Journal of Web Engineering, Vol. 9, No. 2 (2010) 095–115
© Rinton Press

COMPARISON OF COMMON XML-BASED WEB USER INTERFACE LANGUAGES

MIKKO POHJA
Department of Media Technology, Aalto University
P.O. Box 15400, FI-00076 Aalto, Finland
mikko.pohja@hut.fi

Received August 1, 2009
Revised February 25, 2010

In addition to being a platform for information access, the World Wide Web is increasingly becoming an application platform. While web applications have several benefits compared to desktop applications, there are also some problems. With legacy HTML, for example, one cannot produce user interfaces such as those that users have become accustomed to with desktop applications. What worked for static documents is not sufficient for the complicated web applications of today. Several parties have addressed this problem by defining a specific UI description language. In addition, the renewal of HTML aims to enhance support for web applications. This study evaluated five XML-based UI description formats, including HTML 5, in order to determine which language is best suited for modern web application development. The study also assessed what kind of applications are suited to each format. The requirements for a Web UI description language from the literature were revised and three use cases were defined, through which the languages are evaluated. The paper also presents the model differences of the languages.

Keywords: Web User Interface Description Language, Web Application
Communicated by: D. Lowe & O. Pastor

1 Introduction

Commerce and communication tasks, such as the use of e-mail, are common today on the World Wide Web (WWW), as is a trend towards realizing higher interaction tasks, such as information authoring. The WWW has transformed, therefore, from a platform for information access into a platform for interactive services. The User Interface (UI) of an application was traditionally programmed as a stand-alone client using an imperative programming language, such as Java or C++ and component toolkits. The WWW changed that; web browsers can now be used as the client for applications on the Web, with the application UI written in HTML.

Moving applications to the WWW can bring many benefits. Such applications automatically become cross-platform applications, there is no need to install them, and a developer can add new features and bug fixes to a running application [1]. Unfortunately, however, some web technologies are now outdated and, in fact, were not even originally designed for the complex use cases of today's applications. HTML forms, for instance, define the main interaction of the web applications, even though they were not designed to describe complex, higher-interaction UIs. Their usage, along with client-side scripting, has led to poor usability, maintainability, re-use, and accessibility. In order to address those problems, many organizations have developed a declarative User Interface Description Language (UIDL).

The Web's success is partially due to the fact that anyone can create and publish a website. This, in turn, is largely due to declarative languages. The main difference between imperative and declarative languages is the control of the program. In a declarative language, a programmer only provides the logic of the program, with control left to the language. Leaving the control to the language means that there must be a predefined set of functions that the author can use. The programmer of an imperative language must also implement the control. These languages are more powerful, therefore, but also harder to learn. It has been said that people who are not familiar with programming can generally still utilize declarative languages.

Declarative languages, in addition to being modality- and device-independent, are more easily processed by accessibility and other tools, which serves to fix many of the problems found in the approaches with semantically lower levels (such as HTML forms and scripting). A language can either be declarative or imperative, or a combination of the two [2]. The declarative part of the language usually has functional elements, which reduce the need for the imperative part. On the other hand, the lack of functional elements makes the imperative part more important. For practical reasons, it is essential to find a balance between the level of semantics and expressiveness.

Examples of so-called hybrid UI languages include MXML [3], LZX [4], XAML [5], and XUL [6]. They all contain a declarative UI description part, which is complemented by a dedicated programming language. The World Wide Web Consortium (W3C), among others, started working on a declarative web application format. The abovementioned formats motivated this work and there seemed to be a demand for the ultimate standardized format. As it transpired, however, the industry at large was not ready for a completely new format,[a] preferring instead to start developing HTML in a backward compatible manner,[b] using the work done in WHAT WG[c] as a starting point. The development is still in progress, with one of its main goals being to enhance HTML support for the web applications.

This paper assesses whether such a goal is attainable by evaluating the suitability of HTML 5 [7], along with four other languages, to modern web applications. The other formats are XForms, which is an open standard from W3C, and three proprietary formats: LZX, XAML, and XUL. The main criteria for the selected languages were their declarative UI description models, their potential as competitors of HTML 5, their ability to build cross-platform applications, and the existence of several and widespread implementations. The research-oriented UI languages are beyond the scope of this paper.

The research work was conducted by performing a study of the related literature and revising the requirements of a UI description language. In addition, the paper defines use cases, which contain user interaction usually found in desktop applications. The use cases were implemented with the selected languages and had two main purposes: to expose the kind of flaws that the languages might have in terms of implementable features, and to identify

[a] Declarative Formats for Applications and User Interfaces, W3C Working Group Note, http://www.w3.org/TR/dfaui/

[b] Architectural vision for HTML/XHTML2/Forms Chartering, http://www.w3.org/2007/03/vision

[c] Web Hypertext Application Technology Working Group (WHAT WG) is a community that further develops HTML.

usability problems that the languages caused.

The usability of the use cases was evaluated through heuristic analysis [8]. The evaluators went through the applications and evaluated them against 10 usability heuristics. This is a light-weight evaluation method that is believed to identify the vast majority of usability problems [8]. In short, usability is a quality attribute that relates to how easy something is to use [9]. Web usability is an approach with which the usability guidelines are applied to websites. The web usability guidelines focus on how, for instance, layout, navigation, and user interaction methods (such as forms) should be used and how browser chrome must be taken into account when designing a website. [10] Some aspects of web usability also apply to web applications, as well as websites. This paper takes those aspects into account in the form of the usability heuristics and requirements of a UI description language.

This paper is an extension of a conference article [11] that evaluated XForms and XUL. The present paper includes three more formats to the comparison. The paper makes the four main contributions:

- Based on literature, a set of requirements for a Web UI description language is derived.
- Three descriptive use cases are designed and implemented in five different languages, HTML 5, XForms, XAML, LZX, and XUL.
- The languages are evaluated based on the derived requirements and the use case implementations.
- UI development and communication model differences of the languages are presented.

The paper is organized as follows. Sections 2 and 3 provide background information on the topic and review the related work. Section 4 discusses the research scope and reviews the research steps, followed by an introduction to UI development models of the languages in Section 5. The results of the work are presented in Section 6 and discussed in Section 7. Finally, Section 8 concludes the paper.

2 Background

Declarative user interface descriptions date back to before the advent of the WWW and HTML. They are used as part of a model-based UI design, in which the UI is built on a certain model. The model can be a description of tasks, data, or a presentation, among other things. The aim is to identify reusable components of the UI and to capture more knowledge in the model [12]. The author can specify what features the interface should have, rather than write programs that define how the features should work, which saves them from writing a lot of procedural code [13]. One study showed that an average of 48 percent of an application's code is devoted to the user interface portion [14]. Enhancing UI development may considerably boost the application development process.

W3C has recently formed an incubator group in order to determine whether there should be a specification of model-based UIs for web applications.[d] In addition, the ACM Transactions on Computer-Human Interaction journal recently published a special issue on User Interface Description Languages for Next Generation User Interfaces [15] that introduces novel

[d]http://www.w3.org/2005/Incubator/model-based-ui/

UIDL approaches. This paper evaluates how existing declarative user interface description approaches suit Web UI design.

Draheim et al. divided declarative UI description languages into four categories [16]. The first category includes traditional code-based description of UIs, in which the application logic embeds the UI description. The second category is a GUI-oriented document like HTML. Most of the languages discussed in this paper belong to the third category, which includes document-based GUIs. The difference between the second and third categories is the main purpose of a language. Languages that belong to the second category tend to be document description languages. In the third category, languages are tailored to the domain of GUIs, and the fourth category is a document-oriented approach. Draheim et al. introduced a UI description format, which belongs to the last category.

The present paper focuses on GUI-oriented documents and document-based GUIs. It also studies languages whose cross-platform implementations are readily available. This rules out some research-oriented UI languages, which are reviewed in the next Section. Popular Web toolkits such as Dojo[e]and Google Web Toolkit[f]are not included since their outcome is usually HTML + Javascript, which is naturally fairly close to HTML 5.

2.1 XForms

XForms 1.0 Recommendation [17] 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 mark-up to describe the most common operations in form-based applications [18]. It can use any XML grammar to describe the content of the form (the instance data). This also enables the creation of generic editors for different XML grammars with XForms. It is possible to create complex forms with XForms using declarative mark-up without resorting to scripting.

XForms is an abstract user interface description language and one of its design goals was to avoid mandating a certain modality. This means that it can be suited to describing user interfaces, which are realized in different modalities, such as the GUI and Speech.

Several XML vocabularies have been specified in W3C. Typically, an XML language is targeted for a certain purpose (e.g., XHTML for content structuring or SVG for 2D graphics). Moreover, XML languages can be combined. A compound document, which is an XML document that consists of two or more XML languages, can specify the user interface of an application. In this paper, XForms is combined with XHTML+CSS level 2 to realize the use cases. XForms 1.0 directly includes the following W3C specifications: XML Events, XPath 1.0, XML Schema Datatypes, and XML 1.0.

2.2 XUL

Mozilla has developed a UI description language called XML User Interface Language (XUL) [6]. The mark-up consists of widget elements like buttons, menus, etc. XUL applications, which are based on several W3C standards, include HTML 4.0, Cascading Style Sheets (CSS) 1 and 2, Document Object Model (DOM) Levels 1 and 2, JavaScript 1.5 (including ECMA-262 Edition 3 (ECMAScript)), and XML 1.0.

[e]Dojo Toolkit, http://dojotoolkit.org/
[f]Google Web Toolkit, http://code.google.com/webtoolkit/

The goal of XUL is to build cross-platform applications. The applications can be ported to all of the operating systems on which Mozilla runs (for example, Linux, Windows, Windows CE, and Mac OS X). The layout and appearance of XUL applications are separated from the application definition and logic. Moreover, the application can be localized for different languages and regions independently of its logic or presentation.

XUL can be complemented by some of the technologies that Mozilla has introduced. *The eXtensible Bindings Language (XBL)* is a mark-up language that defines new elements for XUL widgets. *Overlays* are XUL files used to describe extra content for the UI. *XPCOM* and *XPConnect* make it possible to integrate external libraries with XUL applications and, finally, *XPInstall* provides a way to package XUL application components with an install script [19].

2.3 HTML 5

HTML 5 [7], the successor to HTML 4.01, is currently a work-in-progress at W3C. HTML 5 aims to fix errors and problems on the previous version and add new features especially for web applications. In addition to HTML 4.01, HTML 5 includes new versions of XHTML 1 and DOM2 HTML API, which were previously defined in separate specifications.

HTML 5 defines two syntaxes, HTML 5 syntax and XML syntax, both of which result in a DOM presentation of a document. While earlier versions of HTML were based on SGML and used SGML parsing rules, HTML 5 has its own parsing rules.

HTML 5 introduces a number of new elements and attributes while removing many presentational elements and attributes. In addition, some elements will have new semantics [20]. HTML 5 is intended to be a complement for CSS and ECMAScript, which are also used in this paper.

2.4 XAML

EXtensible Application Markup Language (XAML) is a user interface mark-up language for Windows Presentation Foundation (WPF), which is a graphics subsystem of the Windows Vista operating system. XAML consists of features from both Microsoft Windows applications and web applications. In WPF, the application UI can be defined with a programming language (such as C#, C++, Visual Basic, etc.) or by XAML [5]. In other words, a developer can use XAML instead of C# or the equivalent to create the UI elements. Using XAML, it is possible to use specialized tools for application development. The tools generate XAML code to run on WPF.

2.5 LZX

LZX is the UI description language of the OpenLaszlo platform for creating web applications. OpenLaszlo is an open source project that consists of LZX and the OpenLaszlo Server, a Java Servlet that compiles LZX applications into either Flash or DHTML depending on the targeted run-time environment. LZX is an XML format that also includes ECMAScript snippets to describe the application logic. In LZX, the UI is described with concrete UI components, which can be tied into a data model.

3 Related Work

While there are a number of research-oriented UI languages, such as XIML [21] and UIML [22], these are outside of the scope of this paper and many of them have been examined in related research. Souchon and Vanderdonckt reviewed XML-compliant user interface description languages [23], comparing the general properties and the UI description capacities of the languages. They found XIML to be the most expressive language, whereas UIML was found to have the best software support. XUL was found to be less expressive.

Trewin et al. examined four XML languages UIML, XIML, XForms, and Alternative Interface Access Protocol (AIAP) [24] intended for abstract user interface representation [25]. They defined the requirements for the representations, which include high-level requirements like applicability to any target and any delivery context, personalization, flexibility, extensibility, and simplicity. They also defined technical requirements, most of which have been collected from the literature [26, 27]. The technical requirements consist of separating purpose from presentation, characteristics of interface elements and functions, flexibility in inclusion of alternate resources, compatibility with concrete user interfaces, support for different interaction styles, and support for remote control. XForms and AIAP were found to fulfill the requirements most fully, especially in terms of separating data from presentation and flexibility in resource substitution.

The requirements for a generic user interface description format are discussed in [28], which also presents an implementation of an integrated description of user interfaces for both graphical and voice modality. The proposed requirements are device independence, modality independence, and customizability concerning layout without restricting device independence.

Finally, there are numerous XML-based languages for the desktop GUI, including Glade,[g] while InfoPath addresses office applications [29]. Several proprietary formats are also available, including XAL[h]and IDEAL [30]. Finally, W3C's Device Independent Authoring Language (DIAL) [31] is close to the XForms + XHTML compound, which is evaluated in this paper.

4 Research Scope and Steps

The research area of the paper is *web application user interface technologies*, while the scope for the research is desktop-style user interfaces in the WWW environment. The following list enumerates the research steps, while the scoping, defined above, applies to all of the research steps:

1. The web application use cases are selected
2. Requirements of a UI description format from literature are collected
3. The selected languages are evaluated against the requirements
4. The use case implementations are evaluated through heuristic analysis [32]

The requirements for UI description languages are collected from different sources and have been assessed as being suitable for the Web context. In addition to requirements from the literature, a set of requirements were defined, which were considered to be important for Web UI description formats. These requirements are discussed in more detail in Section 4.2. The evaluation of the requirements is based on documented features of the languages and on

[g]Glade. Available at: http://glade.gnome.org/
[h]XAL, http://www.openxal.org/

an implementation experience gained during the study. In the heuristic analysis, the use case implementations are examined with 10 recognized usability principles. The analysis reveals possible usability problems in the languages.

4.1 Use cases

The selected use cases are from an existing content management system of an Internet magazine. The application is used on the WWW and is originally implemented with HTML 4.01 and CSS. Three user interfaces were selected from the system, each of which is difficult to realize properly with HTML 4.01. Firstly, the wire frame models of the use cases were designed, using general usability guidelines without considering possible restrictions of the languages. Users of the system are mainly journalists, who have experience using word processing programs and are familiar with concepts like *copy-paste*. Since file system operations are not within the focus of the paper, interfaces for saving and loading files have not been included in the user interface designs.

The design of the user interfaces in this paper is based on usability best practices [33] and user interface design patterns [34, 35, 36]. The usability of the interfaces has been validated by usage simulation [37] and heuristic analysis [8, 32].

Document Editor. The purpose of this user interface is to create and modify simple structural documents that could be web pages. The data in the document is limited to text, pre-existing images and pre-existing tables (created, for example, by the Table Editor user interface). Figure 1 shows a wire frame model of the Document Editor.

The structure of the document can be modified by marking text blocks with different existing styles (for example, heading 1, heading 2, text paragraph, notice, etc.). The marking is targeted to a selected text box. For simplicity, all styles are block-level styles, that is, they are always attached to the whole text block.

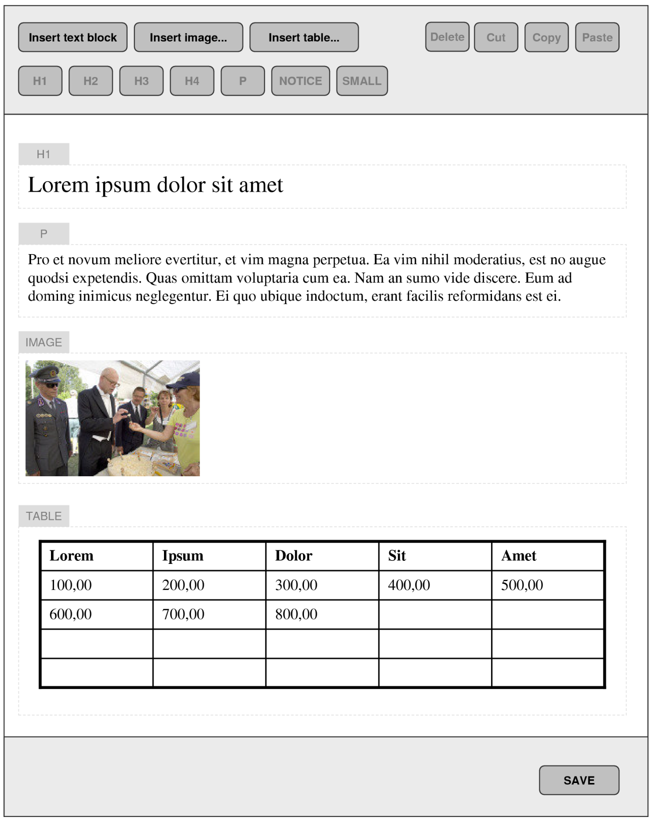
In order to keep the focus on the structure of the document in the interface, the images and tables cannot be modified in the document editor interface. A possible use case for the document editor is a journalist who reviews a laptop and completes it with images of the laptop and a table of its features.

Table Editor. This user interface creates and modifies simple tabular data, which can be displayed, for example, in a web page. The type of data in the table is limited to characters and numbers. Figure 2 shows a wire frame model of the Table Editor.

The user can also edit the structure of the tabular data by marking some of the columns or headers as headings and by entering header text for the entire table. The number of rows and columns in the table is user-editable. For simplicity, table cells are not allowed to span multiple columns or rows.

Tree Editor. The purpose of this user interface is to create and modify a tree structure for which the nodes of the tree have multiple editable attributes. The nodes are used in this paper to represent website areas for the site of a certain magazine, although they could represent anything. The Tree Editor is depicted in Figure 3.

Users can create new nodes, edit attributes, move nodes around in the tree, and delete nodes. A possible use case for the tree editor would be to manage the structure of an online magazine.



Lorem ipsum dolor sit amet

Pro et novum meliore evertitur, et vim magna perpetua. Ea vim nihil moderatius, est no augue quodsi expetendis. Quas omittam voluptaria cum ea. Nam an sumo vide discere. Eum ad doming inimicus neglegentur. Ei quo ubique indoctum, erant facilis reformidans est ei.

Lorem	Ipsum	Dolor	Sit	Amet
100,00	200,00	300,00	400,00	500,00
600,00	700,00	800,00		

Fig. 1. Wire frame model of the document editor.

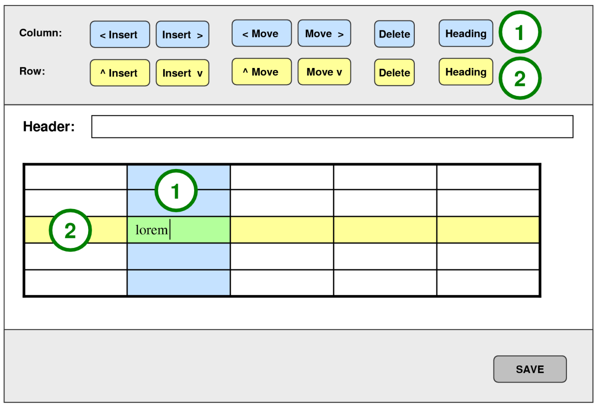


Fig. 2. Wire frame model of the table editor. (1) The active column is color coded to match the coloring of the buttons for manipulating the column. (2) The active row is color coded to match the coloring of the buttons for manipulating the row.

4.2 Requirements of a UI Description Language

The languages were evaluated against the requirements found from the literature, the results of which are shown in Tables 1-3 and discussed in the following subsections. The evaluation

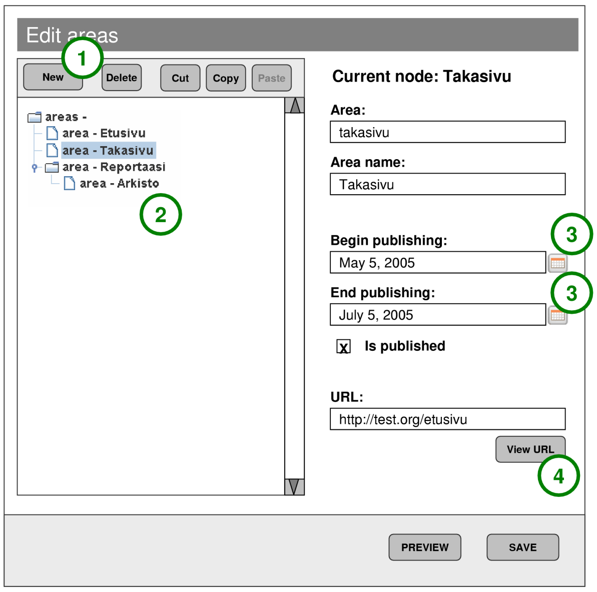


Fig. 3. Wire frame model of the tree editor. (1) Creates a new area as a child of the currently selected node. Data for the newly created area is entered from the form on the right. (2) Nodes in the tree are moved by dragging-and-dropping them. (3) Opens a calendar widget to select the date. (4) Opens the URL in a browser.

was conducted on a three-level scale, using the following levels and corresponding markers:

++ A built-in property. A language supports the property natively.

+ Possible to add or implement with another technology, such as scripts.

- Not possible with the language.

4.2.1 General Requirements

The general requirements in Table 1 are device and modality independence and customizability [28]. In other words, the UI representation must be generic and independent of any specific client device technology. The UI description format shall not restrict the modality and the format must provide a high degree of control over layout and graphical appearance. While all the languages meet the requirements, they do so on different levels.

By design, XForms is device and modality independent. XForms has an abstract UI description that makes it device independent, and it also uses data types, which makes it easy to utilize different modalities [38] because a user's input can be handled more specifically. In voice modality, for instance, grammar-based recognition can be made more accurate. Otherwise, the expected type of input must be defined for each modality separately.

While the UI description of other formats does not restrict the selection of devices, a developer must consider the target devices and implement respective UI versions. Other languages' UI elements can be transferred to other modalities, but a lack of data types requires extra work from a developer.

Every format provides control over layout and graphical appearance. In XForms, the UI elements are abstract, whereas the other formats have specific UI elements that are easier to customize.

Table 1. The general requirements of the UI description language [28].

Requirement	XForms	XUL	HTML 5	XAML	LZX
Device Independence	++	+	+	+	+
Modality Independence	++	+	+	+	+
Customizability	+	++	++	++	++

4.2.2 Technical Requirements

Trewin et al. introduced universal usability and technical requirements for abstract UI representations [25]. In their paper, XForms is evaluated against these requirements among three other languages. XForms and AIAP most closely met the requirements defined in the paper, particularly with regard to the separation of data from presentation and flexibility in resource substitution. With regard to the languages evaluated in this paper, all of the formats met the requirements of universal usability. The differences were identified in the technical requirements, which are as follows:

Separation of Interface Elements from their Presentation: Data and presentation information must be separated.

Interface Elements: The ability to define dependencies between interface elements: a UI description must enable automatic UI generation for any target environment and it must support data types.

Presentation Related Information: There must be a method for logically grouping the elements of a presentation. The presentation information must include resources such as labels and help text. To allow presentation replacement, it should be possible to share the core UI description between alternative UIs.

Run Time and Remote Control: Local computation reduces the latency of a network communication. The current state must be explicitly available at run time. Finally, there must be two-way communication between the target and the controller.

The evaluation against the technical requirements is shown in Table 2. The interface elements are not separated from their presentation in any format besides XForms, in which the data model can be accessed through a separate binding layer. While the interface elements can have dependencies in all formats, only in XForms and XAML is it a well built-in property. In other formats, the dependencies have to be realized through scripts. XForms is also easier to use in any target since its UI description is more abstract. HTML 5 and XForms support data types, whereas others do not.

The presentation can be grouped well with all languages. XForms provides an explicit of including labels and help texts, while in other formats they can be realized with normal text. All formats make it possible to provide an alternative presentation.

Table 2. The technical requirements [25].

Requirement	XForms	XUL	HTML 5	XAML	LZX
Separation of Interface Elements from Presentation					
Separation of Data/Pres.	++	-	-	-	-
Interface Elements					
Dependencies	++	+	+	++	+
Any Target	++	+	+	+	+
Data Types	++	-	++	-	-
Presentation Related Information					
Logical Groupings	++	++	++	++	++
Labels & Help Text	++	+	+	+	+
Presentation Replacement	+	+	+	+	+
Run Time and Remote Control					
Local Computation	++	+	+	+	+
Serialization	++	+	+	++	++
Synchronization	+	+	+	+	+

Local computations (such as data validation) can be realized with scripts in all formats. In addition, XForms provide native features for local computation. The state of the UI can be stored in a data instance in XForms, XAML, and LZX. LZX has some flaws when using multiple pointers for a single data instance. With these languages, the state of the UI is always available. In XUL and HTML 5, the state must be stored and queried separately. These differences are discussed in more detail in Section 5.

While two-way communication and synchronization between a target and controller is possible with all the formats, it is not actually a format dependent problem. In particular, server push, which would provide updates from a controller to a target, is more an issue with current Internet protocols than with the formats. However, HTML 5 defines a feature called Server Sent Events, which enables information push. In addition, the server push can be emulated, for instance, by Comet [39].

In summary, XForms most closely meets the requirements for universal interaction, while HTML 5 and XAML qualify slightly better than LZX and XUL.

4.2.3 Additional Requirements

The requirements in the literature were extended with a more detailed *typical interaction patterns* requirement set from the application scenario [11]. The additional requirements are:

Paging & Dialogs: A large UI is considered more usable if it is divided into smaller sections. This can be achieved, for instance, with wizard-type paging or with tabs.

Repeating constructs: Helps a developer to manage large data sets on the UI.

Nested constructs: The data sets are typically structured data, such as XML. Nested constructs help when presenting them on the UI.

Copy-paste: Helps users to edit content in an application.

Undo-redo: Users must be able to withdraw their actions.

Drag-and-drop: Direct manipulation interfaces are considered easy to use [40].

The results of evaluation against the additional requirements are shown in Table 3. Repeating structures, paging, and dialogs are natively supported by XForms and XAML, while XUL also supports paging. Implemention requires some script programming with other formats. Nested constructs are supported by all formats, while XUL, XAML, and LZX provide them natively, as does XForms, as long as a proposed XForms tree module is used [11]. HTML 5 requires scripting in order to complete the nested constructs.

Table 3. The additional requirements proposed in [11].

Requirement	XForms	XUL	HTML 5	XAML	LZX
Paging & Dialogs	++	++	+	++	+
Repeating constructs	++	+	++	++	++
Nested constructs	++*	++	+	++	++
Copy-paste	+	+	++	++	++
Undo-redo	+	+	++	++	+
Drag-and-drop	+	++	++	++	++

* Using the proposed tree extension.

Copy-paste can be used with all of the formats. HTML 5, XAML, and LZX support it natively (LZX only for text), whereas it can be implemented in XForms and XUL. Undo-redo and drag-and-drop have native support in HTML 5 and also in XAML with certain elements, while they can be implemented in XForms, XUL and LZX. For XForms, a developer must alter the data model accordingly, which makes it harder than with the other two. In summary, XAML handles the typical interaction patterns best. However, all formats qualify well from these requirements, even XForms, which is the most abstract of the formats covered here.

4.3 Use Case Implementations

The use case implementations are discussed below. Heuristic analysis was also conducted for the implementations according to the heuristics defined by Nielsen and Mack [32]. No major problems were found from the interfaces, mainly because the wire frame models were already designed according to the heuristics.

4.3.1 XForms

The XForms implementation of the use cases was done using the XForms 1.1 Working Draft[41] (W3C Work In Progress), which is implemented in the X-Smiles browser [42]. XForms1.1 has several features that make it possible to minimize scripting. The main features utilized from XForms 1.1, to avoid the need to use scripting, are duplicate and destroy actions and media type-aware output rendering.

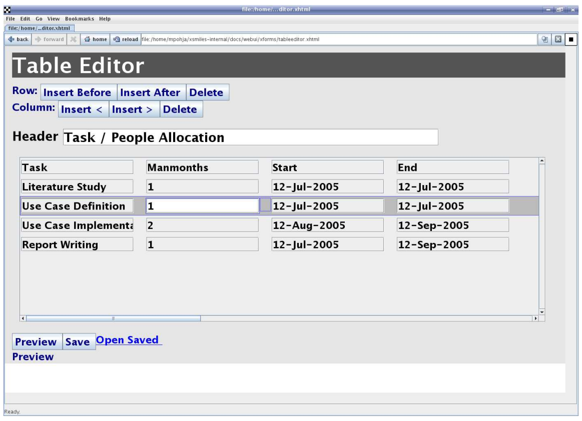
The XForms language was extended with a tree module [11], since there is no way in XForms 1.1 to select nodes from a recursive structure. It is implemented, as a proof of concept, in the X-Smiles XForms implementation.

The user interface state is completely contained in the XForms model, and can therefore be automatically serialized and submitted to a server without any additional scripting.

Document Editor: The document editor relies on XForms repeat and dynamic UI bindings. It requires a few XForms 1.1 features, namely destroy and duplicate and output@mediatype, which it utilizes heavily. This UI has no scripting.

Tree Editor: The tree editor uses the proposed XForms Tree extension. All other dynamic features are done using XForms UI bindings. This UI has no scripting.

Table Editor: This table editor UI (cf. Figure 4) is written in XForms 1.1, but it has a small script for inserting, deleting and moving columns. This could be avoided if XForms had repeating and conditional action containers (such as *for* and *if*).


file:/home/...ditor.xhtml
File Edit Go View Bookmarks Help
back forward home reload
file:/home/mpohja/xsmiles-internal/docs/webui/xforms/tableeditor.xhtml
Table Editor
Row: Insert Before Insert After Delete
Column: Insert < Insert > Delete
Preview Save Open Saved
Preview
Ready.

Header Task / People Allocation

Task	Manmonths	Start	End
Literature Study	1	12-Jul-2005	12-Jul-2005
Use Case Definition	1	12-Jul-2005	12-Jul-2005
Use Case Implementa	2	12-Aug-2005	12-Sep-2005
Report Writing	1	12-Jul-2005	12-Sep-2005

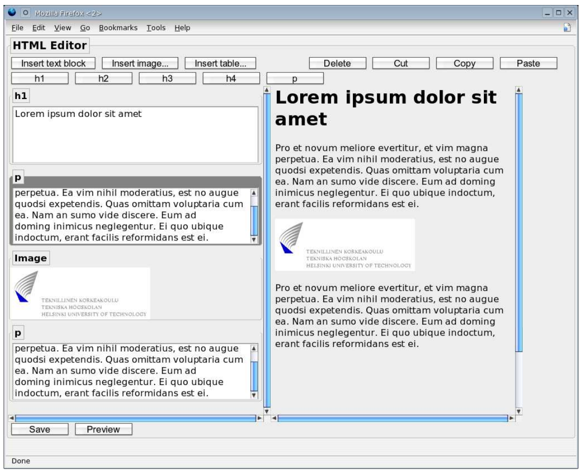
Fig. 4. XForms Table Editor.

4.3.2 XUL

In addition to the wire frame model designs, XUL made it possible to use context menus in Document and Tree Editors. In addition, Document Editor has a real-time preview of a document. XUL interfaces require a lot of scripting; all the button functions, drag-and-dropping, and focusing of elements has to be realized through scripts. It was possible to implement use cases with XUL and, for instance, drag-and-drop was straightforward to realize. The XUL Document editor is depicted in Figure 5.

4.3.3 XAML

The XAML implementations were compiled to XAML Browser Applications (XBAP). XBAP applications are web applications that are run on Internet Explorer. XAML was used to describe the UIs of the use cases and the functionality was implemented in C#. The UI development was straightforward with XAML because it provides a wide set of UI components that sped up implementation of the use cases. Implementing the functionality with C# was also easy because XAML elements are effectively C# objects. Drag-and-drop is natively supported only for selected elements; authors must implement it for other elements by themselves. In addition, it is practically impossible to use it with elements that are created from a data model because the system cannot direct the drag operation to a correct element. The Document Editor in XAML is depicted in Figure 6.



Lorem ipsum dolor sit amet

perpetua. Ea vim nihil moderatius, est no augue quodsi expetendis. Quas omittam voluptaria cum ea. Nam an sumo vide discere. Eum ad doming inimicus neglegentur. Ei quo ubique indoctum, erant facilis reformidans est ei.

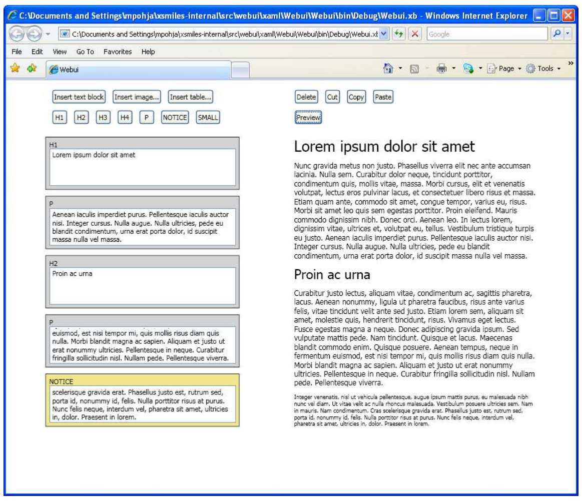
perpetua. Ea vim nihil moderatius, est no augue quodsi expetendis. Quas omittam voluptaria cum ea. Nam an sumo vide discere. Eum ad doming inimicus neglegentur. Ei quo ubique indoctum, erant facilis reformidans est ei.

Lorem ipsum dolor sit amet

Pro et novum meliore evertitur, et vim magna perpetua. Ea vim nihil moderatius, est no augue quodsi expetendis. Quas omittam voluptaria cum ea. Nam an sumo vide discere. Eum ad doming inimicus neglegentur. Ei quo ubique indoctum, erant facilis reformidans est ei.

Pro et novum meliore evertitur, et vim magna perpetua. Ea vim nihil moderatius, est no augue quodsi expetendis. Quas omittam voluptaria cum ea. Nam an sumo vide discere. Eum ad doming inimicus neglegentur. Ei quo ubique indoctum, erant facilis reformidans est ei.

Fig. 5. XUL Document Editor.



C:\Documents and Settings\mpohja\xsmiles-internal\src\webui\xaml\Webui\Webui\bin\Debug\Webui.xt

Lorem ipsum dolor sit amet

Aenean iaculis imperdiet purus. Pellentesque iaculis auctor nisi. Integer cursus. Nulla augue. Nulla ultricies, pede eu blandit condimentum, urna erat porta dolor, id suscipit massa nulla vel massa.

Proin ac urna

euismod, est nisi tempor mi, quis mollis risus diam quis nulla. Morbi blandit magna ac sapien. Aliquam et justo ut erat nonummy ultricies. Pellentesque in neque. Curabitur fringilla sollicitudin nisl. Nullam pede. Pellentesque viverra.

scelerisque gravida erat. Phasellus justo est, rutrum sed, porta id, nonummy id, felis. Nulla porttitor risus at purus. Nunc felis neque, interdum vel, pharetra sit amet, ultricies in, dolor. Praesent in lorem.

Lorem ipsum dolor sit amet

Nunc gravida metus non justo. Phasellus viverra elit nec ante accumsan lacinia. Nulla sem. Curabitur dolor neque, tincidunt porttitor, condimentum quis, mollis vitae, massa. Morbi cursus, elit et venenatis volutpat, lectus eros pulvinar lacus, et consectetuer libero risus et massa. Etiam quam ante, commodo sit amet, congue tempor, varius eu, risus. Morbi sit amet leo quis sem egestas porttitor. Proin eleifend. Mauris commodo dignissim nibh. Donec orci. Aenean leo. In lectus lorem, dignissim vitae, ultrices et, volutpat eu, tellus. Vestibulum tristique turpis eu justo. Aenean iaculis imperdiet purus. Pellentesque iaculis auctor nisi. Integer cursus. Nulla augue. Nulla ultricies, pede eu blandit condimentum, urna erat porta dolor, id suscipit massa nulla vel massa.

Proin ac urna

Curabitur justo lectus, aliquam vitae, condimentum ac, sagittis pharetra, lacus. Aenean nonummy, ligula ut pharetra faucibus, risus ante varius felis, vitae tincidunt velit ante sed justo. Etiam lorem sem, aliquam sit amet, molestie quis, hendrerit tincidunt, risus. Vivamus eget lectus. Fusce egestas magna a neque. Donec adipiscing gravida ipsum. Sed vulputate mattis pede. Nam tincidunt. Quisque et lacus. Maecenas blandit commodo enim. Quisque posuere. Aenean tempus, neque in fermentum euismod, est nisi tempor mi, quis mollis risus diam quis nulla. Morbi blandit magna ac sapien. Aliquam et justo ut erat nonummy ultricies. Pellentesque in neque. Curabitur fringilla sollicitudin nisl. Nullam pede. Pellentesque viverra.

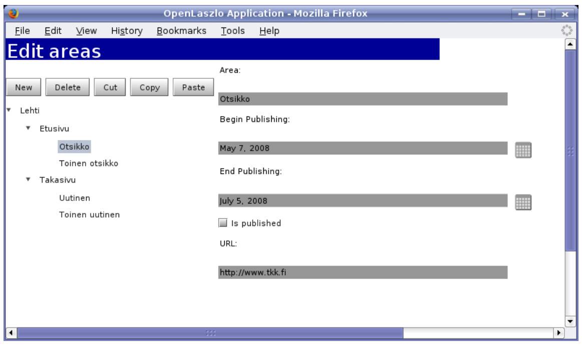
Integer venenatis, nisl ut vehicula pellentesque, augue ipsum mattis purus, eu malesuada nibh nunc vel diam. Ut vitae velit ac nulla rhoncus malesuada. Vestibulum posuere ultricies sem. Nam in mauris. Nam condimentum. Cras scelerisque gravida erat. Phasellus justo est, rutrum sed, porta id, nonummy id, felis. Nulla porttitor risus at purus. Nunc felis neque, interdum vel, pharetra sit amet, ultricies in, dolor. Praesent in lorem.

Fig. 6. XAML Document Editor.

4.3.4 LZX

The LZX implementations ran on an OpenLaszlo platform version 4.0.2 and the programs were compiled to DHTML. The Flash compilations differ to some degree from DHTML. For instance, it is not possible to access the local file system from a Flash application, while the UI controls are naturally different.

Even though LZX provides a data instance for an application, its usage is not straightforward. The data instance can provide data for an input element but it will not be updated automatically when a user modifies the data. An author must create a handler to update the data instance. In addition, the tree editor (cf. Figure 7) had to be implemented with two data instances. The tree editor form could not utilize the data instance of the tree, instead it had to have a data instance of its own that was updated along with the focus changes.


OpenLaszlo Application - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Edit areas
New Delete Cut Copy Paste
Lehti
Etusivu
Otsikko
Toinen otsikko
Takasivu
Uutinen
Toinen uutinen

Area: Otsikko

Begin Publishing: May 7, 2008

End Publishing: July 5, 2008

☐ Is published

URL: http://www.tkk.fi

Fig. 7. LZX Area Editor.

There were also some difficulties implementing drag-and-drop functionality and the HTML preview of the document editor. While the UI elements can be dragged, dropping them at certain locations and moving other elements accordingly is not easy and authors must implement these actions themselves. HTML previews are possible, but LZX supports a minimal set of embedded HTML elements, which means that an author must define the appearance of most of the elements.

While the overall concept of LZX is good, the implementation is not very sophisticated. Some design decisions, such as putting the content of the UI, styling, and logic into a single file, would require major revisions.

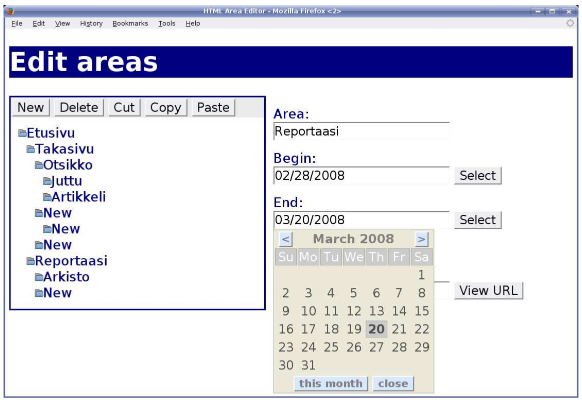
4.3.5 HTML 5

HTML 5 is still a work-in-progress at W3C and, understandably, no full implementations are available yet. Nevertheless, parts of the specification have been implemented in browsers such as Firefox, Internet Explorer, Konqueror, Opera, and Safari. Depending on the feature, therefore, the use case implementations could be tested using several browsers. In any case, a feature like undo-redo, for example, is verified only in the specification.

Compared to HTML 4.01, HTML 5 contains several improvements in terms of web applications, which has, of course, been the goal of the renewal campaign. In order to make the new version backwards compatible with earlier versions, the new specification must retain some fundamental structural choices, such as tying the data to the form elements. In addition, HTML 5 lacks the tree element. The area editor (cf. Figure 8) is realized with *div* elements and ECMAScript. Apart from this, the required features could be implemented with HTML 5.

5 Model Differences between Languages

Although usability did not differ greatly in the implementation of the selected use cases on a desktop computer, there are differences in the languages from a developer's perspective, most significantly in developing the user interface design and communication between the server and the UI.



Edit areas

Area: Reportaasi

Begin: 02/28/2008

End: 03/20/2008

Fig. 8. HTML 5 Area Editor.

5.1 *UI Development Model*

Most of the XML-based user interface definition languages, including XUL, LZX, and XAML, are widget-based. HTML 5 can be included in this category. Being widget-based means that they are quite concrete and authors add widgets, such as buttons and text areas, to the user interface. This makes it easy to create a user interface layout graphically, but, depending on the language, all or part of the user interface logic has to be programmed using a programming or scripting language. XForms, in contrast, starts by defining an XML data model and all operations are done to the data model using declarative actions and XPath expressions. User interface is automatically kept up-to date with a dynamic dependency tracking.

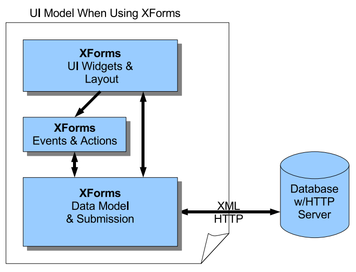
5.2 *Communication between UI and Back-end System*

For the communication between UI and back-end system, the user interface state has to be serialized for transmission. Having received a serialized reply from the server, it must then be transformed into application state. That serialization is automatic in XForms (cf. Figure 9a), since the data model is a live XML document object model, which is automatically serialized and de-serialized. In addition, the submission is automated.

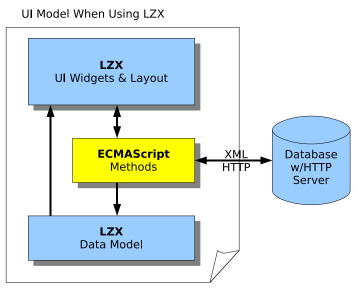
LZX and XAML also have data models, which means that, while serialization is automatic, the submission must be realized through a programming language. In addition, in LZX an author must implement a logic to keep the data model up-to-date (see Figure 9b). The data model provides data for UI, but the UI cannot automatically update the data model; that must be implemented separately for each case. In XAML (cf. Figure 9c), the data model updates are automatic; otherwise, it is similar to the LZX model.

XUL and HTML 5 have no explicit data models, and communication between a backend process and the user interface has to be re-implemented using ECMAScript for each user interface, as shown in Figure 9d. For example, when the server sends back updated structured content, there has to be a script that updates the corresponding DOM. This means that authoring and maintaining XUL- or HTML 5-based applications is more complicated than it is for others.

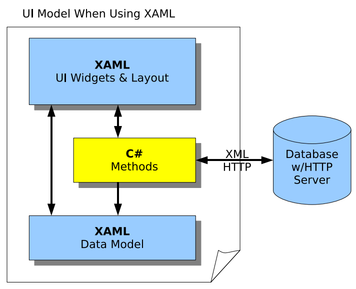
In summary, the more a developer has to implement logic to serialize and utilize the data, the more difficult the whole development becomes. The automatic serialization and dynamic


UI Model When Using XForms
XForms UI Widgets & Layout
XForms Events & Actions
XForms Data Model & Submission
XML HTTP
Database w/HTTP Server

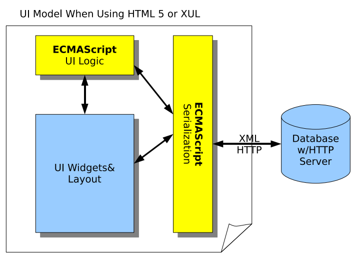
(a) XForms



(b) LZX



(c) XAML



(d) HTML 5 and XUL

Fig. 9. Communication between the UI and the back-end system.

bindings of XForms makes it easiest to build a UI, while the use of XAML and LZX data models is partially automated. HTML 5 and XUL require the most work in terms of using and serializing the UI data of an application. All the logic for handling the data must be implemented for each application separately.

6 Result of the Evaluation

The languages were evaluated against three requirement sets. In addition, the paper assessed the usability of the use case implementations and the communication model between the UI and the server. The results are summarized in Table 4. The summary uses a three-level scale with the following levels and corresponding markers:

+++ Excellent. A language qualifies well against a criterion without significant flaws.

++ Good. A language may have flaws, but still qualifies well.

+ Moderate. A language has some major problems, but it is still usable.

In conclusion, XForms and XAML were the best languages overall, despite having some flaws. HTML 5 and LZX were the second-best options, qualifying evenly in all categories without excelling in any. Although XUL had flaws in every category, its end result was equal to those of the other implementations. In other words, the difference is only significant for developers.

Table 4. Summary of results.

Criterion	XForms	XUL	HTML 5	XAML	LZX
General Requirements	++	+	+	+	+
Technical Requirements	+++	+	++	++	+
Additional Requirements	+	+	++	+++	++
Usability of the UC Implementations	++	++	++	++	++
Communication Model	+++	+	+	++	++

7 Discussion

The requirement conformance also reveals what types of applications the languages suit best. XForms copes best with the universal interaction requirements, but is weakest with typical interaction patterns, which are best fulfilled with languages targeted to application UIs. XForms' main target is complicated forms and, as a result, it simplifies the management of large data set on the client-side. XForms also excels when there are accessibility requirements. With XForms' abstract UI description, it is easier to develop applications for different modalities and devices. Customizability is important when using an application in different contexts [43], and is easiest in languages with specific UI elements, including XUL, HTML 5, XAML, and LZX. Finally, a desktop-like UI with direct manipulation is best achieved with XAML.

The different types of requirements are somewhat at variance with each other. For instance, conforming well to technical requirements may make it difficult to implement all the typical interaction patterns. As an example, realizing drag-and-drop with XForms' UI independence is rather complicated.

A data model makes it easier to develop a web application because the application data can be in the same format throughout the application, even on the server-side. On the client-side, a single data can be used by several UI components and its modification is reflected in all elements by which it is referenced. Unfortunately, this is the case only in theory. In practice, the usability of the data models differs considerably among the technologies. XForms offers a flexible data model, which can be referenced and accessed from any UI element or binding. In addition, the submission of the data is integrated into the XForms model. LZX, in contrast, has very limited data model usage. It cannot be utilized to a full extent, because UI elements do not update the data model and referring is not flexible. The blinkered data model usage vitiates a versatile UI description language design such as LZX. In fact, LZX would notably benefit from the XForms data model; after all, XForms allows any format to be used to describe the UI, or any modality for that matter.

Like data models, the drag-and-drop implementations vary significantly between the formats. At minimum, there must be way to define draggable objects and drop spots, and operation must reflect the serialized data structure. In HTML 5 and XUL, the data are tied to UI elements, so the dragged element changes the order of the data automatically. In other formats, an author must keep the data model up to date. XAML provides drag and drop only for selected elements and, with LZX, the drop spots cannot be specified. In summary, HTML 5 and XUL fulfilled the requirements most fully, whereas XForms did not provide any drag and drop support.

HTML 5 specification says in its scope definition: "For sophisticated cross-platform applications, there already exist several proprietary solutions (such as Mozilla's XUL and Macromedia's Flash)." However, in light of the results of this paper, HTML 5 can evenly compete with the abovementioned proprietary technologies. Even though it lacks some of the features that other formats have, a notable non-functional benefit of HTML 5 is that it is backward compatible with HTML 4.01, the most popular UI format ever.

The level of semantics and expressiveness varies between the languages. XForms provides a rich declarative use of UI data and it is easy to define interdependencies of the data and the UI. XAML and LZX also have data models, but dependencies must be mainly defined by a programming language, especially in LZX. While HTML 5 has several declarative functionalities compared to legacy HTML, it still relies heavily on scripting, as does XUL.

8 Conclusions

This paper studied five UI description languages, which were evaluated against a wide set of requirements found in the literature. In addition, three use cases were implemented with all the languages. The use cases were typical to the web application UIs, but are difficult to realize properly with legacy HTML. Finally, the model differences of the UI formats were compared.

XForms, HTML 5, and XAML fulfilled the requirements best, while XUL and LZX had slightly lower compliance with the requirements. The best use case implementations were done in XUL, HTML 5, and XAML, which were most like the original wire frame models. Even though the XForms and LZX implementations lack some designed features, the use cases could be realized well with them. Therefore, there is no single format that would suit all types of applications; the choice depends on the required features and scope of the application. XForms is best suited for data-intensive applications and applications that have accessibility requirements. XAML is best for providing a desktop-like UI, while context-sensitive applications are easiest to implement with HTML 5, XUL, LZX, and XAML.

The results show that developing legacy HTML incrementally has been sufficient to date, and the work-in-progress HTML 5 can evenly compete with the other formats. Taking its previous dominance as a web application description language into account, it is difficult to replace by any format.

Implementation experience shows that the use of a data model eases the development of an application. One can start designing the data model and how it is accessed, and the data will eventually become the centerpiece of an interactive application. The data model might, however, be difficult to integrate with the drag-and-drop functionality.

Acknowledgements

The author would like thank his co-authors of the preceding article, particularly Mikko Honkala, who implemented the use cases in XForms and wrote half of the article, Miemo Penttinen, who designed the wire frame models of the use cases, and Petri Vuorimaa, the leader of the research group.

References

1. Mehdi Jazayeri. Some Trends in Web Application Development. In *FOSE '07: 2007 Future of Software Engineering*, pages 199–213, Washington, DC, USA, 2007. IEEE Computer Society.
2. Mikko Honkala. *Web User Interaction - A Declarative Approach Based on XForms.* PhD thesis, Helsinki University of Technology, Finland, December 2006.
3. Adobe. Adobe Flex 3 - Developer Guide. Technical report, Adobe, 2008.
4. Laszlo Systems, Inc. OpenLaszlo Application Developer's Guide. Technical report, Laszlo Systems, Inc., February 2008. Available online: http://www.openlaszlo.org/lps4/docs/developers/.
5. David F. Sklar and Andy van Dam. An Introduction to Windows Presentation Foundation. Windows Vista Technical Articles, Microsoft Developer Network (MSDN), September 2005.
6. David Hyatt. XML user interface language (XUL) 1.0. Mozilla.org, 2001.
7. Ian Hickson and David Hyatt, editors. *HTML 5.* W3C Working Draft, January 2008.
8. Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 249–256, New York, NY, USA, 1990. ACM Press.
9. Jakob Nielsen and Hoa Loranger. *Prioritizing Web Usability.* New Riders, Berkeley, CA, USA, 2006.
10. Jakob Nielsen. *Desinging Web Usability: The Practice of simplicity.* New Riders, CA, USA, 1999.
11. Mikko Pohja, Mikko Honkala, Miemo Penttinen, Petri Vuorimaa, and Panu Ervamaa. Web User Interaction – Comparison of Declarative Approaches. In *Web Information Systems and Technologies*, volume 1 of *Lecture Notes in Business Information Processing*, pages 190–203. Springer Berlin Heidelberg, August 2007.
12. James Foley, Won Chul, Srdjan Kovacevic, and Kevin Murray. The User Interface Design Environment. *SIGCHI Bull.*, 20(1):77–78, 1988.
13. Angel R. Puerta and Pedro Szkeley. Model-Based Interface Development. In *CHI '94: Conference companion on Human factors in computing systems*, pages 389–390, New York, NY, USA, 1994. ACM.
14. Myers, Brad A. and Rosson, Mary Beth. Survey on user interface programming. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 195–202, New York, NY, USA, 1992. ACM.
15. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(4), November 2009.
16. Dirk Draheim, Christof Lutteroth, and Gerald Weber. Graphical user interfaces as documents. In *CHINZ '06: Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction*, pages 67–74, New York, NY, USA, 2006. ACM Press.
17. Micah Dubinko, Leigh L. Klotz, Roland Merrick, and T. V. Raman. XForms 1.0. W3C Recommendation, 2003.
18. Richard Cardone, Danny Soroker, and Alpana Tiwari. Using XForms to simplify web programming. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 215–224, New York, NY, USA, 2005. ACM Press.
19. Peter Bojanic. The Joy of XUL. Available online http://www.mozilla.org/projects/xul/joy-of-xul.html, December 2003.
20. Anne van Kesteren. HTML 5 differences from HTML 4. Working Draft, W3C, January 2008.
21. Angel Puerta and Jacob Eisenstein. XIML: a common representation for interaction data. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 214–215, New York, NY, USA, 2002. ACM Press.
22. Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. UIML: an appliance-independent XML user interface language. In *WWW '99: Proceeding of the eighth international conference on World Wide Web*, pages 1695–1708, New York, NY, USA, 1999. Elsevier North-Holland, Inc.
23. Nathalie Souchon and Jean Vanderdonckt. A review of XML-compliant user interface description languages. In *Proceedings of the 10th International Workshop on Interactive Systems. Design, Specification, and Verification: DSV-IS 2003.* Springer, 2003.
24. Gottfried Zimmermann, Gregg Vanderheiden, and Al Gilman. Prototype Implementations for

a Universal Remote Console Specification. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 510–511, New York, NY, USA, 2002. ACM.

25. Shari Trewin, Gottfried Zimmermann, and Gregg Vanderheiden. Abstract user interface representations: how well do they support universal access? In *CUU '03: Proceedings of the 2003 conference on Universal usability*, pages 77–84. ACM Press, 2003.
26. J., Myers B., Harris T.K., Rosenfeld R., Shriver S., Higgins M., and Hughes J. Nichols. Requirements for automatically generating multi-modal interfaces for complex appliances. In *ICMI '02: Proceedings of the 4th IEEE International Conference on Multimodal Interfaces*, page 377, Washington, DC, USA, 2002. IEEE Computer Society.
27. Gottfried Zimmermann and Gregg Vanderheiden. Technical Requirements for a Delivery Context Independent User Interface Model. In *W3C Workshop on Device Independent Authoring Techniques*, September 2002. Available online: http://www.w3.org/2002/07/DIAT/posn/trace.html.
28. Rainer Simon, Michael Jank Kapsch, and Florian Wegscheider. A generic uiml vocabulary for device- and modality independent user interfaces. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 434–435, New York, NY, USA, 2004. ACM Press.
29. Michael Hoffman. Architecture of microsoft office infopath 2003. Microsoft Developer Network, October 2003.
30. José Manuel Cantera Fonseca, Ignacio Marín Prendes, Javier Soriano, and Juan J. Hierro. Declarative Models for Ubiquitous Web Applications. In *W3C workshop on declarative models of distributed web applications*, April 2007. Available online: http://www.w3.org/2007/02/dmdwa-ws/Papers/jose-m-c-fonseca.html.
31. Kevin Smith. Device Independent Authoring Language (DIAL). Working Draft, W3C, May 2006. http://www.w3.org/TR/dial/.
32. Jakob Nielsen and Robert L. Mack, editors. *Usability Inspection Methods*. John Wiley and Sons, New York, NY, USA, 1994. Also available at http://www.useit.com/papers/heuristic/heuristic_list.html.
33. Alan Cooper. *About Face: The Essentials of User Interface Design*. John Wiley & Sons, August 1995.
34. Jenifer Tidwell. Common Ground: A Pattern Language for Human-Computer Interface Design. Available online http://www.mit.edu/~jtidwell/interaction_patterns.html, May 1999.
35. Jenifer Tidwell. *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly Media, Inc., 1. edition, November 2005.
36. Sari Laakso. User Interface Design Patterns. Available online http://www.cs.helsinki.fi/u/salaakso/patterns/, September 2003.
37. Jennifer Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design*, chapter 13. Wiley, 1st edition, January 2002.
38. Mikko Honkala and Mikko Pohja. Multimodal Interaction with XForms. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 201–208, New York, NY, USA, 2006. ACM.
39. Alex Russell. Comet: Low Latency Data for the Browser. Weblog, March 2006. Available online: http://alex.dojotoolkit.org/?p=545.
40. Ben Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8):57–69, 1983.
41. John Boyer, David Landwehr, Roland Merrick, and T. V. Raman. XForms 1.1. W3C Working Draft, 2004.
42. Petri Vuorimaa, Teemu Ropponen, Niklas von Knorring, and Mikko Honkala. A Java based XML browser for consumer devices. In *17th ACM Symposium on Applied Computing*, pages 1094–1099, Madrid, Spain, March 2002.
43. Gerti Kappel, Birgit Proll, Werner Retschitzegger, and Wieland Schwinger. Customisation for ubiquitous web applications: a comparison of approaches. *Int. J. Web Eng. Technol.*, 1(1):79–111, 2003.