Proceedings of 2001 IEEE Human-Centric Computing Languages and Environments, Stresa, Italy, 5-7 September, 2001, IEEE CS Press, 312-319.

A Visual Approach to XML Document Design and Transformation

Kang Zhang[1,2] Da-Qian Zhang[3] Yi Deng[2]

[1]State Key Lab for Novel Software Technology, Nanjing University, Nanjing 210093, China

[2]Department of Computer Science, University of Texas at Dallas, Texas 75083-0688, USA
{kzhang, yideng}@utdallas.edu)

[2]Coral Corporation, 1600 Carling Ave., Ottawa, Canada
daqianz@coral.com

Abstract

This paper presents a visual approach to the representation and validation of multimedia document structures specified in XML and transformation of one structure to another. The underlying theory of our approach is a context-sensitive graph grammar formalism. The paper demonstrates the conciseness and expressiveness of the graph grammar formalism. An example XML structure is provided and its graph grammar representation, validation and transformation to a multimedia representation are presented.

1. Introduction

An important aspect of engineering software and information systems in the Internet environment is the representation, manipulation and presentation of multimedia documents. Based on the notion of multidimensional languages, Chang [3] proposes a conceptual framework for multimedia software engineering (MSE) that covers syntactic, semantic, pragmatic, and system aspects. A multidimensional language (ML) is considered a language that has objects of different media types as its primitives and spatial and temporal operators as its operators. The ML syntax describing how a complex multimedia objects are constructed from primitive or other complex media objects addresses the syntactic aspect of the MSE framework. Semantic aspect of MSE describes how dynamic media objects respond to internal or external events. Pragmatic aspect of MSE includes the tool support for building multimedia applications based on patterns of multimedia structures. Finally, system aspect of MSE relates to the quality of service and quality of presentation in systems design.

The work reported in this paper addresses primarily the syntactic and pragmatic aspects of MSE. When considering how to construct multimedia objects based on patterns of multimedia structures and how to build a tool to support the construction of quality presentation over the Internet, XML (eXtended Markup Language) [20] becomes our natural choice for defining multidimensional languages. XML has been increasingly accepted as the standard for Web document markup.

The central idea of XML is to allow users to create any documents of their preferred structures and share with other people. This opens a way for different types of document structures to be created to facilitate communications for various professional domains. Such a type of document structure is realized by a set of element tags that can be used to delimit data items in a document of the specific domain. If all the delimiting tags are properly placed in a document, the document is said to be *well-formed*. However, whether an XML document is valid is determined by the Document Type Definition (DTD), a formal grammar for specifying the structure and permissible values of XML documents. The tag set of a specific domain is called the "vocabulary" of that domain. People of the same domain could use the same basic syntax, parsers, and assisting tools of the vocabulary.

A useful design feature of XML is its separation of the content and the style of a document. The XML markup does not include formatting information about the document presentation. The need for a styling language for XML's presentation and for translating from one XML vocabulary to another has motivated the proposal of the extensible styling language (XSL) [21]. XSL specifies and supports the transformation of an input document to another structure, and describes how to present transformed information. XSL specifications are, however, hard to express intuitively in a linear textual form even though the structure of a XML vocabulary is essentially a tree structure. It is most natural to represent the XML tree graphically so that specifications and transformations can be performed intuitively.

This paper presents a graph grammar-based approach to the specification and transformation of multimedia documents written in XML. The main objective is to offer a graphical and intuitive representation of any document structure

Proceedings of 2001 IEEE Human-Centric Computing Languages and Environments, Stresa, Italy, 5-7 September, 2001, IEEE CS Press, 312-319.

(essentially a tree) and a powerful mechanism for automatic syntax-checking of such a structure and transformation of the document from one representation to another. Instead of using DTD to define an XML grammar and XSL to specify transformations, we propose a well-defined graph grammar formalism to define the syntax and semantics of an XML vocabulary, as well as to specify the transformations to any target XML vocabulary, SMIL [22] or XHTML. We provide a graphical document transformation mechanism for the designer of any multimedia presentation structure. Once the designer has defined the structure and specified transformation using our visual language generation system [25], a graphical document construction and transformation tool will be automatically generated. A novice user will be able to use the tool to construct multimedia presentations visually on the graphical editor and convert them to different presentations if desired. The major advantages of our approach are the following:

- A visual approach to constructing and transforming multimedia documents is easier to comprehend than the textual form. A novice user without any programming experience would find the visual approach intuitive if the visual representation could reflect one's mental image of a document structure.
- Automatically generated by a visual language generator, the graphical document construction and transformation tool that is provided to the user is capable of syntactic checking of any constructed document. The transformation process from one representation to another can also be visualized to assist the user in understanding the structural relationship between the two representations.

The rest of the paper is organized as the following. Section 2 introduces the graphical representation of any XML structure suitable for the interpretation and processing of our graph grammar formalism that is explained in Section 3. Section 4 demonstrates how an XML document can be validated and transformed to a representation of different structure by going through an example. Section 5 describes additional features of text operations through Java code and extension of cross-references, followed by the discussion of related work in Section 6. Finally, Section 7 concludes the paper.

2. Visualizing Document Structure

We will illustrate our ideas of graph transformations for XML document validation and conversion by going through an example: the markup of a simplified museum structure. Consider the XML tree for the simplified museum structure in Figure 1. The XML document for a museum may look like the following

```
<?xml version="1.0" ?>
<Museum>
    <Name>Dallas Science Museum</Name>
    <Direction>Directions to the museum …</Direction>
    <Hall>
        <Topic>Computers</Topic>
        <Item>
            <Name>Babbage Engine</Name>
            <Photo src="babbage24x24.jpg"></Photo>
            <Demo src="babbage_demo.avi"></Demo>
        </Item>
    </Hall>
    <Hall>
        <Topic>Printing Machinery</Topic>
            ……
        </Hall>
</Museum>
```

Assume that we wish to transform the museum XML to an SMIL representation and the target multimedia presentation structure expressed in SMIL is illustrated in Figure 2. SMIL (Synchronized Multimedia Integration Language) [22] is a simple markup language like HTML that facilitates the authoring of TV-like multimedia presentations on the Web. An SMIL presentation can be composed of streaming audio, streaming video, images, text or any other media type. The element <par> in Figure 2 dictates that all the medias enclosed in it will be played simultaneously, assuming that we would like the name (in the form of *h3*), photo (*img*), and video clip (*video*) of each museum item to be shown at the same time. The meanings of other elements are similar to those of HTML.

To perform the transformation using XSL, part of XSL code for "Hall" may look like

```
<xsl:template match="Hall">
  <div style="margin-left:40pt; margin-bottom:15pt; text-align:left">
        <xsl:apply-template />
  </div>
</xsl:template>
```

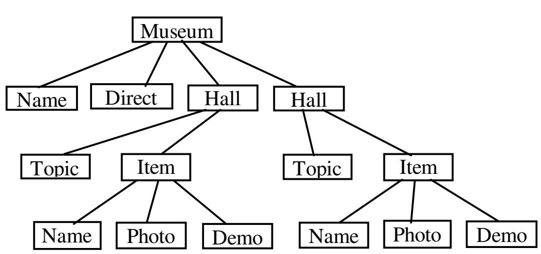


Figure 1: An XML tree for museum structure

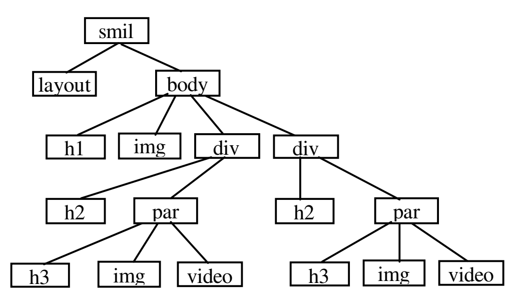


Figure 2: An SMIL tree for multimedia museum

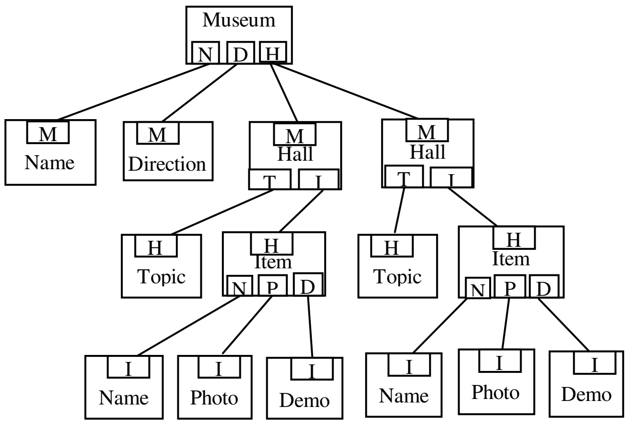


Figure 3: The node-edge form of the museum XML tree

We first translate the tree diagram in Figure 1 into a graphical form whose syntax is suitable for our grammar interpretation. The translation is very straightforward as shown in Figure 3, ignoring any arrows since the direction is unimportant in our graph grammar representation. A node in the node-edge representation is a two-level structure. The first level is the large surrounding rectangle, which is called a *super-vertex*. The small rectangles embedded in a super-vertex are the second level called *vertices*. A vertex can be connected by one or more edges depending on whether it is marked as explained in Section 3. Each edge is unique and is determined by the two vertices that it connects. A super-vertex is also considered as a vertex that can be linked to other vertices. The RGG does not impose semantic difference between connecting a vertex and connecting a super-vertex.

In a node-edge diagram, all the vertices in a node are uniquely identified. For convenience and simplicity, we use a capital letter (e.g. I) to identify each vertex according to which other node (e.g. "Item") it may be connected to.

We propose a graph transformation mechanism that supports

- Validation of any input document structure according to a predefined XML grammar, and
- Transformation of one document structure to another according to a set of predefined transformation rules extended from the above grammar.

Our system also supports suppression of document contents and can re-arranging elements in a document (such as swapping the presentation order of two media elements).

3. Reserved Graph Grammar

Graph grammars with their well-established theoretical background may be used as a natural and powerful syntax-definition formalism [13] and the parsing algorithm based on a graph grammar may be used to check the syntactical correctness and to interpret the language semantics.

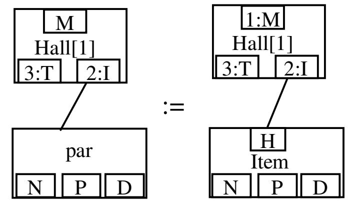


Figure 4: A graph rewriting rule

In a graph grammar, a graph rewriting rule as shown in Figure 4, also called a *production*, has two graphs called *left graph* and *right graph*. It can be applied to another graph (called *host graph*) in the form of an *L-application* or *R-application*. A *redex* is a sub-graph in the host graph which is isomorphic to the right graph in an R-application or to the left graph in an L-application. A production's L-application to a host graph is to find in the host graph a *redex* of the left graph of the production and replace the redex with the right graph of the production. The L-application defines the language of a grammar. The language is defined by all possible graphs that have only terminal labels and can be derived using L-applications from an initial graph (i.e. λ). An R-application is a reverse replacement (i.e. from the right graph to the left graph) that is used to parse a graph.

One obstacle for the application of graph grammars is that even for the most restricted classes of graph grammars the membership problem is NP-hard [14]. Consequently, most graph grammar parsing algorithms proposed so far are either unable to recognize interesting languages of graphs or tend to be inefficient when applied to graphs with a large number of nodes and edges.

Another problem is that nearly all known graph grammar parsing algorithms [7][8][17] deal only with context-free productions. A context-free grammar requires that only a single non-terminal is allowed on the left-hand side of a production [19]. A context-sensitive graph grammar, on the other hand, allows left-hand and right-hand graphs of a production to have arbitrary number of nodes and edges. Most existing graph grammars for visual languages are context-free. Yet not many visual languages can be specified by purely context-free productions. Additional features are required for context-free graph grammars to handle context-sensitivity. It is therefore difficult for context-free grammars to specify varieties of visual languages.

A context-sensitive graph grammar, on the other hand, allows left-hand and right-hand graphs of a production to have arbitrary number of nodes and edges. Rekers and Schürr [12] proposed *layered graph grammars* (LGGs) in specifying visual languages. Being context-sensitive makes the graph grammars expressive. The LGG formalism is intuitive and thus easier to understand and to use than textual formalisms when specifying visual languages. However, although being expressive, a layered graph grammar cannot be efficiently implemented since its parsing complexity generally reaches exponential time. It is reported that parsing layered graph grammars using the Rekers-Schürr algorithm is extremely hard [16].

We have proposed a context-sensitive graph grammar called the *reserved graph grammar* (RGG) [24], which was motivated by the development of a general-purpose visual language generator and influenced by the LGG. Considering that the targets of the generator are visual languages, their grammars are better specified using a graph formalism.

The RGG is developed based on the layered graph grammar by using the layered formalism to allow the parsing algorithm to determine in finite steps whether a graph is valid. It combines the approaches of the embedding rule and the context elements for handling the interconnection relationships between a replacing sub-graph and its surrounding graph elements (traditionally called *embedding*). By introducing context information, simple embedding rules can be sufficiently expressive to handle complicated programs.

In order to identify any graph elements that should be reserved during the transformation process, we *mark* each isomorphic vertex in a production graph corresponding to these elements by prefixing its label with a unique integer. The purpose of marking a vertex is to preserve the context.

Using the marking mechanism, we introduce an embedding rule as the following. If a vertex v in the redex of the host graph has an isomorphic vertex v' in the corresponding production's right graph and neither v nor v' is marked, then all edges connected to v should be completely inside the redex. The above simple embedding rule brings multiple benefits:

1. Each application of a production can ensure that a graph can be embedded in a host graph without creating dangling edges.
2. The cases of single-edge and multiple-edge connectivity can be easily defined and distinguished in the grammar.
3. The parsing orders that may lead to incorrect semantics can be prevented easily by the simple marking mechanism.

The node structure enhanced with the notion of marked and unmarked vertices in the RGG simplifies the transformation specification and also increases the expressiveness. Moreover, the wildcard mechanism used in the LGG is not needed in the RGG. The examples in Figures 5 and 6 illustrate the first two benefits in the R-application process, where the host graphs have isomorphic graphs (enclosed in dotted boxes) of the right graph of the production in Figure 4. The third benefit will be demonstrated in the example of the parsing process in Section 4.

In Figure 5(a), the isomorphic graph is a redex. The marked vertices and the vertices corresponding to the isomorphic vertices marked in the right graph of the production are painted gray. The transformation deletes the redex while keeping the gray vertices. Then the left graph of the production is embedded into the host graph, as shown in Figure 5(b), while treating a vertex in the left graph the same as the corresponding gray vertex. We can see that the marking mechanism allows some edges of a vertex to be reserved after transformation. For example, in Figure 5(a), four edges connecting to the outside of the redex are reserved after transformation.

In our notion of museum XML tree, a "Hall" node may connect to multiple "Item" nodes, while a "Item" node is allowed to connect to only one "Hall" node, because a museum's display hall may have many items on display but not vice versa. We show how such a constraint can be expressed and maintained in the RGG. The solution is simple: we mark the I vertex of the "Hall" node and leave H vertex of the "Item" node unmarked in the definition of the production (as illustrated in Figure 4). According to our embedding rule, the isomorphic graph in dotted box in Figure 6(a) is not a redex, because the isomorphic vertex of the unmarked vertex H in the "Item" node has an edge that is not completely inside the isomorphic graph. Therefore, the graph in Figure 6(a) is invalid. On the other hand, the graph in Figure 6(b) is valid according to the embedding rule. There is a redex (the isomorphic graph in the dotted box) in the graph, because the isomorphic vertex of I in "Hall" connecting to "Item" in the right graph of the production is marked, even though it has an edge connected outside the isomorphic graph. Therefore, the marking mechanism helps not only in embedding a graph correctly, but also in simplifying the grammar definition.

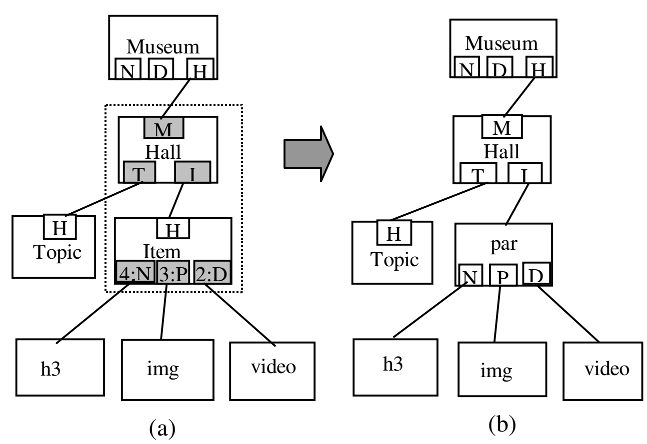
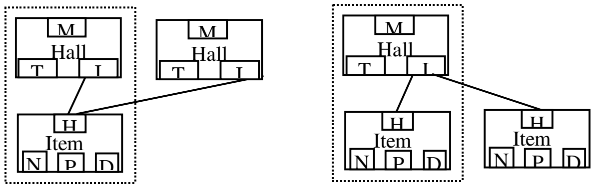


Figure 5: Reserving edges in R-application

Proceedings of 2001 IEEE Human-Centric Computing Languages and Environments, Stresa, Italy, 5-7 September, 2001, IEEE CS Press, 312-319.



(a) Illegal (b) Legal

Figure 6: Determining connectivity in R-application

As shown above, an RGG is explicit in describing the syntax of a wide range of diagrams. This embedding mechanism in the RGG allows implicit representation to avoid most of the context-specifications while being more expressive. This greatly reduces the complexity of visual expressions, and in turn increases the efficiency of the parsing algorithm.

The RGG formalism has been used in the implementation of a toolset called VisPro, which facilitates the generation of visual languages using the Lex/Yacc approach [25]. As a part of the VisPro toolset, a visual editor could be used to create visual programs based on the grammar specifications and parsing algorithms are automatically created according to the grammar.

4. Document Specification and Transformation

We now show how graph transformation and associated syntax-directed computation can be performed to achieve the same functionality as that of XSL. We demonstrate how one XML vocabulary can be easily transformed to another once the transformation rules extended from the XML grammar have been defined and applied.

The graph transformation rules shown in Figure 7 explicitly and precisely define the syntax of the museum XML when the sub-graphs with dotted lines are excluded; and the mechanism for transforming the museum XML syntax (Figure 1) to the multimedia version in the SMIL syntax (Figure 2) when the sub-graphs with dotted lines are included. In the following discussions, we will refer to the former rules as the *grammar* (for the museum XML), and the latter, extended from the grammar, as the *translator* (from the museum's XML presentation to the multimedia presentation in SMIL). Figure 7 shows both the grammar and the translator together (differentiated by the dotted lines), consisting of a set of productions, where a box labeled <i> is Production i. The productions are defined to take advantage of the tree structure of any XML document that can be parsed (transformed) efficiently from tree leaves to the root. Based on our embedding rule stated earlier, parsing from any sub-graph that has no leaf node is not allowed since at least an unmarked vertex in the sub-graph would be connected to the outside.

Carrying out the general parsing process is time-consuming as it needs to attempt the R-applications for all productions. Although a general RGG parsing algorithm has the exponential time complexity, the parsing time can be made polynomial by introducing a constraint (due to the embedding rule) into the RGG. It is not yet known how this constraint limits the application scope, but we find that even the grammar of a complicated control flow diagram satisfies the constraint. With this constraint, a parsing algorithm of polynomial time complexity and an algorithm for checking whether an RGG satisfies the constraint have been developed [24]. Further improvement in parsing similar grammars more efficiently has also been reported [1].

We have developed a simple parsing algorithm, called *selection-free parsing algorithm (SFPA)*, which only tries one parsing path. It has been proven that when parsing any graph using SFPA, if one parsing path fails, any parsing path will fail [24].

To be able to use SFPA, a graph grammar must satisfy the selection-free condition. Informally, a graph grammar with

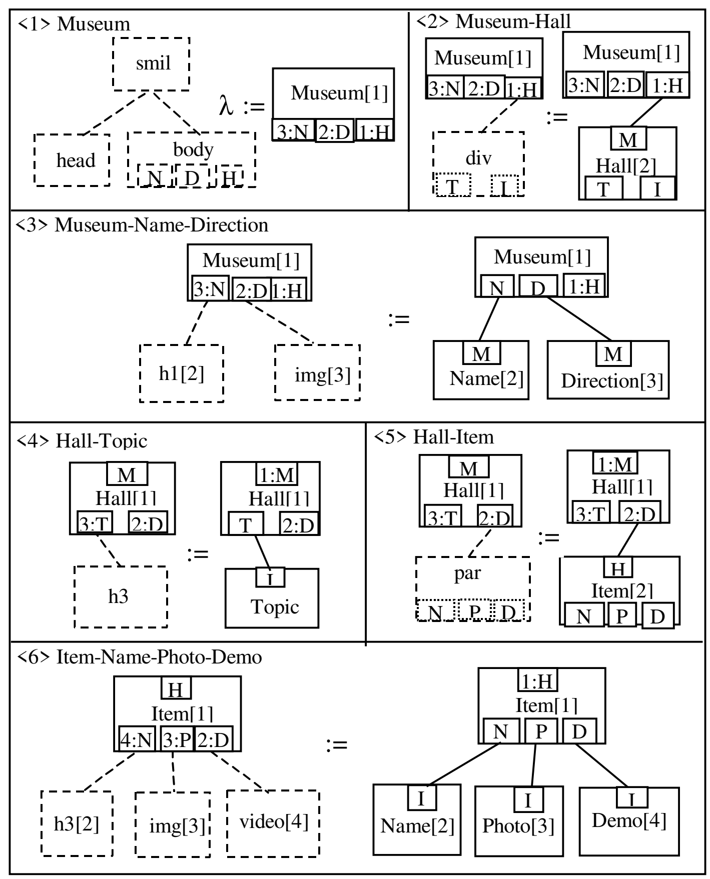


Figure 7: A grammar specifying the museum XML and its transformation to a multimedia representation

a set of productions is said to be selection-free if a graph has two redexes corresponding to two productions' right graphs and the applications of the two productions to the graph in different orders generate the same resulting graph. The formal definition of the condition and a proof of polynomial parsing time under the condition are treated elsewhere [23].

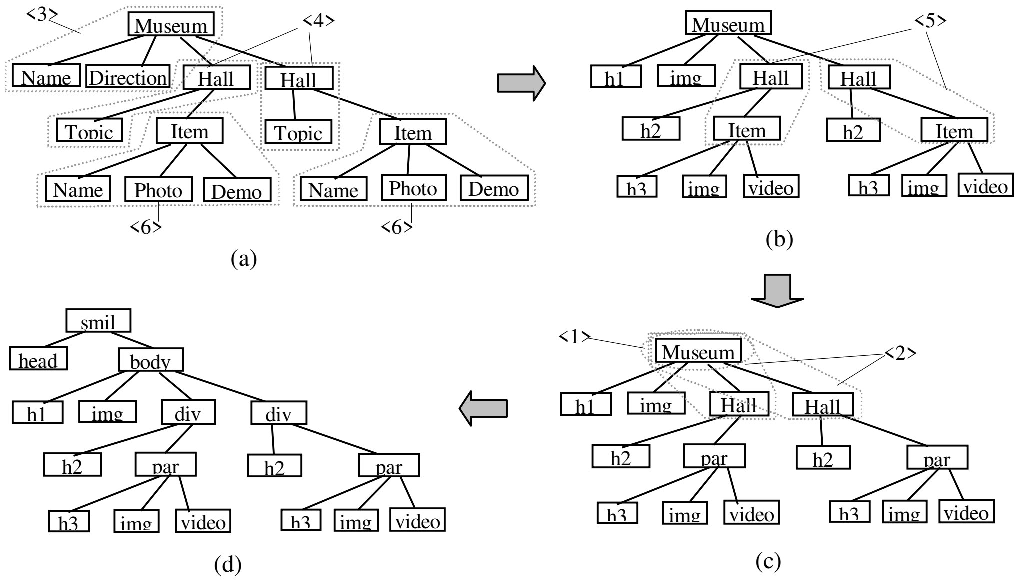


Figure 8: Graph transformations from museum XML to SMIL when productions are applied

Both the museum XML grammar and XML-SMIL translator illustrated in Figure 7 are selection-free under the definition, so we can use SFPA to parse any diagrams representing document structures to check if they are valid museum XML documents. We can also use SFPA to translate a valid XML documents to the SMIL presentation. To take advantage of the tree structure of any XML documents, we define our production rules in such a way that efficient parsing and translation can be performed from tree leaves to the root. The extended part (from the grammar) in the translator would not affect the transformation efficiency since its sub-graphs could all be regarded as terminals.

The initial node-edge graph of the museum XML shown in Figure 5 has all the vertices unmarked. By applying the R-application of the grammar in Figure 7 (excluding the sub-graphs with the dotted lines) repeatedly to a specific diagram, we can determine whether the diagram represents a valid museum XML document. Similarly, by repeatedly applying the R-application of the translator in Figure 7 (including all the sub-graphs, except "λ"), a museum XML document can be transformed into an SMIL document for multimedia presentation.

The process of transforming the museum XML tree drawn in Figure 1 to the corresponding museum SMIL tree in Figure 2 is illustrated in Figure 8, where the labels in arrow brackets indicate the corresponding productions. The R-applications may be applied in different orders but will produce the same result.

In Figure 8(a), the seven sub-graphs in the dotted boxes are possible redexes, which can be applied with productions <3>, <4>, and <6> to produce the graph in Figure 8(b). Similarly, the graph in Figure 8(b) can be transformed into the graph in Figure 8(c), and so on. Finally, the graph is transformed into an SMIL representation. Part of the generated SMIL document is shown below:

```
<smil>
    <head> ……</head>
    <body>
        <h1>Dallas Science Museum</h1>
        <img src="map12x10.jpg" />
        <div style="margin-left:40pt; margin-bottom:15pt; text-align:left">
            <h2>Computers</h2>
            <par>
                <h3>Babbage Engine</h3>
                <img src="babbage24x24.jpg"/>
                <video src="babbage_demo.avi" />
            </par>
        </div>
        <div>
            <h2>Printing Machinery</h2>
            ……
        </div>
    </body>
</smil>
```

5. Auxiliary Support

In XSL, a template may contain elements that serve as instructions for creating result tree fragment. The instructions are executed when the template is instantiated. In our approach, these instructions, as well as any entities that are applied to the original XML document, such as information filtering, text re-arrangement, are specified in Java (called *action*) and are attached to the corresponding production. For example, to add the dimension of the "Hall" display in the target SMIL, we will attach the following action code to Production<2>:

```
action(AAMGraph g)
{ Attributes attributes=g.getAttributes);
    attributes(2).put("style", "margin-left:40pt; margin-bottom:
    15pt; text-align:left");
}
```

Different actions can be performed on different attributes of the redex of a production to achieve the desired text processing results. Writing an action code is like writing a standard exception handler in Java by treating each attribute as an object.

Another feature of XML is cross-referencing in its document specification through the Document Type Definition (DTD). There are several ways in XML to represent cross-reference relationships. For instance, assume that the museum XML document specifies an additional museum activity called “Talk”, describing various seminars relating to specific display items with their venues and dates as shown in Figure 9(a). We further assume that a seminar relating to an item may be presented on multiple dates. In XML DTDs, this could be achieved by defining an ID attribute “id” in each “Talk” and an IDREFS attribute “schedules” in each “Item” that refers to one or more “id” values, as illustrated in the following code

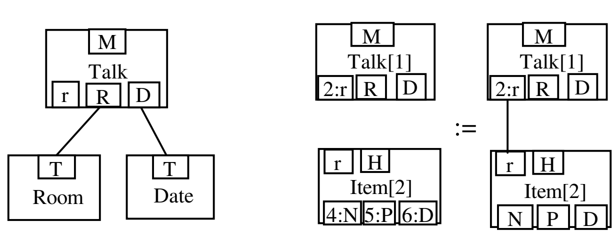
```
<Talk id="t008">
      <Room>E218</Room>
      <Date>2001-6-22</Date>
</Talk>
<Talk id="t015">
      <Room>N226</Room>
      <Date>2001-9-5</Date>
</Talk>
......
<Item schedules="t008 t015">
      <Name>Babbage Engine</Name>
      ......
</Item>
```

We designate a special vertex labeled “r” to support such a cross-reference connection. The distinction between a single reference (IDREF in DTDs) and multiple references (IDREFS) is easily made by the marking mechanism as demonstrated earlier in Section 3 (Figure 6). A production rule for the relationship of one “Item” to multiple “Talks” is shown in Figure 9(b).

The use of cross-reference links would potentially introduce cycles in an XML document structure so that it is no longer a tree. During parsing and transformation, we start processing (thus removing) cross-reference links by identifying the “r” vertices, before processing tree leaves.

6. Related Work

Using visual programming and visual language approaches to work with XML is not new, though most previous work



(a) Additional elements (b) Cross-reference

Figure 9: Example of cross-referencing

has been on visual query languages, such as Delaunay [4], Lore [9], VQL [15], XML-GL [2], and Xing [6]. For example, Xing is a visual language for querying and transforming XML data. It achieves XML transformation and restructuring using some rules that combine the patterns of queries and results returned by queries. It uses nested boxes to represent XML data such what each element tag is written on top of a box while all the attributes of the element are enclosed in the box. Cross-referencing of the IDREF(S)->ID relationship is explicitly visualized by using an arrow (arrows). The general Xing document metaphor does not visualize overall structural relationships among document elements. It facilitates visual queries rather than document design and validation.

On the other end of the spectrum, VRDL (Visual Repository Definition Language) [11] aims at harnessing the massive amount of data on the Web by imposing a logical structure visually and encapsulating chunks of original information into metadata entities. Similar to our approach, VRDL is generated by a visual language generator [9] and thus supports graphical and syntactic editing. The visual notation used in VRDL is based on and adapted from the Nassi-Shneiderman diagram. But the system is not designed for visual construction and transformation of multimedia documents written in XML-based languages.

More related to our work is that of Weitzman and Wittenburg [18] that uses a formalism known as Relational Grammars to support the authoring of multimedia documents. The formalism governs the document layout and structure through spatial specifications, and the document presentation sequence through temporal constraint specifications. The approach adapts visual language generation process to the automatic generation of multimedia design through syntax-directed translation. It parses the document elements and user-specified relations to build one or more derivation trees, each representing an independent presentation. The relational grammar formalism is constraint-based and supported by a constraint solver. Our approach uses a context-sensitive graph grammar formalism which we believe is more intuitive.

Inspired by the work of Weitzman and Wittenburg, Cruz and Lucas developed a visual querying and presentation system called Delaunay^{MM} [5]. As a multimedia extension to Delaunay that was designed to visualize object-oriented databases, Delaunay^{MM} allows the user to specify the layout of a multimedia document in a set of style sheets and provides a query language interface for the user to query object-oriented database and the Web. It combines the user-defined style sheets with the answers to the queries to automatically generate a visual document that includes pages with multimedia objects. The layout of each page is determined by an associated style sheet, specified in a grid and applied with various layout constraints. The approach used in Delaunay^{MM} is not based on language generation mechanism and thus does not provide correctness checking of layout structures.

7. Conclusion

This paper has presented a visual approach to automatic syntax checking of any structured documents written in XML and transformations between different XML vocabularies, for applications such as multimedia presentation. The approach uses a formal graph transformation mechanism, extended from the reserved graph grammar (RGG) that has been used for the automatic generation of domain-oriented visual programming languages. An RGG is a collection of graph rewriting rules represented by labeled graphs. It is context-sensitive and its right-graph represents a pattern to be matched against elements in the source XML tree and left-graph represents a template to be instantiated to create part of the resulting tree. The source and result trees could represent any XML, SMIL or XHTML vocabularies.

8. Acknowledgments

We would like to thank Andy Schürr for his comments and suggestions on the first draft of the paper.

References

[1] Bottoni and A. Schürr, Efficient parsing of visual languages based on critical pair analysis and contextual layered graph transformation, *Proc. 2000 IEEE Symp. on Visual Languages*, Seattle, USA, 10-13 Sep. 2000, 59-60.

[2] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tancia, XML-GL: a graphical language for querying and representing XML documents, *Proc. 8th Int. World Wide Web Conf.*, 1999.

[3] Shi-Kuo Chang, *Multimedia Software Engineering*, Kluwer Academic Publishers, 2000.

[4] I.F. Cruz, M. Averbuch, W.T. Lucas, M. Radzyminski, and K. Zhang, Delaunay: a database visualization system, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1997, 510-513.

[5] I.F. Cruz and W.T. Lucas, A visual approach to multimedia querying and presentation, *Proc. ACM Multimedia'97,* Seattle, USA, 8-14 November 1997.

[6] M. Erwig, A Visual Language for XML, *Proc. 2000 IEEE Symp. on Visual Languages*, Seattle, USA, 10-13 Sep. 2000, 47-54.

[7] E.J. Golin, A method for the specification and parsing of visual languages, PhD Thesis, Brown University, May 1991.

[8] M. Kaul, Parsing of graphs in linear time, *Proc. 2nd Int. Workshop on Graph Grammars and Their Application to Computer Science*, LNCS 153, 1982, 206-218.

[9] J. McHugh and J. Widom. Query Optimization for XML. *Proc. 25th Int. Conf. on Very Large Data Bases*, Edinburgh, Scotland, September 1999.

[10] M. Minas and G. Viehstaedt, DiaGen: A generator for diagram editors providing direct manipulation and execution of diagrams, *Proc. 1995 IEEE Symp. on Visual Languages*, Darmstadt, Germany, September 1995, 203-210.

[11] M. Minas and L. Shklar, Visual definition of virtual documents for the world-wide Web, *Proc. 3rd Int. Workshop on Document Processing*, LNCS 1293, 1996, 183-195.

[12] J. Rekers, and A. Schürr, Defining and parsing visual languages with layered graph grammars, *J. Visual Languages and Computing*, 8(1), Feb. 1997, 27-55.

[13] G. Rozenberg, (ed.*), Handbook on Graph Grammars and Computing by Graph Transformation: Foundations*, Vol.1, World Scientific, 1997.

[14] G. Rozenberg and E. Welzl, Boundary NLC graph grammars -- basic definitions, normal forms, and complexity, *Information and Control*, 69, 1986, 136-167.

[15] K. Vadaparty, Y.A. Aslandogan, and G. Ozsoyoglu, Towards a unified visual database access, *Proc. ACM SIGMOD Conf. on Management of Data*, 1993, 357-366.

[16] J.T. Vermeulen, Viability of a parsing algorithm for context-sensitive graph grammars, Technical Report, Leiden University, 1996.

[17] L.M. Wills, Automated program recognition by graph parsing, PhD Thesis, MIT AI Lab, Cambridge, Massachusetts, Technical Report 1358, 1992.

[18] L. Weitzman and K. Wittenburg, Automatic presentation of multimedia documents using relational grammars, *Proc. ACM Multimedia'94*, San Francisco, USA, 15-20 Oct. 1994.

[19] K. Wittenburg and L. Weitzman, Relational grammars: theory and practice in a visual language interface for process modeling, *Proc. AVI'96*, Gubbio, Italy, May 27-29, 1996.

[20] W3C, Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/REC-xml.html, Oct. 2000.

[21] W3C, XSL Transformations (XSLT), http://www.w3.org/TR/REC-xslt-19991116, Nov. 1999.

[22] W3C, Synchronized multimedia integration language, http://www.w3.org/TR/REC-smil, Sep. 2000.

[23] D-Q Zhang, K. Zhang, and J. Cao, A context-sensitive graph grammar formalism for the specification of visual languages, to appear in *The Computer Journal*, 2001.

[24] D-Q. Zhang and K. Zhang, Reserved graph grammar: a specification tool for diagrammatic VPLs. *Proc. 1997h IEEE Symp. on Visual Language,* Capri, Italy, Sep. 23-26, 1997, 341-348.

[25] K. Zhang, D-Q. Zhang, J. Cao, Design, construction, and application of a generic visual language generation environment, *IEEE Trans. on Software Engineering*, Vol.27, No.4, Apr. 2001, 289-307.