Creating virtual worlds using adapting XML

Vlastimil Miléř
Postgraduate Student
Czech Technical University
Karlovo náměstí 13
121 35, Praha 2, Czech Republic
xmiler@fel.cvut.cz

Bohuslav Hudec
Associate Professor
Czech Technical University
Karlovo náměstí 13
121 35, Praha 2, Czech Republic
hudec@cslab.felk.cvut.cz

ABSTRACT

This paper deals with two aspects of virtual world creation – consistency of data and isolation of independent concepts. These concepts include among others 3D graphics rendering, physical simulation, audio processing, or user interaction. Current systems for virtual world creation focus more on concrete concepts and less on their independence thus limiting extensibility. They usually define static core functionality that quickly becomes outdated due to the fast development in computer industry. Adapting XML on the other hand focuses on extensibility, defining rules for cooperation (data sharing) and providing mechanisms for concept isolation. This makes it suitable as the base layer for extensible virtual world languages.

Keywords
Virtual World, XML, 3D Graphics, Data Adaptation

1. MOTIVATION AND INTRODUCTION

Building a working virtual world is hard. It is so hard, because there is a large amount of concepts in the virtual world, which must be synchronized and cooperate.

It is a trivial task to model a tree today. A bit harder is to simulate its growth. New entities are required – sun light, substrate, water and nutrients, gravity or wind to name a few. This problem has potential exponential characteristics. Each new element may influence the existing ones. Therefore adding it into a virtual world may as a consequence cause invalidation of current data structures and algorithms.

Our goal is to minimize this unpleasant property. One of the possibilities to reduce the exponential characteristics of the interactions in virtual world is to reuse data and algorithms. Once the algorithms for rendering, occlusions, collision detection, and event routing or physical simulation are developed, they must be applicable on as much real world data as possible without compromises. Each algorithm should work with most suitable data.

In this paper, we describe a data storage mechanism that mimics the properties of the real world and that:

- Keeps the data and concepts as much isolated as possible and thus lowers the exponential growth in complexity.
- Allows sharing of selected chunks of data to prevent duplication and inconsistency.

2. ADAPTING XML

Adapting XML is based on best features of current technologies for virtual world description (XML+DTD [1], XML Schema [4], VRML-97 [5], X3D [6], MPEG-7 [7]). These features are:

- Transparent structure.
- Selection of most important concepts.

While it tries to eliminate the problematic ones:

- Missing references support in parser.
- Missing custom atomic types.
- Poor isolation of concepts.
- Monolithic architecture.

To accomplish this, we employ following techniques:

- References are processed in parser. Their existence is transparent for client code.
- Type-libraries – a type-library is a module cooperating with the parser providing means

for storing, parsing, serialization and validation of atomic types. No atomic types are defined by the parser – there are only custom atomic types in type-libraries.

- Adapting engine – adapting engine is a layer above the parser that extracts the relevant bits of data from the whole file and creates a "writable view" on the data.

Architecture of adapting XML

The adapting XML inserts a new layer between XML parser and the application. The new layer is called *adapting engine* and its task is to adapt the data in the XML file using *converters* (Fig. 2-1).

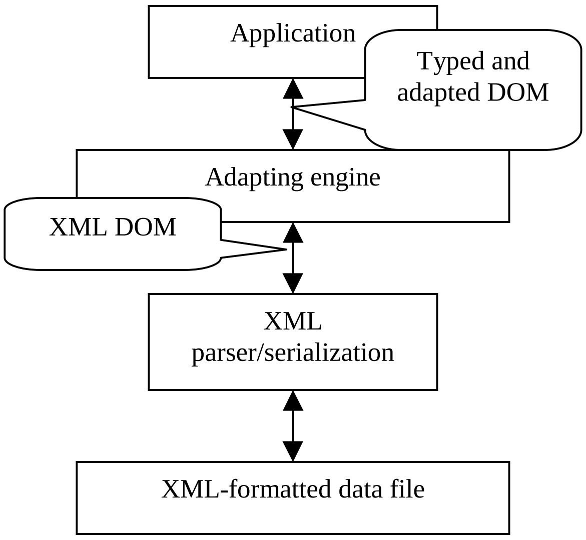

Fig. 2-1 Adapting engine in the system.

The typed and adapted DOM is similar to the XML DOM [1] interface. But instead of using simple elements, the elements implement two interfaces – a common interface and a type-specific interface.

Using the common interface, the element can be serialized or de-serialized from a string representation in the XML file. Also, the element can be uniquely identified using this interface.

The type-specific interface provides access to the internal data For example, if an atomic type represents a triangle mesh, it can have methods for manipulation and rendering of the mesh. Figure 2-2 shows the role of adapting engine in an application. Each part of an application receives suitable data through the typed and adapted DOM. This separation is a way to achieve one of our primary goals – the separation of concepts and minimization of dependencies.

The components in the application can cooperate at different levels [10]. Among others, they may use the file in the file system and serialize access to it or they may use the typed and adapted DOM.

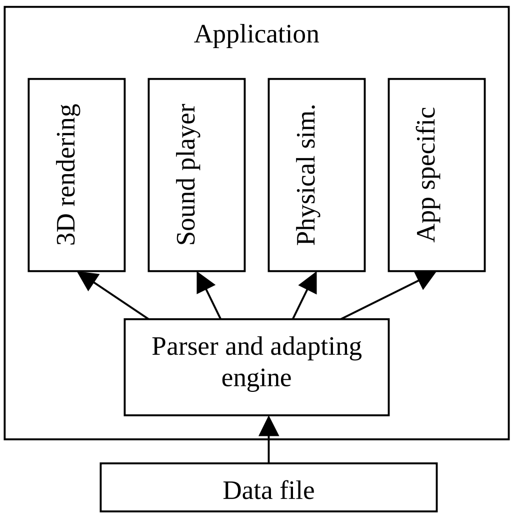

Fig. 2-2 Adapting engine separates concepts and promotes componentized architecture instead of monolithic architecture.

3. TYPES AND ELEMENTS IN ADAPTING XML

Atomic types

XML defines one atomic type – a string (CDATA). For some applications, this may be not suitable. They require multiple atomic types optimized for different tasks. These applications would usually encode these types into a string. But each application that uses this type must provide the validation code on its own. It would be desirable to prevent the code duplication by moving it into a shared type library. This type library is then used by the *adapting engine*. The library needs to be written in a full featured programming language that is able to interoperate with the *adapting engine*.

The *adapting engine* instantiates the type from the library and uses it to parse and validate the string. An application interfacing with *Adapting engine* using the *Typed and adapted DOM* and would see this type instead of the generic XML element.

By allowing custom atomic types and defining the way of connecting them to the parser and adapting engine, the mechanism for concept separation was given a solid foundation. Application may parse, serialize and validate files while not explicitly understanding the semantics and syntax of their contents.

Structured types

The ability to define transparent structured types is paramount for flexible virtual world description.

XML Schema definition of structured data types is sufficient for our needs.

In the examples in the rest of this article, XML Schema syntax will not be used due to limited space. The structured types will be represented graphically as on figure 3-1.

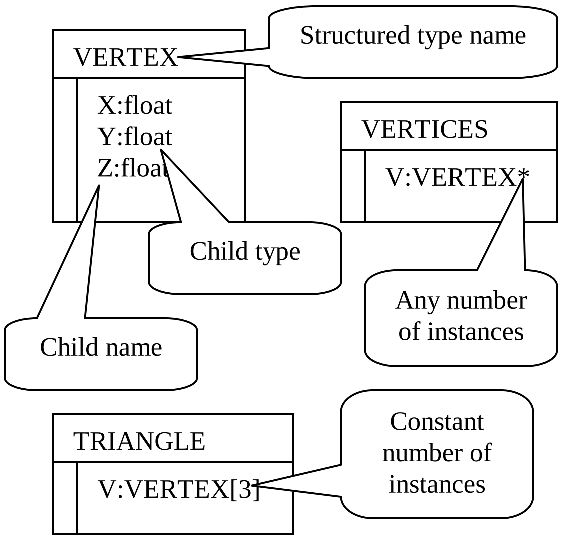

Fig 3-1 Definition of structured types VERTEX, VERTICES and TRIANGLE.

4. CONVERTERS

The problem of data adaptation may have different complexness. Sometimes it is enough to ignore portion of the data, rename data fields or fill missing fields with default values. Alternatively, the conversion may be very complex.

A *converter* is an object cooperating with the adapting engine. Its declaration specifies a source type and a destination type. When given an element of source type, it creates another element of destination type. The created element is a “view” on the source element through destination type. If it makes sense for given pair of types, the changes made to the created element should be reflected in the original one.

Adapting XML defines two types of converters – scripted and native converters. They differ in implementation. Native converters are declared in XML file and implemented in type library. Scripted converters are completely defined in the XML file.

5. VIRTUAL WORLD CONCEPTS IN ADAPTING XML

This chapter describes the so far verified concepts and enumerates the alternatives how each concept may be implemented if constraints are different.

3D Graphics

Figure 5-1 shows types in a commonly used hierarchical 3D scene composed of Segments. This definition was used for testing of adapting XML.

The described scene definition was used due to its simplicity and similarity with already existing definitions. It uses only discrete atomic types (int, float) and does not make use of advanced features of adapting XML.

Since 3D graphics data are usually large in volume and it would be ineffective to have object for each floating point number of each vertex, an atomic type for vertex can be defined in a type library. With the recent movement to pixel shaders and opaque vertex data [8], this will suit most applications.

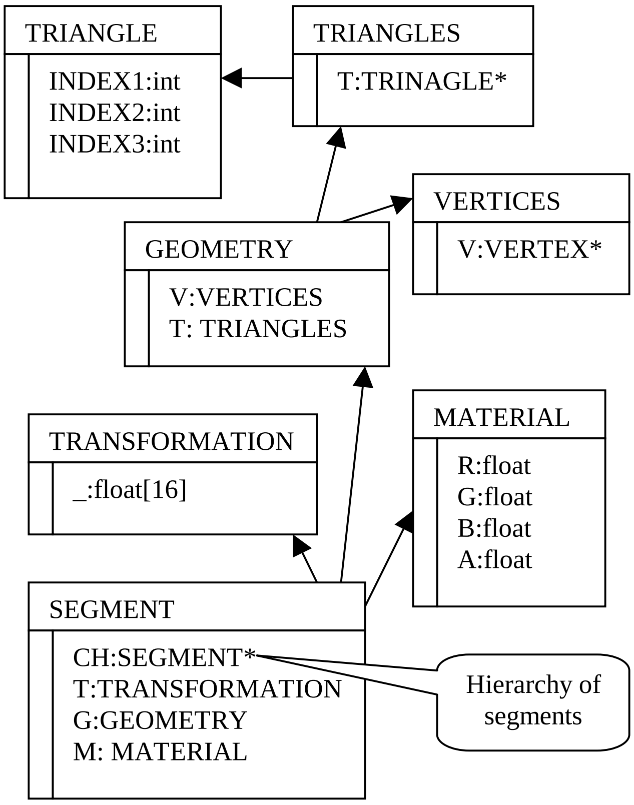

Fig. 5-1 Structure of a simple hierarchical scene. The arrows mark dependencies between types.

In each application, there must be a module able to interpret the semantics of the Segment. The definition of types, the type library and the interpreting module form logical system component. It makes sense to distribute them together as a package. Support of similar functionality is one of the X3D goals.

Grip Points

It is virtually impossible to create 3D graphics and other complex data used in virtual worlds without the use of suitable tools. Grip points [9] is a simple method for editation of data transformable to points in 3D space (typically vertices positions or normals). Types GripPoint and GripPoints (vector of GripPoints) were defined. An accompanying library was able to draw them as 3D points and user was able to move them in 3D space.

Example

Cooperation of isolated concepts using scripted converters was tested on a simple example. A user type – Box – was defined. It consisted of 6 floating point numbers named X1, Y1, Z1, X2, Y2, Z2. These represented the two points in 3D space, where the box was positioned (fig. 5-2).

Two scripted converters were defined:

- Box to Geometry – this converter maps a Box node to a Geometry node. The geometry node contains 12 triangles and 8 vertices. The child nodes in Vertices reference the children of Box.
- Box to GripPoints – the Box node is converted to a GripPoints node containing two GripPoint nodes (X1, Y1, Z1) and (X2, Y2, Z2).

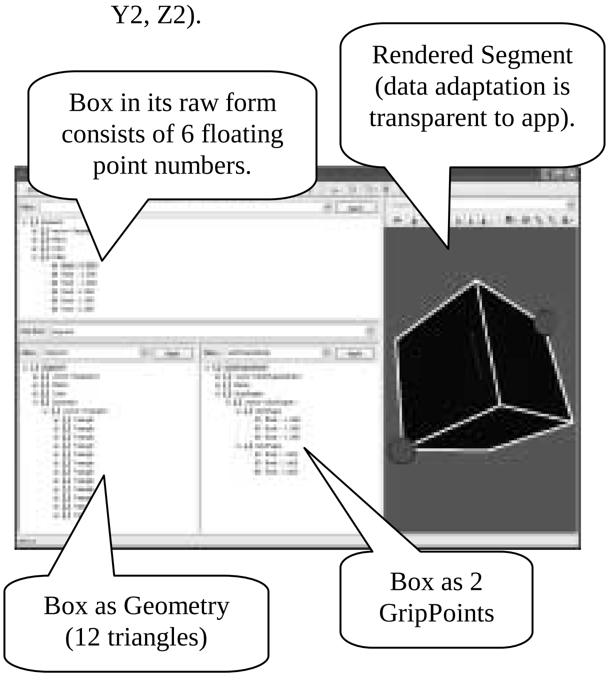
Box in its raw form consists of 6 floating point numbers.
Rendered Segment (data adaptation is transparent to app).
Box as Geometry (12 triangles)
Box as 2 GripPoints

Figure 3-2 User defined type Box is rendered as Geometry and edited as GripShapes (independent concepts cooperate).

This example, however simple, demonstrated that features of adapting XML fulfill our primary goals. The two concepts (Segment, GripPoints) were independent and the user defined Box type made use of them.

6. SUMMARY AND CONCLUSION

In this paper, the basic principles of adapting XML were described – architecture, definition of atomic and structured types and features of converters. Then it was demonstrated, how some of virtual world concepts could be implemented in adapting XML.

The adapting XML simplifies the creation of virtual worlds in by isolating concepts and thus allowing to postpone decision making. At its current stage it is not meant to replace X3D or MPEG-7 as they are more specific and suitable for today's industry. The virtual world creation is still a very immature discipline and many virtual world implementations contain few distinct concepts and sometimes the data in the file are misused for inadequate tasks (for example using graphics data for collision detection).

As the number of concepts in virtual world rises, the advantages of adapting XML become more visible. In the future, set of independent libraries will be implemented (for 3D rendering, collision detection, user interaction, physical simulation, sound processing). The creation of virtual world will be simplified to selecting appropriate libraries, implementing own concepts and connecting them using scripted converters. Adapting XML tries to follow a famous quote of Albert Einstein: "Make everything as simple as possible, but not simpler".

7. REFERENCES

[1] Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (editors): XML+DTD specification http://www.w3.org/TR/REC-xml
[2] Clark J. (editor): XSLT specification http://www.w3.org/TR/xslt
[3] Clark J., DeRose S. (editors): XPath specification http://www.w3.org/TR/xpath
[4] XML Schema specification http://www.w3.org/XML/Schema
[5] VRML97 specification (ISO/IEC 14772-1:1997) http://www.web3d.org/technicalinfo/specifications/vrml97/index.html
[6] X3D specification http://www.web3d.org/x3d.html
[7] MPEG-7 specification (ISO/IEC 15938-x:200x)
[8] OpenGL architecture review board: OpenGL Extension Registry http://oss.sgi.com/projects/ogl-sample/registry/
[9] Frey D.: AutoCAD 2000, Sybex 1999, ISBN 07-821249-8-4
[10] Miler V., Hudec B.: Communication in Componentized System with 3D Graphics and Multimedia, WSCG 2003 – Posters, ISBN 80-903100-2-8
[11] Gamma E., Helm R. Johnson R., Vlissides J: Design Patterns, Addison-Wesley Pub. Co. 1995, ISBN 02-016336-1-2