Design Patterns in XML Music Representation

Perry Roland
Digital Library Research & Development Group
University of Virginia
Charlottesville, VA 22903-4149 USA
pdr4h@virginia.edu

Abstract

Design patterns attempt to formalize the discussion of recurring problems and their solutions. This paper introduces several XML design patterns and demonstrates their usefulness in the development of XML music representations. The patterns have been grouped into several categories of desirable outcome of the design process – modularity, separation of data and meta-data, reduction of learning requirements, assistance to tool development, and increase in legibility and understandability. The Music Encoding Initiative (MEI) DTD, from which the examples are drawn, the examples, and other materials related to MEI are available at http://www.people.virginia.edu/~pdr4h/.

1 Introduction

Design patterns attempt to formalize the discussion of recurring problems and their solutions (Gamma, Helm, Johnson & Vlissides, 1995). Since common problems with (hopefully!) common solutions occur in many domains, patterns are used in almost every part of development. Design patterns are an effective way to share design decisions that actually work.

Several XML design patterns, drawn mostly from (Lainevool, n.d.), are introduced and their usefulness in the development of XML music representations is demonstrated. Because XML Schemas do not support specialized entities and parametric references, limiting the user's ability to extend the schema in an ad hoc fashion, and because they are more verbose and more complex (Valentine, Dykes & Tittel, 2002), this paper concentrates on the design of a (mostly) data-centric, i.e., designed to be processed by a machine rather than read by a human, Document Type Declaration (DTD). Of course, there are additional design patterns that are not covered. It is, however, the author's hope that enough patterns have been covered to generate additional discussion of the design of DTDs for music representation.

The Music Encoding Initiative (MEI) DTD, from which the examples are drawn, is both philosophically and technically related to the Text Encoding Initiative DTD (TEI, 2002). The complete DTD, the examples, and other materials related to MEI are available at http://www.people.virginia.edu/~pdr4h/. Readers unfamiliar with XML may wish to refer to a general introduction, such as *The XML Companion* (Bradley, 2000).

2 Modular Design

Modular design offers several advantages. First, it eases the burden of DTD maintenance by isolating changes. In addition, extensibility is increased since any module may be replaced in its entirety at will. Also, when each DTD module declares a single document element, it may be used independently of the main DTD, allowing the creation of the markup to be divided among several authors who need not have knowledge of the entire DTD. Finally, it enables the utilization of significant portions of the markup, such as the file header or MIDI performance data, in other contexts. The primary technique for achieving a modular design is to employ multiple secondary DTDs. The principal DTD typically takes the form of a "driver" which references the subsidiary modules.

3 Separation of data and meta-data

It is generally agreed that a clear separation between data and meta-data is highly desirable. In fact, separating these is the primary goal of DTD design as the separation itself reflects the worldview of the DTD creator. In other words, it clarifies the intended use of the DTD as well as the structure of any markup that is to be validated against the DTD. In addition, separating meta-data from its corresponding data allows the meta-data to be shared with other entities to which it also applies.

Elements are best used for structurally significant information while attributes are best used for atomic characteristics of an object that have no identity of their own. In other words, an attribute models part of the internal state of an object. (Stuhec, 2002) When an atomic characteristic is repeatable or must have an internal hierarchical structure, it may be represented by an element. However, these requirements can often be satisfied through the use of multiple attributes or repetition of the principal object with different attributes.

Just as the entity->property, or "has a", relationship is best modeled using attributes for each property, relationships

among entities, such as those characterized as "is a", "complements", etc., are best expressed via attributes as well. (Jelliffe, 1998) The representation of multiple hierarchies, that is, multiple parent-child relationships among a set of entities, particularly benefits from this kind of treatment.

4 Reduction Of Learning Requirements

With large DTDs, users are required to learn the content models and attributes for a large number of entities. Aside from simply reducing the number of entities, making the DTD less useful, there are a number of techniques that can be used to reduce the learning requirements of a new user and assist the experienced user in remembering the details of how to apply the DTD.

New users often require longer, descriptive names while experienced users prefer shorter, mnemonic ones. Localized names, i.e., in a different language, for a special user group, such as students or application programmers, or for reduction of storage requirements, are also often necessary. Furthermore, the name of an entity may change during the DTD development process or may even change between stages of encoding or processing.

Perhaps even more important than the selection of appropriate identifiers, is the selection of entities that match the granularity of the problem domain. In addition to the techniques listed in section 3, proper use of container and collection elements increases usability.

5 Assisting Tool Development

While a DTD should ideally be designed without any particular application in mind, in practice, without some general idea about how the data will be used, there's no point in creating the DTD in the first place!

In a data-centric representation, attributes are easier to mechanically process. Also, since accessing attributes does not require recursive or iterative processing, unlike embedded elements, using them may result in speedier processing.

Machine processing of a representation can be improved by also encoding the meta-data before the data it refers to. For example, because an HTML table's width and height are recorded before the actual table data, the table can be laid out and the data "dropped in". Similarly, if the meta-data for a music score is encoded before the actual data, such as notes and rests, then rendering decisions can be made much more effectively than if encoding intermingles the data and the meta-data.

Also, when meta-data is recorded first, several entities may refer to a single instance of the meta-data, reducing the size and complexity of the markup instance as well as the tool to manipulate it.

Because all of the ways in which it might be processed cannot always be foreseen, flexibility and extensibility should be built into a DTD. Extensibility is defined as the ability to add to an element's content model while leaving the basic declaration unchanged. Flexibility is defined as editing an entire content model for a particular purpose – the basic content model may be extended or restricted. These qualities are necessary in order to extend the useful life of the DTD. This is especially important for music representation because the repertoire to be encoded is vast and encoding is expensive. Because XML Schemas do not support parameter entities, this level of flexibility is difficult to implement using them. The usual way of providing an extensible content model in a DTD is to introduce a parameter entity into an element's content model. A similar degree of flexibility and extensibility may be achieved for attributes by declaring most, if not all, attributes for an element inside a parameter entity as well.

6 Increasing Legibility And Understandability

If a DTD is well designed, its useful lifetime will probably extend beyond the lifetime of the tools that use it. Therefore, despite the existence of tools that hope to hide the details of the DTD from the inexperienced, eventually someone, somewhere must be able to read and understand it, even if it is just to write another tool of the same sort. For this reason, it is wise for DTD creators to employ design patterns that increase the legibility and understandability of the DTD.

A flyweight (Gamma, et al., 1995) abstracts common markup declarations into a single reusable entity. Using a flyweight design pattern can ease the burden of DTD maintenance by eliminating errors that might be introduced when the same information is repeated and by reducing the size of the DTD.

Since the usability of a DTD depends to a certain extent on consistency, often a set of attributes is provided that can be placed on all elements, or on significant subsets of elements. If users can expect common attributes on every element, their ability to use the DTD can be enhanced. Also, it is easier to process a document that has a consistently applied set of common attributes. The common attributes are typically declared in a flyweight in order to enhance maintainability.

References

Bradley, N. (2002). *The XML Companion* (3rd ed.). London: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R, & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison Wesley.

Jelliffe, R. (1998). *The XML and SGML Cookbook: Recipes for Structured Information*. Charles F. Goldfarb Series on Open Information Management. Upper Saddle River, NJ: Prentice Hall PTR.

Lainevool, T. (n.d.). *XML Patterns.com Home Page*. Retrieved May 2, 2003, from `http://www.XMLPatterns.com/`.

Stuhec, G. (2002). *Elements versus Attributes* in *Position Papers*. Retrieved May 2, 2003 from `http://oasis-open.org/committees/ubl/200203/ndrsc/review/draft-ndr-20020316.pdf`.

TEI Consortium (2002). *TEI Guidelines*. Retrieved May 2, 2003 from `http://www.tei-c.org/Guidelines2/index.html`.

Valentine, C., Dykes, L., & Tittel, E. (2002). *XML Schemas*. Alameda, CA.