated data type that can serve as a basis for scripts capable of responding to frame events. Unfortunately, the external SAI has no inherent

support for per-frame processing. Hence, developers are forced to apply custom solutions which, albeit often highly effective, may compromise transparency, reusability and implementation life-span.

4 ARCHITECTURAL PATTERN

To tackle issues such as those discussed in Section 3.2 with transparency, standards-compliance, reusability and extendibility in mind, we propose the design pattern shown on Figure 1.

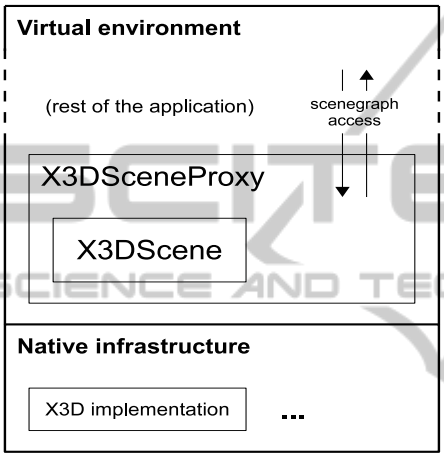

Figure 1: Architectural overview of a virtual environment system using X3D through X3DSceneProxy.

The figure shows a virtual environment system maintaining an internal scene graph as a *X3DScene* object. (In this text, *X3DScene* refers to the concrete implementation of the *SAIScene* data type provided by the X3D implementation used, while *SAIScene* is the corresponding abstract X3D data type).

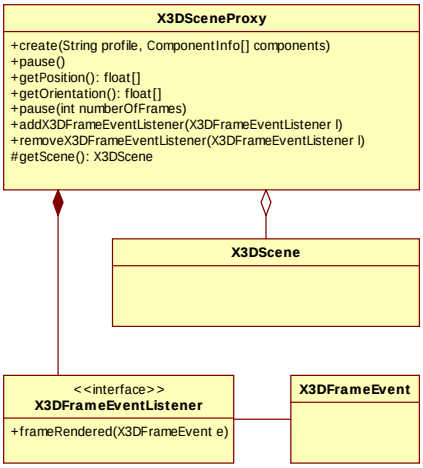

Figure 2: X3DSceneProxy and related classes.

However, the *X3DScene* object is not used directly but through an object of the *X3DSceneProxy* class – a *scene proxy*. As its name implies, *X3DSceneProxy* is a proxy for *X3DScene* (Gamma, Helm, Johnson and Vlissides, 1995). Meaning to mediate all interaction between application code and X3D scene graphs, each *X3DSceneProxy* object fully encapsulates and manages a *X3DScene* object, only allowing access to it according to strict specifications. Figure 2 depicts *X3DSceneProxy* and all related classes (only critical members are shown).

In the following text, aspects of the functionality that can be made available thanks to the proposed pattern are discussed along with implementation guidelines. Please note that the latter are but possible methods to address the issues discussed in Section 3.2; other mechanisms can be implemented just as well without any compromise on transparency and reusability thanks to the proposed pattern.

4.1 Initialization

The scene proxy handles creation and initial configuration of the internally-managed *X3DScene* object. More specifically, it creates a *TimeSensor* and a *ProximitySensor* node. The *TimeSensor* node is set to loop forever, while the *ProximitySensor* node's size field is set to a sufficiently large value, which implies that it is to monitor the entire scene for events. In addition, a field event listener is added to the *fractionChanged* field of the *TimeSensor* node, which means that, as per the X3D specification, the listener's *readableFieldChanged* method – referred to as the *frame event handler* in the following text – will be called on every frame.

4.2 Viewpoint Tracking

The scene proxy enables access to the currently-bound viewpoint's position and orientation through its *getPosition* and *getOrientation* methods based on information provided by the *ProximitySensor* node discussed in Section 4.1. This is provided as a convenience to applications that need transparent run-time access to that kind of information, for instance, to respond when user avatars move to specific locations in the virtual world.

4.3 Application Code Execution Delay

The scene proxy can be used to delay execution of application code for a specific number of frames by means of its *pause* methods which use internal counters to monitor the number of frames rendered

since their invocation and block execution until a specified count is reached. This particular bit of functionality can be used to effectively overcome obstacles created as a result of X3D's asynchronous execution model, thus enabling synchronized interaction between application code and the X3D scene graph. For instance, the example in Section 3.2.1 can be modified as follows:

```
float before = initial;
((SFFloat) foo).setValue(before);
scene.pause();
float after = ((SFFloat) foo).getValue();
boolean test = before == after;
```

In the above code snippet, the variable *scene* is a reference to the scene proxy. Thanks to a call to the *pause* method, the above code will wait one frame before reading the value of the *foo* field. This means that, as per the X3D specification, pending event cascades will have (most probably) been processed and values will have been written to output fields when the field is subsequently read, resulting to a *test* variable value of true, in contrast with the case in Section 3.2.1.

4.4 Per-frame Processing

The scene proxy can be used to apply per-frame processing. To that end, it maintains an internal list of frame event listener references, that is, references to objects of class *X3DFrameEventListener* that have registered an interest to receive notifications about frame events. A frame event is represented by a *X3DFrameEvent* object. Implementation-specific frame event handling occurs in the *frameRendered* method the *X3DFrameEventListener* interface.

5 CASE STUDY

We have implemented the proposed architectural pattern using Xj3D, a Web3D consortium-endorsed, LGPL-licensed, multi-platform, extendible X3D toolkit written in Java (Xj3D, 2011), as part of the REVE Worlds virtual environment system.

The REVE Worlds system encodes virtual objects as it*ems*. An item contains several *item aspects*, each describing a certain representational aspect of the virtual object, for example, its physical properties, perceivable semantics and accessible functionality. Information pertaining to a virtual object's appearance, geometry, structure and all other properties that can be subjectively regarded as physical are encoded by the respective item's *physical aspect* in the form of X3D data. In particular, for each virtual object, a X3D scene is accessed through an external data source (for example, a X3D file). Then, part of it is loaded and appended to a sequence of specialized functionality nodes which are created by the system and are responsible for application-specific tasks, such as positioning the virtual object in space, aligning it with the world coordinate system, adding annotations such as a wireframe bounding box and nameplates, and more. The mechanism is highly flexible, as different parts of a single X3D scene can be used for different virtual objects, enabling virtual world designers to organize their resources into libraries. Also, the same part of a single X3D scene can be reused by different virtual objects, which facilitates the definition of several discrete virtual objects with similar physical properties.

Figure 3 below shows two discrete virtual objects sharing the same X3D scene graph.

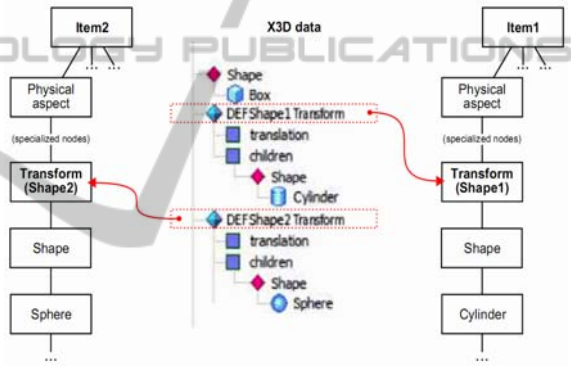

Figure 3: Two discrete virtual objects whose physical aspects draw data from the same X3D scene graph.

A sample virtual world in the REVE Worlds system is shown in Figure 4 below.

Figure 4: Sample virtual world in REVE Worlds.

In the REVE Worlds system, a virtual object's functionality is exposed as a set of *functions* which can be executed, in potentially rapid succession, by a virtual agent's effectors as part of the execution of complex actions, such as animated motion and other kinds of virtual object manipulation. This introduces the requirement for accurately-timed read- and write-access to the virtual environment's scene graph which, in X3D's case, turns out to be problematic because of its asynchronous execution model. In addition, it is necessary that the scene graph is evaluated and updated on a per-frame basis so that the virtual world's state is at all times consistent with all laws in effect, which is also problematic because of X3D's lack of inherent per-frame processing support. We were able to address those requirements effectively and elegantly by incorporating the proposed patter into the REVE Worlds system's architecture. We have also found modifications of relevant application code to be straightforward and transparent, thus increasing the maintainability, extendibility and life-span of our implementation.

6 CONCLUSIONS

In this paper, we present specific elements of the X3D specification that may pose problems to virtual environment developers. We also propose solutions to those problems and describe an object-oriented architectural pattern for the transparent incorporation of those solutions into concrete virtual environment designs. In particular, we demonstrate how to address field output timing issues that may arise due to X3D's asynchronous event model, as well as how to implement per-frame processing according to an event delegation mechanism. We have been unable, however, to overcome the inability to access primitive geometry node size-related fields (for instance, for the purposes of automatic bounding box calculation), as that is a restriction that arises from the X3D specification itself.

ACKNOWLEDGEMENTS

We would like to thank all members of the X3D Public mailing list, the X3D developer message boards and the Web3D Consortium for all the exciting discussions we had and the invaluable insights they provided us with over the past few years of our involvement with X3D and Xj3D.

REFERENCES

Anastassakis, G., Panayiotopoulos, T., 2011. Intelligent Virtual Environment Development with the REVE Platform: An Overview. In Vilhjalmsson, H. H., Kopp, S., Marsella, S. and Thorisson, K. R. (Eds.), *Intelligent Virtual Agents*, Lecture Notes in Computer Science, Vol. 6895 (pp. 431-432). Springer.

Aylett, R., Luck, M., 1999. Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments. *Applied Artificial Intelligence* 14(1), 3-32.

Behr, J., Dähne, P., Roth, M., 2004. Utilizing X3D for immersive environments. In *Web3D '04: Proceedings of the ninth international conference on 3D Web technology* (pp. 71-78). ACM.

Bouras, C., Panagopoulos, A., Tsiatsos, T., 2005. Advances in X3D multi-user virtual environments. In *Proceedings of the Seventh IEEE International Symposium on Multimedia (ISM'05)* (pp. 136-142).

Bouras, C., Tegos, C., Triglianos, V., Tsiatsos, T., 2007. X3D Multi-user Virtual Environment Platform for Collaborative Spatial Design. In *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops (ICDCSW '07)*.

Cabral, M., Zuffo, M., Ghirotti, S., Belloc, O., Nomura, L., Nagamura, M., Andrade, F., Faria, R., Ferraz, L., 2007. An experience using X3D for virtual cultural heritage. In *Proceedings of the Twelfth International Conference on 3D Web Technology (Web3D '07)* (pp. 161-164). ACM.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Ibanez-Martinez, J., Delgado-Mata, C., 2006. A Basic Semantic Common Level for Virtual Environments. *International Journal of Virtual Reality*, 5(3) 25-32.

Ieronutti, L., Chittaro, L., 2007. Employing virtual humans for education and training in X3D/VRML worlds. *Computers & Education*, 49(1), 93-109.

Ong, S., Mannan, M., 2004. Virtual reality simulations and animations in a web-based interactive manufacturing engineering module. *Computers & Education*, 43(4), 361-382.

REVEnet, 2011. *REVEnet*. http://kelnet.cs.unipi.gr/reve.

Web3D Consortium, 2011. X3D and related Specifications. http://www.web3d.org/x3d/specifications/

Xj3D, 2011. *The Xj3D project*. http:// http://www.xj3d.org.