An XHTML 2.0 Implementation

Mikko Pohja, Mikko Honkala, and Petri Vuorimaa

Telecommunications Software and Multimedia Laboratory,
Helsinki University of Technology
P. O. Box 5400, FI-02015 HUT, Finland
Tel. +358-9-4515260
{mikko.pohja, mikko.honkala, petri.vuorimaa}@hut.fi

Abstract. The next version of XHTML is at work-in-progress stage in the World Wide Web Consortium. It adds a lot of features to the most used content language of the Web. The most notable change is the addition of XForms, the next generation WWW forms language. This paper describes the XHTML 2.0 specification and an XML user agent implementation for it. The new features of the language are discussed both from the author's and the user agent manufacturer's point of view. In addition, it describes a case study, which takes advantage of the new features.

1 Introduction

HyperText Markup Language (HTML) [1] is one of the greatest factors to the success of the World Wide Web (WWW). It provides an easy way for authors to create content to WWW. Originally, HTML was designed to describe structure of the document. Later, a lot of presentational features were added into it.

To bring HTML back to its origin, it was redefined as an Extensible Markup Language (XML) [2]. First versions of XHTML were just reformulation of HTML 4, but the newest version, XHTML 2.0 [3], which is at work-in-progress stage in the World Wide Web Consortium (W3C), is no longer backward compatible with the earlier versions. As a starting point to a design of XHTML 2.0 has been experiences and problems of earlier web technologies, especially HTML. The intention is to make XHTML 2.0 easy to adopt for authors and to match to original purpose of HTML (i.e., describe the structure of hypertext documents). For instance, XHTML 2.0 removes all presentation elements from HTML and subordinates all presentation to stylesheets. In addition, a lot of functionality has been added to XHTML 2.0. The objective is to reduce the use of scripting languages within XHTML documents. Most common scripts have been replaced by functional elements.

The success of the WWW is also largely based on interactive services such as search engines, online banking, and e-commerce. A key technology used in interactive Web applications is HTML forms. However, requirements for these services have steadily increased since the advent of the technology. Today's high-end forms use complex client-side ECMAScript [4] 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. [5]

The HTML forms has been replaced by XForms in XHTML 2.0. That removes need for scripts from the forms and separates model and presentation of a form. Consequently completing the design principles of XHTML 2.0.

The transition to XHTML 2.0 will be more difficult than transitions between earlier HTML versions. XForms [6] will be the biggest change in XHTML 2.0, but there are also other changes, which are not backward compatible [7]. In this paper, we have assessed the impacts of the transition to both user agent developers and authors.

The paper describes XHTML 2.0 specification and an XML user agent implementation for it. Next section introduces XHTML and XForms modules of XHTML 2.0 and their integration. Sections 3, 4, and 5 discuss implementations of X-Smiles XML browser, XHTML module, and XForms module, respectively. A case study is described in section 6, while discussion about the transition to XHTML 2.0 is in section 7. Finally, section 8 gives the conclusions.

2 XHTML 2.0

XHTML 2.0, which is a W3C Working Draft, is a markup language, which describes structure of a document. It does not represent document's layout. XHTML 2.0 is successor of the earlier HTML languages, but it is not backward compatible with them.

XHTML 2.0 is supposed to be as generic XML as possible. Since XHTML 2.0 is an XML language, it can be combined with other XML languages and use their features. Only special-purpose XHTML features were included in XHTML 2.0. These are, for example, elements for images, tables, menus, etc. In addition, XHTML 2.0 uses facilities like XForms, XML Events [8], and XML Base [9]. In addition, layout of the document is defined by Cascading Style Sheets (CSS). All the presentational elements, such as *font* are removed from XHTML 2.0.

The separation of content and presentation has many advantages. Layout of the document can be easily changed, layout can be specified for different devices, documents are smaller, and user agents are easier to implement. For instance, from the user agent vendor's point of view the user agent's components can be separated and they can be smaller, while the author can more easily reuse styling. This also leads to better accessibility because it is easy to change the styling of a document without touching the contents.

Compared to earlier versions, XHTML 2.0 has better ability to make documents more structured. Content is intended to be divided into sections, which all can have titles and subsections. Sections within sections can nest indefinitely deep. As before, content in sections consist of paragraphs, tables, list, etc.

One aim of XHTML 2.0 is to reduce scripting from the documents. Many elements have now some kind of functionality as a default. That reduces device dependency and eases authors work, because they do not have care about script

languages. It is still possible to use scripting via the Document Object Model (DOM) interfaces. Other benefits of moving from scripts to declarative languages are improved accessibility and device-independence. This results from the fact that it is easier to use automated tools to process declarative markup than scripted documents.

XHTML languages are divided into modules. XHTML 2.0 contains all the modules defined in XHTML Modularization 1.0 [10]. Although, content of the modules have been changed a bit. In addition, it uses modules from XForms, XML Events, and Ruby. XForms module is discussed in more detail in next subsection. XML Events provides an uniform way to integrate DOM Level 2 event interfaces with event listeners and handlers. Ruby module is used to add short annotations to the text. Usually it is used as pronunciation instructions with eastern languages. The Ruby module is out of scope of this paper.

The biggest changes in XHTML module are the addition of structural and functional elements. Functional elements had to be realized by scripts earlier. Now, there are elements, which have some default functionality. For instance, menus are common on the web pages. For that, there is navigation list in XHTML 2.0 [11]. Navigation list is like normal list, but only selected part of the list is shown at a time. The part can be selected for instance by moving cursor on it. It is possible to style such a list as a drop-down menu with CSS, as shown in Fig. 1. XForms specification also contains lot of features, whose intent is to replace scripting.

Fig. 1. Navigation list styled as a drop-down menu

Also, the use of attributes has changed. Attributes can be used more generally than before. For example, *href* attribute can be added to any element. That makes the *a* element useless. In addition, all the elements can have *src* attribute. That enables addition of external sources to the element. Element's normal content is shown, if source is not available or cannot be shown in the device in question.

2.1 XForms

HTML forms has many well-known shortcomings. It has no separation of data and presentation. Building anything more than a simple form requires excessive amounts of scripting, which is hard to implement and maintain. Another often-used approach is to send the form back and forth between the browser and the server, which leads to great amount of round-trips and reduced maintenance of the forms.

XForms 1.0 Recommendation is the next-generation Web forms language, designed by the W3C. It solves some of the problems found in the HTML forms by separating the purpose from the presentation and using declarative ways to describe the most common operations in form-based applications. It can use any XML grammar to describe the content of the form (the instance data). Thus, it is also possible to create generic editors for different XML grammars with XForms.

XForms separates the form into three main layers: Model, Instance Data, and User Interface (cf. Fig. 2 (a)). The Model layer includes the XML Schema and constraints. With the schema, it is possible to define the structure and the data types of the instance data, but XForms can also be authored without a schema. The schema can be processed via a normal XML Schema processor as shown in Fig. 2 (a).

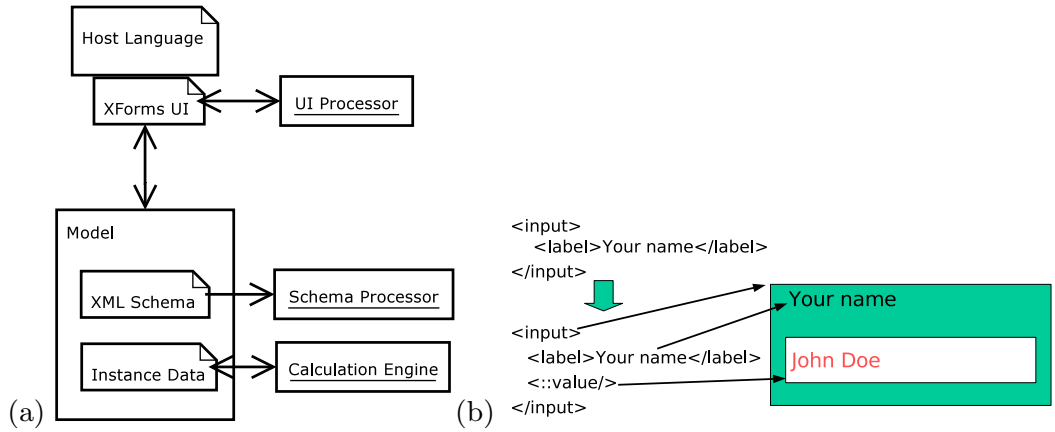

Fig. 2. (a) XForms layers and (b) the pseudo-class ::value inside a form control

It is possible to define dynamic relations (constraints) between parts of the instance data. These relations are defined using XPath expressions. Examples of dynamic relations include inter-field calculations and constraints, which dynamically set the state of an item in instance data to read-only or required. The calculation engine interacts with the instance data and it resolves and computes these relations dynamically while the user interacts with the form [5].

The User Interface is also bound to the instance data using XPath expressions. The choice of form controls covers the range of typical GUI widgets (although they are defined in higher level terms, such as select1 instead of menu).

Some form controls also adapt to the underlying XML Schema data type. For instance, an input control bound to a *date* data type will show a calendar picker instead of a text field. XForms also hosts a collection of dynamic user interface features, such as displaying and dynamically modifying collections of repeating items and switching parts of the user interface on and off. [12]

XForms exposes a lot of the processing to the author via DOM events. For instance, there are events that are thrown when the instance data becomes invalid, read only, required, etc. The author can catch these events declaratively using XML Events. The counterpart to XML Events are XForms actions, which can be executed when a certain event condition is met. This way it is possible to define, in a declarative fashion, certain actions to happen when the user interacts with the form. For instance, the form could be submitted partially when the user selects a certain item from a selection list.

2.2 Integration of XForms and XHTML

XForms is not a self-standing document type, i.e., it needs a host language to define the document's master layout. XHTML is a natural choice as a host language in Web context. The current version of the XHTML 2.0 Working Draft includes XForms as the forms module.

Since XHTML relies heavily on CSS2 layout, it is important that XForms also integrates to this model well. For instance, it 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. 2 (b) shows this; the first code snippet shows what the author writes to create a form control. It contains only the *input* element and a *label* child. To give the author better CSS control of the rendering, the specification describes an additional CSS pseudo-element, called "::value". The element *input* and *label* should be laid out using the CSS rules (such as display:block or display:inline) and the additional pseudo-element ::value is used to define style just for that part of the control, which the user can interact with; 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.

3 The X-Smiles XML Browser

X-Smiles is an open source XML browser developed at the Helsinki University of Technology. The authors of this paper have implemented XHTML 2.0 and XForms support in the browser. In addition, the CSS layout implementation is done by the authors. The authors have also actively participated in the W3C XForms Working Group.

This section describes the main architecture of the browser. The main components of the X-Smiles browser can be divided into four groups: XML processing, Browser Core Functionality, Markup Language Functional Components (MLFCs) and ECMAScript Interpreter, and Graphical User Interfaces (GUIs).

3.1 Operation

The core of the browser controls the overall operation of the browser. It includes browser configuration, event handling, XML Broker, etc. MLFCs handle different XML languages and render the documents. There are several GUIs in the X-Smiles distribution. They are used to adapt the browser to various devices or as virtual prototypes, when prototyping content targeted to diverse range of devices. [13]

Basically, the X-Smiles operation consists of three steps: parsing, creating DOM, and rendering. Parsing and creating DOM are done concurrently. When reading new XML document, the X-Smiles browser first reads the document source. The file is accessed using either the file or http protocol. The Xerces XML parser and DOM implementation construct the DOM model of a document. Parser creates different Simple API for XML (SAX) events, which are used by DOM implementation. The MLFCs handle the presentation of the document. The MLFC layer is done in a modular way; it is possible to add and remove MLFCs for certain browser configurations without rebuilding the browser. It is also possible to show embedded documents. For instance, it is possible to use a SMIL presentation referenced by an object element in XHTML.

3.2 Hybrid Documents

Hybrid XML documents are documents, which contain several XML languages, separated by namespaces. Recently, the usage of hybrid documents has increased. The trend has been to specify XML languages as modules, which are combined to construct complete languages. This way, the modules can be reused across different languages, making implementations smaller and easier. It also reduces the number of different languages that the author must learn.

As the number of languages gets higher, a way to flexibly handle these kinds of documents is required. The X-Smiles browser has a framework that handles hybrid documents. This framework includes a component called XML Broker, which handles the registration of language implementations. A language implementation in the framework is called a MLFC (i.e., Markup Language Functional Component). An important concept in hybrid documents are *host* and *parasite* languages. There is always one *host* language in a document, usually determined by the namespace of the document element. The host language is used for the master layout of the document. Other languages that are embedded in the document are called *parasite* languages. In this paper, XHTML 2.0 is a *host* language, while XForms and XML Events are *parasite* languages. In an implementation, there is a need for communications between the *host* and *parasite* elements in

the DOM. In X-Smiles, this is done by extending the DOM implementation and implementing general browser-defined interfaces. [14]

4 XHTML Module

The XHTML module handles XHTML documents. The documents can be styled by style sheets and contain elements from other XML languages. The operation of the XHTML module is depicted in Fig. 3. An XML parser parses a given XML document and creates DOM document, which is an object tree representation of the XML document. From each tag in the XML document, a node for DOM tree is created. The document is displayed by a layout document, which implements Java Swing's document model. The layout document is formed from DOM tree. Basically, every DOM element have respective layout element in layout document. Layout document is displayed by Swing Views. Each layout element has respective View, which defines its layout. Views render the document in browser window. [15]

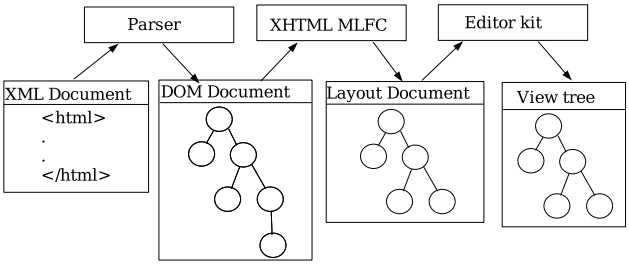

Fig. 3. Operation of the XHTML module

The XHTML elements can be divided into two groups: stylable and non-stylable elements. All the visible elements are stylable. A style is assigned for each stylable element after DOM has been created. Elements get their style from the style sheet object. It is possible to change the visual type of the element by using the CSS property "display" (e.g., inline/block).

Every DOM element has a respective layout element, which is used by layout document. In X-Smiles, XHTML elements and all the other elements in XHTML document have to have layout element in order to get rendered. In the case of XHTML, layout element is an instance of *AbstractElement*, which is an inner class of *XHTMLDocument2*. *AbstractElement* implements also Swing's *Element* interface, whereas *XHTMLDocument2* implements Swing's *StyledDocument* interface. That way they both can be used as a part of Swing document model. [16]

Views are also the part of the Swing document model. They define how elements are rendered in browser window. Views can contain content and other views and they are placed in a row either vertically or horizontally. For instance, inline elements are placed horizontally and paragraphs vertically. Every view has

some basic layout, which can be modified by style sheet. Elements are associated with Views by editor kit, which ties the layout document and views together.

5 XForms Module

The XForms module was also implemented using the MLFC interfaces in the X-Smiles browser. It is an parasite MLFC and always needs a host MLFC (in this paper, XHTML2 MLFC). The host is responsible for the main document layout, while the parasite is responsible for rendering the host language elements.

5.1 Architecture

Fig. 4 depicts the architecture of the XForms MLFC in X-Smiles. There are three main layers in the implementation: XForms model, Meta UI and User Interface. At the lowest level of the XForms model, there are the XML libraries that the implementation uses. For XML parsing and XML Schema, we used the Xerces-J 2.4.0 implementation compiled with DOM level 3 support. DOM level 3 support was needed for Post Schema Validation Infoset (PSVI), which is used to get the data type information. For XPath, Xalan-J 2.5.1 was used. On top of the XML libraries lie the Instance document implementation, and the validation and calculation engine implementations. Data types are used in all layers, for instance to instantiate right types of form controls. The middle layer, Meta UI, contains high-level User Interface constructs, such as repeating items in a collection and switching parts of the User Interface on and off. The topmost layer is the User interface. To further enable multi-platform support, an additional layer, called Abstract Components was created. For each platform, an implementation, or wrappers, of the abstract components were included. Currently, we have support for AWT, Swing and Havi widgets.

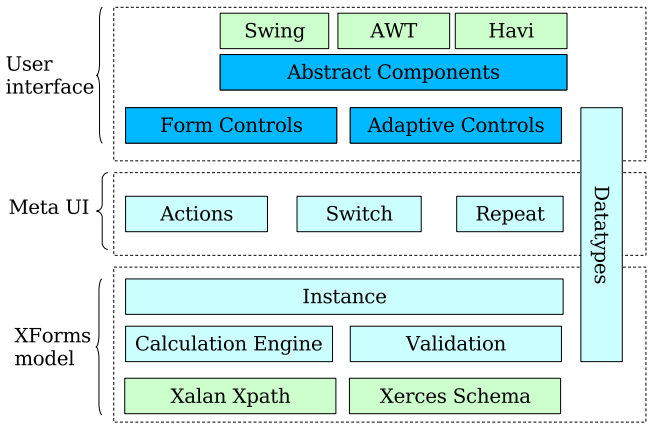

Fig. 4. XForms MLFC architecture in X-Smiles

XML Events was implemented by creating the XML Events MLFC into X-Smiles. The MLFC uses DOM events implementation in the Xerces-J parser to

implement listeners and events. A general action interface was created for the browser, and all XForms action elements implement that interface.

5.2 CSS Integration

Because of the way XForms language was designed, it was not entirely trivial to integrate it with a CSS based layout engine, such as the XHTML+CSS engine in X-Smiles. For instance, there exists a notion of cursor inside a repeated construct, that has no correspondence in the DOM. Also, the element describing the form control contains a label element inside it, making it difficult to style just the form control itself, as discussed in section 2.2. Basically, the implementation of pseudo-classes was quite straight-forward, since it is a concept outside of the DOM tree. On the other hand, pseudo-elements are conceptually in the DOM tree for CSS styling and cascade purposes, and are therefore much harder to implement. For instance, the pseudo-elements ::repeat-item and ::repeat-index should be inserted in the middle of the DOM tree. Fig. 5 depicts this; a simple repeat construct bound to a nodeset containing two nodes leads to more complex run-time object tree containing several pseudo-elements. Note that some DOM elements are treated as children of the pseudo-elements for purposes of CSS styling and cascading.

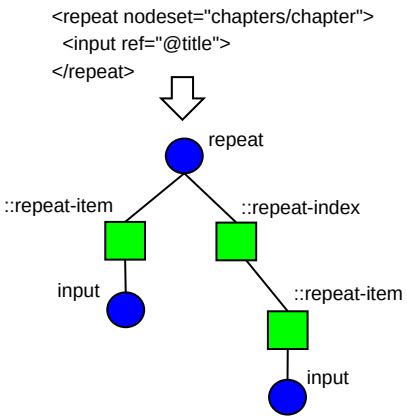

Fig. 5. repeat-specific pseudo-elements

We implemented all pseudo-classes, such as :invalid, describing the state of the form control. Also, we implemented the pseudo-element ::value. The pseudo-elements for styling the repeated structures were left as a future work item.

6 A Case Study: Document Composition

In this section, we introduce a real-world example of how to use XHTML 2.0 to compose and navigate a large document. One of the shortcomings of XHTML 1.0 is that it does not support forms that edit repeating structures, thus requiring

server-side or script processing to provide such functionality. Another shortcoming is the non-existing support for navigation lists, such as pull-down menus. Navigation lists show only the currently selected part of the list, thus helping the user to navigate more easily in a large document hierarchy. Navigation lists have to be implemented with complex scripting in XHTML 1.0, decreasing accessibility and device-independence. The example shows how to use the declarative features of XHTML 2.0 to implement both the structural editing and navigation. The whole example runs in the client and does not need server-side programming. It also does not use any scripting, thus making it accessible and device independent.

The document used in this case study is X-Smiles' Technical Specification document. It has been created using the Doc Book XML format. In this case study, we show how to present and navigate that document in XHTML 2.0 and even edit the document composition. The Doc Book source files are controlled by XForms and transformed to XHTML 2.0 using XSL Transformations.

The components of the example are shown in Fig. 6. The document consists of chapters, which each are in separate XML files. Headings, paragraphs, images, etc. are marked by specific tags in the chapter documents. All the chapter documents are included into a composition document by XML Inclusions (Xinclude)[1]. The composition document is edited by Edit document, which is an XHTML 2.0 document with XForms form. The composition document is an instance of the form. Through the Edit document, user can select, which chapters are included in the final document. The final document is transformed to XHTML 2.0 format by XSL style sheet. The document composition and structure are discussed in more detail below.

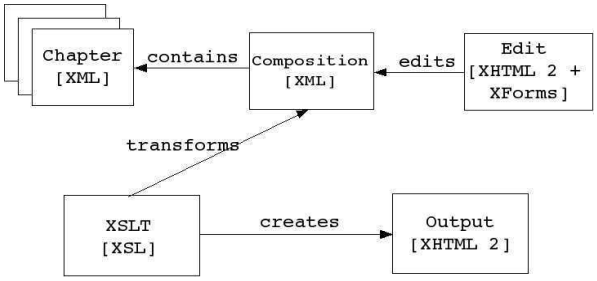

Fig. 6. Test case architecture

The editor document, which is depicted in upper-right corner of Fig. 6, is an XHTML 2.0 document, which includes embedded XForms elements. The XForms instance data is the Composition document. The editor document uses a XForms *repeat* element to create an editor for this document. It is possible to add new sections to the composition, or enable/disable them. The editor UI is depicted in Fig. 7 (a).

[1] XML Inclusions, http://www.w3.org/TR/xinclude/

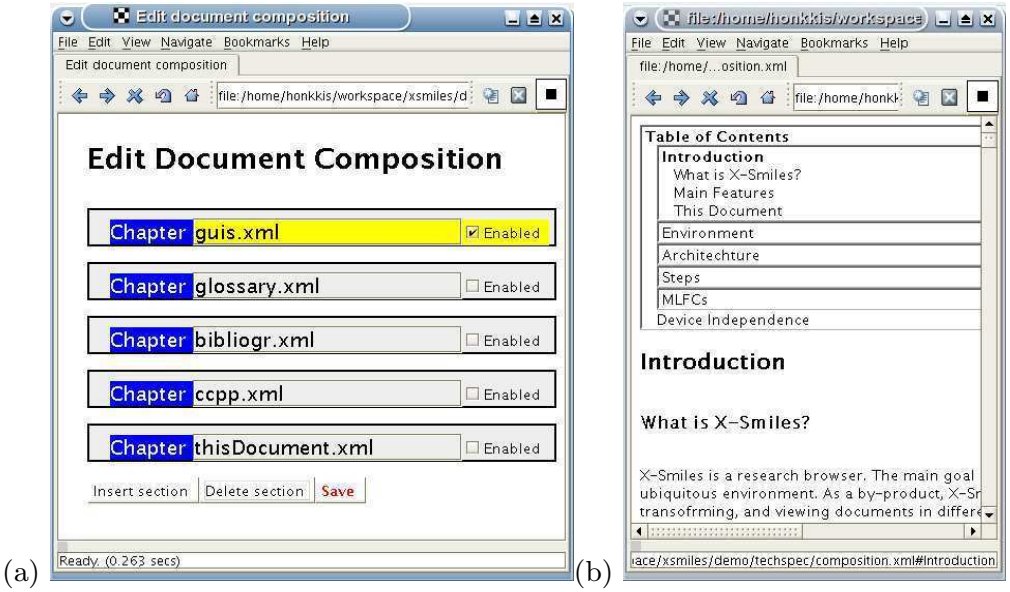

Fig. 7. (a) Editing the composition with XForms and (b) viewing the final document

The XSL style sheet converts all the elements from the composed file to XHTML 2.0 elements. Content of chapter documents is divided to sections, which can contain other sections, paragraphs, headings, etc. The XSL Transformation converts all the chapter and section elements to XHTML 2.0 sections. Therefore, document is structured automatically. In XHTML 2.0, all headings can be represented by *h* elements. Their style depends of amount of embedding sections. So, during transforming, there is no need to decide the level of headings.

The document has a menu in the beginning. It is realized using XHMTL 2.0 navigation list. Navigation list contains all the headings of the document. The navigation list shows only selected part of the list at the time. List items are also links, which refer to the heading in question in the document. Source code of the navigation list with few entries is shown below.

Source code of the navigation list

```
<nl>
  <label>Table of Contents</label>
  <li  href="#Introduction">
    <nl>
    <label href="#Introduction">Introduction</label>
    <li href="#WhatisX-Smiles?">What is X-Smiles?</li>
    <li href="#MainFeatures">Main Features</li>
    <li href="#ThisDocument">This Document</li>
    </nl>
  </li>
  <li  href="#Environment">
    <nl>
    <label href="#Environment">Environment</label>
```

```
    <li href="#RuntimeEnvironment">Runtime Environment</li>
    <li href="#Runningit">Running it</li>
    <li href="#BuildingwithAnt">Building with Ant</li>
    </nl>
  </li>
</nl>
```

The navigation menu could be styled like a regular pull-down menu as shown in Fig. 1, but due to limitations in the CSS layout in the current version of X-Smiles (0.82) that was not done. As discussed in Section 8, we are building a better CSS layout engine into the X-Smiles browser. The resulting document, viewed in the X-Smiles browser is shown in Fig. 7 (b). Part of the navigation list, whose source is shown above, is also visible.

7 Discussion

So far, the appearance of a new HTML version has not caused much headache for authors and user agent developers. Different HTML versions are backward compatible, which means that old documents do not need major revising. Even the transition to XHTML 1.0 is a straight forward process as it is basically a reformulation of HTML 4.01 in XML format. The new XHTML 2.0 specification will cause more problems, though. The biggest change is related to forms. The fifth Working Draft specifies that XForms will be used as the forms technology in XHTML 2.0.

XForms requires major changes especially in user agents. Old HTML form implementations cannot be used as a base for XForms, because the whole concept is different. XForms has a new data model and it contains more functionality than HTML forms. The new data model requires the use of XML Schema and XPath processors. Both of these are large components, which causes problems in restricted devices. W3C is aware of this problem, and thus it has defined the XForms Basic Candidate Recommendation. In XForms Basic, the data model is less advanced, requiring only support for certain data types, in effect removing the need for full XML Schema processor.

The transition to XHTML 2.0 causes also problems for authors. The main obstacle is that they have to learn a new forms language. Fortunately, they do not have to start from scratch. The reason is that XForms is based on XPath. Most XML developers use XSLT regularly. Since both XForms and XSLT are based on XPath, the developers already have a starting point. Another advantage is that XForms is declarative language. Current, HTML forms are heavily based on scripting, which makes updating and reusing of forms difficult. XForms is easier to maintain and it allows the reuse of forms in different XHTML documents.

Another major change in XHTML 2.0 is the real separation of content and presentation. XHTML 2.0 removes all the presentation related features from XHTML 1.0 and concentrates only on defining the structure of the document. All presentation related issues are defined using separate CSS style sheets. In addition, several XHTML 2.0 elements and attributes contain functionality, which

before had to be realized using scripting. Therefore, authors do not have to use scripting as much as before and also their documents will be more widely accepted by the different user agents. Both of these features improve the maintenance of XHTML 2.0 documents.

Based on the above facts, our conclusion is that XHTML 2.0 is a major step in the development of web technology. The new features require major changes in user agents, though. In addition, the authors have to learn how to use the new features. In our opinion, the advantages brought by XHTML 2.0 outweigh the problems. For instance, XHTML 2.0 removes the need for heavy scripting and separates the content from presentation. Therefore, XHTML 2.0 documents are easier to maintain and reuse.

8 Conclusions

In this paper, we have discussed the new XHTML 2.0 specification and its user agent implementation. According to the previous section, the XHTML 2.0 documents are easier to maintain and reuse, because they require less scripting and the content is really separated from the presentation. The introduction of XHTML 2.0 requires major changes in user agents, though.

Our XHTML 2.0 implementation is part of the X-Smiles user agent. The implementation is modular and mainly based on already existing components. No changes had to be made to the parsing and processing of XML documents. The XHTML module uses the well established DOM interface to access the XML data. The DOM documents are rendered using Java Swing document model. The XHTML 2.0 documents can be styled by CSS style sheet and they can contain XForms elements. We reused an already existing CSS processor.

Unfortunately, we could not implement all XForms specific CSS features. Currently, we are removing the Swing dependency form the CSS layout engine. In future, we plan to use the new CSS layout engine with the XHTML 2.0 and XForms components. Then, we expect to be able to implement all XForms specific features. For instance, at the time of writing we have already implemented all the XForms pseudo-elements in the new version of the layout engine.

The results of this paper show that implementation of XHTML 2.0 requires further development of user agents, but is not unrealistic. User agents, which already support XHTML 1.0 can be updated to support XHTML 2.0. Already existing components can be used in most cases. Generally, changes below the DOM interface are not required, albeit XForms requires DOM Level 3 support. Also, existing XML Schema and XPath engines can be used. Some changes to the CSS layout model can be expected, though. In some cases, the XForms module is the only new component.

In our future work, we plan to study the use of XHTML 2.0 in restricted devices. The main research question is how all the components required by XHTML 2.0 can be fitted in to a restricted devices with less memory and processing power than desktop devices. Most problems are related to XForms. At the moment, it remains to be seen whether XForms Basic will help in this problem.

9 Acknowledgments

The author Mikko Honkala would like to thank to Nokia Oyj Foundation for providing support during the research. The research was funded by the XML Devices and GO-MM projects to whose partners the authors would like to express their gratitude.

References

1. Ragget, D. et al., "HTML 4.01 Specification," *W3C Recommendation*, December 1999. Available at http://www.w3.org/TR/html401/
2. Bray, T. et al., "Extensible Markup Language (XML) 1.0," *W3C Recommendation*, February 2004. Available at http://www.w3.org/TR/2004/REC-xml-20040204/
3. Axelsson, J. et al., "XHTML 2.0," *W3C Working Draft*, May 2003. Available at http://www.w3.org/TR/xhtml2/
4. ECMA-262, ECMAScript language specification, European Computer Manufacturers Association (ECMA), 1998.
5. Boyer, J., and Honkala, M., "The XForms Computation Engine: Rationale, Theory and Implementation Experience," in *Proc. of the 6th IASTED International Conference, Internet and Multimedia Systems, and Applications*, (IMSA 2002), August 12-14, 2002, Kauai, Hawaii, USA.
6. Dubinko, M. et al., "XForms 1.0," *W3C Recommendation*, October 2003. Available at http://www.w3.org/TR/2003/REC-xforms-20031014/
7. Pilgrim, M. The Vanishing Image: XHTML 2 Migration Issues," O'Reilly xml.com, July 02, 2003. Available at: http://www.xml.com/pub/a/2003/07/02/dive.html
8. McCarron, S. et al., "XML Events," *W3C Recommendation*, October 2003. Available at http://www.w3.org/TR/xml-events/
9. Marsh. J., "XML Base," *W3C Recommendation*, June 2001. Available at http://www.w3.org/TR/xmlbase/
10. Altheim, M. et al., "Modularization of XHTML," *W3C Recommendation*, April 2001. Available at http://www.w3.org/TR/xhtml-modularization/
11. Kendall, G.C., "XHTML 2.0: The Latest Trick," O'Reilly xml.com, August 07, 2002. Available at: http://www.xml.com/pub/a/2002/08/07/deviant.html
12. Honkala, M., and Vuorimaa, P., "Advanced UI features in XForms," in *Proc. of the 8th International Conference on Distributed Multimedia Systems*, September 25 - 28, 2002, pp. 715-722.
13. Vierinen, J., Pihkala, K., and Vuorimaa,P., "XML based prototypes for future mobile services," in *Proc. 6th World Multiconf. Systemics, Cybernetics and Informatics*, SCI 2002, pp. 135-140.
14. Pihkala, K., Honkala, M., and Vuorimaa, P., "A Browser Framework for Hybrid XML Documents," in *Proc. of the 6th IASTED International Conference, Internet and Multimedia Systems, and Applications*, (IMSA 2002), August 12-14, 2002, Kauai, Hawaii, USA, pp. 164-169.
15. Cogliati, A. et al., "XHTML and CSS Components in an XML Browser," in *Proc. of the 4th International Conference on Internet Computing*, IC 03, Las Vegas, USA, June 2003, pp. 563-569.
16. Pohja, M. and Vuorimaa, P., "Dynamic XHTML Layout Document for an XML Browser," in *Proc. of the 2nd IASTED International Conference on Communications, Internet, and Information Technology*, CIIT 2003, Scottsdale, AZ, USA, November 2003, pp. 355-360.