# WEB USER INTERACTION
## *Comparison of Declarative Approaches*

Mikko Pohja,* Mikko Honkala,* Miemo Penttinen,† Petri Vuorimaa* and Panu Ervamaa†
*Telecommunications Software and Multimedia Laboratory, Helsinki University of Tehcnology
P.O. Box 5400, FI-02015 HUT, Finland

†Frantic Media
Arabianranta 6, FI-00560 Helsinki, Finland

Keywords: Web User Interface, XForms, XUL.

Abstract: The World Wide Web is evolving from a platform for information access into a platform for interactive services. Several applications are already used through Internet and Web browsers. User interface of such an application is defined by HTML. However, HTML has its deficiencies when used as a general UI description language. Several parties have addressed this problem by defining specific UI description languages. Thus, for instance, a web browser could be used as a user interface for any application. We have revised the requirements for a UI description language from literature and evaluated two XML-based UI description formats against the requirements through use cases.

## 1 INTRODUCTION

Commerce and communication tasks, such as using e-mail, are common today in the World Wide Web (WWW). Also, there is a trend to realize higher interaction tasks, such as information authoring, over the WWW. Therefore, WWW is transforming from a platform for information access into a platform for interactive services (Hostetter et al., 1997). Traditionally, application User Interfaces (UI) were programmed as stand-alone clients using procedural programming languages, such as Java or C++ and component toolkits. WWW changed that; any browser can be used as the client when accessing applications in the Web, and the application UI is written in platform independent HTML.

Unfortunately, some of the technologies used in the Web are outdated and, in fact, were not originally designed for the complex use case scenarios of today's applications. For instance, HTML forms are used as the main interaction definition, even though they were not designed to describe complex, higher-interaction UIs. Their usage (along with client-side scripting) has led to bad usability, maintainability, re-use, and accessibility. Therefore, a new paradigm for the Web is needed: the declarative UI. Declarative UI languages have usually a higher semantic level while traditional programming languages have more expressive power. Declarative languages, in addition to being modality and device independent, are more easily processed by accessibility and other tools, therefore fixing many of the problems found in the approaches with lower semantical level (e.g., HTML forms and scripting). For practical reasons, it is essential that a balance between semantical level and expressive power is found.

In this paper, we study two UI description languages and how they suit to Web applications. The languages are XForms (Dubinko et al., 2003) and XUL (Hyatt, 2001). They are selected, because they can be used to build cross platform applications and have already several implementations. The research work has been conducted by doing a literature study of related work, defining requirements for a UI description language, and defining and implementing use cases with selected languages. The results are comprised of evaluation of the languages against the requirements and heuristics analysis of the use case implementations.

The main contributions of this paper are the following:

- Based on literature, a set of requirements for a web user interface definition language is derived.
- Three descriptive use cases are designed and implemented in two different languages, XForms and XUL.
- XForms and XUL are evaluated based on the derived requirements and the use cases.

Pohja M., Honkala M., Penttinen M., Vuorimaa P. and Ervamaa P. (2006).
WEB USER INTERACTION - Comparison of Declarative Approaches.
In *Proceedings of WEBIST 2006 - Second International Conference on Web Information Systems and Technologies - Internet Technology / Web Interface and Applications*, pages 295-302
DOI: 10.5220/0001238802950302

- We propose an extension to XForms language for navigating and editing recursive structures.

The paper is organized as follows. The next Section gives background to the topic and reviews the related work. Section 3 discusses the research scope and the problem. In addition, it defines use cases. Results of the work are presented in Section 4. Finally, Section 5 concludes the paper.

# 2 BACKGROUND

## 2.1 Related UI Languages

The focus of this paper is UI languages, whose cross-platform implementations are readily available. Because of this, some research-oriented UI languages, such as XIML (Puerta and Eisenstein, 2002) and UIML (Abrams et al., 1999), are outside of the scope. In addition to the UI languages reviewed in this paper (XForms and XUL), there exists a whole array of XML-based UI definition languages, whose implementations can be verified. Those are reviewed in related research (Souchon and Vanderdonckt, 2003; Trewin et al., 2004). In addition, there exists numerous XML-based languages for the desktop GUI, including Glade[1], and Microsoft XAML (Rector, 2003), while InfoPath addresses office applications (Hoffman, 2003).

## 2.2 XForms

XForms 1.0 Recommendation (Dubinko et al., 2003) 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 markup to describe the most common operations in form-based applications (Cardone et al., 2005). It can use any XML grammar to describe the content of the form (the instance data). Thus, it also enables to create generic editors for different XML grammars with XForms. It is possible to create complex forms with XForms using declarative markup, without resorting to scripting.

XForms is an abstract user interface description language. One of its design goals was not to mandate a certain modality. Therefore, it can be suited to describe 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. An XML document, which consists of two or more XML languages, is called compound document. A compound document can specify user interface of an application. In this paper, XForms is combined with XHTML+CSS level 2 to realize the use cases. XForms 1.0 includes other W3C specifications directly: XML Events, XPath 1.0, XML Schema Datatypes, and XML 1.0.

[1]Glade. Available at: http://glade.gnome.org/

## 2.3 XUL

Mozilla has developed a UI description language called XML User Interface Language (XUL) (Hyatt, 2001). The markup consists of widget elements like buttons, menus, etc. XUL applications are based on several W3C standards. Those 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.

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 (e.g., 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 few technologies introduced by Mozilla. *The eXtensible Bindings Language (XBL)* is a markup language that defines new elements for XUL widgets. *Overlays* are XUL files used to describe extra content for the UI. *XPCOM* and *XPConnect* enable the integration of external libraries with XUL applications and, finally, *XPInstall* provides a way to package XUL application components with an install script. (Bojanic, 2003)

## 2.4 Requirements

Souchon and Vanderdonckt have reviewed XML-compliant user interface description languages in (Souchon and Vanderdonckt, 2003). The paper compares the general properties and the UI description capacities of the languages. XIML is found out most expressive language whereas UIML has best software support. XUL is found to be less expressive. XForms has not been evaluated in the paper.

Four XML languages for abstract user interface representation are examined in (Trewin et al., 2004). The languages are UIML, XIML, XForms, and AIAP. The paper defines requirements for the representations. Those include high level requirements like applicability to any target and any delivery context, personalization, flexibility, extensibility, and simplicity.

In addition, they have defined technical requirements, which 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 fulfill best the requirements. Especially, in terms of separation of data from presentation and flexibility in resource substitution.

Requirements for a generic user interface description format are discussed in (Simon et al., 2004). They also present 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.

# 3 RESEARCH SCOPE AND METHODS

The research area of the paper is *Web user interaction models*.

Because of the huge variance in interaction scenarios and technologies (ranging from natural language speech interaction to 3D interaction with immersive displays), the research is tightly scoped. The scope for the research is desktop-style user interaction in WWW environment.

The research steps are enumerated in the following list. The scoping, which is 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. XForms and XUL are evaluated against the requirements.
4. The use case implementations are evaluated through heuristic analysis (Nielsen, 1994).

## 3.1 Use Cases

The selected use cases are from an existing content management system, which is used to manage the content of an Internet magazine. The application is used through Web and is originally implemented with HTML and CSS. We selected three user interfaces from the system, which are difficult to realize properly with HTML. First, the wireframe models of the use cases were drawn. The models were designed using general usability guidelines without taking account possible restrictions of the languages. Users are mainly journalists, who have experience in using typical word processing program and are familiar with concepts like *copy-paste*.


Figure 1: Wireframe model of the document editor.

The design of the user interfaces in this paper is based on usability best practices (Cooper, 1995) and user interface design patterns (Tidwell, 2005) and (Laakso, 2003). The usability of the interfaces has been validated by usage simulation (Preece et al., 2002) and heuristic analysis (Nielsen, 1994).

**Document Editor.** The purpose of this user interface is to create and modify simple structural documents, which could, e.g., be displayed as a web page. The type of data in the document is limited to text, pre-existing images and pre-existing tables (created, e.g., by the Table Editor user interface). Wireframe model of the Document Editor is shown in Figure 1.

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

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 creates a review of a laptop and completes it with images of the laptop and a table about its features.

**Table Editor.** The purpose of this user interface is to create and modify simple tabular data, which can be

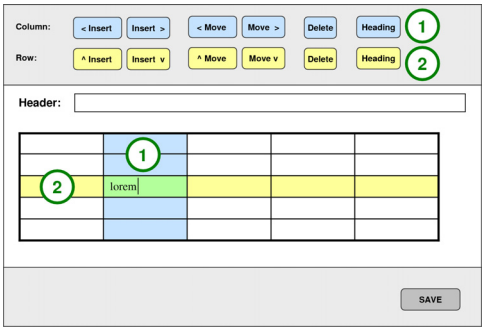


Figure 2: Wireframe model of the table editor. (1) Active column is color coded to match the coloring of the buttons for manipulating the column. (2) Active row is color coded to match the coloring of the buttons for manipulating the row.

displayed, e.g., in a web page. The type of data in the table is limited to characters and numbers. Wireframe model of the Table Editor is shown in Figure 2.

The user can also edit the structure of the tabular data by marking some of the columns or headers as headings and by entering a header text for the whole table. The number of rows and columns in the table is user-editable.

For the sake of simplicity, table cells are not allowed to span multiple columns or rows. Possible use cases for the table editor are: the user wants to create a table documenting the average monthly temperatures in four different locations during one year; or the user wants to create a table presenting the costs estimate for purchasing a new computer setup.

**Tree Editor.** The purpose of this user interface is to create and modify a tree structure where the nodes of the tree have multiple editable attributes. In this paper, we use the nodes to represent web site areas for a site of a magazine. However, the nodes and attributes could represent anything. The tree Editor is depicted in Figure 3.

The user is able to create new nodes, edit the attributes, move nodes around in the tree and delete nodes. A possible use case for the tree editor is: managing the structure of an online magazine.

# 4 RESULTS

The results of the paper are discussed in this Section. We analyzed how the languages fulfill the requirements of a UI description language presented in the literature. In addition, we introduce the use case implementations and the heuristic analysis we did for them. Finally, model differences of the languages are explained.

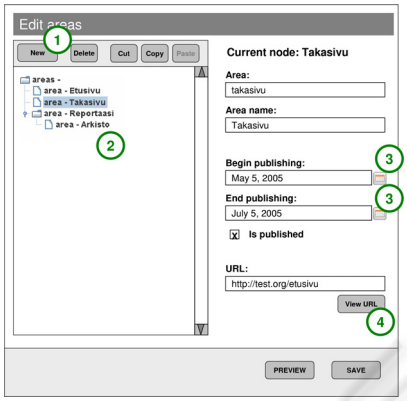


Figure 3: Wireframe model of the tree editor. (1) Creates a new area as a child to the currently selected node. Data for the newly created area is entered from the form in the right. (2) Moving nodes in the tree is done by drag-and-dropping them. (3) Opens a calendar widget for selecting the date. (4) Opens the URL in a browser.

## 4.1 Requirements

The languages were evaluated against the requirements, and the results are shown in Tables 1-3. The general requirements in Table 1 are from (Simon et al., 2004). They are device and modality independence and customizability. Both languages meet the requirements. XForms has abstract UI description. Thus, the concrete UI is totally device independent. Also, the UI description of XUL does not restrict the selection of devices. XForms uses data types, which makes it easy to utilize different modalities. For instance in voice modality, grammar based recognition can be made more specific. XUL widgets can also be transferred to other modalities, but lack of data types makes it more difficult. Both XForms and XUL provide control over layout and graphical appearance. In XUL, the customization is easier because of specific UI elements.

The requirements in Table 2 are represented in (Trewin et al., 2004). In the paper, XForms is evaluated against the requirements among three other languages. XForms, along with Alternative Interface Access Protocol (AIAP), was found best suited to meet the requirements defined in the paper. Especially, regarding to separation of data from presentation and flexibility in resource substitution.

The interface elements are not separated from their presentation in XUL. In XForms, the data model can be accessed through separate binding layer. In both languages, the interface elements can have dependencies. However, in XUL, the dependencies have to be realized through scripts. XForms is also easier to use in any target since its UI description is more abstract. XForms supports data types, whereas XUL does not.

Table 1: The requirements of the UI description language (Simon et al., 2004).

| Requirement | XForms | XUL |
|---|---|---|
| **General requirements** | | |
| Device Independence | Good | Possible |
| Modality Independence | Good | Possible |
| Customizability | Possible | Good |

Table 2: The requirements for universal interaction (Trewin et al., 2004).

| Requirement | XForms | XUL |
|---|---|---|
| **Separation of Interface Elements from Pres.** | | |
| Separation of Data/Pres. | Good | Possible |
| **Interface Elements** | | |
| Dependencies | Good | Possible |
| Any Target | Good | Possible |
| Data Types | Good | No |
| **Presentation Related Information** | | |
| Logical Groupings | Good | Good |
| Labels & Help Text | Good | Possible |
| Presentation Replacement | Possible | Possible |
| **Run Time and Remote Control** | | |
| Local Computation | Good | Possible |
| Serialization | Good | Possible |

The presentation can be grouped well with both languages. XForms provides explicit way to include labels and help texts, while in XUL they can be realized with normal text. Providing an alternative presentation is possible with both formats. Local computations (e.g., data validation) and data serialization are easier to provide with XForms, which has current state always available. They must be realized through scripts in XUL. These differences are discussed in more detail in Subsection 4.6.

The requirements found from the literature were extended with a more detailed *typical interaction patterns* requirement set from the application scenario (c.f. Table 3). Repeating structures (repeat) and paging and dialogs (switch, message) are natively supported by XForms, while in XUL they require some script programming. Nested constructs are supported by the XUL tree control as well as our proposed XForms tree module. Copy-paste, undo-redo, and drag-and-drop can be programmed with scripts in XUL, while in XForms only copy-paste is possible to implement. As a summary, XUL handles the typical interaction patterns better, since it has more desktop-oriented focus.

Table 3: The proposed extensions to requirements.

| Requirement | XForms | XUL |
|---|---|---|
| **Typical Interaction Patterns** | | |
| Paging & Dialogs | Good | Good |
| Repeating constructs | Good | Possible |
| Nested constructs | Good$^{(*)}$ | Good |
| Copy-paste | Possible | Possible |
| Undo-redo | No | Possible |
| Drag-and-drop | No | Possible |

$^{(*)}$ using the proposed tree extension.

## 4.2 Use Case Implementations in XForms

The XForms implementations of the use cases were done using the XForms 1.1 Working Draft (Boyer et al., 2004) (W3C Work In Progress), which is implemented in the X-Smiles browser (Vuorimaa et al., 2002). XForms 1.1 has several features, which make it possible to minimize scripting. The main features from XForms 1.1, which were utilized, are duplicate and destroy actions, and mediatype-aware output rendering. Otherwise these features would have required the use of scripting.

XForms language was extended with a tree module, since in XForms 1.1, there is no way of selecting nodes from a recursive structure. We also implemented the tree module, 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 automatically be 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 few XForms 1.1. features, which it utilizes heavily, namely destroy and duplicate, and output@mediatype. This UI has no scripting.

**Tree Editor.** The tree editor (cf. Fig. 4) 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 is written in XForms 1.1, but it has a small script to insert, delete and move columns. This could be avoided if XForms had repeating and conditional action containers (such as *for* and *if*).

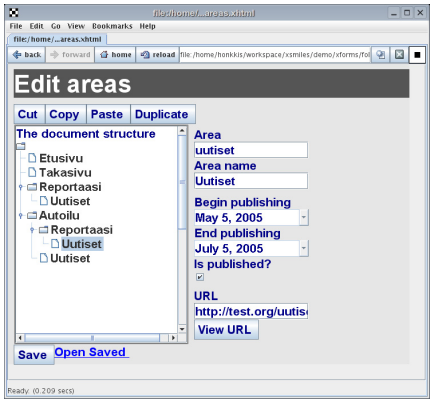


Figure 4: XForms Tree Editor.

## 4.3 Proposed XForms Extension: Tree Module

We have extended the XForms 1.0 specification with a tree module *xforms:tree* and a corresponding XPath extension function *nodeindex*.

This form control displays a tree, which corresponds to the instance tree rooted at the bound node. It must have an id attribute. The item element's label (executed with the context of corresponding node) is used to determine the label of each node.

The XPath function *nodeindex* takes an idref of an tree widget, as an argument and returns the instance node, which corresponds to the currently selected node in a tree widget.

A code example of the tree element's usage is shown in Figure 5. It would display a tree of folders and files. When a user selects a node, an editor for that node is shown in the relevant group.

```
<instance>
  <data>
    <folder name="xxx">
      <folder name="xxx">
        <file name="xxx" description="xxx"/>
      </folder>
    </folder>
  </data name="xxx">
</instance>

<tree ref="/data/folder" prune="true"
    id="folders">
  <label>The directory document</label>
  <item><label ref="@name"/></item>
</tree>

<group ref="nodeindex('folders')">
  <group
    ref="self::node()[localname()='folder']>
    <label>Folder</label>
    <input ref="@name"/>
  </group>
  <group
    ref="self::node()[localname()='file']>
    <label>File</label>
    <input ref="@name"/>
    <input ref="@description"/>
  </group>
</group>
```

Figure 5: Example of tree widget usage.

## 4.4 Use Case Implementations in XUL

In addition to the wireframe model designs, XUL enabled to use context menus in Document and Tree Editors. Also, Document editor has a real time preview of a document. It is remarkable that XUL interfaces require a lot of scripts. All the button functionalities, drag-and-dropping, and focusing of elements have to be realized through scripts. The XUL Document editor is depicted in Figure 6.

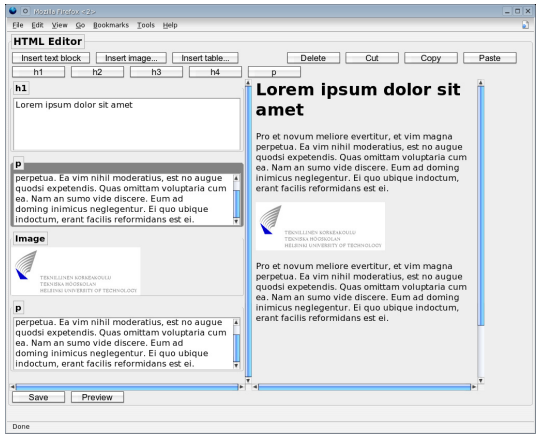


Figure 6: XUL Document Editor.

## 4.5 Heuristic Analysis

We did the heuristic analysis according to the heuristics defined by Nielsen (Nielsen, 1994). We did not find any major problems from the interfaces. Mainly, because the wireframe models were already designed

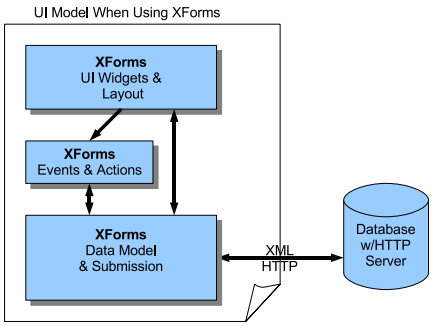


Figure 7: UI Model using XForms.

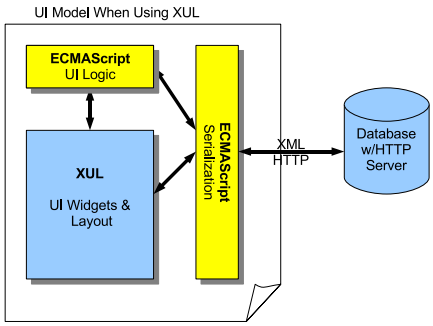


Figure 8: UI Model using XUL.

according to the heuristics. Nevertheless, we were able to identify some problems from all the interfaces. Common deficiencies were lack of undo-redo and help operations. Of course, these should have been considered already in the design phase.

XForms Document Editor does not have drag-and-drop functionality. In addition, the preview function is a bit problematic in XForms editors, because user has to always save the form before previewing it. However, saving is not always desired when previewing. Finally, selected column or headers cannot be highlighted in XForms Table Editor.

## 4.6 Model Differences Between XUL and XForms

Although, in the selected use cases on a desktop computer, the usability between the XForms and XUL user interfaces does not differ, we have noticed a difference in the user interface development model. Like XUL, most of the XML-based user interface definition languages are widget based. This means that they are quite concrete, and the author works by adding widgets, such as buttons and text areas to the user interface. It is therefore very easy to graphically create a user interface layout, but all user interface logic has to be programmed using a programming or scripting language, though. In contrast, XForms starts by defining a XML data model, and all operations are done to the datamodel using declarative actions and XPath expressions, while user interface is automatically kept up-to date with a dynamic dependency tracking.

Maybe the biggest difference is the communication between the user interface and the back-end system. For the communication, the user interface state has to be serialized for transmission. Vice versa, after getting serialized reply from the server, it has to be de-serialized into application state. In XForms (cf. Fig. 7), that serialization is automatic, since the datamodel is a live XML document object model, which is automatically serialized and de-serialized.

On the other hand, in XUL there is no explicit datamodel, and communication between a backend process and the user interface have to be reimplemented using ECMAScript for each user interface, as shown in Fig. 8. This is true also for HTML forms and its derivatives, such as Ajax[2]. For example, when the server sends an updated structured content back, there has to be a script, which updates the corresponding XUL DOM, respectively. This means, that authoring and maintaining XUL-based applications is more complicated than XForms.

It is noteworthy that XUL has a templates mechanism, which allows to use RDF as the datamodel to some extent. Since RDF is more complicated than XML (graph vs. tree), and we would have to serialize the RDF datamodel anyway into the XML document model either at server or client, it was not used in this paper. Using XBL (Hyatt, 2000) combined with XUL should allow the use of XML datamodels, thus removing the need of serializing and de-serializing communications. All user interface logic has to be still written in ECMAScript, though XBL encapsulates the operations in a reusable manner. It is still unknown, whether XUL+XBL removes the need for any serialization and deserialization in the selected use cases.

# 5 CONCLUSIONS

In this paper, two UI description languages were studied, namely XForms and XUL. We collected requirements for a UI description language from literature, extended them with typical interaction patterns, and evaluated the languages against the requirements. In addition, we selected three use cases, which are typical to Web application UIs, but are difficult to realize properly with HTML. First, we designed wireframe

[2]Ajax: A New Approach to Web Applications, http://adaptivepath.com/publications/essays/archives/000385.php

models of the use cases according to the usability guidelines. Based on those, we implemented the use cases with both languages, and did heuristic analysis for the implementations.

XForms fulfilled the requirements slightly better than XUL. On the other hand, with XUL, the use cases could be realized more strictly according to the wireframe models, since, for instance, drag-and-drop is not supported in XForms. As a conclusion, the differences between XForms and XUL+XBL on desktop are not big. We do expect that major differences can arise in device independence and multimodal usage scenarios, where XForms is better.

Also, we feel that both languages should add support for general undo-redo, copy-paste, and drag-and-drop interaction patterns. In special cases, it is possible to support these, but for instance, copy-paste between different types of data input and outputs is not usually supported (for instance, copying the values in a repeating spreadsheet-type of table into a different type of repeating construct). These interaction patterns are so widely available in current user interfaces, that they need to be supported in the Web user interface languages as well, in order to facilitate the deployment of these user interfaces on the Web.

## REFERENCES


Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E. (1999). 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. Elsevier North-Holland, Inc.

Bojanic, P. (2003). The Joy of XUL. Available online http://www.mozilla.org/projects/xul/joy-of-xul.html.

Boyer, J., Landwehr, D., Merrick, R., and Raman, T. V. (2004). XForms 1.1. W3C Working Draft.

Cardone, R., Soroker, D., and Tiwari, A. (2005). 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. ACM Press.

Cooper, A. (1995). *About Face: The Essentials of User Interface Design*. John Wiley & Sons.

Dubinko, M., Klotz, L. L., Merrick, R., and Raman, T. V. (2003). XForms 1.0. W3C Recommendation.

Hoffman, M. (2003). Architecture of microsoft office infopath 2003. Microsoft Developer Network.

Hostetter, M., Kranz, D., Seed, C., and C. Terman, S. W. (1997). Curl, a gentle slope language for the web. *World Wide Web Journal*.

Hyatt, D. (2000). XBL - extensible binding language 1.0. Netscape.

Hyatt, D. (2001). XML user interface language (XUL) 1.0. Mozilla.org.

Laakso, S. (2003). User Interface Design Patterns. Available online http://www.cs.helsinki.fi/u/salaakso/patterns/.

Nielsen, J. (1994). Ten Usability Heuristics. Available online http://www.useit.com/papers/heuristic/heuristic_list.html.

Preece, J., Rogers, Y., and Sharp, H. (2002). *Interaction Design*, chapter 13. Wiley, 1st edition.

Puerta, A. and Eisenstein, J. (2002). 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. ACM Press.

Rector, B. (2003). Introducing "longhorn" for developers. Microsoft Developer Network.

Simon, R., Kapsch, M. J., and Wegscheider, F. (2004). 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. ACM Press.

Souchon, N. and Vanderdonckt, J. (2003). 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.

Tidwell, J. (2005). *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly Media, Inc., 1. edition.

Trewin, S., Zimmermann, G., and Vanderheiden, G. (2004). Abstract representations as a basis for usable user interfaces. *Interacting with Computers*, 16(3):477–506.

Vuorimaa, P., Ropponen, T., von Knorring, N., and Honkala, M. (2002). A Java based XML browser for consumer devices. In *The 17th ACM Symposium on Applied Computing*, Madrid, Spain.