A BROWSER FRAMEWORK FOR HYBRID XML DOCUMENTS

Kari Pihkala, Mikko Honkala, and Petri Vuorimaa
Telecommunications Software and Multimedia Laboratory,
Helsinki University of Technology,
P.O. Box 5400, FI-02015 HUT
Finland

ABSTRACT

Hybrid XML documents are documents, which contain several XML languages, separated by a namespace. Recently, the usage of hybrid documents has increased. The trend has been to specify XML languages as modules, which are combined to construct complete documents. As the number of languages gets higher, a way to flexibly handle these kinds of documents is required. This paper describes a framework for an XML browser to handle hybrid documents. The features of the framework are demonstrated with a hybrid document containing SMIL, XForms, and XML Events.

KEY WORDS: XML, SMIL, XForms, XML Events, X-Smiles, browser.

1. Introduction

World Wide Web Consortium (W3C) has defined a set of markup languages based on eXtensible Markup Language (XML) [1], e.g., XSL Formatting Objects (XSL FO), Synchronised Multimedia Integration Language (SMIL) [2], Scalable Vector Graphics (SVG) [3], XML Events [4], XForms [5], and XHTML [6]. XSL FO is a page layout presentation format. SMIL can be used to define spatial and temporal properties of multimedia applications, while SVG is a similar format for vector graphics and animations. XML Events enables including scripts and logic into XML documents. Finally, XForms is the next generation language for web forms and XHTML is an XML based version of the popular HTML language.

A browser is the most common client to access web applications. Unfortunately, the current commercial browsers have limited support for XML languages. Most of browsers support only few of the XML specifications. This slows the spreading of XML based applications. Therefore, we have developed an open source XML browser called X-Smiles (www.x-smiles.org). The main advantage of the X-Smiles is that it supports most of the W3C XML specifications, e.g., XSL FO, SMIL, SVG, XML Events, XForms, and VoiceXML [7]. Support for XHTML is under work. The browser is implemented in Java, thus enabling porting it to various devices. [8]

Although, different XML based markup languages can be rendered separately in X-Smiles; it can also display hybrid documents. A hybrid document is an XML document, which contains several XML languages, distinguished by a namespace. In this paper, we use the term *host language* to denote the main language of an XML document. A host language usually has the default namespace in the document and it also defines the layout for the document. Typical host languages are XHTML, SMIL, and SVG. A language, which is embedded inside a host language, is called a *parasite language*. XML Events and XForms are examples of parasite languages. These hybrid documents can be validated with hybrid document types [9]. However, building support for them is not as straightforward.

In this paper, we describe a framework, used in X-Smiles, to handle hybrid documents. The idea is that each XML language is implemented as a separate component called *Markup Language Functional Component* (MLFC). Each MLFC knows how to handle a specific XML based language. The framework allows MLFCs to communicate with each other, thus making it possible to handle hybrid documents.

The structure of the paper is the following. The overall X-Smiles architecture is introduced in section 2. In section 3, the framework is presented. Section 4 describes a case study based on the framework, while section 5 draws conclusions.

2. The X-Smiles Architecture

In this section, the overall architecture of the X-Smiles browser is presented.

The X-Smiles XML browser has been implemented to render several XML languages [8]. It is capable of rendering, e.g., SMIL, XSL FO, and SVG documents. This is achieved using several MLFCs, each rendering one specific XML language. Having each MLFC as an

independent component allows them to be added or removed from the browser at will. Figure 1 depicts the overall architecture of the browser. The architecture is composed of four major layers (from bottom to the top): XML processing, Browser Core Functionality, MLFCs, and Graphical User Interfaces (GUIs). At the bottom, the XML Parser and XSL Transformer process the XML documents, converting them into a DOM tree. In the middle, the Browser Core controls the browser's internal state, such as configuration data, document history, etc. The MLFCs render XML languages, as mentioned earlier. There are two special MLFCs (i.e., source and tree MLFCs), which only display the source code of the document. The Graphical User Interfaces (GUI) can be used to customize the browser for various devices.

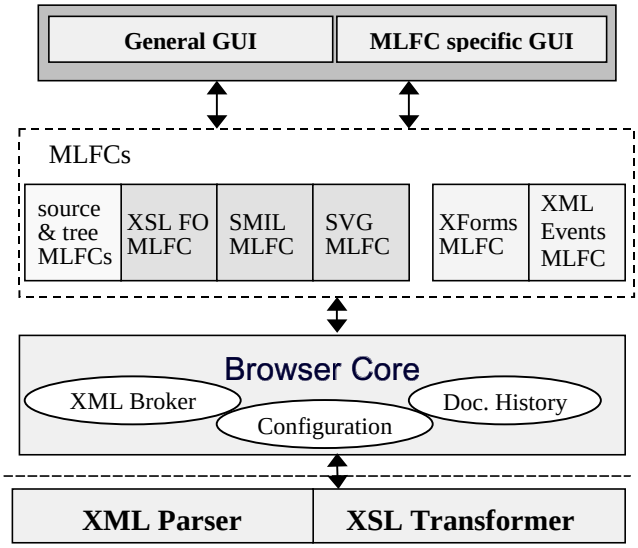

Figure 1. Architecture of the X-Smiles browser.

3. Implementation

In this section, we present the implementation of the framework to handle hybrid documents.

3.1 Overview

The entry point to the framework is the DOM tree construction. The main idea was to be able to use off-the-shelf XML parser or XSLT transformer to perform the DOM tree construction, and therefore be assured of standards compliance, while reading the XML documents. There are the following distinct phases in fetching an XML document:

- *Open Stream*. A stream is opened to the URL of the requested document.
- *Generate DOM*. The XML parser starts to read the stream and generate the XML DOM one node at the time.
- *Transform DOM*. If the XML document contains an XSLT stylesheet reference, it will be transformed with an XSLT transformer into the presentation XML DOM.

Figure 2 describes the X-Smiles modules needed in the framework for fetching an XML document and creating the DOM. The XML parser or XSLT transformer is instructed, using the JAXP [10] interface, to generate an *XSmilesDocumentImpl* instead of normal *DocumentImpl*. *XSmilesDocumentImpl* is derived from *DocumentImpl* and it forwards element creation requests to XMLBroker, which in turn uses MLFCs to create specialized DOM elements. Created elements are stored as the descendants of *XSmilesDocumentImpl*.

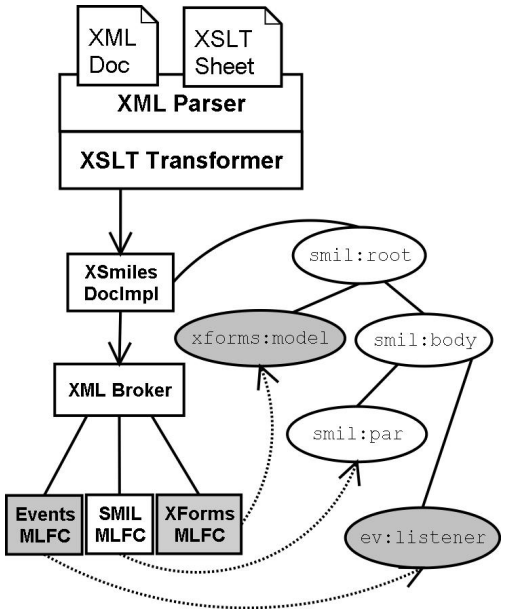

Figure 2. Framework architecture.

3.2 XML brokering

We separated the MLFC registration and general element and attribute creation to an independent module, called XML Broker. XML Broker has three main roles:

- *MLFC Registering*. All MLFCs in the system register themselves to XMLBroker either by a namespace (e.g., 'http://www.w3.org/2000/12/xforms') or by the root element's unqualified name (e.g., 'smil').
- *Dispatching element and attribute creation*. XMLBroker checks the namespace of each element and attribute to be created, and if it matches any of the registered MLFCs, it forwards the request to the corresponding MLFC.
- *MLFC Instantiation*. XML Broker instantiates MLFCs on-demand, and keeps track of the MLFCs that have been instantiated for a document.

The instantiation of MLFCs works in such a way that there is always only one instance of each type of MLFC in a document. The same MLFC (e.g., SMIL MLFC) can register itself under multiple namespaces and root element tag names, but still there is only one instance of it per document. There can be multiple different MLFCs per document (e.g., SMIL MLFC and XForms MLFC). Cf. Section 3.5 discussion about the two types of MLFCs: hosts and parasites.

The root element of an XML document plays a special role, since there can only be one host MLFC per document. It is identified by the root element's name or namespace (i.e., namespace always takes precedence over tag name; element's name support is merely there for legacy XML languages, such as SMIL 1.0, which don' t support namespaces).

3.3 MLFC responsibilities

MLFCs play an important role in the framework. Their responsibilities include:

- *DOM element creation*. MLFC creates language-specific DOM elements, when an XML document is parsed.
- *Element implementation*. MLFC includes specialized implementations of DOM elements.
- *DOM attribute creation*. MLFC creates language-specific DOM attributes, when an XML document is parsed.
- *Attribute implementation*. MLFC includes specialized implementations of DOM attributes.
- *Rendering*. MLFC and the specialized elements and attributes are responsible for rendering themselves.

An MLFC contains DOM element implementations, specialized for each element type. For instance, SMIL MLFC contains implementations for all SMIL elements (e.g., *SMILHeadElementImpl* and *SMILBodyElementImpl*). If the MLFC decides to use a generic DOM element, it returns the element creation request back to XML Broker.

An MLFC can also contain attribute implementations, specialized for each attribute type. Attributes are handled similarly to elements. Again, MLFC can use generic DOM attributes, if specialized attributes are not required.

3.4 Initialization

The initialization phase is used to perform pre-rendering tasks, for instance, XML Events adding event listeners to the DOM tree. One implication of using XML parser and XSLT transformer to generate the specialized DOM tree is that at element creation time, the element's child elements or even its own attributes are not known. Thus, the initialization cannot happen at the DOM creation. The solution is to call a special *initialize()* method for all DOM elements and attributes after DOM creation, when all the elements and attributes are available.

3.5 Host and parasite MLFCs

An MLFC can be a host, a parasite, or both. There is a strict rule: one host MLFC is created per a document. The host is identified by the document's root element and it decides the master layout for the document. The layout model can differ between different host MLFCs. For example, XHTML has a flow type of layout, while SVG uses explicit coordinates for placement.

A parasite MLFC always needs a host to live in. Parasite, such as XForms MLFC, may only define layout for its own elements. Sometimes the parasite does not have visible components (e.g., XML Events MLFC). Figure 2 above depicts the creation of an XML DOM, where SMIL MLFC is the host and there are two parasites: XForms and XML Events. In the figure, the host elements in the DOM have white background, while the parasites have darker background.

3.6 Interaction between the MLFCs

The elements cannot usually live in the DOM without communicating with each other. Consider, for example, an XForms element that lives in a SMIL document. The SMIL document needs to be able to access the graphical component of the XForms element in order to place it on the screen, at the location specified by the SMIL document. Another example is an event listener that must be able to fire event handlers specified by some other language.

One requirement for the framework was that the host MLFC is independent of the possible parasite MLFCs. It must also be possible to add new MLFCs without modifying the existing ones. Therefore, interaction between elements is solved by defining two entities: *Service Provider* and *Service Caller*, as depicted in Figure 3. The Service Provider is an interface that a DOM element or attribute can implement. The Service Caller is another element or attribute that knows how to use the Service Provider's interface. The elements then communicate directly with each other, and not via XML Broker or browser core. Some elements may not need communication, or they may also communicate via DOM events.

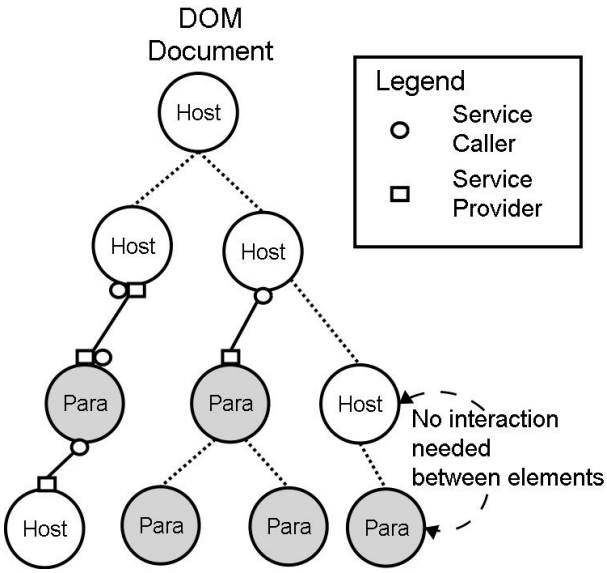

Figure 3. Interaction between DOM elements and attributes.

Service Providers are defined using interfaces. There is, for instance, a general service *VisualComponentService*. A DOM element, which implements this interface, has a method for accessing the visual component as well as activating and de-activating the component, and setting the zoom level. Using these methods the parent element can control the rendering of the component. Both, host and parasite elements, can be Service Providers or Service Callers. For example, XML Events parasite element will call *activate()* for its *EventHandlerService* children, both host and parasite children, when an event is dispatched. The framework itself allows any mix of parasites and hosts (as long as the root is a host element). It is up to the elements to check the type of the parent or child if it needs to communicate with it.

3.7 Limitations and constrains

While the framework described in this article is aimed to be extensible, there are few known limitations to the framework.

First, there is currently no support for specialized document implementations. The framework will always create a default document implementation for the XML document, *XSmilesDocumentImpl*. In some cases, it would be desired for the MLFC to be able to create its own *DocumentImpl*. For example, a HTML-DOM Document interface contains some methods that are not possible to implement with the current version of the framework. This limitation is a result of not knowing, at document creation time, what will be the host MLFC. A possible resolution would be to delay the creation of the *DocumentImpl* until the root element is read from the XML stream.

Secondly, initialization phase may cause performance degradation. The DOM tree needs to be initialized after it has been fully constructed. This may make the framework inefficient for large documents, since the whole document needs to be retrieved and transformed into a DOM tree until it can be rendered. A resolution would be to initialize and render parts of the tree, while constructing the whole tree. SAX interface could also be used to create and initialize the tree.

4. A Case Study: SMIL, XForms, and XML Events

In this section, we describe how the described framework handles a hybrid document containing SMIL, XForms, and XML Events.

4.1 The hybrid document

As an example service, an imaginary car sales service was created, depicted in Figure 4. The user can select desired values using the controls on the left, and the changes will be reflected in the car model shown on the right. The user can press the “Order” button to send the values to the server. To achieve this, the form controls are written in XForms language, capturing of events and control logic in XML Events and ECMAScript, and the layout and visual appearance in SMIL. SMIL is the host language providing the layout, while XForms and XML Events are parasite languages.

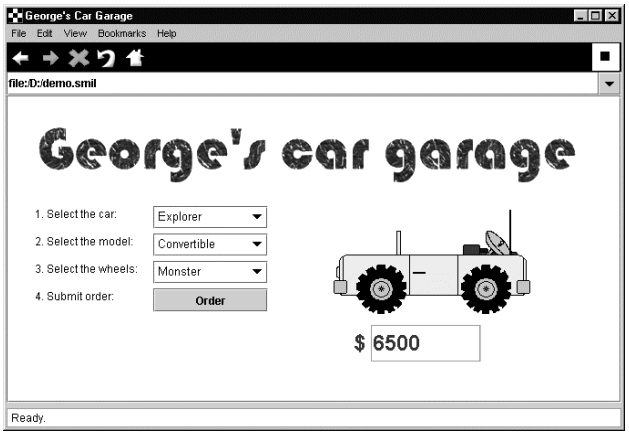

Figure 4. A simple multimedia service.

4.2 SMIL MLFC

The SMIL MLFC in the X-Smiles browser is capable of rendering SMIL 2.0 Basic language [2] documents. The MLFC has been designed to be dynamic (i.e., run-time changes in the SMIL elements will be reflected in the presentation). Thus, a script can modify the appearance of the presentation.

In the example, SMIL controls the spatial and temporal dimensions of the document, providing the layout and timing information for all the elements in the document.

4.3 XForms MLFC

The XForms MLFC implements most of the XForms working draft. Since the specification does not define any layout, the language cannot be used as a host language. Instead, it can be used as a parasite language, letting a host language decide the layout of the XForms controls [11].

In the example, XForms is used to provide the form controls on the left. When the user selects items from the controls, the instance data is updated, and the price is automatically calculated. Instance data is submitted to the server, when the "Order" button is pressed. Figure 5 depicts how XForms is embedded in SMIL. The XForms control is used like a SMIL media element, under the "par" element. The "par" element will define the region and timing information for the control.

```
 <text region="text1" src="data:,1.
Select the car:" begin="1s"/>
 <par region="sel1" begin="1s"
         ev:event="DOMActivate"
         ev:handler="#audiohandler">
   <xfm:selectOne xform="form1"
         ref="order/car"
         selectUI="checkbox">
     <xfm:item value="buggy">
                    Buggy</xfm:item>
     <xfm:item value="explorer">
                  Explorer</xfm:item>
     <xfm:item value="formula">
              Formula 3000</xfm:item>
   </xfm:selectOne>
 </par>
```

Figure 5. Snippet showing XForms elements.

4.4 XML Events MLFC

XML Events MLFC provides scripting functionality in the X-Smiles browser. It assumes that the other MLFCs will dispatch events to the DOM tree. It listens to them and accordingly evaluates event handlers. The event handler code is included in an additional element called "script". Currently, it has not been defined in the XML Events specification, but it has been implemented in our solution to run ECMAScript. ECMAScript can be used to modify the DOM elements, thus controlling the run-time presentation look. XML Events can either be parasite elements or parasite attributes.

In the example, XML Events is used to tie changes made in the form controls to the presentation. Listeners wait for XForms events and run a handler, when a change event is observed. The handler is made of a piece of ECMAScript, which changes the "src" attribute of the audio element. Figure 6 depicts how the audio element is changed with ECMAScript according to the selected car model. The event listener is defined as attributes for the par element shown in Figure 5.

```
<!-- This changes the audio -->
<ev:script id="audiohandler"
            type="text/ecmascript">
  ...
  // Change the audio element
  aud=document.getElementsByTagName(
                   "audio").item(0);
  if (car == "buggy")
     aud.setSrc("buggytus.wav");
  if (car == "explorer")
     aud.setSrc("expl.wav");
  if (car == "formula")
     aud.setSrc("metal.wav");
</ev:script>
<audio id="audioplay"
             src="buggytus.wav"/>
```

Figure 6. Snippet showing how ECMAScript changes the audio track.

4.5 Interfaces

Figure 7 depicts the interfaces of the elements and attributes used in the car demo. Only the SMIL elements are host language elements, other elements are parasites. SMIL media elements can be both. Two interfaces are in use, *VisualComponentService* and *EventHandlerService*. The SMIL time containers (e.g., seq and par), are Service Callers, being able to display any element implementing *VisualComponentService*. Of course, the time containers can also contain the usual SMIL elements.

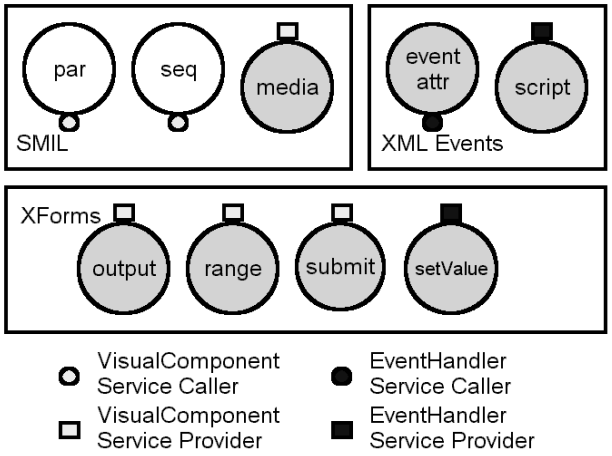

Figure 7. Implemented MLFC interfaces.

The XForms control elements (e.g., output, range, and submit) implement a *VisualComponentService*, allowing any host language to control their appearance. In this case, the SMIL time containers were used. The control elements also send a DOMActivate event, when their

value is changed. The setValue element also implements an *EventHandlerService* interface allowing it used as an event handler.

The XML Events "listener" element listens to DOM events, in the example, the DOMActivate event. The element is a Service Caller, assuming that the event handler will provide *EventHandlerService*. In our implementation, either "script" element or XForms declarative elements can be used as event handlers.

In addition to the presented interfaces, it is possible to embed XForms elements and SMIL media elements as parasites in XHTML, SVG, and XSL FO documents. Also, XML Events can be included in XHTML, SVG, and XSL FO documents.

5. Conclusion

The XML language specifications are evolving towards small sets of elements, which will be combined in hybrid documents. Such an approach has already been taken with XForms and XML Events. Also, modularization of languages will likely produce hybrid documents.

This paper presented a framework to render hybrid documents. The X-Smiles browser uses MLFCs to render XML languages. Each MLFC can render one XML language. A central module, called XML Broker, is used to forward parsed XML tags and attributes to the associated MLFCs. An MLFC creates DOM elements and attributes, which are then appended to a DOM tree. This results in a DOM tree with specialized DOM elements and attributes, each providing functionality for itself. The elements and attributes can communicate with each other via interfaces. This enables embedding one language in another.

As an example, a simple SMIL document was created with XForms and XML Events embedded in it. The document was then rendered using relevant MLFCs, showing how they interact with each other. This showed that the currently implemented interfaces are quite simple, but still powerful. They allow elements to modify others layout, to define timing for displaying, and can fire actions.

The main benefit of the given framework is that it is highly modular. New MLFCs can be created independently, still allowing them to interact with the old ones. New MLFCs can also take advantage of already created MLFCs' functionality. This is in line with the current effort made at W3C to create reusable XML languages and modules. The framework also guides the internal structure of the MLFCs to be robust, and to follow standards.

However, the framework has few drawbacks. Currently, document objects won't implement the standardized interfaces. Also, there may be some performance degradation, because of the initialization phase of the DOM tree. Both of these limitations will be addressed in a future version.

In the future, the need for more interfaces is obvious. The interfaces in the current MLFCs will be revised, and new interfaces will be developed, keeping them as generic as possible. After all, they are the key for XML languages to co-operate. Also, more MLFCs for various XML languages can be implemented. For instance, MathML as a parasite language, to embed it in any host language.

6. Acknowledgement

The authors Kari Pihkala and Mikko Honkala would like to thank Nokia Oyj Foundation for providing the scholarships and support during the research work.

References

[1] T. Bray et al., Extensible Markup Language (XML) 1.0 (second edition), *W3C Recommendation*, Oct. 6, 2000, http://www.w3.org/TR/2000/REC-xml-20001006.
[2] J. Ayars et al., Synchronized Multimedia Integration Language (SMIL 2.0), *W3C Recommendation*, Aug. 7, 2001, http://www.w3.org/TR/smil20/.
[3] J. Ferraiolo et al., Scalable Vector Graphics (SVG) 1.0 Specification, *W3C Recommendation*, Sept. 4, 2001, http://www.w3.org/TR/SVG/.
[4] S. McCarron et al., XML Events, *W3C Working Draft*, Oct. 26, 2001, http://www.w3.org/TR/2001/WD-xml-events-20011026/.
[5] M. Dubinko et al., XForms 1.0, *W3C Working Draft*, Jan.18, 2002, http://www.w3.org/TR/2002/WD-xforms-20020118/.
[6] S. Pemberton, XHTML 1.0: The Extensible HyperText Markup Language, *W3C Recommendation*, Jan. 26, 2000, http://www.w3.org/TR/xhtml1/.
[7] VoiceXML Forum, Voice eXtensible Markup Language version 1.0, *W3C Note*, May 5, 2000, http://www.w3.org/TR/voicexml/.
[8] P. Vuorimaa et al., A Java based XML browser for consumer devices, *the 17th ACM Symposium on Applied Computing*, Madrid, Spain, March 10-13, 2002, pp. 1094-1099.
[9] M. Altheim et al., Modularization of XHTML, *W3C Recommendation*, Apr. 10, 2001, http://www.w3.org/TR/xhtml-modularization/.
[10] R. Mordani et al., Java API for XML Processing, Version 1.1, *Java Community Process Specification*, Feb. 6, 2001, http://java.sun.com/xml/jaxp-1_1-spec.pdf.
[11] M. Honkala and P. Vuorimaa, XForms in X-Smiles, *WWW Journal*, *4*(3), 2001, pp. 151-166.