# XForms in X-Smiles

MIKKO HONKALA and PETRI VUORIMAA Mikko.Honkala@iki.fi; Petri.Vuorimaa@hut.fi
*Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, Finland*

## Abstract

World Wide Web Consortium is currently specifying XForms form language, which is intended to be the next generation language for WWW forms. In XForms, content is separated from presentation. Thus, forms can be versioned easily for different purposes. In addition, the form fields can be pre- and auto-filled. Finally, XForms allows the validation of the data at the client side before it is send to the server. The major browsers do not currently support XForms, which prevents the early utilisation of XForms. In this paper, we describe our implementation of XForms. The implementation is part of our X-Smiles browser, which is an open source XML browser.

**Keywords:** XML, XForms, SMIL, SVG, XHTML, browser.

## 1. Introduction

Since the advent of the World Wide Web (WWW), forms have been a critical building block in interactive WWW applications. They are the main components for capturing user input and transferring it to a content server. In the current WWW paradigm, (which dates back to the original NCSA Mosaic GUI for X Windows, circa 1994 [13]) the form field contents are sent to a CGI script or a servlet as name-value pairs.

XForms is an effort by the World Wide Web Consortium (W3C) to replace the HTML forms with more advanced format. XForms is an Extensible Markup Language (XML) [5] based language for describing forms on different user devices.

The current design of WWW forms doesn't separate the purpose from the presentation of a form. XForms is, in contrast, comprised of separate sections that describe what the form does, and how the form looks like. This allows flexible presentation options, including classic XHTML forms [15], to be attached to an XML form definition [9]. XForms uses XML to describe a piece of instance data, to which form controls are linked. The user-modified XML instance is then validated and submitted back to the content server. One shortcoming of the current WWW forms is the fact that field validation is only possible through (often incompatible) ECMAScripts [10]. XForms has a well-defined model for form content validation.

XForms is designed to work together with different XML based markup languages. The XForms working group mentions XHTML [15], Synchronised Multimedia Integration Language (SMIL) [12], and Scalable Vector Graphics (SVG) [11] as XForms embedding languages. Therefore, XForms can be used as the form language in all kinds of WWW applications.

Unfortunately, the number of XForms implementations is very limited at the moment. None of the so-called major browsers (e.g., Microsoft Internet Explorer 6, Netscape Navigator 6, and Opera 5) supports XForms. This prevents the utilisation of the XForms in WWW applications.

X-Smiles [22] is an XML browser intended for consumer devices, e.g., Personal Digital Assistants (PDA), mobile phones, and digital television set-top boxes (STB). It has been implemented using Java language and it is available as open source at `http://www.x-smiles.org/`. The main advantage of X-Smiles is that it supports most of the XML based presentation languages (i.e., XSL FO [1], SMIL [12], SVG [11], and VoiceXML [21]). Currently, XHTML [15] implementation is under work. Thus, X-Smiles is an ideal platform for XForms; different Java based components can be utilised in XForms implementation and XForms can be integrated with different XML based presentation languages.

This paper is organised as follows. In the next section 2, we discuss the features of the XForms specification in more detail. Next, in section 3, we present the X-Smiles browser and its main architecture. After that, in section 4, we describe the XForms implementation of X-Smiles. Section 5 discusses embedding XForms in different markup languages. Finally, conclusions are given in Section 6.

## 2. XForms language

The XForms has three main layers: *Instance*, *Model*, and *User interface* [9], as depicted in Fig. 1. The User Interface layer can be divided to *Form Controls* and the *Embedding Document*. Khare names the layers in a more abstract way: *Data*, *Logical*, and *Presentation* [13].

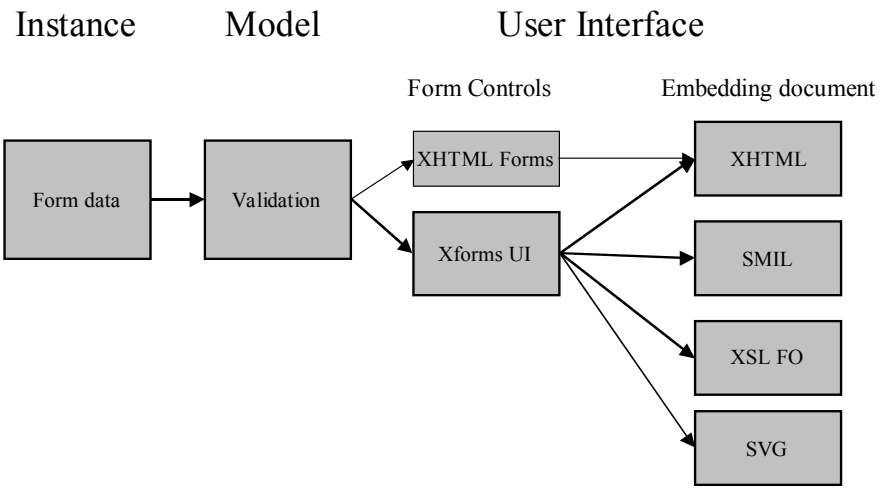



**Fig. 1.** XForms layers.

The *Instance* contains only the data. No presentation or validation information is included. Thus, the instance can be used to store and transfer the data between clients, servers, and applications. It is defined as either an XML document fragment inside the embedding document or as an external XML entity referenced by an XLink [8].

The *Model* provides validation and inter-data constraints for the form. The validation uses the XML Schema [18] as its language. The schema can also be defined either inline or referenced via an XLink. Constraints, such as calculations, are defined using XPath [6]. The model layer provides a powerful way to describe, what is allowed in the form, without

resorting to scripting. With the current WWW forms, scripting is the only way to validate form field or express calculations. Because XML Schema and XPath are used as the languages, the same constraints can validate form content both at the client and server side.

In the *User Interface* layer, one can apply either XHTML form controls [15] or XForms UI controls. XForms also re-uses some XHTML modules, namely the "event" module, rather than re-inventing its own event handlers. For styling, XForms relies on Cascading Style Sheets (CSS) [3].

The *Embedding Document* can be any kind of XML document, which allows embedding XML elements in other namespaces. Thus, XForms can be embedded, e.g., in XSL FO, XHTML, SMIL, or SVG document.

## 3. The X-Smiles XML browser

This section describes briefly the X-Smiles browser and its components. The XForms implementation is one component of the browser.

The architecture of the X-Smiles browser is depicted in Fig. 2. The architecture is composed of four major layers (from bottom to the top): *XML Processing*, *Browser Core Functionality*, *Markup Language Functional Components (MLFC) & ECMAScript Interpreter*, and *Graphical User Interfaces (GUI)*. Each layer is described in the following.

The XML processing module contains the *XML Parser* and *XSL Transformer*. The XML Parser parses the XML documents. The XSL Transformer processes the document according to the related XSL stylesheet, if applicable. The output is stored into a data structure, which can be accessed via the *Document Object Model (DOM) Interface* [17].

After the DOM tree has been constructed, it has to be rendered. Since there are different XML languages (e.g., MathML, ChemicalML, SMIL, XSL FO, and SVG) available, different rendering modules have to be used. In X-Smiles, this is achieved by using different MLFCs. Each MLFC knows how to render one specific type of XML language. The different MLFCs can be used at the same time. For example, an XSL FO document can contain an inline reference to a SMIL document. So far, we have implemented an MLFC for SMIL, XML FO, XForms, Voice XML, and SVG. Currently, we are working on XHTML.

The core of the browser ties the different modules together. The main parts of the core are *MLFC Management and Retrieval*, *ECMAScript Handler, Event Broker, General Functionality*, and *Browser Configuration*. The MLFC Management and Retrieval unit takes care of loading the appropriate MLFC, which can also be retrieved over the network. The Event Broker forwards the events to other units. The ECMAScript Handler co-ordinates the operation of the ECMAScript interpreter. The General Functionality unit controls the GUI, document history, etc. Finally, the Browser Configuration unit is responsible for management of the different features of the browser such as the parser and stylesheet processor, home page, etc.

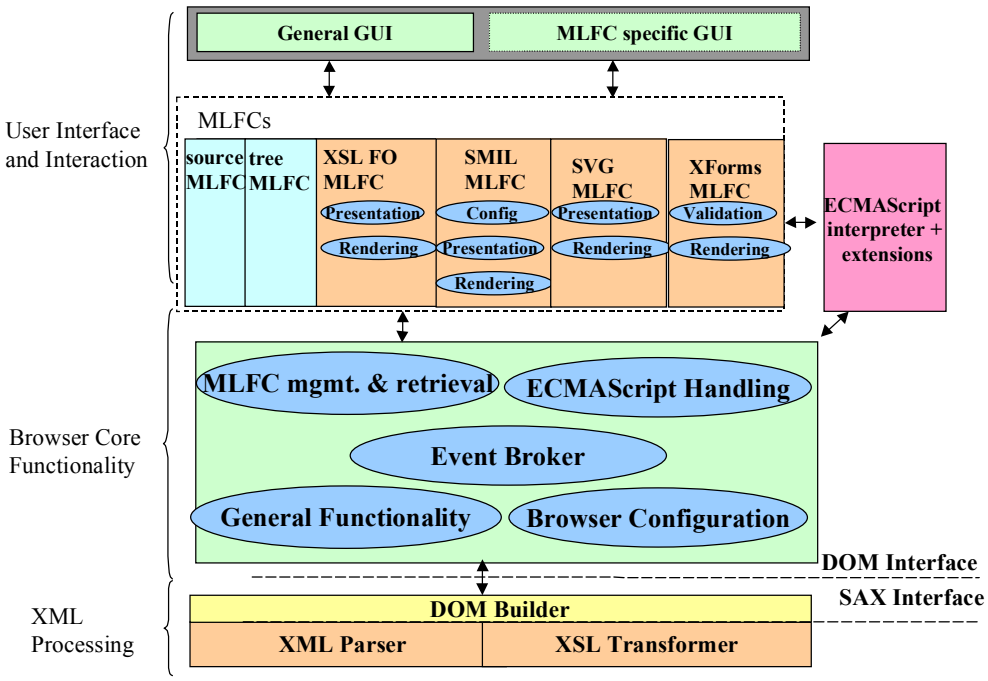



**Fig. 2.** The internal architecture of the X-Smiles XML browser.

The *ECMAScript Interpreter* runs the scripts contained in the XML document. ECMAScript provides service developers a convenient method to implement interactivity. The current ECMAScript interpreter is Rhino [4].

The original X-Smiles GUI was developed for desktop PCs. We have also developed special user interfaces for digital television [20], PDAs, and mobile phones. The different GUIs are interchangeable, and the user can switch them even on the fly.

## 4. Implementation of XForms in X-Smiles

This section describes the technical implementation issues of the XForms MLFC in the X-Smiles browser. The XForms MLFC handles interactive WWW forms. One simple example of such a form is shown in Fig. 3. The figure shows the user configuration of X-Smiles browser, which was implemented using XML, XSL, and XForms.

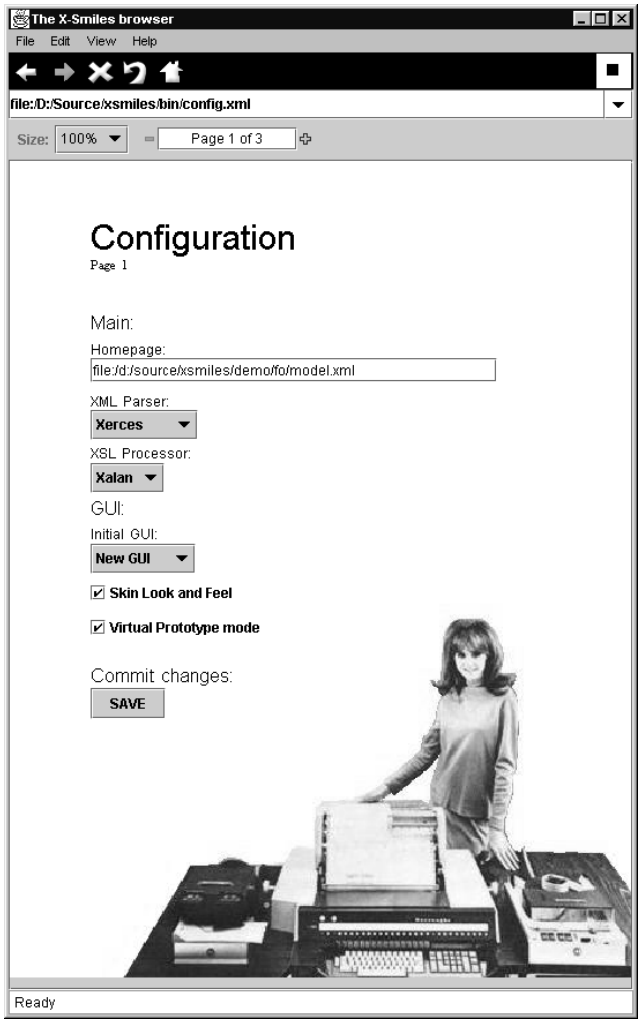



**Fig. 3.** Example of XForms in X-Smiles.

The role of MLFCs is important in X-Smiles. Usually, each MLFC has it own presentation DOM, which contains information specific to the particular markup language. For example, the SMIL presentation DOM contains timing information. The MLFCs also control, how the markup languages are rendered on the screen. XML parser is used just to read the XML stream, and to create element and attribute creation events, much like a simple SAX parser.

Therefore, to implement XForms in X-Smiles browser, we first had to implement the XForms MLFC. Main technical issues that had to be taken into account in the design were:

- How will we draw the form controls? Can/should we use some user interface package?
- Embedding in different markup languages (e.g., XSL FO, SMIL, and SVG) should be made easy.
- Binding between the instance and the GUI so that changes in the instance will be reflected in the UI and vice versa.
- ECMAScript binding and mouse events. It should be possible to access the form data with scripts and add mouse event handlers to XForms elements.
- Validating the form after user has inputted values to the form. Is it possible to use off-the-self validating parser to do the job?

- Enabling the use of CSS2 to style the XForms elements. Is it possible to use any CSS tools for this problem?

The rest of this section focuses on solutions for these questions. Before that, we will describe the basic steps required for the processing of XForms documents, though.

### *4.1. Processing of XForms Documents*

Usually, XForms are not used alone. Rather, they are embedded in a host document. The processing of a host document containing a XForms is done as follows. First, the X-Smiles browser reads an XML Document. During this step the XML and the related XSL file are processed and the appropriate presentation DOM (e.g., SMIL DOM) construction is started.

The required MLFCs are determined by a so-called *XML Broker*. The broker first finds out the host markup language (e.g., XHTML, XSL FO, SMIL, or SVG) and hands the element creation request to the respective MLFC. During the process, the XML Broker encounters the XForms namespace, and hands the XForms element creation over to the XForms MLFC. After the whole tree has been constructed, it is initialized recursively.

The XForms MLFC creates also the internal instance DOM, which contains the form data, as depicted Fig. 4. In addition, it creates the form controls. Since the XForms is rendered by the host MLFC, the controls are actually displayed in the host presentation DOM. Finally, the form controls are attached to the corresponding instance data nodes. The nodes are found by executing the XPath expression in the form control's "ref" attribute. Xalan's XPath engine is used to execute XPath expressions.

The instance data is validated according to the validation model. This can be done during the XForm is filled and finally, when it is submitted. The validation is based on the XML Schema instructions. X-Smiles uses the validating XML parser Xerces to implement validation within XForms.

### *4.2. XForms User Interface*

The XForms specification [9] defines a certain set of basic form controls, such as buttons and drop down lists. Thus, we had to implement them in the XForms MLFC. There were three possibilities to implement the form controls (keep in mind that X-Smiles is a Java application):

- Create new Java user interface module from scratch
- Use Java Abstract Windowing Toolkit (AWT) components, which in turn use the system UI components
- Use Java Swing components, which are pure java

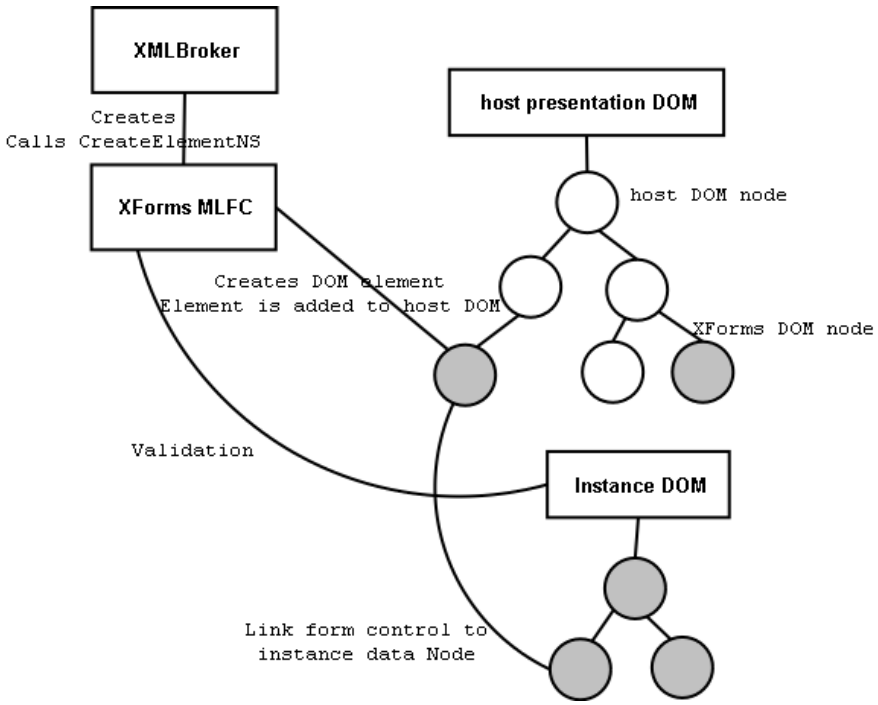



**Fig. 4.** XForms Document Object Models.

From these alternatives, the first one was abandoned, because it would have been too laborious to create the widgets and the corresponding event structures from scratch. The second alternative was first tried, but later replaced by the third one for the following reasons:

- Swing widgets look exactly the same in all platforms
- Swing offers more flexible styling and events
- Swing is used already by the X-Smiles browser

Most of the form controls defined in XForms [9] were implemented in X-Smiles. Currently, only "group" is not implemented.

The drawback of using Swing is that it may not be available in restricted environments that do not fully support Java2. There is a version of Swing that can be used in such environments, but it takes about 2 MB of static storage memory, which may be too much in some cases. Using another component framework remains a future work item.

### *4.3. Binding between the Instance and the GUI*

In an XForms implementation, the instance and the GUI should have a close relationship; whenever a user makes changes in the GUI, those changes should be updated to the instance. And vice versa: if parts of the instance change, the GUI must change accordingly. For the latter, consider the following scenarios:

- Two GUI controls are bound to the same piece of instance data. When a user makes a change in the other, the other must change accordingly.
- An ECMAScript makes changes in the instance: the GUI controls bound to the changed parts must display the changes.

These scenarios lead to one important requirement: the ability to get events about modifications of the DOM. This is just what DOM Events [17] define. Luckily, the DOM implementation used in X-Smiles (i.e., Xerces) implements these events, so all that was left to do was to take these events into use. We added the GUI components as DOM event listeners for the DOM elements they are bound to.

Another way to accomplish this is to make a specialized instance DOM, to which form controls are registered. Whenever a value in the instance DOM changes, the appropriate form controls are signaled about the change. This is a future work item for XForms that will eliminate the need for DOM event implementation.

### *4.4. ECMAScript Binding and Mouse Events*

In the XForms implementation, we wanted ECMAScripts [10] to have full read/write access to the form data. The most natural way to accomplish this is to have the access through the DOM, because in X-Smiles the ECMAScripts already have DOM access (implemented using the LiveConnect feature of Rhino ECMAScript engine [4]). We identified three possibilities for the ECMAScript binding:

- Extend DOM for the "XForm" element. This would provide unified access to both external and internal instances, but would need a proprietary DOM extension.
- Use the inline DOM as the instance data. This option does not require DOM to be extended, but external instance would not be accessible.
- Extend DOM for the "XFormsControl" element. This option would provide unified access, but would not separate content with presentation, since the content is accessed through the presentation.

After analysing these options, we decided to implement the first option: "Extend DOM for the XForm element". This option has only one drawback; we had to specify proprietary DOM extension, since XForms does not have a specified DOM. Because of the extension, scripts can read and modify the instance data through the embedding documents DOM and the corresponding form controls change their display accordingly because of the DOM events generated by the DOM implementation, respectively.

### *4.5. Validating the User's Input*

XForms specification uses XML Schema [18] to validate the instance data and thus the user's input. XML Schema is a relatively large specification, and therefore we decided to use a third party XML Schema validator. The choice of Xerces was obvious, because it is already used as an XML parser and DOM implementation. After an analysis we came up with a few requirements for user input validation using the schema:

- It must be possible to re-validate the instance after it has changed.
- It would be nice to be able to "guide" the user to input only valid data. For example, if an instance data element's data type is "date", the user should not be able to enter anything else than a date. And furthermore, it should be possible to enter the date in a format specified by the user's locale.

It became apparent that the first requirement was possible to implement with the current version of Xerces. On the other hand, the second requirement was not possible, because there was no way to get schema-related information from Xerces.

Current version of Xerces does not give access the Post-Schema Validation Infoset (PSVI), which makes it impossible to revalidate only part of the instance. Because of this, the whole instance must be validated every time a value changes. This makes the validation quite inefficient. A future version of Xerces will probably offer an access to the PSVI.

### *4.6. CSS Styling*

XForms relies on CSS2 [3] for styling the form controls. For example, colours, font properties, and sizes are defined with CSS2.

There are three layers to CSS: parsing, selecting, and styling. The parser's responsibility is to read the CSS style and parse it into a CSS object model. The selecting means that appropriate CSS objects for the DOM element to be styled are selected. Finally, styling applies the style objects for the element.

We implemented CSS style attributes directly in the DOM elements, so the selecting was not necessary to implement. We are extending the implementation, so selecting will also be implemented in the future.

We chose to use a ready-made CSS2 parser by Steady State [19] to parse the CSS2 data within the elements. The parser itself is not very complex. It could have been written also from scratch. Finally, we built a styling module, which takes a style string and a Java UI Component as input. The module then parses the style string and styles the component accordingly. Only colour and text-decoration parts of the CSS2 were implemented.

### *4.7. Submitting*

Submitting means sending the completed form back to a WWW application running at a WWW server. The XForms specification [9] defines three ways of submitting the form content: *application/x-www-form-urlencoded, multipart/form-data,* and *text/xml*. We have implemented only *text/xml,* but implementing the rest is not difficult. In *text/xml*, the updated instance data is serialized and sent to a location defined in the "submitInfo" element. The serialization is performed in X-Smiles by using Xalan XSLT processor through the Java XML Processing (JAXP) [14] interface. The resulting XML stream is then sent using regular HTTP POST operation.

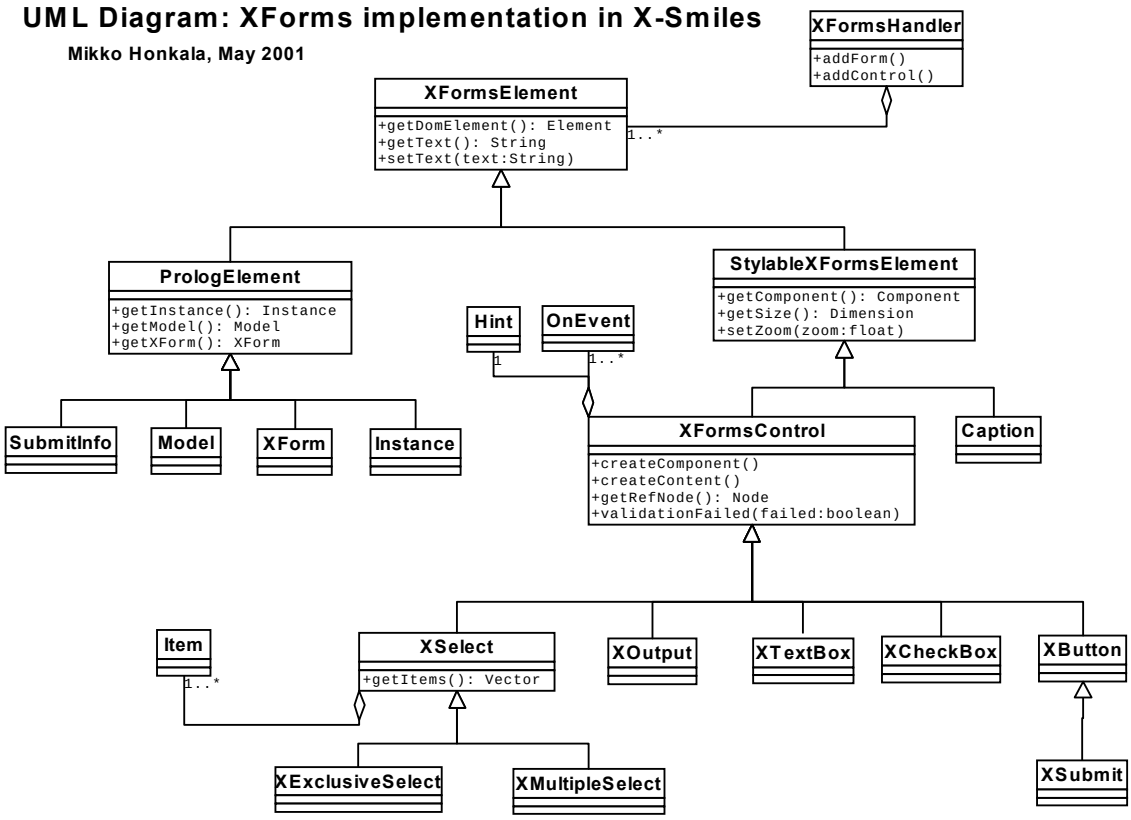



**Fig. 5.** UML diagram of the class structure.

### *4.8. Class Structure*

XForms implementation was designed in an object oriented fashion. Fig. 5. depicts the UML diagram of the XForms implementation. Class *XFormsHandler* has multiple *XFormsElements*, which can be either *PrologElements* of *StylableXFormsElements*. *PrologElements* include *SubmitInfo*, *Model*, *XForm*, and *Instance*, while *StylableXFormsElement* can be either *XFormsControl* or *Caption*. In turn, *XFormsControl* can be any of the XForms form control classes.

## 5. Embedding XForms in Different Markup Languages

This section describes the embedding of XForms into different host markup languages. One goal of W3C XForms Working Group is to be able to define such markup that can be embedded in different type of XML documents. The Working Group lists the following languages: SMIL, SVG, and XHTML. In addition, the group's WWW pages mention Voice Browsers. In X-Smiles, we implemented XForms support in XSL FO, SVG, and SMIL.

### *5.1. XSL Formatting Objects*

The XSL FO MLFC displays XSL Formatting Objects documents [1] and is based on Apache FOP project [2]. We extended FOP to allow XForms elements within the "fo:instream-foreign-object" element as shown in the listing below:

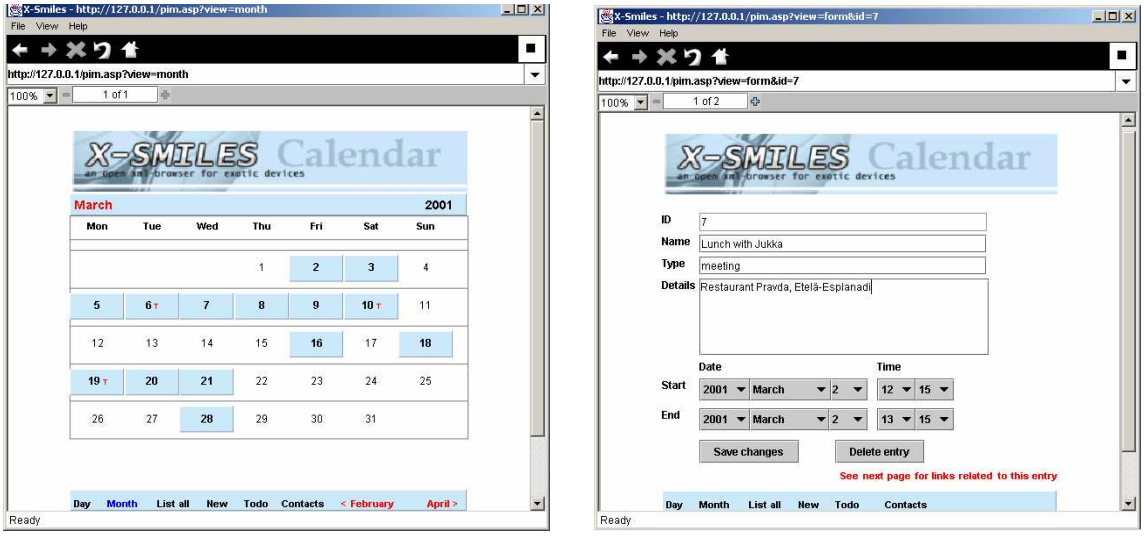
```
<fo:instream-foreign-object space-after.optimum="15pt">
 <xfm:exclusiveSelect ref="config/main/xslprocessor" style="list-ui:checkbox;"
     xform="form1">
  <xfm:item>Xalan</xfm:item>
  <xfm:item>XT</xfm:item>
 </xfm:exclusiveSelect>
</fo:instream-foreign-object>
```

**Fig. 6.** Code snippet for XForms in XSL FO.

The drawback with the current solution is that FOP handles foreign-objects as block objects rather than inline objects and places them always on a new line. Therefore, table constructs have to be used often.

#### *5.1.1. Personal Information Manager (PIM) Example*

The Personal Information Manager (PIM) is an extensive XML application made for the X-Smiles browser. It utilizes a server-side XML database and server-side scripting. The pages are send as XML documents to the browser, which transforms them into renderable XSL FO pages with an accompanied XSL stylesheet. XForms that are embedded in XSL FO documents are used widely in the application.




(a) (b)

**Fig. 7.** (a) Month display of the PIM and (b) XForm to fill a meeting entry.

Fig. 7. shows two screenshots of the PIM application. The Fig. 7. (a) displays the month view of the PIM. The user can select individual days for viewing by clicking the respective cell with a mouse. At the day view, the user can modify the meetings, to-do items, etc. by opening a form depicted in Fig. 7. (b). The figure shows a meeting entry displayed as an

XForm. The user can alter the content, which is then sent to the server script as an XML document by the XForms implementation and consequently saved to the server-side XML database.

### *5.2. SVG*

The SVG MLFC displays Scalable Vector Graphics [11]. It is based on CSIRO SVG toolkit [7]. We extended the CSIRO toolkit with support for namespaces and the "foreignObject" element. XForm elements are placed within the "foreignObject" element, which also gives the coordinates for placement on the screen (cf. Fig. 9. ).

#### *5.2.1. Searchable Map Example*

The Searchable Map example, which is depicted in Fig. 8. , shows a simple client-side use case for XForms within a SVG image. The image is a zoomable map of Finland. The user can type in a name of a city she wants to locate. The input field is implemented as a XForms and embedded in the host SVG document.

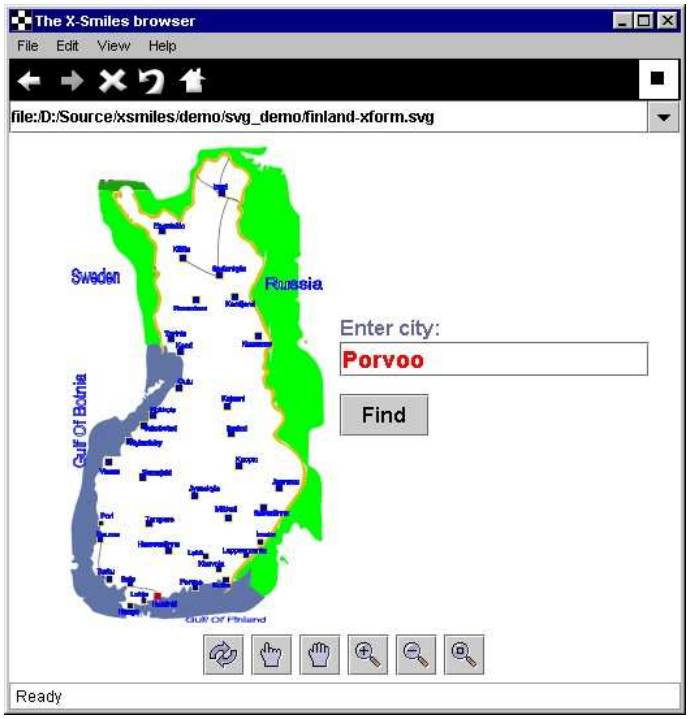



**Fig. 8.** Map search screenshot.

An ECMAScript within the graphics document gets the user's input, and searches for the city name in the graphics. If the city is found, the map is zoomed and the city is highlighted with a bigger red font. If the city is not found in the map, the user is prompted with a pop-up window.

Fig. 9. shows a code snippet from the example. First, the location of the SVG image is defined. After that, the XForm is defined as a "foreignObject" element. The XForm contains the submit and instance information. The model is not necessary in this case. Finally, the location of the XForm is defined. At the same time, textbox control is linked to the instance data.

```
<svg height="474" viewBox="-100, 140, 450, 450" width="257"
          xmlns:xfm="http://www.w3.org/2000/12/xforms">
<foreignObject>
  <xform id="form1" xmlns="http://www.w3.org/2000/12/xforms">
    <submitInfo id="submit1" localfile="temp2.xml" />
    <model>
    <!-- The model is currently ignored -->
    </model>
    <instance id="instance1" xmlns="">
      <mapsearch>
        <city>Sodankyla</city>
      </mapsearch>
    </instance>
  </xform>
</foreignObject>
...
<!-- An UI element -->
<foreignObject height="30" width="200" x="210" y="125">
  <xfm:textbox cols="14" ref="mapsearch/city" xform="form1"/>
</foreignObject>
...
</svg>
```

**Fig. 9.** Code snippet for map search example.

### *5.3. SMIL*

The SMIL MLFC [16] displays SMIL documents and is written completely by the X-Smiles team. It uses swing components to display text and images and Java Media Framework (JMF) for video and audio.

XForms elements can be placed in the SMIL document, where viewable content is allowed, mainly within the “par” and “seq” elements. The XForms elements are recognised by their namespace. SMIL attributes “begin”, “dur”, and “region” are embedded in the XForms elements as shown in the following code snippet:

```
<par id="Main">
 <img alt="XForms in SMIL" begin="0s" id="nolla" region="eka" src="title.gif"/>
 <audio begin="1s" id="1st" reqion="sound" src="tick.wav"/>
 <xfm:textbox begin="1s" cols="30" dur="15s" ref="purchaseOrder/shipTo/name"
          region="toka" rows="1"/>
</par>
```

**Fig. 10.** Code snippet for XForms in SMIL example.

### *5.3.1. Timed Form Example*

This simple XForms in SMIL example shows an XForms form in a SMIL presentation. The elements appear sequentially and also their duration is specified, so the user has only a defined amount of time to fill the form and press the "submit" button.

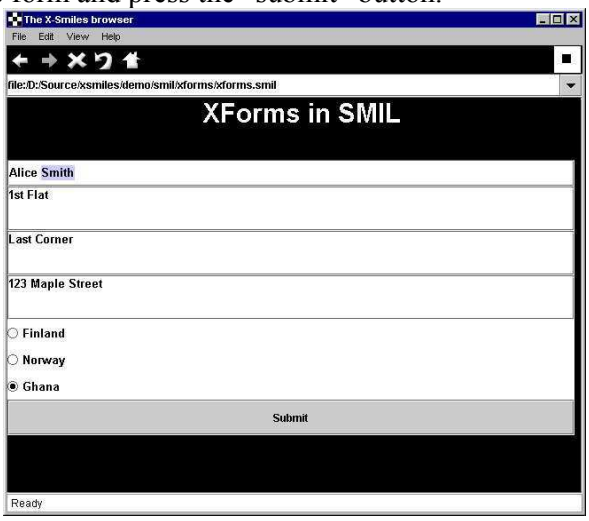



**Fig. 11.** Simple SMIL document with XForms.

Although, the example is simple, it demonstrates the integration of SMIL and XForms. Another way to integrate SMIL and XForms is to use XForms inside SMIL documents.

## 6. Conclusions

Forms have been a critical building block for WWW applications. XForms is W3C's work in progress to define a new WWW form specification that will piece up the shortcomings of the current WWW forms and recast forms into the XML world.

In this paper, we showed that it is possible to implement XForms in a Java based XML browser. We also showed that validation within XForms can be done using a validating XML parser, but then some user interface features cannot be implemented.

When the XForms browser implementation is done in a general fashion, it is easy to embed it into different XML markup languages. In this paper, we demonstrated how XForms can be used with XSL FO, SVG, and SMIL.

## Acknowledgement

The X-Smiles browser was originally developed in a student software project during 1998-99. We would like to thank all the participants of that project. After that, the development has continued in the GO-MM and XML devices projects. From these projects, we would like to thank Mr. Juha Vierinen, M.Sc. Kari Pihkala, Mr. Teemu Ropponen, Mr. Niklas Von Knorring, M.Sc. Anssi Hakkarainen, M.Sc. Tamas Heiszter, and Mr. Farooq Ummar. In addition, we would like to thank Mr. Mikael Kuhn, who is the author of the PIM example.

## References


[1] S. Adler et al., "Extensible Stylesheet Language (XSL) Version 1.0," *W3C Recommendation*, Oct. 15, 2001.

[2] Apache Software Foundation, "FOP: Formatting Objects Project". `http://xml.apache.org/fop/`

[3] B. Bos, et al., "CSS2: Cascading Style Sheets, Level 2," *W3C Recommendation*, May 12, 1998.

[4] N. Boyd, "Rhino: JavaScript for Java," *The Mozilla Org*. `http://www.mozilla.org/rhino/`

[5] T. Bray et al., "Extensible Markup Language (XML) 1.0 (Second Edition)," *W3C Recommendation*, Oct. 6, 2000.

[6] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0," *W3C Recommendation*, Nov. 16, 1999.

[7] CSIRO, "CSIRO SVG Toolkit". `http://www.cmis.csiro.au/svg/`

[8] S. DeRose, E. Maler, and D. Orchard, "XML Linking Language (XLink) Version 1.0," *W3C Recommendation*, June 27, 2001.

[9] M. Dubinko, et al., "XForms 1.0," *W3C Working Draft*, Aug. 28, 2001.

[10] ECMA, "ECMAScript Language Specification 3rd Edition," *Standard ECMA-262*, Dec. 1999. `ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf`

[11] J. Ferraiolo et al., "Scalable Vector Graphics (SVG) 1.0 Specification," *W3C Recommendation*, Sept. 4, 2001.

[12] P. Hoschka et al., "Synchronized Multimedia Integration Language (SMIL 2.0)," *W3C Recommendation*, August 7, 2001.

[13] R. Khare, "Can XForm Transform the Web?," *IEEE Internet Computing*, vol. 4, no. 2, March-April 2000, pp. 103-106.

[14] 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`

[15] S. Pemberton et al., "XHTML™ 1.0: The Extensible HyperText Markup Language," *W3C Recommendation*, Jan. 26, 2000.

[16] K. Pihkala, N. von Knorring, and P. Vuorimaa, "SMIL in X-Smiles," *The Int. Conf. on Distributed Multimedia Systems*, Taipei, Taiwan, Sept. 26-28, 2001.

[17] T. Pixley, "Document Object Model (DOM) Level 2 Events Specification Version 1.0," *W3C Recommendation*, Nov. 13 , 2000.

[18] H. Thompson et al., "XML Schema part 1: Structures," *W3C Recommendation*, May 2, 2001.

[19] D. Sweinsberg, "Steady State CSS2 Parser". `http://www.steadystate.com/css/index.html`

[20] J. Vierinen and P. Vuorimaa, "A Browser User Interface for Digital Television, " in Proc. *The 9th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG'2001*, Plzen, Czech Republic, Feb. 5-9, 2001, pp. 174-181.

[21] VoiceXML Forum, *Voice eXtensible Markup Language VoiceXML*, 2000. `http://www.voicexml.org`

[22] P. Vuorimaa, T. Ropponen, and N. von Knorring, "X-Smiles XML Browser," *The 2nd International Workshop on Networked Appliances*, *IWNA'2000*, New Brunswick, NJ, USA, Nov. 30 – Dec. 1, 2000.