General Interface Description of Websites using CLICK and UIML

Yogita Bhardwaj, Muhammad Abu-Saqer, Manuel A. Pérez-Quiñones
Department of Computer Science,
Virginia Polytechnic Institute
Blacksburg, VA 24060
yogitab@vt.edu, mabusaqe@vt.edu, perez@cs.vt.edu

ABSTRACT
This paper explores the domain of programming paradigm for Multi-Platform User Interfaces using XML based languages. The main focus of this work is User Interface Markup Language (UIML), an XML based language for describing user interfaces in a platform-independent manner. We have explored the capabilities of UIML as an interface description language for describing interactive websites. We have selected an end-user web programming tool called CLICK, which also uses an XML based interface description for the websites created through it. We have analyzed both the representations and devised a conversion process from CLICK XML to UIML. We have found that UIML is expressive enough to represent applications built using CLICK. UIML provides various benefits over the interface description generated by CLICK especially that of facilitating the development of web based interfaces for multiple platforms through CLICK.

Author Keywords
UIDL, CLICK XML, UIML, generic UIML vocabulary.

INTRODUCTION
With the increase in number of computing platforms being commonly used by people, the task of building a user interface for an application for all of these platforms becomes very complex. Apart from being complex, the process is somewhat redundant since the conceptual interface remains the same but presentation and implementation changes with each platform. Researchers have tried to solve this problem by expressing the user interface at various levels of abstraction. Various markup languages have been created to represent the user interfaces at these different levels of abstractions. One such language is the User Interface Markup Language (UIML). UIML is a meta-language that requires an XML specification (or a vocabulary) to provide meaning to the parts used in the description [1]. The language itself is completely independent of any metaphor and just introduces a basic set of tags for defining a user interface structure. This facilitates writing the specification once and rendering it multiple times on different platforms based on different vocabularies. This single authoring approach goes a long way in alleviating the plight of a UI developer developing interfaces for multiple platforms. We have derived the motivation for our work from this powerful concept. We wanted to study to what extent UIML is helpful for developing interactive web based interfaces.

Our target problem domain is an end-user web programming tool called CLICK (Component-based Lightweight Internet-application Construction Kit). CLICK is a research project, at being developed at Virginia Tech, oriented towards providing a toolkit to create interactive websites with most commonly used features like form validations, database connectivity of the form fields etc. An XML representation is inherently platform independent and allows for easy parsing and modification to data. For these reasons, representing the interface in an XML document and providing code generator based on that was the natural choice for CLICK. This XML captures various web pages in a web application and various web interface widgets and their behavior that goes on each page. CLICK is a project at the very early stages of its development and does not intend to make available its XML based representation for manipulation directly by end users yet.

The question that would now arise is that why this tool needed to create its own XML to represent a web interface. If it is simply an oversight of already existing XML based interface description languages, then can UIML serve the same purpose as this custom XML? We tried to answer this question by converting CLICK XML to UIML and thus actually developing an interactive website, which was created using CLICK, again through UIML. Our expectation was that through this process we would be able to uncover any hurdles that UIML may present while developing basic interactive websites. Of course, a benefit that CLICK gets from this is that the UIML representation of a web interface built in CLICK can then leverage the

methodology to create multi-platform user interfaces through UIML [3] for the same application.

RELATED WORK

There has been significant research in the area of device independent UI development and using XML based languages to represent the UI. These languages are commonly called User Interface Description Languages (UIDLs). The goals for these UIDLs have been enumerated in [4]. These efforts are mainly oriented towards providing a way to separate data from presentation so that the presentation of the interface can be easily adapted to changes in platform where these applications are deployed. XForms [11], based on the same goal, is a W3C effort which introduces device independence for form-based web interfaces. This is very restrictive since not all interfaces can be form based and CLICK itself will expand its domain from just forms to letting people create interfaces with dynamic text e.g. VT CS faculty/staff directory. Several different UIDLs have been proposed like USIXML [4], XIML [6], TERESAXML [5] and all of them are essentially based on the requirement of being able to start a user interface description at a conceptual and abstract level, represented usually by a task model, and then moving step-by-step towards more concrete representations and the final UI for a target platform.

These languages follow the model-based interface development approach [7] where task modeling is the initial step. It requires thinking of interface definition process in terms of user tasks, dialog, domain objects and presentation. These languages represent entire development lifecycle unlike UIML which starts at a more concrete level, i.e. with a modality dependent generic vocabulary. With the model based approach, the interface specification can be created at any level and can either be abstracted into a more abstract specification or reified into a more concrete specification [4]. RIML [10], Renderer-Independent Markup Language, is another effort towards single authoring which addresses the layout and pagination issues across different platforms. A comprehensive evaluation of all the XML-based languages is presented in [8, 9]. The languages are mostly compared on the criteria of what models these languages support, how well they separate data and presentation, flexibility, universal usability and ease of use among others. Clearly, not any single language appears to be a winner or a comprehensive solution to all issues.

Ali et al. suggest that creating user interfaces with UIML using platform specific vocabularies is still a very cumbersome process due to limited commonality among these vocabularies [2]. This contradicts the original concept of device independent authoring, since the author requires knowledge of both UIML and the target language. A solution to this problem is the use of a generic vocabulary. A generic vocabulary includes a set of generic elements that can be used for any platform. An important step identified in the model based design process is a task model which is at an abstraction level higher than that is currently possible with UIML. Using a task model in conjunction with UIML [3] will facilitate the development of multi-platform user interfaces as a task model can capture conceptual information about the interface that remains the same across multiple interfaces.

SURVEY OF WEB DEVELOPMENT PRACTICES

The domains of exploration for this project were CLICK and UIML that are research projects whose user base is still very limited. For this reason we limited our survey to that of web development practices. The goal of our survey was to find out how web developers think about their web applications in terms of code, layout and usability. We also wanted to study practices followed for testing the application. The survey was conducted in the form of semi-structured interviews. We interviewed 17 computer science graduate students, out of which 3 have been professional developers. As the participants rated themselves on their experience with web programming, we had a sample set of 3 beginners, 8 intermediate and 6 expert developers in the field.

Our results were not very surprising and most of the web developers are aware and do follow best practices. 11 of these participants expressed that usability and look and feel of the websites were of primary concern to them. This suggested that a website created (for the desktop platform) using a WYSIWYG tool should always look the same no matter how the rendering is done by the backend and must preserve the look and feel as much as possible. On the other hand, we found that developers had the opinion that data and logic would not change when moving from one platform to another, so it should be ported automatically and they would be willing to customize the interface for each platform.

Our findings related to web programming styles suggested that our participants prefer following best practices when developing web applications. 12 of the participants mentioned that they mainly do styling of their websites using the style sheets because it gives good modularity and allows easy change. This suggests that ideally an interface development tool for websites should create style sheets for all the style related information. CLICK generates the code in such a manner that the layout and client-side and server-side logic are all combined. But, 12 of the participants said that they would always separate client side and server side code and only 3 of these participants were aware that even client side and server side code could be combined.

Finally we asked our participants, how do they generally test their websites? 9 participants mentioned that the look and feel of the website are their top priorities, e.g. the information should be correct, design and colors should look good, links should be clickable and should direct to correct pages. 4 of these participants mentioned using usability guidelines to test the work flow. 2 participants mentioned that they also test how their websites look in different browsers. All the participants mentioned that for dynamic forms they test for every possible value that their

application allows and disallows so that the behavior is as they expected.

We observed that the fields of interest, of the participants influence their responses. Those participants who prefer to write backend logic were more concerned about the behavior of an application. On the other hand, the participants who had the knowledge of usability principles focused more on usability of the websites. Of course, neither of the two aspects can be weighed lesser than the other. But the guidelines we derived from this survey were to preserve the layout and the look and feel to the maximum extent and to generate style sheets for the style information for any website and finally to provide clean separation between the client-side and server-side behavior of any website.

UIML OVERVIEW

UIML is modeled by a meta-interface model [13] which separates out interface description i.e. <*interface*>, underlying application logic i.e. <*logic*> and specification of actual rendering to a particular device i.e. <*presentation*> for any application. <*interface*> section is composed of four main subsections. <*structure*> section refers to what interface elements the UI is comprised of. Each element is represented with <*part*> tag and the type of this part, "class", is determined by the vocabulary used. E.g. a label part may have a class JLabel when we use JAVA Swing vocabulary. <*style*> section refers to presentation style e.g. fonts, colors etc., specified as a set of <*property*> tags on a part. <*content*> refers to text or images that go on a UI. Finally, the <*behavior*> section specifies as a set of conditions and actions performed when these conditions are met. The <*behavior*> section specifies the interactive behavior of any interface.

CLICK XML OVERVIEW

CLICK XML specifies an entire web application in one XML file. Each application with root tag, <*app*> is a set of <*page*> tags which refer to each page in the web application. On every page we can define some <*component*> tags which are the web interface elements. CLICK XML does not separate style, content or behavior. Inside the <*component*> tag all the style, content and behavior information is embedded. E.g. location of the component on the screen, text associated with it and interaction data like input constraints, action on click or database connections. Equivalent to a "class" in UIML, CLICK XML interprets each component by its "type". E.g. for a label, the type "htmlText" is used and for a textbox type "inputText" is used. These types decide how the component is rendered in HTML.

MAPPING CLICK XML TO GENERIC UIML

The need for starting an interface specification at the level of a generic UIML for efficient interface development has already been identified [2]. So, we first map the CLICK XML representation to the generic UIML vocabulary. An official specification of a generic UIML is not yet available so we base our discussion on the concept of generic vocabulary as mentioned in [2, 3]. We can suitably adapt to any changes in the generic vocabulary provided the inherent model of UIML does not change. This generic vocabulary can then be mapped to platform specific vocabularies to obtain interfaces for another platform. The process of mapping is shown in Figure 1. An important aspect is that of device families, which refers to a group of platforms with similar layout features. E.g. a desktop family includes all the desktop based interfaces created with different UI toolkits like HTML or JAVA Swing. In order to avoid having to create one generic vocabulary for all the platforms and making it unreasonably complex and bloated, several generic vocabularies are created. For the purpose of our discussion we consider a generic vocabulary for GUIs [2]. In the following sections we discuss how elements from CLICK XML can be mapped to each of the main elements of UIML.

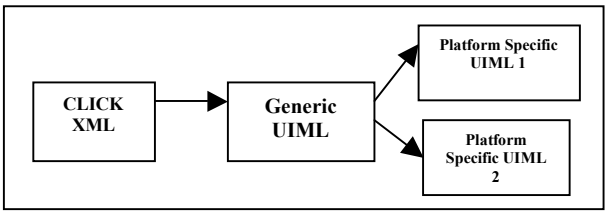

Figure 1. Mapping of CLICK XML to UIML

Interface

CLICK XML represents an entire website as an application that is a collection of related pages. Whereas a single document in UIML is really one single interface which maps to one single web page. This approach is much more scalable when the number of pages in a website grows. On the other hand, CLICK XML is a nice bundle of related web pages with one of the pages among these being a home page. In UIML we can specify one main <*structure*>, as explained below, and specify a <*restructure*> element which dynamically changes the interface based on certain events and can be thought of as analogous to switching between pages on a certain condition. We focus on one single page such that the interface for each page in CLICK XML can be represented by the <*interface*> element in UIML.

Structure

Structure refers to the collection of widgets that make up an interface and their relationship to each other, e.g. spatial relationship between elements in GUI. These widgets represented by a <*part*> element are analogous to the <*component*> element in CLICK XML. Each <*part*> is classified in a particular category by *part-class* in UIML and *type* in CLICK XML. CLICK XML generates code in HTML 4.0 and the application logic in PHP. When using HTML 4.0 vocabulary of UIML we can create a structure in UIML which is exactly similar to CLICK XML. E.g. a component in CLICK of type "inputText" can be represented by a part in UIML of class "GSLTextRegion" representing a generic text field.

Style

In UIML style information is specified as a set of <*property*> tags. Between CLICK XML and generic UIML the style information maps completely. E.g. attributes X, Y for location can be specified as a Location property on a UIML part. The benefit UIML provides is that the style section is separated out from the general structure. The resulting interface description is much more structured and less cumbersome to adapt to changes. From our survey we had found that people prefer all the styling to be done by style sheets so that their applications are modular. The style section in a UIML document is analogous to style sheets for a web interface. Although current renderers do not leverage this feature to auto-generate the style sheets but it is definitely a possibility.

Content

Content in the UIML document refers to all the text, images and sounds used within an interface document. Mapping between content from CLICK XML to UIML is straightforward just like style. Content related to each of the components in CLICK XML is embedded within the <*component*> node. This content can be easily mapped to the <*content*> section in a UIML document. For the purpose of internationalization we can simply replace text strings in CLICK XML by another language. But, the advantage UIML provides is that, due to the clean separation of content from structure, the content section can easily be replaced by another equivalent section. We have the ability to specify multiple content sections in one document and selectively render it based on the target language. A flip side of representation of content in CLICK XML is that the text content in CLICK XML is stored as straight HTML because of the inability of CLICK XML to provide a markup equivalent to HTML, which ties it closely to one particular platform. So, the CLICK XML cannot directly map to other platforms

Behavior

The behavior section defines the interaction between the user and the interface as a set of rules where a certain action is performed whenever a particular condition is met. This is the most important section with respect to the interactive web based interfaces. In a generic UIML behavior can be specified independent of the underlying logic which is closely tied to a platform. Behavior section in UIML simply specifies what function to call on certain event. This function is simply left as a stub to be filled in when platform specific UIML is specified in the <*logic*> section. There are three main interaction features that can be added to the interactive websites through CLICK. These are: database connectivity of the input form fields, client side input validations, and conditional actions on button click.

UIML does not have a specific data model that specifies what data an interface collects or presents. When specifying database connectivity through CLICK XML, each of the components, that require a user input, can be attached to a database field (the database being a default application database), and can save the value entered to this database field on a button click. CLICK XML specifies this connection by adding an attribute "dbFieldName". To do this in UIML we need some external function. This can be a function call in UIML which essentially looks like the code in Figure 2. Function calls in UIML are always called on triggering of an event, e.g. a button click, and database connectivity is persistent information. Another alternative is to leverage the <*logic*> section where the actual interface to application logic is defined. We can use the cleaner data model given by XForms for our purpose here and use it as a part of our application logic.

```
<call name="connectInputToDatabase">
    <param name="inputField">firstName</param>
    <param name="dbField">dbFirstName</param>
</call>
```

Figure 2. External Logic function for database connection

An important aspect to be considered here is that CLICK XML simply says that an input field is saved to a certain database field or can specify a certain text on the web page comes from a database field. It completely hides what the data source is and how it accesses the data. This becomes more important when we talk about developing multi-platform user interfaces.

Client side input validations are easier to specify. CLICK XML specifies an input constraint on an input field by the type of constraint e.g. not empty or between 2 and 15 characters, and a message that is popped up in case the constraint is not specified. In the code generated by CLICK this translates into a Javascript validation function. This can be represented very well as an external Javascript function that can be specified in UIML and triggered off on a button click. An example is shown in Figure 3 and 4. Javascript logic was imported from CLICK directly without changes. HTML renderer by Harmonia Inc. renders these client-side Javascript functions in the same file as the HTML front end and leaves out all the backend processing logic in a separate server-side file. This is exactly how developers who participated in our user survey expect their application code to look like.

Finally, a complex interaction behavior in CLICK can be specified based on various values entered by a user in a form. For example, if the email field is filled up by the user, only then an email is sent to that address as shown in Figure 5 and 6. UIML provides a means to specify complex logic conditions as well. An element called <op> represents a general set of logical operators. With this element basic logical conditions (less than, greater than, equal, not equal, and, or etc.) may be expressed along with the ability to structure complex conditions involving multiple values [12]. We can thus achieve the complex conditional logic

CLICK implements. At this stage of creating a generic UIML description we still leave out the underlying logic description, i.e. the external logic functions, as stubs which need to be described at the platform specific UIML level. Figure 7 shows an abridged XSLT script for transforming CLICK XML to a generic UIML.

```
<inputConstraint type="notEmpty" min="" max="" acceptEmpty="0">
    First Name cannot be left empty
</inputConstraint>
```

Figure 3. Specification of input validation in CLICK.

```
<behavior><rule>
  <condition>
    <event class="OnClick" part-name="submit"/>
  </condition>
  <action>
    <call name="form.isEmpty">
      <param name="name">firstName</param>
      <param name="errorMsg">First Name cannot
       be left empty</param>
    </call>
  </action>
</rule></behavior>
```

Figure 4. External Logic function for input validation.

Overall, while mapping the behavior section to generic UIML we were able to capture direct meaning of the tags in CLICK XML but we could only capture the inferred meaning. For example as described above, most of the interaction behavior specified by CLICK can be represented in UIML using external logic function. We would need to rely on the developer to infer the correct requirements for these functions through function names and input parameters.

```
<actionRule>
  <conditions connector="and">
    <condition fieldId="sendEmail" operator="Yes" />
  </conditions>
  <actions>
    <actionSendEmail from="someone@somewhere.com"
to="yogitab@vt.edu" subject="Hi"> <![CDATA[
Message]]></actionSendEmail>

  </actions>
</actionRule>
```

Figure 5. Specifying conditional action through CLICK

```
<condition>
  <op name="and">
    <event class="buttonClicked" part-name="Submit"/>
      <op name="equal">
        <property name="value" part-name="sendEmail"/>
        <constant value="Yes"/>
      </op>
  </op>
</condition>
<action>
<call name="form.sendEmail">
<param name="from">someone@somewhere.com</param>
<param name="to">yogitab@vt.edu</param>
    ...
</call>
</action>
```

Figure 6. Equivalent representation of Figure 5 in UIML

```
<xsl:template match="app/page[@id='Page']">
<uiml><interface><structure>
<part class="GTopContainer">
   <xsl:attribute name="name">
      <xsl:value-of select="@id"/>
   </xsl:attribute>
<part class="GArea">
  <xsl:attribute name="name">
  <xsl:value-of select="@id"/>Form</xsl:attribute>
<xsl:apply-templates select="component"/>
</part>
</part>
</structure></interface></uiml>
</xsl:template>
<xsl:template match="component">
<part>
<xsl:attribute name="name">
   <xsl:value-of select="@id"/></xsl:attribute>
<xsl:attribute name="class">
<xsl:choose>
  <xsl:when test="@type = 'htmlText'">GLabel</xsl:when>
  <xsl:when test="@type = 'button'">GButton</xsl:when>
      …
</xsl:choose>
</xsl:attribute>
<style>
<property name="location">
   <xsl:value-of select="@x"/>,<xsl:value-of select="@y"/>
</property>
…
</style>
</part>
</xsl:template>
```

Figure 7. Abridged XSLT script for CLICK XML to generic UIML

PLATFORM SPECIFIC RENDERING

We discussed in the previous section, the creation of a generic UIML from CLICK XML. Our final step was to map the generic UIML to a platform specific UIML, add platform specific logic functions and then finally render it using the renderers provided by Harmonia Inc. We picked up the HTML 4.0 vocabulary generate again the original website created by the end user with same the behavior and the look and feel. Converting generic UIML to platform specific UIML is another simple XSL transformation. However, the process of going from generic UIML to a platform specific UIML is not entirely automatic but a developer's intervention is required in order to achieve best quality results. Achieving the same look and feel of the original website with the HTML 4.0 vocabulary based UIML was possible. These extra properties we added were mainly about converting the location property from generic UIML to part class "DIV" with "style" property specifying location, so as to achieve exact placement of components on screen in an HTML document.

Implementation of the behavior for our target platform was limited by the freely available renderer for HTML 4.0 vocabulary by Harmonia Inc. as it adhered to UIML 2.0 specifications and we had identified several powerful features in UIML 3.0 that were relevant to our topic e.g. <*restructure*> and <*op*>. CLICK XML writes the backend logic in PHP, where as the current renderers work best for JAVA servlets based backend logic. Figure 8 shows an example of a guestbook application created using UIML which was originally developed using CLICK. This application saved the data to the application database and was directed to a 'thank you' page. First could not be left empty. All the input validations and form submit actions were specified as actions to be performed on the "Submit" button click in UIML. This application preserved the look and feel and input validations that were initially generated by CLICK. The input validation logic was borrowed from CLICK. We wrote the backend logic for this application using JAVA servlets. Saving form fields into a database was not directly derived from the UIML and we have to write a function tailored for this application since we could not specify what data is collect from the form. The behavior of the application was captured in the generic UIML as stubs. The only task left for this stage was to appropriately fill those stubs.

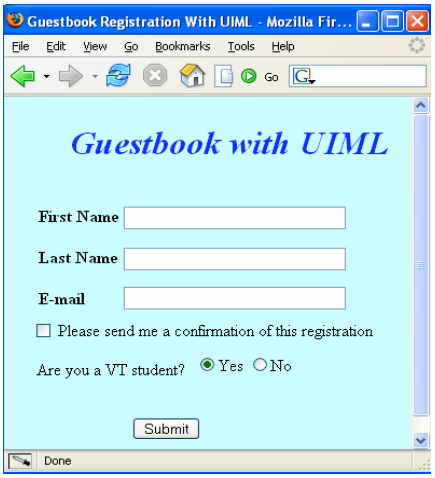

Figure 8. HTML 4.0 based interface rendered by the HTML renderer by Harmonia Inc.

We also tried to convert the generic UIML to the JAVA Swing platform. Just as HTML 4.0 platform, this was trivial as well but modifications were required to get the desired layout. We also had to customize the logic section to tailor it for JAVA format. Figure 9 shows the JAVA based equivalent of the web application developed in CLICK.

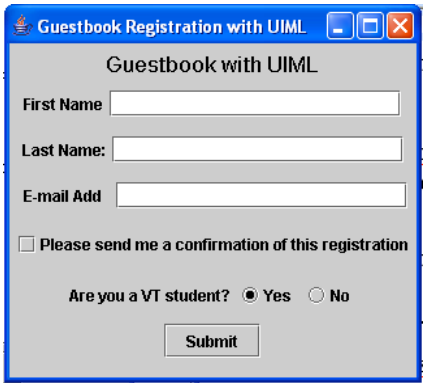

Figure 9. JAVA Swing based interface rendered by the JAVA renderer by Harmonia Inc.

A question that could arise here is if it is simply a matter of code generation, we can simply take the CLICK XML and implement a new code generator around it which can provide any platform specific interface then why use UIML. As we have discussed already, in the process of mapping CLICK XML to a generic UIML we gain a lot of advantages in terms of clean design and scalable architecture. But the most significant aspect is the concept of vocabularies which makes selective rendering of each of the components possible. For each of the parts in the UIML, we can specify what class it maps to in a vocabulary and changing rendering of the part is simply changing its class. Similarly if the class is not supported by the vocabulary the part is not rendered at all. In such a case platform specific vocabularies provide much more flexibility and control over a black box code generator.

CONCLUSION

Our work demonstrated the capabilities of UIML for specifying interface for an interactive website created using an end-user web programming tool, CLICK. From our user survey we derived certain guidelines on the design of the backend code generated by both of our automated tools. We found that with UIML, leaving out the limitations of renderers freely available, it is possible to generate code for a web application that developers would prefer to work with. This included generating style sheets, separation of client side and server side logic and finally preserving of the look and feel of an interface. Also, the code generated by CLICK is undergoing a lot of changes and has not yet been evaluated against web programming best practices. Due to these reasons we could not evaluate the benefits of converting CLICK XML to UIML in terms of the code generated. UIML is a research project and evaluation of this language itself is not in the scope of this project. Although the conversion process from CLICK XML to UIML is tedious it is entirely possible to preserve the interface look and feel and interaction behavior from one representation to another. We also identified several advantages an interface description using UIML provides.

FUTURE DIRECTIONS

We were able to map the CLICK XML to a generic UIML and then from the generic UIML to HTML 4.0 specific UIML and finally to the code. While it is still possible to map this generic UIML to a PDA platform or WML platform, there are issues related to interface migration which make this process much complex. Also, a generic vocabulary is most efficient when this generic vocabulary spans across a family of devices which have the similar layout features e.g. the desktop family, PDA family, WML family [2]. If we try to merge varied platforms we end up with a bloated generic UIML. The answer then would be to raise an abstraction level beyond the device differences. A task model can be used as a starting point for the MPUI development [2] such that it captures conceptual information about the interface which remains same across multiple interfaces. We would like to reverse engineer the task model of a website from the CLICK XML. Given a task model in the CTT [5] notation we can follow the approach devised by Ali et al [2] to generate interfaces for multiple platforms using CLICK.

REFERENCES

1. Abrams, M., Phanouriou, C., UIML: An XML Language for Building Device-Independent User Interfaces. in *XML'99*. 1999. Philadelphia.

2. Ali, M.F., Abrams, M., Simplifying construction of multi-platfrom user interfaces using UIML. in *European Conference UIML 2001*. 2001. Paris: Harmonia & Aristote.

3. Ali, M.F., Pérez-Quiñones, M.A., Abrams, M., and Shell, E., Building Multi-Platform User Interfaces with UIML. in *CADUI*. 2002. France.

4. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., and Trevisan, D., USIXML: A User Interface Description Language for Context-Sensitive User Interface. in *Developing User Interfaces with XML: Advances on User Interface Description Languages, a Satellite Workshop of Advanced Visual Interfaces*. 2004. Gallipoli, Italy, 55-62

5. Paternò, F., Mancini C., Meniconi S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. in *Interact*. 1997. Sydney: Chapman&Hall, 362-369

6. Puerta, A., Eisenstein, J., XIML: A Universal Language for User Interfaces. 2001, RedWhale Software.

7. Puerta, A.R., A Model-Based Interface Development Environment. *IEEE Software*, 1997, 40-47.

8. Souchon, N., Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages. in *DSV-IS*. 2003. Berlin: Springer-Verlag, 377-391

9. Trewin, S., Zimmermann, G., and Vanderheiden, G. Abstract User Interface Representations: How well do they Support Universal Access? in *CUU*. 2003. Vancouver, British Columbia, Canada, 77-84

10. Ziegert, T., Lauff, M., Heuser, L, Device Independent Web Applications - The Author Once - Display Everywhere Approach. ICWE, 2004, 244-255.

11. World Wide Web Consortium, "XForms - The Next Generation of Web Forms," http://www.w3.org/MarkUp/Form

12. UIML3.0 Draft specification,

http://www.uiml.org/specs/uiml3/DraftSpec.htm

13. Eaton, C. and Memon, A.M., Evaluating Web Page Reliability across Varied Browsing Environments. in *The 15th IEEE International Symposium on Software Reliability Engineering (ISSRE'04)*, (Saint-Malo, Bretagne, France, 2004).

14. SELVAKUMAR, M. Automated testing for Web applications. Dr. Dobb's Journal of Software Tools 24, 5, 24 (5). 88, 90, 92, 95-96, 1999.

15. Sampath, S., Mihaylov, V., Souter, A. and Pollock, L., “Composing a Framework to Automate Testing of Operational Web-Based Software,” Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04), Chicago, Illinois, 2004.

16. XMLUnit - JUnit and NUnit testing for XML, *http://xmlunit.sourceforge.net*

17. XsltUnit, *http://xsltunit.org/*

18. HttpUnit, *http://httpunit.sourceforge.net/*