This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of Helsinki University of Technology's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org.

By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

©2004 IEEE. Reprinted, with permission, from Proc. of the IEEE Sixth International Symposium on Multimedia Software Engineering (MSE2004), Miami, FL, USA, December, 2004, pages 232-239.

A Configurable XForms Implementation

Mikko Honkala
Helsinki University of Technology, TML
P.O. Box 5400, FIN-02015 HUT, FINLAND
Mikko.Honkala@hut.fi

Petri Vuorimaa
Helsinki University of Technology, TML
P.O. Box 5400, FIN-02015 HUT, FINLAND
Petri.Vuorimaa@hut.fi

Abstract

XForms is a new language for defining dynamic forms and user interfaces for the World Wide Web. In order to take advantage of the user interaction related features in the language, a client side processor is needed. This paper describes a configurable open source software implementation of XForms. The main goal of the implementation is to conform to the World Wide Web Consortium's XForms Recommendation. The other goals are external to the XForms specification and are related to the portability and configurability of the processor. The important questions are related to implementing an XForms processor for diverse environments, and the integration of XForms and other XML languages with different layout models. In the paper, more detailed requirements are gathered from these goals. Also, the design and implementation are presented in detail, in order to give insight to the more difficult and non-obvious parts of the software. The results of the paper cover the runtime requirements of the XForms processor.

1. Introduction

Although HyperText Markup Language (HTML), the main content language of the World Wide Web, was originally just a document description language, lately WWW has been transformed into a platform for interactive services [1]. HTML, being a device and operating-system independent, is still used to describe the user interface of the services. However, the GUI technology behind the services, HTML forms, is now outdated. The complexity of today's applications is far greater than what was imagined when the technology was first invented. Therefore, today's high-end forms have to use complex client-side ECMAScript [2] programming to achieve form field validation and simple computations (or bounce the form back and forth to the server).

Heavy use of scripting inevitably leads to low maintainability and accessibility [3]. Also, for the author, using both scripting and markup can be cumbersome, since it creates semantic and performance discontinuities [4].

Interaction in an application often depends on the device and environment at hand. For instance, the interaction with a mouse and a desktop computer differs from the interaction with a smartphone, let alone speech interaction with a normal phone. The traditional interaction technologies usually target just one environment or device. Applications running in the WWW can be accessed with different kinds of devices, but still the interaction is usually designed to just one or few different usage environments. Due to limitations in HTML forms, it is not easy to target the same application to different devices.

Due to these problems, W3C has developed a successor to HTML forms, XForms [5]. It removes need for scripts, and separates content from the presentation. Furthermore, it uses an abstraction of an user interface, enabling interaction across different modalities and devices. Other important questions are related to implementing an XForms processor for diverse environments, and the integration of XForms and other XML languages with different layout models.

Currently, the most interoperable option for developing a cross-platform client, (i.e., a browser), is Java. Programs written in Java can be run in desktops, smartphones and digital-TV set-top boxes. All of them have different libraries available, though.

There are XML-based form technologies that predate XForms, including XFDL [1] and XFA [2]. Also, there is currently a lot of effort put into XML-based GUI technologies. Some of them are mainly designed for defining GUIs for stand alone applications, including Glade [3], Netscape XUL [4] and Microsoft XAML [5] , while some address the office

1 Extensible Forms Description Language (XFDL) 4.0, John Boyer, Tim Bray, Maureen Gordon, 2 September 1998. Available at: http://www.w3.org/TR/NOTE-XFDL

2 XFA-Template Version 1.0, Gavin McKenzie, 14 June 1999. Available at: http://www.w3.org/1999/05/XFA/xfa-formcalc.html

3 Glade. Available at: http://glade.gnome.org/

4 XML User Interface Language (XUL) 1.0, David Hyatt (ed.), Available at: http://www.mozilla.org/projects/xul/xul.html

application space, namely InfoPath [6] . The configurable implementation reported in this paper is based on an earlier prototype the authors have made [6]. Dynamic calculations, and their implementation within XForms have been studied in [3]. Dynamic user interfaces, such as repeating constructs have been studied in [7]. The relationship between the latest XHTML 2.0 and XForms has been described in [8].

2. The XForms language

XForms 1.0 Recommendation [5] is the next-generation Web forms language. It solves some of the problems found in the HTML forms by separating the purpose from the presentation and using declarative markup to describe the most common operations in form-based applications.

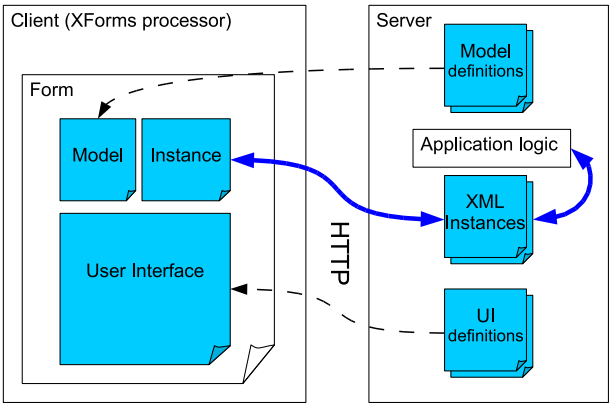

Figure 1. The XForms layers.

XForms separates the form into three main layers: Model, Instance, and User Interface (cf. Fig. 1).

Instance An arbitrary XML document that is modified by clientside user interaction, then submitted to a server.

Model Uses XML to define the constraints on items of the instance, which includes data types and ranges as well as computational relationships.

User Interface Defines how the form is shown and expresses bindings to instance items. User input is governed by rules in the model for the instance item being modified (through a bound input control).

The Model layer includes the *XML Schema* and the *Model Item Properties*. The structure and the data types of the instance data can be defined using the schema, but

5 XAML, Available at: http://msdn.microsoft.com/.

6 Microsoft InfoPath, Available at: http://www.microsoft.com/office/infopath/prodinfo/default.mspx

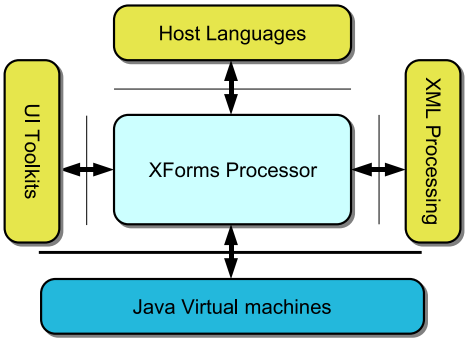

Figure 2. The external requirements.

it's use is optional to the author. The Model Item Properties (MIP) are dynamic constraints written in XPath. They can reference other values in the instance data and are dynamically evaluated by the XForms Processor. With MIPs, it is possible to define dynamic calculations and cross-value checks, which are not possible with XML Schema.

Fig. 1 also depicts the typical usage scenario of XForms. It consists of a client, which includes an XForms processor and an HTTP server with application specific software. The user interface of the application is defined with static documents, written in XForms combined with a host language, such as XHTML. Similarly, the structure of the data is described with static XML Schema documents. The application logic is responsible for sending and receiving XML instance data, which correspond to the data gathered from the user.

Compared to current Web applications with HTML, a lot of the user interface processing is transferred to the client in XForms. This frees bandwith and improves the user experience. Additionally, the application data is separated from the user interface, which helps to build modular software, which is easier to maintain.

Because of the quite heavy dependencies on XML processing tools, such as XML Schema and XPath, XForms may be difficult to implement in a restricted device, such as a smartphone. To overcome this problem, W3C has specified a smaller profile of the language, called XForms Basic [9], which does not include full Schema capabilities.

3. Requirements for the XForms processor

The requirements for the XForms processor come from multiple directions. The main goals can be divided into five categories, as depicted in Fig. 2. The main goal was to have a *conforming XForms processor*. The other goals were external to the XForms specification and were related to the portability and configurability of the processor. They are *Java VM support*, *XML Processing Libraries*, *UI Toolkits*, and *Host Languages*. The requirements that we derived

from these goals are described in detail in the following list. These requirements are the focus of the implementation part of this paper.

XForms Processor: Requirements.

XForms Full XForms 1.0 Support.

XForms Basic Can be configured to run in XForms Basic mode with reduced processing.

Java VM Support: Requirements.

Desktop Java 2 Standard Edition (J2SE) support.

Personal Java Runs in Personal Java 1.1 and Java 2 Micro Edition (J2ME) Personal Profile.

MHP Multimedia Home Platform (MHP) support.

XML Processing: Requirements.

XML Should allow third party XML parser, XPath, and XML Schema libraries.

UI Toolkits: Requirements.

Swing Supports Swing Toolkit in Desktop.

AWT Supports Abstract Windowing Toolkit (AWT).

Havi Uses Havi Toolkit in digital tv set-top boxes.

Host Languages: Requirements.

CSS Languages Supports CSS flow and box model.

Absolute positioning Supports absolute positioning in SVG and SMIL.

4. Design of the XForms processor

The XForms implementation discussed in this paper is done completely by the first author. Some of the libraries used (e.g., the CSS renderer), also have contributions from the first author. The first author is also a participant of the XForms Working Group and therefore is one of the authors of the specification [5]. The processor was one of the three interoperable implementations of XForms, which were required in order to raise the status of the language into W3C Recommendation. The XForms processor reported in this paper is part of the Open-Source XML Browser X-Smiles [10][11].

4.1. Architecture

Fig. 3 depicts the architecture of the XForms implementation. There are three layers: the XForms model, Meta UI, and the User Interface. XForms model implements XML Parsing, XPath and XML Schema. It also includes a calculation engine, datatypes, and the implementation of the instance data. The Meta UI layer has implementations of repeating user interface constructs (i.e., dynamic lists, enumerations, and tables), and switching parts of the user interface on and off dynamically.

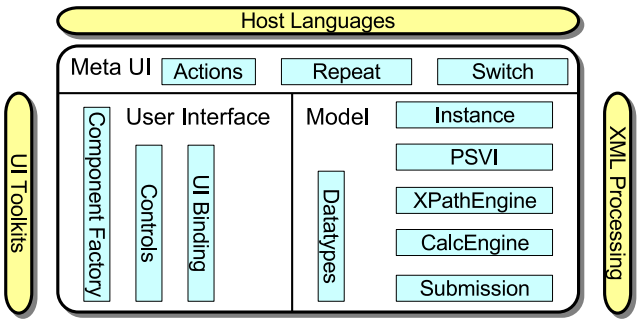

Figure 3. The architecture of the processor.

```
 <bind nodeset="items/item/total"
   calculate="../units * ../price"
   relevant="../units > 0" />
<bind nodeset="totals/subtotal"
   calculate=
   "sum(../../items/item/total)"/>
<bind nodeset="totals/tax"
   calculate="../subtotal *
              ../../info/tax"/>
<bind nodeset="totals/total"
   calculate="../subtotal + ../tax">
```

Figure 4. An example of a model. [3]

Finally, the user interface layer contains both high and low level implementations of all form controls in the XForms specification.

4.2. Instance data

XML parser is used to read the instance data into a DOM representation. We used Xerces 2.4.0, which has been built with a special option to include DOM L3 PSVI information and have extended the DOM implementation, so that it includes the XForms MIPs. It would also be possible to store the MIPs, for instance, in a Hashtable, which would remove the need to extend a certain DOM implementation, but it would degrade the performance. One solution for storing the MIPs is to use the User Data functionality in DOM Level 3, when that gets supported in the DOM implementations.

4.3. Calculation engine

The calculation engine takes care of calculating and keeping track of the model item properties of the form. For instance, the bind statements in Fig. 4 define multiple MIPs and assigns them to a group of nodes.

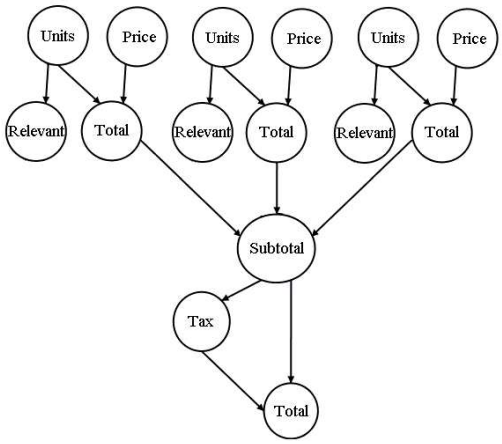

Figure 5. A dependency graph example. [3]

It is the Calculation Engine's responsibility to analyze the expressions and evaluate them at right time. The XForms specification gives an algorithm, which ensures that all expressions in the model are evaluated only once when they are needed. The algorithm has an important pre-requisite: it needs to obtain the referents of each XPath expression. For instance, the first calculate expression in Fig. 4 refers to 2 nodes: @units and @price. Once this information is present, the expressions can be analyzed for all the nodes in the nodeset and a *dependency graph* can be built. The calculation engine uses a topological sorting algorithm to define the optimal calculation order. The details about the algorithm and our implementation is documented in [3].

Fig. 5 depicts an example of a dependency graph. This graph is the runtime representation of the example in Fig. 4, where there exists three items in the "items/item" nodeset. The arrows represent the flow of calculation, so for instance, all of the "item/total" calculations must be performed before the "subtotal" calculation can be done.

4.4. User interface

XForms supports *dynamic dependencies* in UI. Implementing them needs special treatment. Additionally, dynamic dependencies are underspecified in XForms 1.0 specification. Although dynamic dependency is specified in section 7.5.1 [5], it is not specified when to re-evaluate (or *rewire*) the UI bindings. Consider the following:

```
<input ref="/user[@id=/selection]/name"/>
```

The expression in the *ref* attribute has a predicate, which means that a certain element from the collection of *user* elements is selected based on its id. This creates a dynamic dependency if the id is compared to something that might change during the lifetime of the form.

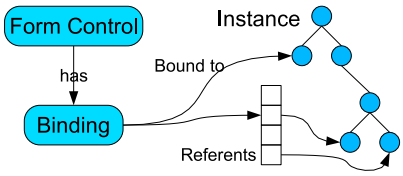

Figure 6. Dynamic UI dependencies.

Implementation of dynamic dependencies differs from *dependency graph*. The main difference is that the dependencies are never longer than one step: one binding cannot be dependent of another one (although the context is inherited to the ancestor bindings, which need to be re-evaluated if the bound node changes). Other difference is the dynamic nature of the dependencies: the graph for dynamic bindings must be re-created automatically if a change is detected, while the model's dependency graph is static until structure of the instance changes. Both require obtaining the XPath referents.

In our implementation, UI binding is represented by a *UI Binding* object. It is instantiated for every element that has UI binding attributes (i.e., *ref*, *nodeset*, or *bind*). The Binding object takes care of creating the listeners in the instance data as shown in Fig. 6.

In order to optimize re-wiring, our strategy is to keep the result of the expression cached. Then, when a referent is changed, the result (nodeset or node) is compared to the cached, and action is only taken when there is a difference between them. In our experience, *repeat* and *itemset* benefit a lot from this simple optimization.

Normally, the re-wiring changes the bound instance node, which means that the value of the form control must be changed. In some rare occasions, the type of the value might be different from the type of the previously bound node. If the form control is sensitive to the type of the node, the form control itself must be changed. This is left as future work in this paper.

4.5. XPath engine

XPath is one of the core technologies in XForms. Model Item Properties and UI binding must be authored using XPath expressions. There exists a lot of commercial and open source implementations of XPath, but not all of them fullfill the requirements that XForms presents. XForms implicitly sets the following requirements for the XPath engine:

DOM support XPath expressions must be evaluated on top of a dynamically changing DOM.

DOM Expression Referents It must be possible to obtain the referents of an XPath expression, in addition to the

result of the expression. This requirement is explained in more detail in section 4.3.

Context support The engine must support user defined context, including namespace context, context node, and the context nodeset.

Extension Functions The XPath engine must support adding extension functions.

In our implementation, a generic *XPathEngine* interface, which supports all of the requirements in the list above, was created. For performace reasons, the parsing of XPath expression into a object model is a separate operation. This allows for the XForms engine to parse expressions only once, while possibly executing them multiple times.

The interface supports two kinds of evaluation operations, one without tracing for the referents and one with the tracing. The context information is passed in the evaluation call, and includes the context node, the context nodeset, the namespace element (namespace evaluation context), and an empty list of referents. We have two implementations of the XPath interface. One uses Xalan 2.5.1 library, while the other uses Jaxen. Jaxen is more efficient, compared to Xalan, when used on top of a DOM tree. It is also smaller in size (cf. section 7.1). Some environments, such as Sun J2SE 1.4 already include Xalan, and that is why it is also supported. Both Jaxen and Xalan had to be modified in order to support *"DOM Expression Referents"* requirement. Other requirements listed in this section were quite easily met just be using the provided interfaces in the libraries.

4.6. XML Schema processing

XML Schema is used in XForms for two purposes: to define the datatypes within the instance data and to define the structure of the instance data. Datatypes are more important for XForms, since the structure of the instance is quite static, except for few operations (insert, delete, and copy). Additionally, datatypes are part of the basic profile whereas structure is not.

It is quite essential that the datatypes are implemented efficiently. For instance, it is possible to author a form, using the *incremental* attribute, so that every keypress modifies the instance data. If there are the datatypes of the instance data available at that time, all that is needed is to check the changed text against the schema datatype.

In XForms Full, there is the need to parse the provided XML Schemas and extract the Post Schema Validation Infoset (PSVI) information for each node. For datatypes, we used the datatypes implementation in Xerces 2.4.0, which also provides the PSVI implementation for the XForms Full mode.

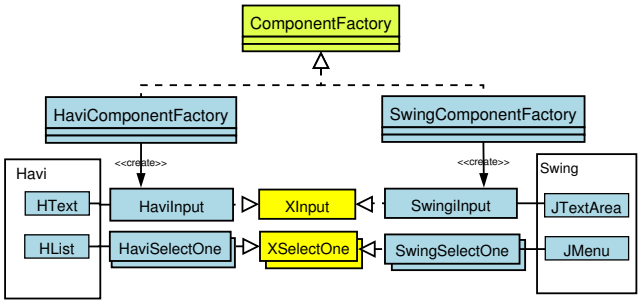

Figure 7. Component Factory's classes.

5. Integration to UI toolkits

Although the layout of the document containing XForms is the responsibility of the host language (e.g., XHTML) implementation, it is still necessary to render the form controls according to the device at hand.

In a perfect world, there would be a single UI Toolkit, whose look and feel could be altered to different environments and input/output methods. Applications would only need to support one toolkit. Unfortunately, even in Java, different environments use different UI Toolkits. Although this is clearly a design mistake, we still had to support it as stated in our requirements. Our requirement was to support Swing, AWT, and Havi. Swing is used in desktop environments, AWT, e.g., in hand-held devices, and Havi in Digital-TV set-top boxes.

On the other hand, XForms form controls are intent-oriented rather than presentation-oriented. For instance there is no form control called *calendar*, but instead an *input*, which is bound to a data with *date* datatype, might produce a calendar control in some platforms. Therefore it is natural to divide the architecture of the UI between intent-oriented and presentation-oriented parts. The intent-oriented part was discussed in section 4.4, and the rest of this section focuses on the presentation oriented part.

5.1. Component Factory

The presentation-oriented part of the user interface was implemented by creating a factory interface Component Factory, along with interfaces for different types of widgets. Then, for each Java component toolkit, such as Swing, an implementation of the Component Factory along with the implementations of each of the widgets is provided. This requires quite many classes, but since these classes are just wrappers for the real widget implementations, there is not that much programming involved.

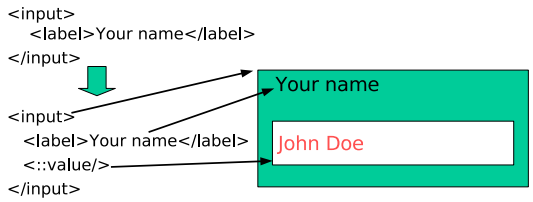

Figure 8. The pseudo-class ::value inside a form control. [8]

Fig. 7 depicts the class relationships between the factory and widget interfaces and their implementing classes. *ComponentFactory* is an Java interface that *HaviComponentFactory* and *SwingComponentFactory* implement. There is also other interfaces for the widgets themselves. All of these interfaces are implemented by the corresponding component toolkit wrapper. Each of the implementing classes uses components from the underlying component toolkit to provide the functionality. For events, AWT event classes and listeners are used where possible, thus removing the need for defining extra event classes.

6. Integration to host languages

As noted earlier, the division of work between the host language and XForms is such that the host language is responsible for the main layout of the document, and XForms is responsible for presenting the form controls and collecting user feedback. There are some features in the XForms language, which do not strictly adhere to this rule of thumb. Among these features are the styling of visible elements, such as form controls, labels, groups, repeating constructs, and switching parts of the user interface on and off dynamically.

The basic integration of form controls and the host language is done by reserving a rectangular area from the host languages document area, and the XForms processor can draw a component and handle it's events from that area. This functionality is provided by the X-Smiles browser, in order to support mixed namespace documents [11].

6.1. CSS based languages

To integrate well with CSS based layout model, XForms uses the notions of pseudo-class, and pseudo-element from CSS. For instance, the "::value" pseudo-element is used to define the appearance of the UI widget itself (it would otherwise be hard to style just the widget, since in XForms, the form control element also encapsulates a label element). Fig. 8 shows this; the first code snippet shows what author writes to create a form control. It contains only the *input* element and a *label* child. The element *input* and *label* should be laid out using the CSS rules and the additional pseudo-element ::value is used to define style just for the value part of the control; in this case an input box.

Other pseudo-elements defined non-normatively in the XForms specification are ::repeat-item and ::repeat-index. The ::repeat-item pseudo-element is used to style all the items in a repeating collection, while ::repeat-index pseudo-element is used to style the currently selected item.

Our implementation was integrated to the CSS layout and rendering model in X-Smiles. This allows, for instance, word wrapping in labels and groups. Also, it enables the use of CSS (e.g., display:block; or display:inline;) to determine how to layout the form controls. We needed to add good support of pseudo-elements in the rendering engine for the purposes of this integration. Currently, pseudo-elements can be added to the rendered document, and they behave like normal DOM elements for layout purposes.

6.2. Languages with absolute layout

The requirement for the processor was to support all of the languages supported by the X-Smiles browser. Some of these have non-CSS layout models. Examples of these languages are SMIL and SVG.

We implemented SVG and SMIL support as a prototype [6]. Our implementation experience shows that implementing repeat is not feasible for these languages. We had an experimental repeat implementation for SMIL [12], but that was later abandoned. We are currently implementing Timesheets [13], which integrate to the CSS layout model well, thus allowing the use of repeating constructs in timed documents.

SVG support is problematic as well; SVG is a vector drawing language, which supports zooming in and out, a feature, which is not normally supported by UI toolkits. We have simulated zooming by changing the font size in form controls depending on the zoom level. Future work item would be to draw the form controls in SVG itself, thus allowing for real zooming and integration to SVG drawing model.

7. Configurability

Most of the requirements in section 3 are related to running the processor in different environments with different UI toolkit and XML processing libraries. In order to fullfill these requirements, the processor was designed to be configurable. Different configuration options are depicted in Fig. 9. On the top, there are the host languages supported by the processor. The supported UI toolkits are shown on the left side, while XML processing libraries lay to the right. At the bottom, there are the supported Java Virtual Machines.

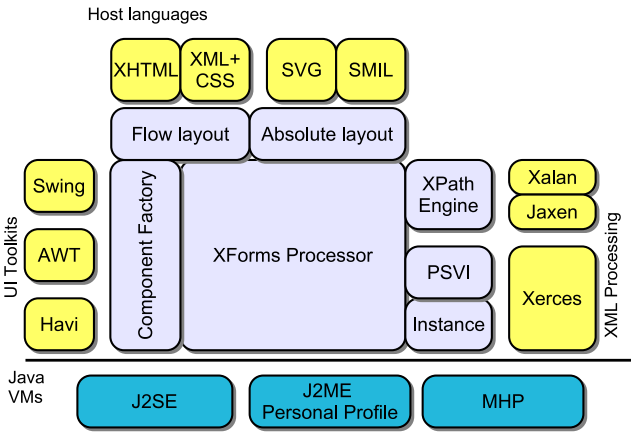

Figure 9. Configuring the processor.

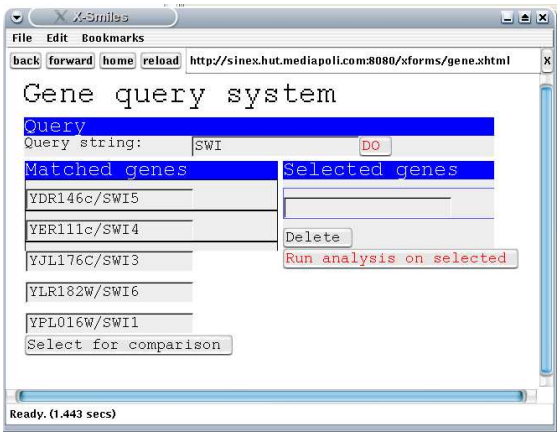

Figure 10. The processor running in J2ME Personal Profile reference implementation.

Table 1 lists few of the configurations that we have created and used. The first is the *Desktop* configuration, which is a full XForms implementation running inside the X-Smiles browser. It integrates to all of the host languages that are supported by X-Smiles. The second one, *J2ME Embedded*, is an embedded XForms basic processor targeted to a handheld device running J2ME Personal Profile. It supports few of the host languages and has a more compact XPath processor and uses AWT as the UI toolkit. This configuration is depicted as a screenshot in Fig. 10, where it is running on top of Sun's reference implementation of J2ME Personal Profile. The final one is the *Digital-TV Browser*, running XForms basic on top of Havi UI toolkit.

The main difference between the XForms basic and full profiles is that basic does not require schema processing, but instead uses just the pre-defines schema datatypes. We have implemented XForms full by parsing the instance DOM into a PSVI-extended DOM using Xerces. XForms basic was therefore implemented as an configuration option, which uses a normal DOM instead of the PSVI DOM. For datatypes, the datatypes implementation in Xerces was used also in basic configuration mode.

7.1. Memory requirements

The storage memory requirements of the implementation are dependent of the configuration that is being used. Normally, some of the libraries are already provided by the system. For instance, every system usually has the UI Toolkit libraries pre-installed. Therefore we measured the size of the processor itself. We wanted to show also the size of the minimal browser as a reference value.

The storage size of the processor is reported in table 2. We also used an open-source tool called *ProGuard* [7] , to minimize the storage size of the jars. It successfully decreased the size of xsmiles-personaljava.jar by 35%. The sizes reported in table 2 are the sizes after the minimization. The size of the XForms basic processor including the XPath library is 505 KB, while the size of the XForms full processor is 734 KB. This does not include the basic XML parsing and DOM, which is included in the browser part, being 1052 KB in total.

The run-time memory usage depends on the size of the form application. The set of our demo applications, written in XHTML+XForms, run with 10 MB of Java heap. The most comprehensive application, a configuration file frontend, includes about 100 interactive form controls accessing different parts of the instance data. It also includes switching and repeating user interface contructs. The java heap includes the run-time memory used by the browser core, XHTML + CSS renderer, and the XForms processor.

8. Conclusion

XForms is a next generation markup language for defining user interfaces in the WWW. It provides ease of authoring, reuse, device independence, and accessibility. In order to deploy XForms, support in the user agent is needed. We have presented a configurable XForms processor that is portable to different environments. The processor supports different XML processing libraries found typically in Java environments. It is also independent of the UI toolkit in use. For instance, the Havi toolkit, uses the remote control and television set as the input/output devices. The processor is also independent of the host language. We have integrated the processor with XML+CSS, XHTML, SMIL, XSL FO, and SVG. In order to facilitate for the XForms integration,

7 ProGuard, Available at: http://proguard.sourceforge.net/.

Configuration	Profile	Java VM	XML Parser	XPath	UI Toolkits	Host Languages
Desktop	Full	J2SE	Xerces	Xalan	Swing	XHTML, XML+CSS SMIL,SVG, XSL FO
J2ME Embedded	Basic	J2ME Basic Profile	XercesDTD	Jaxen	AWT	XHTML XML+CSS, SMIL
Digi-TV Browser	Basic	JDK 1.1 Personal Java	XercesDTD	Jaxen	Havi	XHTML XML+CSS, SMIL

Table 1. Examples of working configurations.

Component	Basic (KB)	Full (KB)
Browser core		
Browser core + GUI	258	”
CSS Layout and Renderer	64	”
CSS Parser (Steady State)	149	”
XML Parser (Xerces DTD)	446	”
XML APIS (W3C / Xerces)	134	”
Total	1052 KB	1052 KB
XForms Processor		
XForms processor	164	”
XPath engine (Jaxen)	228	”
XML Schema Datatypes	113	”
XML Schema Structures	0	230
Total	**505 KB**	**734 KB**
Grand Total	**1557 KB**	**1786 KB**

Table 2. The storage memory requirements.

changes are usually needed in the host language implementations. We have shown that XHTML integration is possible with some additions to the host language implementation. For languages, which support only absolute positioning, such as SMIL and SVG, the integration is not straightforward. Some XForms features, such as repeat which require flow layout are impossible, while some host language features, such as SVG zooming, require features, which are usually missing from the UI toolkits.

Our processor was one of the three conforming processors when XForms raised as W3C Recommendation. We have successfully ran our browser in variety of environments, such as MHP for Digital TV and Java 2ME Personal Profile, which is targeted at embedded devices.

The paper gives the memory requirements for the processor for different operating configurations. The storage memory size in the smallest configuration is 1,5 megabytes including browser infrastructure. The XForms processor's minimum size is 500 kilobytes including an XPath processor and Schema datatypes implementation. If the environment already has these, the size can be as low as 164 kilobytes. In our opinion, this shows that is is possible to include an XForms processor in most environments, possibly counting out low-range mobile phones.

The research was funded by the GO-MM project to whose partners and researchers the authors would like to express their gratitude. We would also like to thank Pablo Cesar for valuable comments about this paper.

References

[1] M. H. Butler. Current technologies for device independece. Technical Report HPL-2001-83, Hewlett Packard Laboratories, March 2001.

[2] ECMA-262m. Ecmascript language specification, 1998.

[3] J. Boyer and M. Honkala. The xforms computation engine: Rationale, theory and experience. In *6th IASTED International Conference, Internet Systems, and Applications, IMSA 2002*, 2002.

[4] M. Hostetter, D. Kranz, C. Seed, and S. Ward C. Terman. Curl, a gentle slope language for the web. *World Wide Web Journal*, 1997.

[5] M. Dubinko et al. (eds.). Xforms 1.0. W3C Recommendation, 2003.

[6] M. Honkala and P. Vuorimaa. Xforms in x-smiles. *WWW Journal*, pages 151–166, 2001.

[7] M. Honkala and P. Vuorimaa. Advanced ui features in xforms. In *8th International Conference Systems, DMS 2002*.

[8] M. Pohja, M. Honkala, and P. Vuorimaa. An xhtml 2.0 implementation. In *International Conference on Web Engineering, ICWE 2004*.

[9] M. Dubinko and T.V. Raman (eds.). Xforms 1.0. W3C Candidate Recommendation, 2003.

[10] P. Vuorimaa, T. Ropponen, N. von Knorring, and M. Honkala. A java based xml browser for consumer devices. In *17th ACM Symposium on Applied Computing*, Madrid, Spain, March 2002.

[11] K. Pihkala, M. Honkala, and P. Vuorimaa. A browser framework for hybrid xml documents. In *Internet and Multimedia Systems and Applications, IMSA 2002*. IMSA, August 2002.

[12] K. Pihkala, M. Honkala, and P. Vuorimaa. Multimedia web forms. In *SMIL Europe 2003*.

[13] W. ten Kate, P. Deunhouwer, and R. Clout. Timesheets - integrating timing in xml. In *Multimedia on the Web Workshop, 9th international World Wide Web Conference, WWW9*.