Active Documents in XML

Luca Bompani, Paolo Ciancarini, Fabio Vitali
Dept. of Computer Science, University of Bologna
{bompani|ciancarini|vitali}@cs.unibo.it

Abstract

XML is an extremely promising approach to the standardization of most data structures, but the rendering semantics proposed with XSL is not sufficiently general, lacking support for arbitrary graphic objects and sophisticated notations.

In this paper we introduce XMLC, a generic, modular XML browser that can display any kind of notation required for XML documents, and we also discuss the use of XMLC for active documents.

Introduction

The introduction of XML has provided the World Wide Web with a powerful data format to express most kinds of data structures. XML can be used to represent structured text documents, specialized data structures, and any kind of sophisticated notations for specific needs.

Currently, XML tools are either meant to display text-oriented documents (belonging to one of the myriad document types expressed in XML DTDs) or to rely on XML comfortable syntax for their run-time parameters. Although specialized, non textual notations have been defined in XML (for instance for mathematics and chemistry), they were meant for specialized browsers developed just for the display of the specific notation.

Currently, no tool exists that allow the display of all kinds of XML documents, including text structures, specialized notations, and any mix of structured documents and specific rendering needs. To overcome these limitations we have created a generic, modular XML browser called XMLC. Each XML element is associated to a special rendering module, called *displet*, that creates the visual representation of the element [CRV98].

But the power of displets goes beyond the simple visualization of XML documents. Since we can associate any module to an XML element, we can associate a *behavior* to XML documents. The rendering of the document becomes thus just one of the possible behaviors activated upon it. We provide thus a general environment for *active documents.*

While a traditional document is a consistent piece of information created to be subjected to an action (for instance to be displayed or printed), in our view an active document is a document that can provide some autonomous active behavior: the same document may be displayed, be printed, be searched, perform computations, produce additional documentation, perform animations, etc.

Active documents are not unheard of: just to name one case, Active-X has been marketed with this evocative term. "Traditional" active documents may have some clearly distinguished passive and active parts (as is the case of Active X objects within a traditional word processor or spreadsheet document), or they may contain code executable by the display engine (for instance, a complex PostScript page), or they may have special localized markup that activates some particular behavior in the display engine (for instance, a scripted HTML page, or a Word document containing some macros). The model of active documents we are suggesting, on the other hand, does not distinguish "active" from "passive" documents, since they are all expressed in plain XML format, but are associated to program modules on request, depending on the application they are subjected to. Thus the same document may be displayed, printed, or "activated" for some complex computation, without any change to the source.

In this paper, we discuss the current status of the languages connected to XML, and provide some background information to XMLC and its use as both a general rendering application for XML documents, and as a general runtime environment for activatable XML documents.

XML and related standards

Formed in 1994 by the special interest group on SGML of the World Wide Web Committee, the XML working group set to work to provide an SGML-based generalized markup language that could provide most of the functionalities of SGML, while retaining the same simplicity of HTML for the casual author and the developer of tools. The working group divided the effort in several independent but related languages that cooperate in order to provide a complete and sophisticated markup environment. For our purposes, the most important proposals of the XML family are the XML generalized markup language itself, the XSL stylesheet language, and the DOM document object model, which will be discussed singularly in the following sections.

XML 1.0

XML 1.0 (Extensible Markup Language, [BPS98]) is the first and most stable proposal of the XML family. It is a proper subset of SGML, rather than an application of it (like HTML). XML is a generalized markup language, where authors can decide the set of tags they are going to use in writing their documents, and it is used at its best to express the structure of the document and the semantically relevant elements, rather than their typographical properties.

XSL

Just like SGML documents, XML documents lack any machine-interpretable semantics. That is, XML dictates about the syntax of the markup, not about its meaning. This means on the one hand that authors are free to assign whatever meaning they want to the elements they choose to employ, but also that there must be a way for programs to make use of these elements in an appropriate way.

The solution taken within the XML family is to *map* (or *rewrite*) the source XML document (containing elements meaningful to the author) into a different XML document containing elements meaningful for the program that has to perform the application. Thus, a document containing elements such as section, titles, formulas, sentences, etc., all meaningful to its human author, will be rewritten into a new document containing elements such as blocks, paragraphs, inline styles, white regions, and so on, all meaningful to a displaying or printing program.

This is done by XSL (Extensible Stylesheet Language, [CD98]). XSL is a mapping language for XML elements. Each XSL stylesheet is made of rules composed of a pattern and an action. The pattern identifies an element of the XML source to which the action should apply; a pattern may specify all elements of a given type, all elements contained in a given subtree, all elements having a given attribute, etc. When the most appropriate pattern is found for the current XML element, the action part is considered. XSL actions are simply XML subtrees that are written in the destination document in place of the element being considered.

DOM

The Document Object Model is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Through DOM it is possible to develop applications that use XML documents in a very general way.

Displets for the activation of XML elements

Since XML elements have no predefined meaning or rendering semantics, it is up to the visualization software to provide it. This happens by mapping the document's element names to the program's element names through XSL stylesheets. Thus, the sophistication of the final rendering is independent of the markup, but heavily depends on the sophistication of the rendering program. This is an extremely sensible consideration, but has its weak points: we can write extremely complex documents in XML, but we can display only those that do not have extremely complex rendering needs. In the current use, we can map XML documents onto HTML and display them on standard WWW browsers, but specialized DTDs for strange notation (mathematics and chemistry come to mind) still require a specialized browser to be displayed correctly.

XSL also provides a set of required visualization objects, that is, objects that are known to be available in all XSL implementations, but these only cover standard typographical aspects of the visualization of text, and do not provide support for non standard requirements.

The displet approach [CRV98], on the other hand, provides a generic solution to this problem. The idea behind displets is to create a generic rendering browser that can load and activate little independent software modules tailored to create specific visualization objects. The rendering browser would then activate these modules depending on the content of the document, and deliver all kinds of required rendering needs. *Displets* (*display applets*) such little software modules, designed to be fully interoperable and to be activated by a generic display module.

XMLC

We have designed and created an architecture for rendering displets. In figure 1 we show the general architecture of the application, called XMLC (XML Compiler).

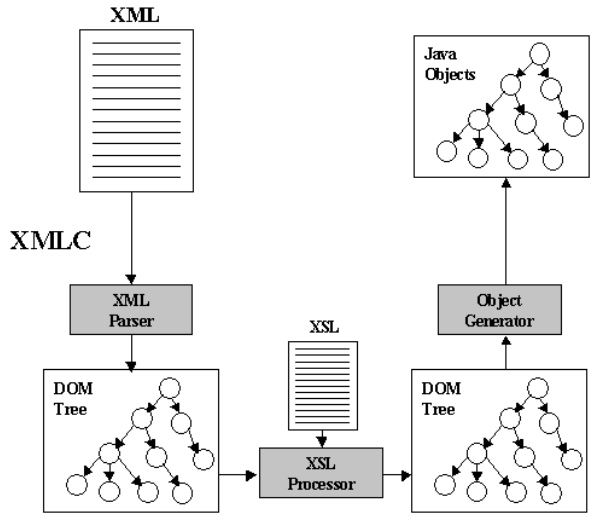

Figure 1: *The general architecture of the XMLC prototype*

The XML document is read and transformed by a normal XML parser into a internal tree representation using DOM. Then one or more layers of XSL stylesheets are applied to the DOM tree through the use of an XSL processor. This transforms the tree, at the end of all these passages, into another DOM tree that has an important property: for every element name in the tree there exist an available Java class, called *displets*, that provides the rendering. The XMLC application will then activate all the Java classes of the required displets, creating a tree of runnable Java objects.

The Java classes are free to be of any type, but using the newly introduced concept of JavaBeans it is easy to create sophisticated and interoperable displets: the use of JavaBeans Containers and Components, which can be easily organized in hierarchies, nicely fits with the hierarchical nature of DOM trees and XML documents.

XMLManager

XMLC is the environment we have implemented for associating Java displets to XML elements and for activating them. XMLC main use is inside a Java applet that renders an XML document within well known Internet browsers, as shown in figure 2.

The main advantage of this approach is that the browser itself does not have a fixed and limited set of visualization objects, but can load and activate any Java class for the purpose of specialized rendering needs. This means that there is no limit to the number of different notations and sophisticated graphic objects that can be satisfied by the XMLC approach.

Furthermore, since the XML elements transform into JavaBeans objects, it is easy to add complex and behaviors during the lifetime of the visualization, providing support for hypertext jumps, animations, interactions with the reader, and in general all the computational capabilities of the Java language. The kind of currently supported notations is the topic of the next sections.

Text, images and links

We have implemented support for text oriented XML elements. The level of support is comparable to that of HTML 1.0 text elements: basic blocks (P, UL, OL and header elements) and inline chunks (like I, B, TT elements, etc.), plus an image tag and a simple inline hypertextual link.

Furthermore, a very simple type of XLink extended links [MD98] has been implemented. The Locator class can display an inline extended link with multiple destinations. When the user clicks on the corresponding word, the link object activates and shows a pop-up menu with all the specified destinations.

The Z notation

A complete support for the Z notation has been implemented (see also [CVM99]). The DTD for the notation we use derives from the ZIF Interchange Format [BN92], but, through the use of additional XSL stylesheets, any other syntax can be used. The support for Z elements is provided through the use of a single displet class, zElement, for the all the boxes that are present in Z specifications (schema, axioms, etc.), and a special downloadable font for all the mathematical glyphs specific of the Z language. All the other elements of the Z language are mapped onto plain HTML elements such as P, DIV and SPAN. An additional layer of XSL stylesheet will then transform them into Paragraph and Word objects as needed. We show here a small fragment of a Z specification (expressed in ZIF):

```
<schemadef style="vert" purpose="state">
  PhoneDB
  <decpart>
    <declaration>
      _known:&pset;NAME
    </declaration>
    <declaration>
      phone: NAME &fpfun; PHONE
   </declaration>
  </decpart>
  <formals> K,L,Z </formals>
  <axpart>
    <predicate>
      known = dom phone
    </predicate>
  </axpart>
</schemadef>
```

This fragment is transformed in a displayable XML document and rendered on the display as in figure 3.

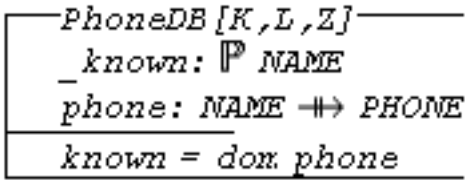

Figure 4: *The display of a simple Z structure.*

TBJava

A prototype that allows XMLC to execute Toolbook books within an Internet browser has been implemented [BOMPANI98]. TBJava translates Toolbook books in XML documents, and displays and activates them in an XMLC applet within a browser, as shown in figure 5.

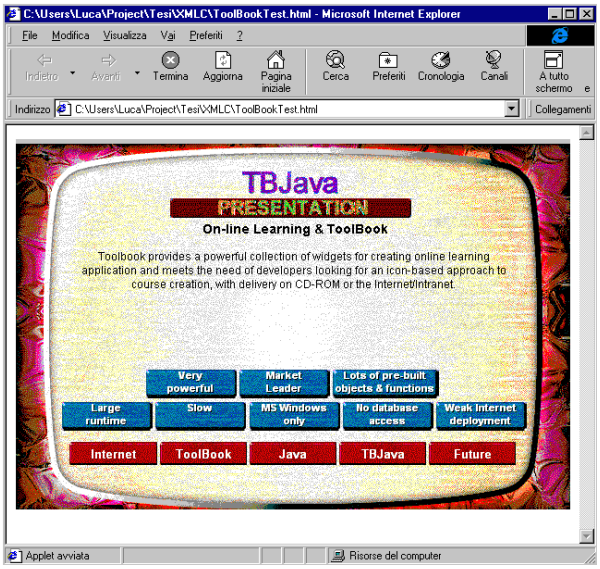

Figure 5: *TBJava, an active displet*

ToolBook is an authoring tool that allow the creation of multimedia applications combining together several types of predefined objects and specifying their behavior with scripts written in a specific language called OpenScript.

TBjava is composed by three different components:

- TBK2XML: a ToolBook filter that transforms ToolBook books into XML documents
- A Java-based OpenScript interpreter
- A set of JavaBeans that implement the ToolBook runtime objects

In figure 6 we show the architecture of TBJava.

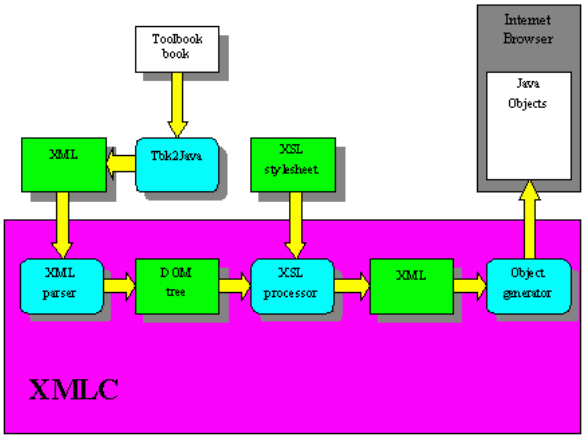

Figure 6: *The general architecture of TBJava*

TBJava relies on XMLC as the runtime environment of the ToolBook book. Since all ToolBook objects have associated properties, behaviors and scripts, XMLC supports them and supports several specific paradigms for message passing and event handling, such as the one of ToolScript and the one of Javascript.

Conclusions

Displets provide an exciting working environment for XML. Since they support but are not limited to traditional text documents, any kind of different applications can be foreseen.

Currently XMLC is a prototype, and thus is not comparable in speed to other, professional applications. We are planning a new implementation of the XSL engine that should considerably increase the speed of the application.

Furthermore, we are continuously enriching the set of displet classes, and are going to provide soon some support for XLinks.

For more information, please consult our site http://www.cs.unibo.it/projects/displets/

Acknowledgements

We wish to thank Cecilia Mascolo, Alfredo Rizzi, Stefano Pancaldi and Milena Roncarati, who helped study, design, implement, and test our implementation.

References

[ABC98] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, T. Research, R. Sutor, C. Wilson and L. Wood *Document Object Model, (DOM) 1.0*, W3C Recommendation 1-October-1998, **http://www.w3.org/TR/1998/REC-**

DOM-Level-1

[BN92] S.Brien and J.Nicholls, *Z Base Standard*, Programming Research Group, Oxford, UK, 1992.

[BOMPANI98] L. Bompani, *Documenti Attivi in XML*, "Laurea" degree thesis, University of Bologna, December 1998 (in Italian).

[BPS98] T. Bray, J. Paoli, C. M. Sperberg-McQueen, *Extensible Markup Language, (XML) 1.0*, W3C Recommendation 10-February-1998, **http://www.w3.org/TR/REC-xml**

[CD98] J. Clark, S. Deach, *Extensible Stylesheet Language, (XSL) 1.0*, W3C Draft 18-August-1998, **http://www.w3.org/TR/1998/WD-xsl**

[CRV98] P. Ciancarini, A. Rizzi and F. Vitali, "An Extensible Rendering Engine for XML and HTML", *Computer Networks and ISDN Systems*, 30(1-7):225-238, 1998.

[CVM99] P. Ciancarini, F. Vitali, C. Mascolo, "Managing complex documents over the WWW: a case study for XML", IEEE Transaction on Knowledge and Data Engineering, 1999, to appear.

[MD98b] E. Maler, S. DeRose, *XML Linking Language (XLink)*, World Wide Web Consortium Working Draft 3 March 1998, http://www.w3.org/TR/WD-xlink