A Review of XML-compliant User Interface Description Languages

Nathalie Souchon and Jean Vanderdonckt

Université catholique de Louvain, Institut d'Administration et de Gestion
Place des Doyens, 1 - B-1348 Louvain-la-Neuve, Belgium
{souchon, vanderdonckt}@isys.ucl.ac.be

Abstract. A review of XML-compliant user interface description languages is produced that compares a significant selection of various languages addressing different goals, such as multi-platform user interfaces, device-independence, content delivery, and user interfaces virtually defined. There has been a long history and tradition to attempt to capture the essence of user interfaces at various levels of abstraction for different purposes. The return of this question today gains more attraction, along with the dissemination of XML markup languages, and gives birth to many proposals for a new user interface description language. Consequently, there is a need to conduct an in-depth analysis of features that make all these proposals discriminant and appropriate for any specific purpose. The review is extensively conducted on a significant subset of such languages based on an analysis grid and user interfaces that we tried to implement across these languages.

1 Introduction

For years, Human-Computer Interaction (HCI) witnessed a perennial race for the ultimate User Interface (UI) Description Language that would ideally capture the essence of what a UI could be or should be. A *UI Description Language* (UIDL) consists of a high-level computer language for describing characteristics of interest of a UI with respect to the rest of an interactive application. Such a language involves defining a syntax (i.e. how these characteristics can be expressed in terms of the language) and semantics (i.e., what do these characteristics mean in the real world). It can be considered as a common way to specify a UI independently of any target language (e.g., programming or markup) that would serve to implement this UI.

The issue of UIDL was first raised when it was required to develop a UI like a module of an interactive application rather than merely a series of lines codes. In a second time, the issue was reinforced when the desire appears to model a UI by a set of specifications so as to communicate these specifications and to share them across stakeholders. Or to (semi-)automatically generate the code of the UI, as desired in model-based approaches for developing UIs. When a UI was required to be run simultaneously on different computing platforms, this

J.A. Jorge, N. Jardim Nunes, J. Falcão e Cunha (Eds.): DSV-IS 2003, LNCS 2844, pp. 377–391, 2003.

need took shape in some language that would be exchanged from one platform to another without any changes to avoid any extraneous development effort.

For some years, the race progressively slept. The wide availability of markup languages and the capability of introducing any language based on XML meta-language, along with the multiplicity of today's available platforms (e.g., mobile phone, smart phone, pocket PC, handheld PC, Tiqit PC, tablet PC, laptop, traditional PC, and even wall screens) have awaken this race and have exacerbated it to a point where today more than a dozen of UIDLs exist that focus on some of the desired characteristics. To shed light on this proliferation of UIDLs, we conducted a systematic comparison based on an analysis grid. The paper focus only on XML-based languages because XML is a well established standard that is easily extensible and that could work with yet-to-be-invented appliances without many changes. Furthermore, it is declarative and can be use by non-programmers or occasional users.

For the purpose of the survey, we gathered and analyzed as much literature as possible on each UIDL. Then, depending on available tools, we systematically developed a multi-platform or multi-context UI for a simple dictionary so as to identify the capabilities of the UIDL and the ability of this UIDL to be supported by editing, critiquing, analysis tools, and, of course, tools for producing runnable UIs, both by compilation/execution and by interpretation.

The remainder of this paper is structured as follows: section 2 reports on some related work that have been considered as significant milestones in the race for **THE** UIDL. Section 3 respectively describes each UIDL that has been retained as significant in the comparison and identifies the main goals pursued by each UIDL. Section 4 similarly describes alternate UIDLs that have been considered for the comparison, but which are more restricted in scope or goals. Section 5 defines the comparison criteria to be used in the comparison analysis and provides the final analysis grids. Section 7 concludes the paper by keeping only salient features of high interest for each considered UIDL.

2 Related Work

Historically, many approaches have emerged to solve the problem of the portability (without any extraneous development effort) of UIs on multiple platform. Some approaches can be identified in the race for the ultimate UIDL [4]:

1. **Binary emulation**: This approach allows an application to be used on different platforms without having to be recompiled, thanks to a software emulator that executes the Intel instruction set, the operating and the windowing systems. *Wabi* (Windows Application Binary Interface), from SunSelect uses this approach.
2. **Virtual toolkits**: Virtual toolkits have been introduced to reduce the effort of development: the developer writes a unique code using a virtual Application Programming Interface (API) that is executed on all the platforms for which the API exists. In other words, this approach provides a software

layer between the application and the environment that makes the translation from one API to another. Two kinds of virtual toolkits exist:

- **by actualisation** (layered toolkits): the toolkit binds the virtual Abstract Interaction Object (AIO) [25] to the real CIO of the platform by actualizing them. For instance, X Virtual Terminal (Xvt) [26] gives C, C++ interface for the presentation in OSF/Motif, Open Look, Macintosh, Ms-Windows et IBM OS/2. The main benefit of this approach lies in the large range of virtual primitives. Nevertheless, its usage is limited by the presence of a massive run-time library. Indeed, the toolkit supports only those features already supported by both the source and the target platforms.
- **by re-implementation** (emulation toolkits): the toolkit re-implements each virtual AIO for each platform, it emulates the look and feel of each native environment. For instance, Galaxy [10] offers a library of CIOs that have the same layout on each platform.

Although these tools certainly contain some abstractions that are expressed in a platform-neutral format, it does not accommodate with many platform constraints.

3. **Ported APIs**: The tools based on this approach support native APIs (usually Windows) on other environments. It concentrates only on the source and the destination platforms of the application and so, it supports a high percentage of the source functionality on the destination platform. Windows Interface Source Environment (*WISE*) from Microsoft is an example of such a tool.
4. **Tools generating adaptive UIs**: Tools that generate a user interface which can be adapted at runtime depending on the context of use. An example is the BOSS-system [22] which is a component of the FUSE-architecture. The user interface generated by BOSS is very flexible, e.g. the layout style of the UI can change at runtime. Thanks to a hierarchical internal representation of the UI that can be modified at any time by restructuring rules, and that is consequently reflected by a UI refresh.
5. **Multi-context tools at the logical level**: Those tools generate at design time a concrete UI for a specific context, from an abstract description of the UI. The abstract description is written in a specific language that differs from one tool to another. Among those tools is Liquid UI [3]. Most languages of the two next sections belong to this approach, but only XML-compliant languages will be considered in this paper.

3 Significant Contributions

In this section, the main contributions of XML-compliant languages for the definition of UIs are analyzed, based on the available literature and tools.

3.1 UIML

The User Interface Markup Language [3] is a meta-language that allows designers to describe the user interface (UI) in generic terms, and to use a style description to map the UI to various operating systems, languages and devices. UIML was created by Virginia Tech's Center for Human Computer Interaction, Harmonia Inc., and other organizations on uiml.org (http://www.uiml.org). Work on UIML began in 1997.

A UIML document contains three different parts [2]: a UI description, a peers section that defines mappings from the UIML document to external entities (target platform's rendering and application logic), and finally a template section that allows the reuse of already written elements. In UIML, a UI is described as a set of interface elements with which the end-user interacts. For each part, a presentation style is given (e.g. position, font style, color), along with its content (text, images etc.) and possible user input events and resulting actions.

The interface description is then rendered according to the specification of the presentation component, and communicates with the application logic via the logic definitions. The renderer either interprets UIML on the client device (similar to the way a web browser renders an HTML file) or compiles it to another language (like WML, HTML).

One big shortcoming of UIML is that, as it just offers a single language to define the different types of user interfaces, it does not allow the creation of user interfaces for the different languages or for different devices from a single description: there is still a need to design separate UIs for each device.

UIML and the related products (LiquidUI) are still under development. Many bugs exist and the installation of the tool is quite hard (installation issues are not specified anywhere).

UIML version 3.0 was released last year. Conferences are organized each year.

3.2 AUIML

In 1998, IBM undertook an Advanced Technology project to develop a Device Independent Markup Language in XML. This project (previously called *DRUID*) ended up with a XML vocabulary called Abstract User Interface Markup Language (AUIML) [14].

AUIML allows defining the intent (or purpose) of an interaction with a user instead of focusing on the appearance. This means that the designers have to concentrate only on the semantics of the interactions. Indeed, AUIML is intended to be independent of any client platform, any implementation language, and any UI implementation technology [6]. A single intent should run on many devices. A UI is described in terms of manipulated elements (a data model that structures the information required to support a particular interaction), of interaction elements (a presentation model that specifies the look of the UI - choice, group, table, tree) and of actions which allow to describe a micro-dialogue to manage events between the interface and the data.

Besides the specification of the appearance of the UI, the presentation model allows flexibility in the degree of specificity of what is expected of the renderer: the designer can either decide to precisely control what is to be displayed or only specify the interaction style, leaving the decision to the renderer. As AUIML is mostly developed for internal use at IBM, most information is confidential. So far, no editor tools are available and little is done in publicity. The rendering engine remains confidential.

3.3 XIML

The eXtensible Interface Markup Language (XIML), the follower of MIMIC [20], provides a way to describe a user interface without worrying about its implementation. It was initially developed by the research laboratories of RedWhale Software Corp. It is now supported by the XIML forum (http://www.ximl.org), an industrial organization dedicated to the research, the dissemination, the adoption, and the standardization of XIML. The goal of XIML is to describe the UI abstracts aspects (e.g., tasks, domain and user) and concrete aspects (i.e., presentation and dialogue) throughout the development life cycle. Mappings from abstract to concrete aspects are similarly supported [9].

XIML is a hierarchically organised set of interface *elements* that are distributed into one or more *interface components* [19]. Theoretically, the language does not limit the number and types of components that can be defined and there is also no limit on the number and types of elements within each component. In a more practical sense, however, XIML predefines five basic interface components, namely: (1) the **task component** that captures the business process and/or user tasks that the interface supports; (2) the **domain component** which is a set of all the objects and classes used; (3) the **user component** that captures the characteristics of the (group of) users that can use the application; (4) the **dialog component** that determines the UI interaction, and (5) the **presentation component**[19].

Besides the *interface components*, a XIML description is composed of attributes and relations. An attribute is a feature or a property that has a value and belongs to a component. A predefined set of attributes already exists. A relation links one or several component(s) together, within a same model component or across several ones.

The definition of the language is on the way to be finished. An editor is proposed to manage the different conception levels (which is still limited and difficult to use). Although XIML specifications are intended to lead to code generation at design time and code interpretation at runtime, no tool is available.

Nevertheless, a tool converts any MOBI-D [21] model specification into an XIML specification and another one reverse engineers HTML pages into XIML (Vaquita) [8].

3.4 Seescoa XML

Seescoa (Software Engineering for Embedded Systems using a Component Oriented Approach) is a project that started in October 1999 and has to be finished

in September 2003, involving a research consortium of four Belgian university partners. The main objective of Seescoa project is to adapt the software engineering technologies to the needs of embedded software [13].

The Seescoa project proposes an architecture for runtime serialization of Java user interfaces into a XML description. This XML description provides an abstraction of the user interface, which is described as a hierarchy of Abstract Interaction Objects (AIO) [25]. Once a user interface has been serialized, and a XML description produced, the description has to move to another device, where it can be "deserialised" into a user interface for the target device. This deserialisation involves mapping the platform independent AIO onto platform specific CIO. Indeed, while parsing the XML document that contains an abstraction of the user interface, the renderer of the target platform is free to choose other ways to present the same functionality on the user interface. For every system a XSLT is defined which maps the AIO of the abstract user interface description to CIO on the foundations of the constraints of each platforms [12].

The XML description of the UI is an interface components (AIO) decomposition. It describes the "look and feel" of the UI. Besides the presentation tags (up to now, six different interactors available), the *action* tag specifies the action to be fired if the interactor is manipulated. Seescoa XML is still under development and there is no stable version of the language up to now. A shortcoming of this language is that it has only a conversation mechanism for Java User Interface, although more powerful mechanisms are being studied that would use technologies as XML-RPC and WSDL (Web Services Description Language).

3.5 Teresa XML

Teresa XML is the XML-compliant language that was developed inside the Teresa project, which is intended to be a transformation-based environment designed and developed at the HCI Group of ISTI-C.N.R (http://giove.cnuce.cnr.it). It provides an environment that supports the design and the generation of a concrete user interface for a specific type of platform [11]. The Teresa project take place inside an European project (Cameleon IST).

The Teresa XML language is composed of two parts: (**i**) a XML-description of the CTT notation [18] which was the first XML language for task models; (**ii**) a language for describing user interfaces. Teresa XML for describing UIs specifies how the various AIO composing the UI are organized, along with the specification of the UI dialog.

Indeed, a UI is a set of one or more presentation element(s). Each presentation element is characterized by a structure, that describes the static organization of the UI (the AIOs [25]) and 0 or more connections, that gives information about the relationships among the various presentations elements of the user interface (it identifies the presentation element whose activation triggers the transition to another presentation element). Each structure element can be either an elementary AIO or a composition of them. Each AIO can be either an *interaction AIO* or an *application AIO* depending on whether or not an interaction between the user and the application is involved [15].

Teresa XML is used in a tool (TERESA) that supports the generation of task models, abstract UIs, and running UIs. This tool is still under development. Some bugs still exist, that are often removed, as new versions are steadily produced. The UIs generated by TERESA may contain some errors (e.g. links broken).

3.6 WSXL

The Web Services Experience Language (WSXL), released by IBM, is a Web services centric component model for interactive Web applications. It is intended for applications providing a user experience across the Internet [5]. The two goals of WSXL are firstly to give a way to build web applications to a wide variety of channels and secondly to create web applications from other ones. WSXL is built on widely established and emerging open standards, and is designed to be independent of execution platform, browser, and presentation markup languages.

WSXL uses base components to allow easy migration and adaptation of web applications. An Adaptation Description can be associated with a WSXL base component. It describes how the markup generated by the component can be adapted to new channels. WSXL enables applications to be built out of separate presentation, data, and control components; this helps developers to separate design issues and facilitates the reassembly of multiple alternative versions of the components in order to meet the requirements of separate channels, users, and tasks.

The WSXL presentation component implements portTypes used to describe and maintain DOM-accessible instances of presentation in WSXL applications. The namespaces for elements used in presentation components are not fixed by WSXL, though commonly useful "widget" sets may be available such as those defined in the XFORMS UI draft. WSXL presentation components may generate output markup in any target XML language and should indicate which languages may be requested. WSXL is currently not yet developed for mobile user interfaces. WSXL is just designed to be the next piece of the set of web services.

4 Other Contributions

4.1 XUL

The Extensible User Interface Language (XUL) is a Mozilla's XML-based language for describing window layout. The goal of XUL is to build cross platform applications, making applications easily portable to all of the operating systems on which Mozilla runs [1].

XUL provides a clear separation among the client application definition and programmatic logic, presentation ("skins" consisting of CSS and images), and language-specific text labels. As a result, the "look and feel" of XUL applications can be altered independently of the application definition and logic.

A UI is described as a set of structured interface elements (windows, menubar, scrollbar, button ...), along with a predefined list of attributes. Scripts are added

that allow interaction with the user. Furthermore, to build cross platform web applications, some bindings can be made between XUL and other technologies introduced by Mozilla: (**i**) *the eXtensible Bindings Language (XBL)* which is a markup language that defines new elements (methods, content, properties) for XUL widgets; (**ii**) *Overlays* that are XUL files used to describe extra content for the UI; (**iii**) *XPCOM/ XPConnect* that allows the integration of new libraries and (**iiii**) *XPInstall* that provides a way to package XUL application. XUL has its focus on window-based graphical user interfaces. This focus is also a limit. XUL is not applicable to interfaces of small mobile devices. Furthermore, there are no abstractions of interaction functionality available.

4.2 XISL

The Extensible Interaction Sheets Language (XISL) is a multi-modal interaction description language. It is designed for describing interaction using multi-modal inputs and outputs [7]. It separates the description of interactions from XML contents and enables the XML contents to be used independently [17].

An interaction is described in terms of **users operations** (e.g. click, speech input) for a XML element and **actions** (e.g. screen update, speech output) based on the users operations. Only interactions are described. As XISL is a multi-modal description language, it is designed to control and support parallel and sequential inputs/outputs as well as alternative input.

The XISL execution system consists of three modules: a front-end module (a UI that has audio capabilities, e.g. a microphone or a speaker), a dialog manager module (it interprets XISL documents, manages dialog flows, and controls inputs and outputs), and a document server module (a general web server).

4.3 AAIML

The Alternate User Interface Access standard (AAIML) is being developed by the V2 technical committee of the National Committee for Information Technology Standards (NCITS).

To overcome the problem of accessibility to UIs for disable persons, the concept of *"Universal Remote Console"* (URC) has been introduced. It allows people with disabilities to remotely control a variety of electronic devices (target device/service), such as copy machines or elevators, from their personal remote control device [27]. Because all those electronic devices are manufactured by different companies, a standard must be found, that allows the personal remote control device (typically a mobile device) to control them.

When a target device or service is accessed, it transmits an abstract user interface to the remote console which, in turn, provides the particular input and output mechanisms that are appropriate for the user. V2 is currently working on the definition of a XML-based language to convey an abstract UI description from the target device or service to the URC. This language would be structured as a set of abstract interactors for input and output operations. On the URC,

this abstract description would be mapped to a concrete description, available on the platform.

4.4 TADEUS-XML

TADEUS-XML was developed for the purpose of the model-based approach of the same name [16]. In a TADEUS-XML description, a UI is made up of two parts: a model component (abstract interaction model), that describes the feature of the UI on a high level of abstraction, and a presentation component.

The XML-based interaction model is a hierarchically structured set of User Interface Objects (UIO). Each UIO has different attributes specifying their behavior. Besides this model, a XML-Based Device Definition is available, that transforms the former model into a device dependent abstract model, which is still on an abstract level but integrates some constraints specific to the target platforms (mapping of the UIO to concrete UIO). Finally, a XSL-based model description is derived, based on the knowledge of the availability of UIOs for specification representation and a running interface is generated. TADEUS-XML is on development stage. The tool supporting the mapping and design process is not yet developed.

5 General Comparison

In the two previous sections, a description of the different UIDLs for UI description was given. The purpose of this section is to make a general comparison of all the previously cited languages together in a general overview. Table 1 compares the general properties of the different UIDLs according the six following criteria's:

- **Component models**: This criteria gives the aspects of the UI that can be specified in the description of the UIs. The task model is a description of the task to be accomplished by the user, the domain model is a description of the objects the user manipulates, accesses or visualizes through the UIs, the presentation model contains the static representation of the UI and the dialog model holds the conversational aspect of the UI.
- **Methodology**: Different approaches to specify and model UIs exist:
  - Specification of a UI description for each of the different contexts of use. As a starting point, a UI specification for the context of use considered as representative of most case, the one valid for the context of use considered as the least constrained or finally the one valid for the context of use considered as the most comprehensive is specified. From this starting UI specification, corrective or factoring out decorations [24] (e.g., to add, remove, or modify any UI description) are applied so that UI specifications can be derived for the different contexts of use.
  - Specification of a generic (or abstract) UI description valid for all the different contexts of use. This generic UI description is then refined to meet the requirements of the different contexts of use.

- **Tools**: Some of the languages are supported by a tool that helps designer and renders the specification to a specific language and/or platform.
- **Supported languages**: Specify the programming languages to which the XML-based language can be translated.
- **Platforms**: Specify the computing platform on which the language can be rendered by execution, interpretation or both.
- **Target**: A context of use [23] is made up of three different models: the user model, the environment model (that represents different configuration of the physical conditions in which the application is used) and finally the platform model (represents any property of the platform). This criteria is aimed at indicating which model variation the markup language was designed for (i.e., mono/multi-platform, mono/multi-user or mono/multi-environment).

Table 2 compares UIDLs according to the five following criteria:

- **Abstraction level**: each UIDL may exhibit the capability to express a runnable UI (instance level), one or many models involved in the development of this UI (model level), how these models are built (meta-model level), and what are the fundamental concepts on which this operation is based (meta-meta-model level).
- **Amount of tags**: to reach the above level of abstraction, each UIDL manipulates a certain amount of tags, which is also highly depending on the coverage of the concepts.
- **Expressivity of the language**: this criteria denotes not only the capability of the UIDL to express concepts of the real world, but also the easiness and the usability of manipulating them with the UIDL. If, for a same expressible concept, a first UIDL needs 5 lines of specification and another one, only 2, the latter will be said to be more concise.
- **Openness of the language**: this criteria informs the designer whether a UIDL sees its concepts or tags fixed or user-modifiable. A UIDL can have a fixed amount of tags while keeping the capability to introduce new concepts that have not been specified in the canonical definition.
- **Coverage of concepts**: depending on the level of abstraction, each UIDL may introduce some specific vs. generic concepts (e.g., a given presentation model vs. any model, each custom-defined), their properties (e.g., to what extent can a concrete presentation be specified), and their relations.

6 Acknowledgements

We gratefully acknowledge the support from the European Commission through the CAMELEON IST project of Vth framework programme (http://giove.cnuce.cnr.it/cameleon.html). The authors would like also to thank Loubna Id-Bouharia for providing a first version of the Cameleon document "D1.3 Companion-Comparison of XML-based languages for specifying user interfaces".

	Models	Methodology	Tools	Supported languages	Supported Platforms	Target
UIML	Presentation and dialog	Specification of multiple UI presentations and factoring out/corrective decoration.	Liquid UI: rendering engine, code editor and generator	Java, HTML, WML, VoiceXML, C++, PalmOS	Handheld and desktop PC; Smart, standard and mobile phone; vocal UI.	Multi-platform.
AUIML	Presentation and dialog.	Specification of a generic UI description. The decoration can be done either by the renderer or by the developer.	Rendering engine	HTML, DHTML, Java Swing, PalmOS, WML	Handheld and desktop PC	Multi-platform (available interactors, displays)
XIML	Any model.	Specification of multiple UI descriptions or of a generic one.	Code editor	HTML, WML, Java	None for the moment. In the future: Handheld and desktop PC ; mobile phone ; Java Terminal	In theory, multi-platform, -user and -environment; in practice, multi-platform
Seescoa XML	Presentation and dialog	Specification of a generic UI description.	Rendering engine	Java Swing and AWT, HTML	Handheld and desktop PC	Multi-platform
Teresa XML	Presentation and dialog ; Task, domain and platform	Specification of a generic UI description.	TERESA	XHTML, Voice XML	Handheld and desktop PC, mobile phone	Multi-platform
WSXL	Presentation, dialog and data.	Specification of multiple UI descriptions and factoring out/corrective decoration	WSXL SDK	Markup languages: HTML, XUL, UIML, ...	Desktop PC	Mono -platform, -user, -environment
XUL	Presentation and dialog.	Specification of multiple UI descriptions and factoring ou/corrective decoration	Rendering engine Gecko, XPCOM/ XPConnect, XPInstall, Mozilla	XUL	Desktop PC (Web application using Mozilla)	Multi-platform
XISL	Dialog.	Specification of multiple UI descriptions and factoring out/corrective decoration	XISL Interpreter	XML-based languages	Mobile phone, Desktop PC, digital TV with multi-modal capabilities	Multi-platform
AAIML	Presentation and dialog.	Not yet defined	Not yet developed, Prototypical architecture of the URC.	Not yet defined	Simulation on Handheld PC, Smart TV	Multi-user on multiple platforms
TADEUS XML	Presentation (based on a user, a task and an object models)	Specification of a generic UI description	TADEUS XML converter to be developed	Not specified	Not specified	Multi-platform

Table 1. Comparison of UIDLs general properties.

	Level	Tags	Expressivity	Openness	Concepts
UIML	Model level	36 tags	Moderate	No	Interface, structure, style, content, behavior, part, peers, logic, presentation
AUIML	Model level	No clear information available, at least 55 tags	Moderate	No	Date-group, group, actions
XIML	Meta-model level	33 tags.	High: everything can be expressed, as the language is open	Yes	Component models, model element, relation_definition, feature_definition, attribute_definition
Seescoa XML	Model level	On the way to be completed, no stable DTD available	Low	No	Group, interactor, action
Teresa XML	Model level	32 tags for the UI description	High	No	Presentation, structure, AIO, interaction_AIO, application_AIO, AIO_composition and connection
WSXL	Instance and model level	No limit, XFORMS	Low: web-application, only graphical UI	No	
XUL	Instance and model level	At least 60 tags	Low: limited to windows-based graphical UI	No	Window, box, hbox, vbox
XISL	Model level	53 tags	High: multimodal UI on multiple platforms. Input Modalities supported: DTMF, speech, pointing, keyboard. Output Modalities: window, speech, video, audio, agent.	Yes	Dialog, exchange, operation, action, input, output
AAIML	Model level	Not yet defined	Moderate	Not specified	Not yet defined
TADEUS XML	Model level	Not specified	Low	No	Uio, input, output, trigger

Table 2. Comparison of UIDLs capacities.

7 Conclusion

The previous investigation and comparison of the most significant UIDLs, if not all, reveal that there might be a plethora of UIDLs, from which it may seem hard to pick up one. We believe that this choice is more dictated by the goals to be pursued if one decides to adopt one of these UIDLs rather than only the different criteria that have been compared.

For instance, XUL is an official Mozilla initiative which received considerable attention from the international audience. However, XUL is mainly intended to support different viewing capabilities that are required to be supported by different computing platforms. Per se, it does address some requirements for supporting multiple platforms, but it is not intended to be a genuine and complete UIDL, as it is probably the less expressive one.

On the other end of the expressiveness continuum is located XIML which demonstrates the highest expressivity possible since it is located at the meta-model level (the only one in the comparison). Therefore, XIML is particularly appropriate to specify UIsfor multiple platforms, multiple contexts of use, even for custom situations that have not been thought before, as it is an open language. But its tool support is less advanced at the time of writing this paper than tools provided by UIML. UIML seems to be one of the most restrictive UIDLs, but the one which is the most supported by software. The real attractiveness of a UIDL heavily depends from this: it is meaningless to possess a refined specification of a UI that cannot be rendered or only partially.

Thus, we believe that XIML should be more appreciated for its interoperability qualities for exchanging UI descriptions between stakeholders (e.g., from one software to another), while UIML should be more accepted for true generation. AUIML is dedicated to accessibility issues and should probably be used only in these circumstances. AUIML is today more part of the internal processes of IBM than in a complete suite of tools, although IBM WebSphere became a truly operational software with wide scope in mind. Like UIML, AUIML only supports some predefined features of the presentation and the dialog models. It does not support other models that are manipulated in context-sensitivity. Moreover, it is impossible to expand the language. IBM today focuses more on the development of Web services, through the WSUI and the WSXL languages.

Furthermore, one of the main conclusions of the survey is that tools are not only difficult to use, but that they often result in low visual quality user interfaces.

In this study, we did not consider Xforms (see http://www.w3.org/MarkUp /Forms/), which is a W3C initiative to express forms-based UIs at a level that is more abstract than supposed-to be physical HTML descriptions. In some way, this initiative addresses the question of multiple computing platforms. Although Xforms is promoted by the W3C, thus giving it the widest potential audience abroad, implementation is only at the beginning. Xforms is basically aimed at expressing forms-based UIs with presentation and some dialog aspects, but does not necessarily support other UI modalities (e.g., vocal UIs).

Finally, we did not consider the legal issues of using one of these languages in a tool to be used or developed. XIML is protected by copyright by the XIML Consortium. Any software that is XIML compliant can consequently be distributed only if the future user of this software already possesses a XIML license. Although this license can be freely obtained from the XIML consortium, this registration process may be interpreted as a burden and a potential reduction of the audience. The vast majority of the other UIDLs are totally free of use.

References

[1] XUL tutorial, 2003. http://www.xulplanet.com/tutorials/xultu/.
[2] M. Abrams. Device-independent authoring with UIML. In *W3C Workshop on Web Device Independent Authoring*, Bristol, 2000.
[3] M. Abrams, C. Phanouriou, A.L. Batongbacal, S. Williams, and J. Shuster. UIML: An Appliance-Independent XML User Interface Language. In A. Mendelzon, editor, *Proceedings of 8th International World-Wide Web Conference WWW'8 (Toronto, May 11-14, 1999)*, Amsterdam, 1999. Elsevier Science Publishers.
[4] M. Argollo Jr. and C. Olguin. Graphical user interface portability. *CrossTalk: The Journal of Defense Software Engineering*, 10(2):14–17, 1997.
[5] A. Arsanjani, D. Chamberlain, and et al. (WSXL) web service experience language version, 2002. http://www-106.ibm.com/developerworks/library/ws-wsxl2/.
[6] P. Azevedo, R. Merrick, and D. Roberts. OVID to AUIML - user-oriented interface modelling. In N. Nunes, editor, *Proceedings of 1st International Workshop "Towards a UML Profile for Interactive Systems Development" TUPIS'00 (York, October 2-3, 2000).*, York, 2000.
[7] T. Ball, Ch. Colby, P. Danielsen, L.J. Jagadeesan, R. Jagadeesan, K. Läufer, P. Matag, and K. Rehor. SISL: Several interfaces, single logic. Technical report, Loyola University, Chicago, January 6th, 2000.
[8] L. Bouillon, J. Vanderdonckt, and N. Souchon. Recovering alternatives presentation models of a web page with vaquita. In *Proceedings of 4th Int. Conf. On Computer-Aided Design of User Interfaces CADUI2002 (Valenciennes, 15-17 May 2002)*, pages 311–322, Dordrecht, 2002. Kluwer Academics Pub.
[9] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of ACM Conference on Intelligent User Interfaces IUI'2001 (Albuquerque, January 11-13, 2001)*, pages 69–76, New York, 2001. ACM Press.
[10] Galaxy Application Environment. Visix Software Inc., 11440 Commerce Park Drive, Reston (VA 22091), 1993.
[11] Paternò. F and Santoro. C. One model, many interfaces. In Ch Kolski and J. Vanderdonckt (Eds.), editors, *Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002)*, pages 143–154, Dordrecht, 2002. Kluwer Academics Publishers.
[12] K. Luyten and K. Coninx. An XML-based runtime user interface description language for mobile computing devices. In *Proceedings of the 8th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2001, (Glasgow June 13-15 2001)*, pages 20–29, Berlin, 2001. Springer Verlag.
[13] K. Luyten, C. Vandervelpen, and K. Coninx. Adaptable user interfaces in component based development for embedded systems. In *Proceedings of the 9th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2002, (Rostock, June 12-14, 2002)*. Springer Verlag, 2002.

[14] R. Merrick. Device independent user interfaces in XML, 2001. http://www.belchi.be/event.htm.

[15] G. Mori, F. Paternò, and C. Santoro. Tool support for designing nomadic applications. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces IUI 2003 (Miami, January 12-15)*, pages 149–157, New York, 2003. ACM Press.

[16] A. Müller, P. Forbrig, and C. H. Cap. Model-based user interface design using markup concepts. In Ch. Johnson (Eds.), editor, *In Proc. Of 8th International Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2001 (Glasgow, 13-15 Juin 2001)*, pages 16–27, Berlin, 2001. Springer-Verlag.

[17] T. Nitta et Al. XISL: An attempt to separate multimodal interactions from XML contents. In *Eurospeech 2001*, pages 1197–1200, Aalborg, 2001.

[18] F. Paternò. *Model Based Design and Evaluation of Interactive Applications*. Springer-Verlag, Berlin, 1999.

[19] A. Puerta and J. Eisenstein. XIML: A common representation for interaction data. In *Proc. Of the 7th International Conference on Intelligent User Interfaces (Santa Fe, United States, January 2002)*, pages 69 – 76., New York, 2002. ACM Press.

[20] A. R. Puerta. The mecano project: Comprehensive and integrated support for model-based user interface development. In J. Vanderdonckt, editor, *Proc. Of the 2nd Int. Workshop on Computer-Aided Design of User Intercace CADUI'96 (Namur 5-7 June 1996)*, pages 19–37, Namur, 1996. Presses Universitaires de Namur.

[21] A. R. Puerta. A model-based interface development environment. *IEEE Software*, 14(4):40–47, 1997.

[22] S. Schreiber. Specification and generation of user interfaces with the BOSS system. In J. Gornostaev et al, editor, *Proceedings East-West International Conference on Human-Computer Interaction EWHCI'94 (St. Petersburg, August 2-6, 1994)*, Moskau, 1994. Springer.

[23] N. Souchon, Q. Limbourg, and J. Vanderdonckt. Task modelling in multiple contexts of use. In *Pre-Proceedings of the 9th International Workshop on Design, Specification and Verification of Interactive Systems Workshop DSV-IS'02 (Rostock, June 12-14, 2002)*, 2002.

[24] D. Thevenin. *Adaptation En Interaction Homme-Machine : Le Cas de la Plasctricité*. PhD thesis, Université Joseph Fourier, 21 December 2001.

[25] J. Vanderdonckt and F. Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White, editors, *Proceedings of the ACM Conference on Human Factors in Computing Systems InterCHI'93 (Amsterdam, 24-29 April 1993)*, pages 424–429, New York, 1993. ACM Press.

[26] XVT. XVT Software, Inc., 4900 Pearl East Circle, Boulder, CO, 80301, USA, 1996.

[27] G. Zimmermann, G. Vanderheiden, and A. Gilman. Universal remote console - prototyping for the alternate interface access standard. In N. Carbonell and C. Stephanidis, editors, *Universal Access: Theoretical Perspectives, Practice and Experience - 7th ERCIM UI4ALL Workshop (Oct. 2002, Paris, France)*. Springer-Verlag, 2002.