CAMELEON Project

R&D Project IST-2000-30104

Title of Document:	XML specification for User Interface Modeling Language
Author(s):	Id-bouharia Loubna, Jean Vanderdonckt, Quentin Limbourg Revised by Fabio Paternò, Carmen Santoro, Revised by Joëlle Coutaz Revised by Jean Vanderdonckt
Affiliation(s):	UCL, CNR
Date of Document:	September 2nd, 2002
CAMELEON Document:	D1.3
Distribution:	Reviewers
Keywords list:	AUIML, Language, Model-based approach, Specification language, User interface model, User interface specification, UIML, XIML, XML, XML-based languages, Tadeus XML, SISL, XISL, WSXL
Version:	*Edited by CNR from two D1.3 companion documents sent by UCL. Updated with new chapter 8 by UCL and other slight changes.*
Comment:	

CAMELEON Partners:

CNR, Pisa, Italy

Université catholique de Louvain, Belgium

University of Grenoble, France

MTCI, Motorola, Turin, Italy

IS3-GICE, Paris, France

Title: XML specification for User Interface Modeling Language	**Id Number: D 1.3**

Abstract

The aim of the CAMELEON Deliverable 3.1 is twofold: first, to provide an analysis of existing XML-based languages that can be used to express user interfaces at a certain level of abstraction. Second, to lay the foundations for identifying what characteristics should have a CAMELEON notation in order to be suitable to exchange models among partners involved in the project.

Title: XML specification for User Interface Modeling Language	**Id Number: D1.3**

Table of contents

Title: XML specification for User Interface Modeling Language	**Id Number: D 1.3**

Introduction

There has been a perennial race during several years between research and development teams to propose, promote, and widespread a user interface specification language and establish it as a de facto standard. Up to now, no common agreement exists on what the ultimate user interface specification language should be. The XML revolution however exacerbated the situation, although improving it by promoting XML-based languages rather than ad hoc languages. Many initiatives exist today both from the industry and the academia to reintroduce a user interface specification language that can serve as a common platform between models, methods, and tools. Also in the CAMELEON Project, the issue of identifying a common language suitable to be used for expressing any potential component model of any user interface model manipulated in the project, has been addressed. With this regard, various existing languages for specifying user interfaces have been considered and to this aim the possibility of introducing a new CAMELEON language has been analysed.

In this document we first provide an overview of the main issues concerning the various languages for specifying user interfaces (Chapter 1). We provide an analysis and a comparison of existing languages for specifying user interfaces (Chapter 2 - Chapter 6). Then, we move on to a hypothetical CAMELEON language, by identifying the requirements for a CAMELEON notation, intended to be used for exchanging models among partners involved in the project (chapter 7). Lastly, we include a contribution that aims at providing the formalization of a number of relevant concepts (appendix).

Chapter 1 *Languages for UI specification: main issues*

Devices and user interfaces diversity.

Because of the explosion of devices, computing platforms, contextual conditions, user interfaces need to adapt to multiple configurations of the context of use. Nowadays, there is an increasing need for modelling tasks which can be supported in multiple contexts of use. Many works have already addressed the issues raised by context-sensitivity which often starts with change of computing-platform or cross-computing platforms changes. This approach becomes more crucial because some UIs are particularly tailored to specific computing platforms, sometimes with Web-based add-ins. Equally important are the UIs which are specialised and adapted to a series of computing platforms, for instance different Web-appliances, since more limited platforms, such as cellular phones or Personal Digital Assistants (PDAs), cannot support the advanced features offered by classic desktop computing platforms.

Sometimes in the same computing platform, there are variations that are caused for example by the programming language, display capacities, CPU speed and storage, etc.. Figure 1 shows some possible variations.

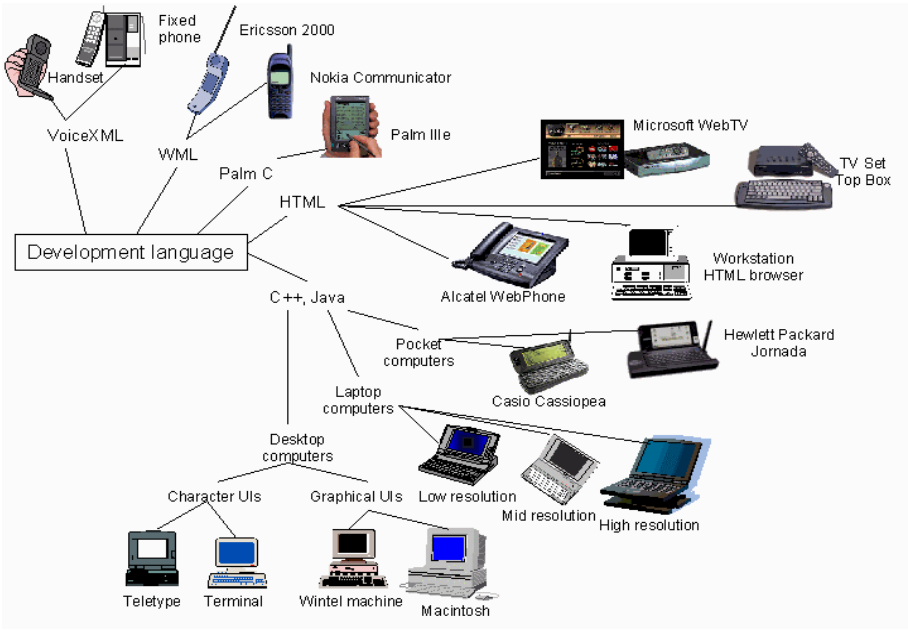


Fig. 1[1]: Example of a hypothetical decision tree for selecting a target computing platform.

Since few years there has been significant efforts to integrate Web technologies into various devices (e.g., mobile, TV sets) other than the traditional Web access equipment such as PCs.[2]In the market today there are at least four categories of devices that can run a computer-generated user interface: desktop computers[3], palmtop and handheld computers[4], smart-phones[5], and traditional voice phones[6].

[1] Costin Pribeanu, Quentin Limbourg, Jean Vanderdonckt, *Task Modelling for Context-Sensitive User Interfaces.*

[2] http://www.w3.org/2001/di/

[3] Desktop have graphical displays, ample memory and computing power, voice capabilities, and the ability to download, store, and run executable code.

[4] Handheld devices have small displays and limited computing power, but they usually allow loading of small executable code segments

[5] Smart-phones have small displays, a few lines of text, little or no processing power, and no downloading of code.

[6] Voice phones use speech synthesis or recorded sounds for output and voice recognition for input.

Moreover, today's Internet appliances feature user interface technologies almost unknown a few years ago: touch screens, styli, handwriting and voice recognition, speech synthesis, tiny screens, and more. This raises a need to support a constantly evolving set of constraints representing the characteristics of web appliances that need to be supported or other computing platforms. This richness of characteristics argue for one more step in abstracting user interfaces good for both today's computer interfaces and future interfaces but creates different kind of problems. First, we are often faced with the problem that services for those devices may not inter-operate with each other or with the existing Web.

Second, different appliances use different languages: WML for cell phones; SpeechML, JSML, and VoxML for voice enabled devices such as phones; HTML and XUL for desktop computers, and so on. Thus, developers must maintain multiple source code families to deploy interfaces to one information system on multiple appliances. Third, user interfaces differ seriously in complexity (e.g., PC versus cell phone interfaces). Thus, developers must also manage interface content. Fourth, developers risk writing appliance-specific interfaces for an appliance that might not be on the market tomorrow. A solution is to build

interfaces with a single, universal language free of assumptions about appliances and interface technology[7].

The existence of different kind of devices and the richness of its characteristics introduce a serious costly problem: the developer must maintain several different bodies of source code for the different interfaces. The effort required to keep all interfaces consistent, or maintain the interfaces is expanded, grows linearly with the number of interfaces. In contrast, the use of an appliance-independent UI language would greatly reduce the amount of code that needs to be maintained and make the maintenance cost sub-linear.[8]

Table 1 Typical characteristics of major Internet appliances at start of 1999[9].

Internet Appliances					
Characteristic	PC	Handheld PC	Palm	Cellular Phone	Voice Phone
Can download applications	Yes	Yes	Yes	No	No
Output devices	>= 800x600 color & speaker	240x480 mono, gray, or color	Portrait orientation mono, gray	3 or 4 line character text	Speaker
Input devices	Keyboard, pointing device, mike	Keyboard, stylus, touch screen	Keypad, stylus, touch screen	Keypad, mike	Keypad, mike

[7] http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html

[8] The maintenance cost is sub-linear, but not constant. That is because the appliance-independent portion can be maintained independent of the number of appliances used, while the portion that maps to specific devices grows with the number of devices (from http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html)

[9] http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html

UI metaphor	GUI, multiple windows	GUI, single window plus message boxes	GUI, single window plus message boxes, handwriting recognition	Cards	Speech synthesis, voice recognition, keypad input
UI development tools	DHTML, XwingML, XUL, many more	OS specific toolkit[10], Java	OS specific toolkit[11]	WML	VoxML, SpeechML, JSML
Graphics	Yes	Yes	Yes	No	No
Primary Memory, Mb	4-256	4-64	1-8	<1	None
Secondary Memory, Mb	Gigabytes	None	None	None	None
Processor speed, MHz	233-450	75-190	16-75	<=50	None

Why a New Language?

According to Marc Abrams, Constantinos Phanouriou, and al., it is necessary to start from scratch and design a new language. An alternative approach is to augment or modify an existing language to match any arbitrary appliance.

[10] Toolkit is the markup language or software library upon which an application's user interface runs.

[11] Toolkit is the markup language or software library upon which an application's user interface runs.

Existing languages were designed with inherent assumptions about the type of user interfaces and appliances for which they would be used. For example, the original HTML started as a language for describing documents (with tags for headings, paragraphs, and so on), and was augmented to describe forms (a different approach has been followed with XHTML). As another example, Javascript events correspond to a PC with a GUI, mouse, and keyboard. In theory, it is possible to modify languages to handle any type of appliance, but this produces a stress on the language design.

A language like Java, which contains fewer assumptions, would be more adequate as a universal, appliance-independent language. But this would require Java to run on all appliances (which may never occur), and appliance-specific code (e.g., for layout) would be needed.

Based on these arguments, it would be more suitable to create a new language based on XML.

Why based on XML?

Because XML permits new tags to be defined, this is appealing in designing a language that must work with yet-to-be-invented appliances.Here below there is a summary of each makup language based on, or compliant to, XML, that will be described in detail.

- *The User Interface Markup Language (UIML)* is an XML-compliant language that allows a declarative description of a user interface in a device-independent manner[12]. To create a user interface, one writes a UIML document, which includes presentation style appropriate for devices on which the user interface

[12] Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S., Shuster, J.: UIML: An Appliance-Independent XML User Interface Language. In: Mendelzon, A. (ed.): Proceedings of 8th Int. World-Wide Web Conference WWW'8 (Toronto, May 11-14, 1999). Elsevier Science Publishers, Amsterdam (1999). Accessible at http://www8.org/w8-papers/5b-hyper text-media/uiml/uiml.html and Abrams, M., Phanouriou, C.: UIML: An XML Language for Building Device Independent User Interfaces. In: Proceedings of XML'99 (Philadelphia, December 1999).

will be deployed. UIML is then automatically mapped to a language used by the target device of a particular computing platform, such as HTML, Java/JFC, WML, HTML, PalmOS and VoiceXML.

- *The Abstract User Interface Markup Language (AUIML)*[13] is an XML vocabulary designed to allow the intent of an interaction with a user to be defined. It allows device independent encoding of information. All the interaction information can be encoded once and subsequently rendered using device dependent rendering.

The eXtensible Interface Markup Language (XIML)[14] is an extensible XML specification language for multiple facets of multiple models in a model-based approach. An XIML specification can lead to both an interpretation at runtime and a code generation at design time.

- *The Seescoa XML* describes the graphical user interfaces as well as its runtime state[15]. By doing so, user interfaces can migrate between platforms dynamically.

Elsevier Science Pub., Amsterdam (1999). Accessible at http://www.harmonia.com/resources/xml99Fi nal.pdf

[13] Azevedo, P., Merrick, R., Roberts, D.: OVID to AUIML - User-Oriented Interface Modelling. In: Nunes, N. (ed.): Proceedings of 1st Workshop "Towards a UML Profile for Interactive Systems Development" TUPIS'00 (York, October 2-3, 2000). Accessible at http://math.uma.pt/tupis00/submissions/azevedoroberts/azevedoroberts.html .

[14] Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S., Shuster, J.: UIML: An Appliance-Independent XML User Interface Language. In: Mendelzon, A. (ed.): Proceedings of 8th Int. World-Wide Web Conference WW'8 (Toronto, May 11-14, 1999). Elsevier Science Publishers, Amsterdam (1999). Accessible at http://www8.org/w8-papers/5b-hyper text-media/uiml/uiml.html

[15] Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

- *The TERESA XML* language supports a transformation based environment for Web applications accessible through heterogeneous interaction platforms.Xxthe previous sentence is unclear. I can't rephrase itXX

Chapter 2 *User Interface Markup Language*[16]

U.I.M.L = User Interface Markup Language.

UIML can be viewed as a meta- or extensible language, analogous to XML. UIML does not contain tags specific to a particular user interface toolkit (e.g., <WINDOW> or <MENU>). Instead, it uses a set of generic tags (e.g., <part>, <property>).

UIML captures the elements that are common to any user interface: an enumeration of the user interface parts, events that occur for those parts, presentation style, content, and interconnection to application logic.

U.I.M.L objectives

The User Interface Markup Language (UIML) is an XML-compliant language.

One objective of UIML is to permit a UIML document to be mapped to any type of user interface. UIML allows insulating the interface designer from the peculiarities of different appliances through style sheets. A measure of the power of UIML is that it can replace hand-coding of Java AWT or Swing user interfaces.

According to Marc Abrams, Constantinos Phanouriou, and al., UIML allows designers to describe the user interface in generic terms, and then use a style description to map the interface to various operating systems (OSs) and appliances. Thus, thanks to its universality, UIML makes it possible to describe a rich set of interfaces and reduces the work in porting the user interface to another

[16] http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html

platform (e.g., from a graphical windowing system to a hand-held appliance) by changing the style description.

UIML represents an interface in five parts: the interface structure, presentation style, content (words, images, and sounds), and actions taken in response to user interaction, and interconnection of the interface to application logic.

UIML environment

UIML considers a user interface as a set of interface elements interacting with the user. For the different categories of users and the different families of appliances, these elements can be organized differently. Each interface element has data (e.g., text, sounds, images) used to communicate information to the user. Interface elements can also receive information from the user using interface artifacts (e.g., a scrollable selection list) from the underlying appliance. Since the artifacts vary from appliance to appliance, the actual mapping (rendering) between an interface element and the associated artifact (widget) is done using a style sheet.

Runtime interaction is done using events which can be local (between interface elements) or global (between interface elements and objects that represent an application's internal program logic [i.e., the backend, or functional core]).

Since the interface typically communicates with a backend to perform its work, a runtime engine provides for that communication. The runtime engine also facilitates a clear separation between the interface and the functional core.

UIML possibility to generate user interfaces.

A UIML document can be used in one of several ways[17]. First, as shown in Figure 2, a UIML document can be stored on a server.

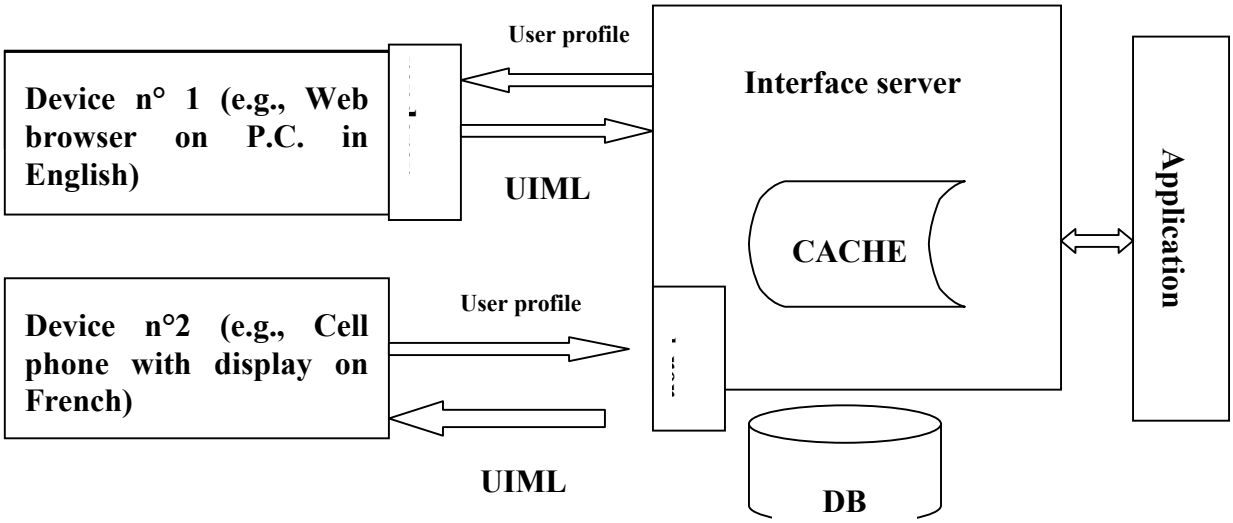


Fig. 2: Use of an UIML document.

When an application is invoked, the UIML document, either is compiled[18] to a target platform's language, or is interpreted[19] as the interaction takes place with the interface. Interpretation is more flexible; As shown in Figure 1 XXI am not sure this is shown in Figure 1XX, the Java interpretive renderer permits the entire UIML interface to appear as a Java bean, so that the interface may be manipulated programmatically by the application logic. UIML can also be used without a server; in this case, UIML or the compiled code or markup language is installed

[17] MarcAbrams and ConstantinosPhanouriou

[18] Compilation on the server side is mandatory for devices that cannot download applications, such as cellular phones, and where memory is limited.

[19] Interpretation is what a Web browser does with HTML.

along with the application logic on end-user devices. At runtime, the rendered interface directly communicates with the application logic.

An advantage of running an interpreter on the client side is that it avoids the need for sending bulky executable code to the devices. For example, to-day, Java applets are time consuming to load over the Internet. In contrast, a Java interpretive renderer installed on a desktop computer requires only UIML to be transmitted over the Internet. UIML is a small text file compared to the equivalent executable code.

Another advantage of sending UIML instead of executable code over the networks is security. UIML is a declarative language, which means it describes what needs to happen but not how it happens.

Examples created with UIML

Example n°1

Tigure 3 shows a single window with a menu. The associated code is described below;

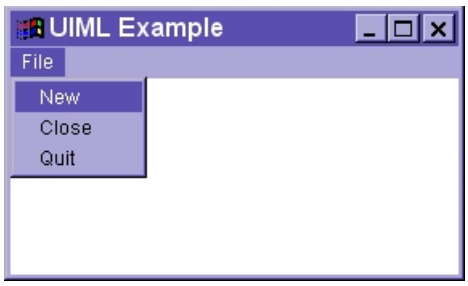


Fig.3[20]**:** A single window with a menu.

[20] http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html#Fig_4

```
<?xml version="1.0" standalone="no"?>
      <uiml version="2.0">
      <interface name="Figure4" class="MyApps">
      <description>
            <element name="Main" class="Main"/>
            <element name="File" class="ActionGroup"/>
            <element name="NewAction" class="ActionItem"/>
            <element name="CloseAction" class="ActionItem"/>
            <element name="QuitAction" class="ActionItem"/>
      </description>

      <structure>
            <element name="Main">
                  <element class="Bar">
            <element name="File">
            <element name="NewAction"/>
            <element name="CloseAction"/>
                  <element class="Separator"/>
            <element name="QuitAction"/>
            </element>
                  </element>
            </element>
</structure>

<data>
<contentname="Main">Example</content>
<contentname="File">File</content>
<contentname="NewAction">New</content>
```

```
<contentname="CloseAction">Close</content>

<content name="QuitAction">Quit</content>

</data>

      <style>

            <attribute        class="Main"        type="rendering"
value="java.awt.Frame"/>

            <attribute class="Main" type="size" value="100,80"/>

            <attribute     class="ActionItem"     type="rendering"
value="java.awt.MenuItem"/>

            <attribute      class="Separator"      type="rendering"
value="wrapper.MenuSeparator"/>

            <attribute     class="ActionGroup"     type="rendering"
value="java.awt.Menu"/>

            <attribute         class="Bar"         type="rendering"
value="java.awt.MenuBar"/>

      </style>


      <events>

            <event      name="SelectQuit"      class="ActionSelect"
source="QuitAction" trigger="Select">

                  <action target="Main" method="exit"/>

            </event>

      </events>

      </interface>

      <logic>

      </logic>

      </uiml>
```

Example n°2[21]:

One will find below, an example of mapping a credit card dialog to WML, VoiceXML and Java AWT code.

```
<?xml version="1.0"?>

<!DOCTYPE uiml PUBLIC

"-//UIT//DTD UIML 2.0 Draft//EN"

"http://uiml.org/dtds/UIML20.dtd">

<uiml>

<head>

<meta name="Description"

content="Use of templates for CreditCard entry form"/>

<meta name="Author" content="Constantinos Phanouriou"/>

<meta name="Email" content="uiml-editor@uiml.org"/>

</head>

<peers>

<presentation name="WML">

<component name="Dialog" maps-to="wml:card"/>

<component name="TextField" maps-to="wml:input"/>

<component name="Button" maps-to="wml:DO type='ACCEPT'"/>

<component name="InputEvent"
```

[21] Abrams, M., Phanouriou, C.: UIML: An XML Language for Building Device Independent User Interfaces. In: Proceedings of XML'99 (Philadelphia, December 1999). Elsevier Science Pub., Amsterdam (1999). Accessible at http://www.harmonia.com/resources/xml99Fi nal.pdf

```
maps-to="wml:event type='onenterforward'"/>
</presentation>
<presentation name="VoiceXML">
<component name="Dialog" maps-to="vxml:form"/>
<component name="TextField" maps-to="vxml:prompt"/>
<!-- Button does not have a rendering in Voice -->
<component name="Button" maps-to=""/>
<component name="InputEvent" maps-to="vxml:filled"/>
</presentation>
<presentation name="Java-AWT">
<component name="Dialog" maps-to="java.awt.Dialog"/>
<component name="TextField" maps-to="java.awt.TextField"/>
<component name="Button" maps-to="java.awt.Button"/>
<component name="InputEvent"
maps-to="java.awt.event.ActionEvent"/>
</presentation>
<logic name="HTTP">
<component name="CreditApp">
<method name="SendCredit"
maps-to="http://<server>:port/cgi-bin/acceptCredit.pl?">
<param name="cnum"/>
</method>
</component>
```

```
</logic>
<logic name="Java">
<component name="CreditApp" maps-to="myBiz.creditApp.class">
<method name="SendCredit" maps-to="acceptCredit">
<param name="cnum"/>
</method>
</component>
</logic>
</peers>
<template name="CreditCard">
<part name="CreditContainer" class="CreditDialog">
<style>
<proprety name="title">Credit Card Entry Form</property>
</style>
<part name="CreditNum" class="number"/>
<part name="AcceptNum" class="Accept">
<style>
<proprety name="content">Accept</property>
</style>
</part>
</part>
</template>
<interface>
```

```
<structure>
<part name="ECommerceApp">
<!-- UI description -->
<part name="MyCredit" source="#CreditCard"/>
</part>
</structure>
<style>
<property name="rendering" part-class="MyCredit.CreditDialog">
Dialog</property>
<property name="rendering" part-class="MyCredit.number">
TextField</property>
<property name="rendering" part-class="MyCredit.Accept">
Button</property>
<property name="rendering" event-name="InputEvent">
InputEvent</property>
<property name="rendering" method-name="SendCredit">
SendCredit</property>
</style>
<behavior>
<rule>
<condition>
<event name="InputEvent" part-name="MyCredit.AcceptNum"/>
</condition>
```

<action>

<method name="SendCredit">

<param name="cnum">

<property name="content"

part-name="MyCredit.CreditNum"/>

</param>

</method>

</action>

</rule>

</behavior>

</interface>

</uiml>

UIML advantages and drawbacks

UIML is a declarative language. It allows the UI designer to specify just what needs to be done, but not how.

UIML distinguishes *which* elements are present in an interface, *what* the structure of the elements are for a family of similar appliances, *what* natural language text should be used with the interface, *how* the interface is to be presented or rendered using cascading style sheets, and *how* events are to be handled for each user interface element. Here is the list of the advantages of the UIML approach[22]:

- Interface designers learn one language, yet can use any appliance.

[22] http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html

- UIML offers the expressive power of programming languages but with the advantages of a declarative language. UIML offers also a quiet fast download time, a better security, and usability for non-programmers.
- UIML is XML-compliant, thereby providing a natural way for XML users to create user interfaces for client-server applications, or to embed interfaces in documents and databases.
- Designers of new interface hardware can offer a simple migration path by mapping the universal language to their appliance.
- UIML is designed to manage a family of interfaces with different capabilities. This is a distinguishing characteristic compared to other markup or programming languages.
- The ability to manage a family of interfaces simplifies the task of designing several versions of a user interface to address accessibility issues.

At runtime, we can face two kinds of events: On one hand, we have events that are localized within the interface and which involve no more than simple attribute assignments, and on the other hand, events that propagate outside the interface and which involve complex computations.

In a U.I.ML skeleton, the *event* section declares the events in generic terms and resolves them to actual appliance events at runtime by using the style sheet. Localized events are largely sufficient for user interfaces used as demonstrations and prototypes.

As mwntioned above, one of the UIML's goals is to break the dependency between the interface and the backend. XXNext sentence is unclear. RephraseXXConsequently, the interface contains no calculations which are with the backend integration actions completely hidden. This is achieved through a runtime engine that monitors all of the events. Application/interface integrators map generic interface events to scripts or application methods.

One advantage of U.I.M.L is that its supports two extreme situations. In fact, the backend can appear as an invisible interface element and accept events defined in the event section. Or, interface elements can appear as objects and the backend can manipulate them just like any other software component.

However, it is necessary to design the user interface and the backend in isolation to maintain a clean separation between them. The `<logic>` section of UIML provides the mapping between the two sides. This allows interfaces (or at least portions thereof) to communicate with any application, given the appropriate event and style mappings, and vice-versa.

The learning curve associated with any new technology can and often have a profound effect upon that technology's acceptance and usefulness. In order to flatten this learning curve, UIML allows for domain-specific variants which adopt vocabularies closely aligned with specific appliances or even application domains. A variant is supported through the definition of a suitable set of XSL transformations.

In one variant of UIML known as UIML/GUI, the example in the section 5 can be expressed as follows:

```
<interface name="DSUIML Example" class="JavaAWT">
      <window name="Main" content="UIML Example">
        <menubar name="Selections">
          <menu name="FileSelection" caption="File" >
            <menuitem      name="itemF0"      caption      ="New"/
shortcut="Ctrl-N">
            <menuitem name="itemF3" caption ="Close"/ >
            <menuseparator/>
            <menuitem name="itemF4" caption ="Quit"/>
          </menu>
        </menubar>
      </window>
```

```
</interface>
```

This example is more compact and easier to read than the first one but it has as drawback to be specific to an appliance with a graphical user interface.

According to MarcAbrams and Constantinos Phanouriou, one drawback of UIML is that the language uses its own style sheets instead of CSS or XSL.

In fact according to these authors, it would be better if the W3C's style sheet standards were *meta-languages*. By this, they mean that the CSS and XSL specifications would be more powerful if the vocabulary for formatting objects and style properties were *not* part of the specification. In this way, the style sheets could be adapted to any type of device without modifying the core CSS and XSL specifications, to permit use on user interfaces available today or invented in the future.

Contrarily to the other domain-specific user interface languages (e.g., XwingML, XUL), UIML/GUI can be mapped to UIML, which can then be remapped to other appliance types (e.g., ones without graphical user interfaces).

Other drawbacks of UIML is that it does not capture context data, it is not intended to support knowledge-based system functions, does not target operation and evaluation functions, and it does not clearly separate the rendering of the interface from its definition[23].

UIML structures

Here is a skeleton of a UIML ***document***:

< ?xml version= »1.0 » ?>

< !DOCTYPE uiml PUBLIC

[23] Angel Puerta and Jacob Eisenstein, XIML: *A Universal Language for User Interfaces*.

« -//UIT//DTD UIML 2.0 Draft//EN »

« http ://uiml.org/dtds/UIML20.dtd »>

<uiml>

<head> <meta> … </meta> </head>

<peers>

<presentation> <component> … </component> </presentation>

<logic> <component> … </component> </logic>

</peers>

<template> … </template>

<interface> … </interface>

</uiml>

- Head element

The *head* element contains metadata about the current UIML document. Elements in the head element are not considered part of the interface, and have no effect on the rendering or operation of the user interface.

- Peers element

The *peers* element contains the mappings between abstractions in the interface description (i.e., interface parts, properties, events, and methods) and their actual implementations (i.e., toolkit widgets and attributes, platform events, and application logic components). This element specifies the vocabulary for toolkits to which UIML is mapped.

In other word, peers element present what widgets in the target platform and what methods or functions in scripts, programs, or objects in the application logic are associated with the user interface.

- Template element

The *template* element allows reuse of interface elements. When an element appears inside a template it can be included ("sourced") elsewhere in a UIML document.

- Interface element

The *interface* element contains the interface description

Here is a skeleton of the *interface* element:

A user interface is described by U.I.M.L with five sections: description, structure, data, style, and events.

<interface>

<structure> <part> … </part> </structure>

<style> <property> … </property> </style>

<content> <constant> … </constant> </content>

<Behaviour> <rule> … </rule> </Behaviour>

</interface>

- Description

The `<description>` section contains the individual elements that collectively form an application's user interface.

Each element has a unique name in the user interface and has a particular function. However, this section does not specify how each element is rendered or even what its function is. Elements can communicate with the backend and with other elements.

The <description> section has to contain two attribute *name* and *class*. In fact each element must have a *name* and a *class* attribute. The name is unique within the interface description. The class attribute specifies an object type; the element's "name" uniquely identifies an instance of that type.

- Structure

The `<structure>` section specifies *which* elements from the description section are present for a given appliance, and *how* the elements are organized. Generally, a structure section is a subset of elements listed in the description section. The reason of this is that some appliances (e.g., handheld appliances) cannot support all the available functionality for a given application.

The <structure> section contains two subset *element name* and *element class*. The element names are selected from the list in the description section. In many cases, it is desirable to introduce elements that are not part of the application for usability reasons. These element names have only a *class* attribute, and cannot receive or generate events.

- Data

The `<data>` section contains data or information that is presented to the user. This facilitates internationalization and creation of user interfaces with varying wording (e.g., for expert versus novice users), and facilitates accessibility. The <data> section has a subset *content name*.

- Style

The `<style>` section gives the style sheet information and data that are appliance-dependent. The style element specifies the mapping of interface parts to a vocabulary of names of user interface widgets in the platform to which the user interface will be mapped. The style element also represents the mapping of event names and classes to events on the underlying platform.

The `<style>` section has a subset *attribute class* which contains the set of interface elements that have the same class attribute. We can specify several style

attributes (e.g., colour, font size) for specific appliances. Different style sheets for different appliances are created. A special attribute called *rendering* maps all the elements with the same class attribute to a widget class in the native UI toolkit.

- Events

The `<events>` section describes the runtime interface events, which may be communicated between the interface elements and the backend. Multiple event description may be contained in the <events> section. Each event is identified with a name that is unique within the interface description. The class attribute is used to resolve events to events from the target appliance, using a style sheet. This section allows elements to synchronize with each other. Events are both appliance-dependent and application-dependent. UIML allows programmers to specify events in generic terms that avoid them rewriting the event handlers for each appliance and application combination. A generic event has a trigger and one or more of source element name, destination element name, and action. It can be triggered by the user (e.g., when the user enters some text), by the application (e.g., when the backend displays data), or by the underlying system (e.g., when a timer expires or an exception is raised).

And finally, the skeleton of the behavior element:

<behavior>

<rule>

<condition>

<event .../>

</condition>

<action>

<property ...> ... </property>

<method ...> ... </method>

<event ...> ... </event>

</action>

</rule>

...

</behavior>

The behavior element may contain one or more rules. Each of those rules may consist of a condition, triggered either when an event occurs, or when an event occurs and an attribute of the event has a specified value. When triggered, the action associated with the condition is executed. The action could change the property of interface parts, invoke a function in a script, or call a function or method in the application logic.

Chapter 3 *Abstract User Interface Markup Language*

AUIML = Abstract User Interface Markup Language

In 1998, **IBM** undertook an Advanced Technology project to develop a Device Independent Markup Language in XML. This project went through a number of phases but ended up with an XML vocabulary called **A**bstract **U**ser **I**nterface **M**arkup **L**anguage (AUIML)[24].

AUIML objectives

AUIML is an XML vocabulary, which allows the intent of an interaction with a user to be defined. In other words, one objective of AUIML is to keep "intent" separate from rendering. This can be done by specify the purpose, or intent, of an interaction rather than its appearance. It

allows designers to concentrate on semantics of the interactions without having to concern themselves with which particular device type(s) need to be supported.

AUIML is an intent-based language for defining interactions with users, focusing on semantics rather than on implementation details. The AUIML renderers already created produce visual representations of what is defined by the AUIML code.

Another AUIML objective is to allow designers to use multiple form factors. That means that single intent should run on many devices.

AUIML is a declarative language that allows programmers or designers to explain just what has to be done but not how to do it.

---

[24] http://www.google.be/search?q=cache:5XmjtqsCto0C:www.belchi.be/download/merrick.pdf+AUIML&hl=nl&ie=UTF-8 document done by Roland A. Merrick 21- 24 IBM Copyright IBM 2001

AUIML environment

AUIML allows device independent encoding of information. All the interaction information can be encoded once and subsequently rendered using "device dependent rendering" this allows users interact with the system. Xxthe end of the previous sentence is unclearXX

AUIML possibility to generate user interface

AUIML is therefore intended to be independent from the client platform on which the user interface is rendered, as well as from the implementation language and the user interface implementation technology[25].

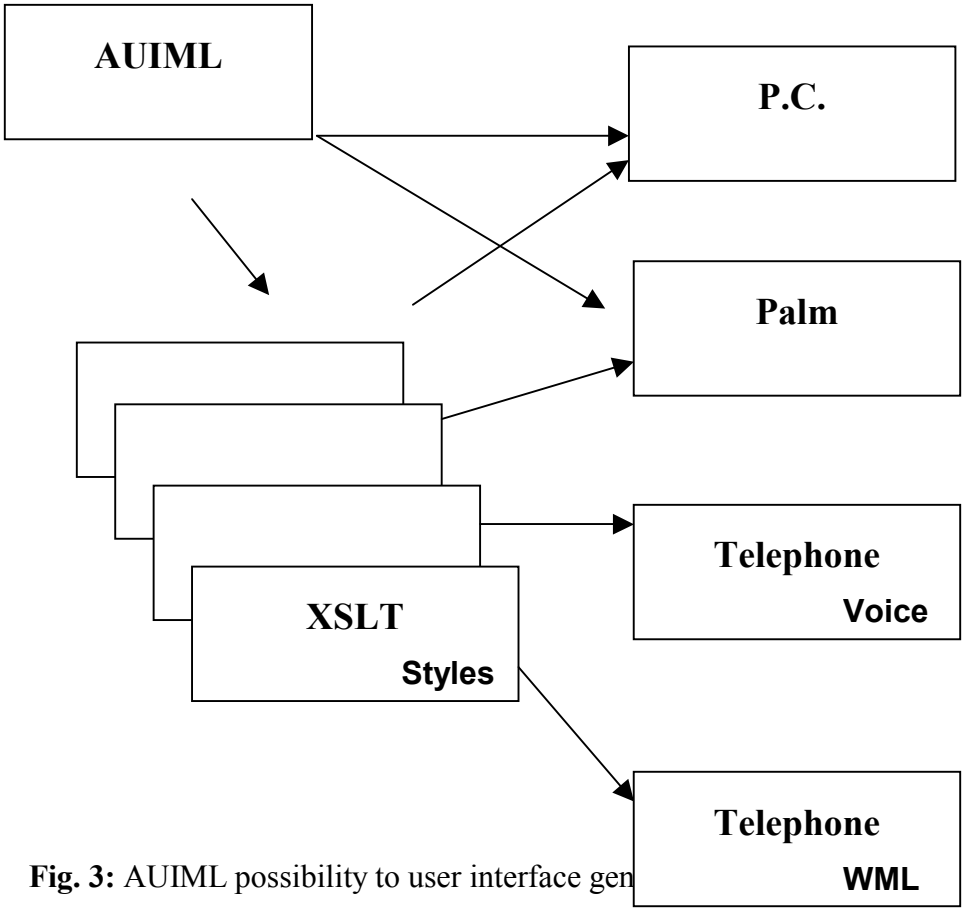


Fig. 3: AUIML possibility to user interface gen

[25] http://math.uma.pt/tupis00/submissions/azevedoroberts/azevedoroberts.html

Example created with AUIML

The example below describes an interface that asks user his name. In this interface, the user should be given the four listed choices for his title, and also entry fields for first name, initial and last name[26].

```
<GROUP NAME="PersonName">
  <CAPTION><META-TEXT>Person's complete name</META-TEXT></CAPTION>

  <CHOICE NAME="PersonTitles" SELECTION-POLICY="SINGLE">
    <CAPTION><META-TEXT>Title</META-TEXT></CAPTION>
    <HINT><META-TEXT>This is a set of valid titles you may choose from.</META-TEXT></HINT>
    <STRING NAME="MR">
      <CAPTION><META-TEXT>Mr.</META-TEXT></CAPTION>
    </STRING>

    <STRING NAME="MRS">
      <CAPTION><META-TEXT>Mrs.</META-TEXT></CAPTION>
    </STRING>

    <STRING NAME="MISS">
```

[26] http://math.uma.pt/tupis00/submissions/azevedoroberts/azevedoroberts.html

```
    <CAPTION><META-TEXT>Miss</META-TEXT></CAPTION>
   </STRING>

   <STRING NAME="MS">
    <CAPTION><META-TEXT>Ms.</META-TEXT></CAPTION>
   </STRING>

  </CHOICE>

  <STRING NAME="FirstName">
   <CAPTION><META-TEXT>First Name</META-TEXT></CAPTION>
  </STRING>

  <STRING NAME="Initial">
   <CAPTION><META-TEXT>Initial</META-TEXT></CAPTION>
  </STRING>

  <STRING NAME="LastName">
   <CAPTION><META-TEXT>Last Name</META-TEXT></CAPTION>
  </STRING>

</GROUP>
```

Xxthe end of this sentence is unclear. RephraseXX Here below, we have the interface described by the AUIML code above might look like after being implemented.

Example n°2: Xxwhy is it in red?XX

<DATA- GROUP NAME=" MySample">

<CAPTION>

<META- TEXT> Questionnaire</ META- TEXT>

<META- IMAGE SRC=" Ques. gif" />

</ CAPTION>

<GROUP NAME=" Person">

<CAPTION> Who are you?</ CAPTION>

<HINT> This section collects information about your identity. It would be a good idea if you filled in this page.</ HINT>

<GROUP NAME=" PersonName">

<CAPTION> Person Details</ CAPTION>

<HINT> Specify the person's title, first and last names, and middle initial.

If you choose a title of MRS you will be asked for a maiden name.</ HINT>

<CHOICE NAME=" PersonTitle" SELECTION- POLICY=" SINGLE"><CAPTION> Title</ CAPTION>

<HINT> This is a set of valid titles for a person.</ HINT>

<STRING NAME=" Mr">

<CAPTION>< META- TEXT>< MN> M</ MN> r.</ META- TEXT></ CAPTION>

</ STRING>

<STRING NAME=" Mrs">

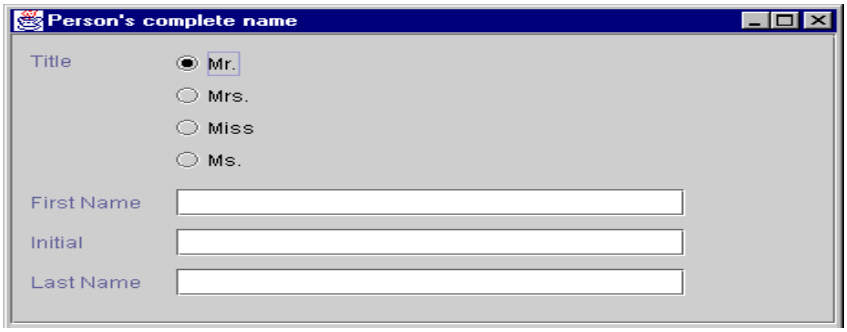
<CAPTION>< META- TEXT> M< MN> r</ MN> s.</ META- TEXT></ CAPTION>

</ STRING>

<STRING NAME=" Miss">

<CAPTION>< META- TEXT> M< MN> i</ MN> ss.</ META- TEXT></ CAPTION>

</ STRING>



<STRING NAME=" Ms">

<CAPTION>< META- TEXT> M< MN> s</ MN>.</ META- TEXT></ CAPTION>

</ STRING>

</ CHOICE>

<STRING NAME=" FirstName" OPTIMUM- LENGTH=" 12">

<CAPTION> First Name</ CAPTION>

<HINT> The person's first, or given, name</ HINT>

<VALUE> Roland</ VALUE>

</ STRING>

<STRING NAME=" Initial" MAX- LENGTH=" 1">

<CAPTION> Initial</ CAPTION>

<HINT> The person's middle initial</ HINT>

<VALUE> A</ VALUE>

</ STRING>

<STRING NAME=" LastName" MANDATORY=" TRUE" OPTIMUM- LENGTH=" 12">

<CAPTION> Last Name</ CAPTION>

<HINT> The person's last, or family, name</ HINT>

<VALUE> Merrick</ VALUE>

</ STRING>

<GROUP NAME=" mn" ITEM- NAME=" Mrs" TEST=" SELECTED" CONDITION=" TRUE"><CAPTION> Maiden Name</ CAPTION>

<STRING NAME=" MaidenName" OPTIMUM- LENGTH=" 12">

<CAPTION> Maiden Name</ CAPTION>

<HINT> Your Maiden</ HINT>

</ STRING>

<GROUP></ GROUP>

</ GROUP>

</ DATA- GROUP>

AUIML advantages and drawbacks

AUIML structure

AUIML contains three kinds of elements: simple data types, structural elements and actions.

Simple Data types can be semantics & representation, DATE, TIME, NUMBER, STRING, BOOLEAN or IMAGE, AUDIO.

Structural Elements are CHOICE, GROUP, TABLE, TREE,…

And finally, Actions can be ACTION, ACTION- GROUP…

AUIML has also some common metadata elements[27]:

<CAPTION>

This is the most concise item of descriptive information. In a Graphical User Interface this would typically be a text, which is rendered as a field prompt or caption when the parent element is displayed.

<HINT>

Provides a short description of the purpose for this data item. It should require little, if any, explicit action from the user to cause it to be rendered.

With a visual rendering this would typically be rendered in the style of Hover help. With an audio system, an unexpectedly long pause could be the trigger.

<HELP>

Provides a longer detailed description of the data item, typically a number of sentences. This would not normally be rendered unless a user explicitly requests it.

[27] http://www.google.be/search?q=cache:5XmjtqsCto0C:www.belchi.be/download/merrick.pdf+AUIML&hl=nl&ie=UTF-8 document done by Roland A. Merrick 21- 24 IBM Copyright IBM 2001

Chapter 4 *eXtensible Interface Markup Language*

XIML = eXtensible Interface Markup Language

XIML (eXtensible Interface Markup Language) is based on XML language. Its initial development has taken place at the research laboratories of RedWhale Software[28].

XIML objectives

XIML provides a way to completely describe a user interface without worrying about how it will be implemented[29]. Because XIML is a markup language, it is based on textual contentstructured by markup tags. Those tags will be given by an XML schema, which defines the language. The schema tells applications that read XIML what to expect to find in an XIML document[30].

XIML represents the attributes and the relationships of the important elements of a user interface. In other words, it enables a framework for the definition and interrelation of interaction data items. As such, XIML can provide a standard mechanism for applications and tools to interchange interaction data and to

[28] RedWhale Software is a leading provider of professional services, products, and technology infrastructure for the design and development of user-centered solutions for e-business, 277 Town and Country Village, Palo Alto, CA 94301

[29] Jacob Eisenstein, *The XIML Files, Paper 1 XIML: The eXtensible user Interface Markup Language,* November 18, 1999

[30] Jacob Eisenstein, *The XIML Files, Paper 1 XIML: The eXtensible user Interface Markup Language,* November 18, 1999

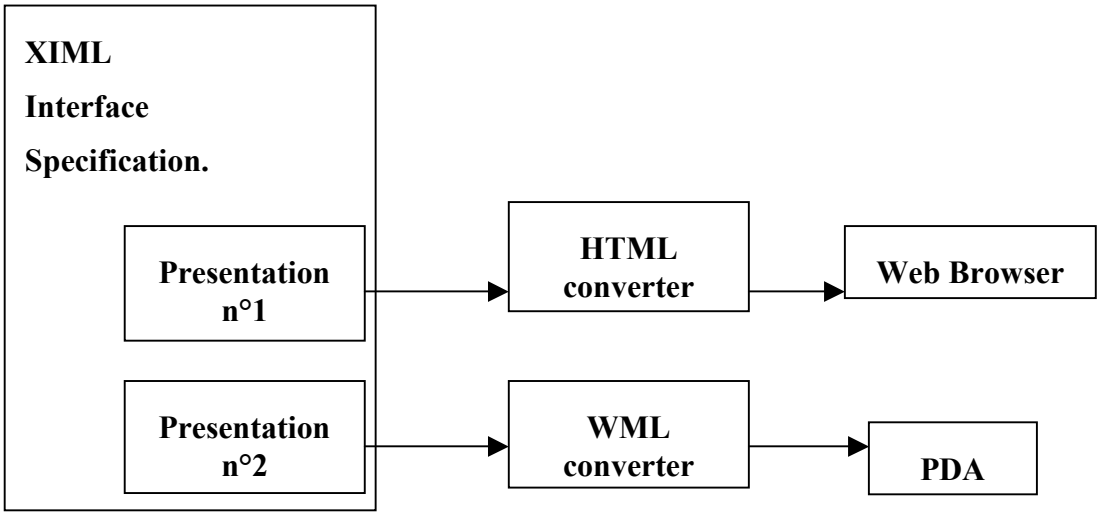
interoperate within integrated user-interface engineering processes, from design, to operation, to evaluation[31].

XIML (eXtensible Interface Markup Language) is proposed as a common representation for interaction data. According to Angel Puerta and Jacob Eisenstein, XIML fulfils the requirements essential for a language of its type:

- it supports design, operation, organization, and evaluation functions,
- it is able to relate the abstract and concrete data elements of an interface, and
- it enables knowledge-based systems to exploit the captured data.

XIML environment

XIML can be used to display a single interface definition on any number of target devices. XIML has this possibility because XIML makes a strict separation between the definition of a user interface and the *rendering* of that interface.



[31] Angel Puerta and Jacob Eisenstein, XIML: *A Universal Language for User Interfaces*.

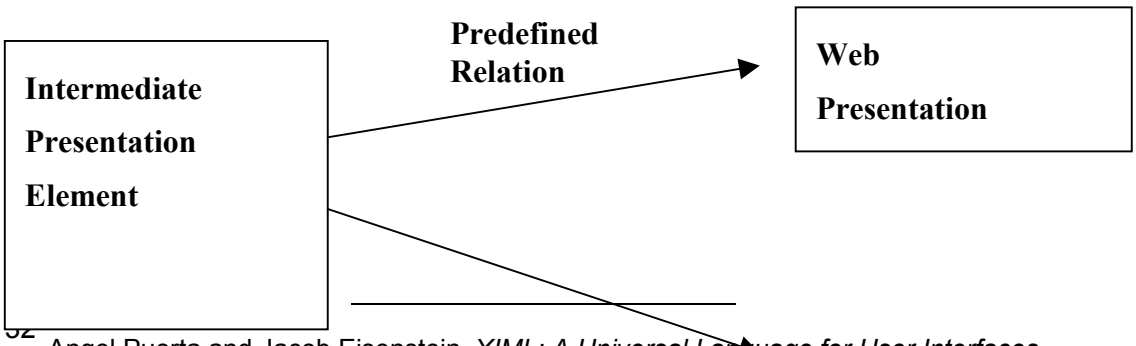
Fig. 6[32]**:** XIML provides a framework for the development of user interfaces that have multiple target displays.

Figure 6 shows an example of the XIML framework for multi-platform development for a couple of sample device targets.

Of course, the language is not restricted to those two types of devices but can theoretically support many types of stationary and mobile devices. In this example, there is a single XIML interface specification for the data to be displayed, the navigation to be followed and the user tasks to be supported. The entire specification can support multiple platforms by simply defining one presentation component per target device. This simply means determining what widgets, interactors, and controls will be used to display each data item on each of the target devices. If the interface is rendered, an XML-enabled device is able process an XIML specification directly. If the target device is not XML-enabled, a converter must be used to produce the target language.

This kind of multi-platform framework helps to save development time and ensure consistency.

In addition, XIML offers capabilities that can provide a high-degree of automation for the multi-platform UI development process. Figure 7 illustrates this automation framework.



[32] Angel Puerta and Jacob Eisenstein, *XIML: A Universal Language for User Interfaces.*

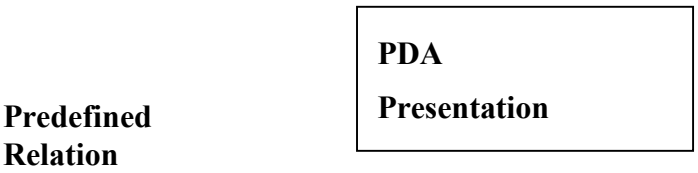


Fig. 7: Automation framework for multi-platform interface development in XIML.

The single "intermediate" presentation component allows developers to work just on one single presentation instead of creating and managing one presentation component per target device. However, this "intermediate" presentation is too inflexible for the practical use.

XIML support for generating user interfaces

There are many considerations that go into selecting an appropriate widget to use in a given instance. These considerations would include screen size, what other elements are on the screen at the same time, user preferences, and contextual issues and so on. Therefore, intelligent tools are developed to handle the task of creating and updating the relations between intermediate and device elements.

XIML provides a resource for the management of a user interface at runtime. Here below, we will exanimate the three runtime functions of XIML:

(a) Dynamic presentation reorganization,

(b) Personalization, and

(c) Distributed interface management.

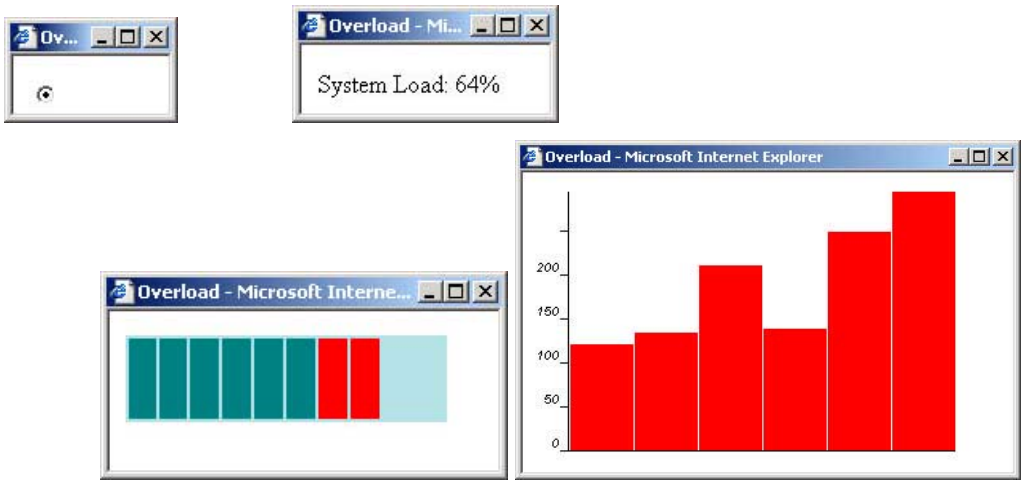


Fig. 8[33]**:** Dynamic presentation reorganization based on available display area.

Dynamic presentation re-organization

Figure 8 represents a sequence of views of a single web page that displays the system load of a server. According to the space available, the page displays different widgets. When that space is minimal, the most basic data item will be displayed. As the area increases, additional text and then a graphical view may be added. Finally, when the display space is maximized, the most sophisticated control available for that target data item will be displayed. A simple application reads the XIML specification for the interface and dynamically adjusts the presentation component of the specification according to a set of thresholds on the value of the display space available.

Fig. 9: Various widgets available for personalization in an XIML specification.

[33] Angel Puerta and Jacob Eisenstein, *XIML: A Universal Language for User Interfaces.*

Personalization

Figure 9 represents a simple example of a personalization feature. XIML specification indicates that there are a number of widgets that can be used to display data source (e.g., the system load as in the previous example). This example shows that XIML has capabilities to support personalization features and that the XIML framework can offer the value of a single repository of interaction data to support many user-interface management functions.

Distributed interface management.

According to Angel Puerta and Jacob Eisenstein, one of the drawbacks of any client-based software application is that the update of the client software is problematic since each individual client needs to be updated.

Server-side applications reduce that problem but droved the trade off that a server updates affect every user of the application at the same time even if these users don't desire the change.

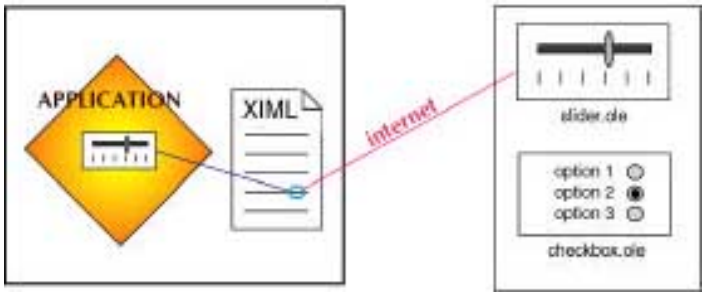


Fig.10[34]**:** XIML mechanism for distributed interface management.

As we have seen from the two previous examples, the XIML specifications can easily change the widgets that should be displayed on a page. But the framework does not make any assumptions about the server or the client: it just treats the widget as a black box that performs a function.

[34] Angel Puerta and Jacob Eisenstein, XIML: *A Universal Language for User Interfaces*.

That flexibility allows the widget to simply be available somewhere on the network, be it on a client, a peer, or a server machine. Xxthis sentence is unclear. rephraseXXThe XIML specification can be set to link to providers of the widget or it can rely on a search-and-supply application.

Task Modeling

XIML has the ability to represent abstract concepts such as user tasks, domain objects, and user profiles. This capability can be referred to as *task modeling*.

RedWhale has developed a number of model-based interface development tools that include among others, an informal user task model specification tool called U-TEL[35], and a formal interface-model development environment called MOBI-D[36].

Both of these tools have the capabilities to represent interface models, including the contextual concepts of user tasks, domain objects, and user profiles.

Those tools model a wide variety of applications such as a military-logistics management tool, a medical-data visualization application, and a supply-chain management tool...

The interface modeling language used by U-TEL and MOBI-D is a frame-based language that shares some characteristics with XIML[37].

RedWhale built a converter that can take any MOBI-D model specification and convert it into an XIML specification.

---

[35] . Tam, R.C.-M., Maulsby D., and Puerta, A. "U-TEL: A Tool for Eliciting User Task Models from Domain Experts". In Proc. *IUI98: 1998 International Conference on Intelligent User Interface*s. San Francisco, CA. ACM Press.

[36] Puerta, A.R. "A Model-Based Interface Development Environment". In *IEEE Softwar*e. 1997. pp. 40-47.

[37] Angel Puerta and Jacob Eisenstein

Reverse Engineering

As a substantial amount of interface code is HTML, the ideal situation would be that XIML could be converted into HTML.

A research group—working independently from RedWhale has successfully accomplished the reverse engineering of HTML into XIML[38].

XIML advantages and drawbacks

These are some of the main features of the language:

XIML is independent from the operating system, software development environment, and even from the programming language. Once a program is written that converts XIML to hard program code for a given platform, any XIML-based user-interface can be ported to that platform instantaneously[39]. This means that software developers will be able to write XIML code once, and that code will run on any operating system, on any device (e.g. cell phone, palm pilot, desktop computer,…) and will be able to communicate with program code written in any language[40].

It represents the concrete aspects of a user interface (such as presentation and dialog) but also its abstract aspects (such as context). It fills the gap between the design and the development of user interfaces by providing a single

[38] Vanderdonckt, J., Bouillon, L., and Souchon, N., "Flexible Reverse Engineering of Web Pages with Vaquita". In Proc. *WCRE'200: IEEE 8th Working Conference on Reverse Engineering.* Stuttgart, October 2001. IEEE Press.

[39] Jacob Eisenstein, *The XIML Files, Paper 1 XIML: The eXtensible user Interface Markup Language,* November 18, 1999

[40] Jacob Eisenstein, *The XIML Files, Paper 1 XIML: The eXtensible user Interface Markup Language,* November 18, 1999

representational framework for both processes thereby facilitating their integration and tool support.

It fills the gap between design and development of user interfaces by providing a single representational framework for both processes thereby facilitating their integration and tool support.

It provides a ready knowledge repository for runtime operations such as personalization, adaptation, context-aware and agent-based interaction.

XIML conceives the design and implementation of a user interface as a series of refinements from an abstract representation (of user context for example) to a concrete representation (of widgets and interaction techniques for example).

Figure 4 graphically summarizes the major types of requirements which, according to Angel Puerta and Jacob Eisenstein, are essential for XIML.

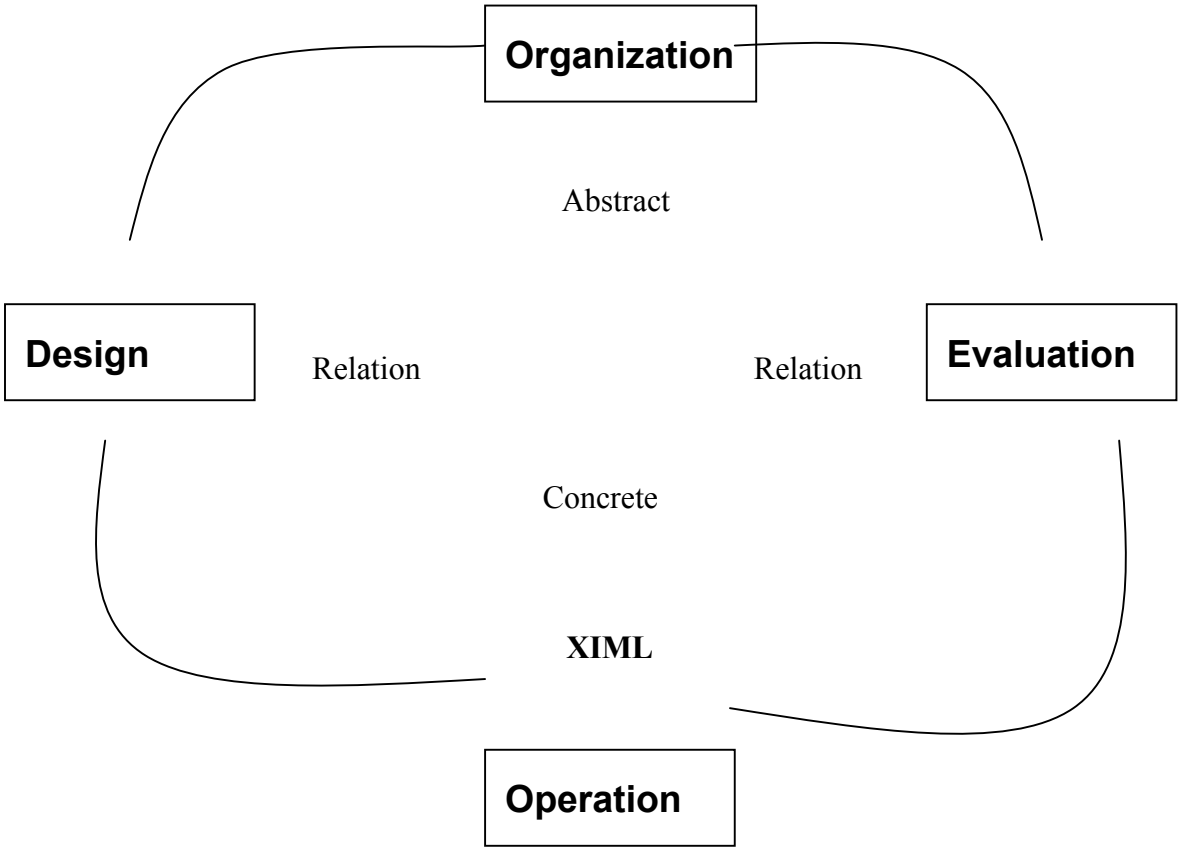


Fig. 4[41]**:** XIML represents abstract, concrete and relational interface data items. It also enables user-interface-engineering functions of design, operation, evaluation, and organization.

- **Central repository of data**. XIML has to allow a comprehensive, structured storage mechanism for interaction data, which may cover one or a collection

[41] Angel Puerta and Jacob Eisenstein, *XIML: A Universal Language for User Interfaces*.

of user interfaces. In this manner, purely organizational or knowledge management functions can be supported by XIML.

- **Comprehensive lifecycle suppor**t. XIML has to allow support functionality throughout the complete lifecycle of a user interface, which includes design, operation, and evaluation phases.

- **Abstract and concrete elements.** XIML must be able to represent not only the abstract aspects of a user interface, such as the context in which interaction takes place, but also the concrete aspects, such as the specific widgets that are to be displayed on a screen.

- **Relational support.** XIML must be able to effectively relate the various elements captured within the scope of its representation, which is particularly important in the case of relating abstract and concrete elements of interaction data. The relational capabilities of the language are what enable the development of knowledge-based support throughout the lifecycle of a user interface[42].

- **Underlying technology.** XIML adheres at two implementation requirements, which allow it to be useful within an industry-based new computing model.

[42] Puerta A. and Eisenstein, J. "Towards a General Computational Framework for Model-Based Interface Development Systems". In *Knowledge-Based System*s, Vol. 12, 1999, pp. 433-442 and. Szekely, P. et al. "Declarative Interface Models for UserInterface Construction Tools: the MASTERMIND Approach". In *Engineering for Human-Computer Interactio*n, L.J. Bass and C. Unger (eds), Chapman & Hall, London, 1995, pp 120-150.

The first implementation requirement is the use of an underlying technology that is compatible with that computing model. To full this first requirement, XIML is as we said previously based on XML language which is the representational centerpiece of the new computing model.

The second implementation requirement is that the language must not impose any particular methodologies or tools on the design, operation, and evaluation of user interfaces. And we know that XIML is able to coexist with existing methodologies and tools.

XIML structure

Figure 5 shows the basic structure of XIML.

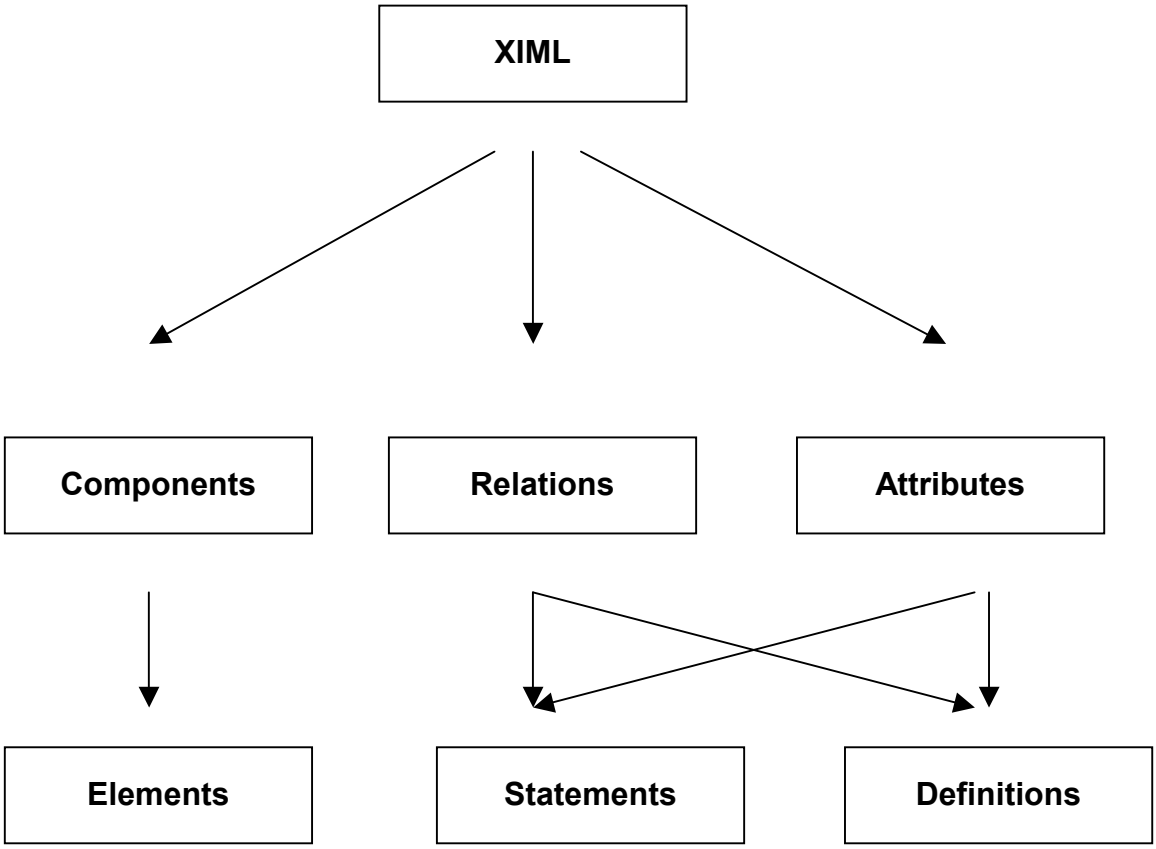


Fig. 5[43]**:** The basic representational structure of the XIML language.

Components

XIML is an organized collection of interface *elements* that are categorized into one or more major interface *component*s. 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 under each component. But this is just on the theoretical sense. In a more practical sense, however, its first version (1.0), XIML

43 Angel Puerta and Jacob Eisenstein, *XIML: A Universal Language for User Interfaces.*

predefines five basic interface components, namely task, domain, user, (which can be characterized as contextual and abstract) dialog, and presentation (which can be described as implementational and concrete).

Task.

This component captures the business process (that requires an interaction with a user) and/or user tasks that the interface supports.

It defines a hierarchical decomposition of tasks and subtasks that also defines the expected *flow* among those tasks and the *attributes* of those tasks. As example of tasks, we have “Enter Date”, “View Map”, or “Perform Contract Analysis”.

Domain.

The *domain* component is defined as an organized collection of data objects and classes of objects structured into a hierarchy. Objects are defined via attribute-value pairings.

Objects that are included in this component are restricted to those that are viewed or manipulated by a user and can be either simple or complex types.

“Date”, “Map”, and “Contract” are all examples of domain objects.

User.

The *user* component defines a hierarchy of users, which can represent a user group or an individual user. Therefore, an element of this component can be a “Professor” or can be “Professor Jean Vanderdonckt”. The characteristics of these users are defined by Attribute-value pairs. The user component of XIML captures data and features that are relevant in the functions of design, operation and evaluation.

Presentation.

The presentation component defines a hierarchy of interaction elements that comprises the concrete objects that communicate with users in an interface. As examples of these, we have a window, a push button, a slider, or a complex widget such as an ActiveX control to visualize stock data. The logic and operation of an interaction element are separated from its definition. This feature allows the rendering of a specific interaction element to be left entirely to the corresponding target display system.

Dialog.

The *dialog* component defines a structured collection of elements that determine the interaction actions available to the users of an interface.

As example of interaction actions, we can quote a “Click”, a “Voice response”, and a “Gesture”. This component also specifies the flow among the interaction actions that constitute the allowable *navigation* of the user interface.

Relations

A *relation* in XIML represents a definition or a statement that links any two or more XIML elements either within one component, or across components. Because XIML captures relations in an explicit manner, it creates a body of knowledge that can support design, operation, and evaluation functions for user interfaces. The runtime manipulation of those relations constitutes the *operation* of the user interface.

Attributes

Attributes are features of elements that can be assigned a *value*, which can be one of a basic set of data types or an instance of another existing element. Multiple values are allowed as well as enumerations and ranges.

Chapter 5 *Seescoa XML*

Seescoa XML

SEESCOA is a project that started in October 1999 and has to be finished in September 2003. This project implies an important collaboration of several groups of researchers from different universities: Distrinet from KULeuven, Programming Lab from the VUB and RUG from Paris.

Xxplease rephraseXX Each research group has its self work and each is work is a part of a puzzle[44].

SEESCOA is the acronym for 'Software Engineering for Embedded Systems using a Component-Oriented Approach'.

The main objective of SEESCOA (IWT-project, STWW-programme, Technology and Economics) is to adapt the software engineering technologies to the needs of embedded software[45]. To achieve this objective is to make reuse of components in different compositions possible. The components should be applicable in a broad range of situations and have a generic character.

The definition of a SEESCOA component[46]:

A SEESCOA component is an object offering a coherent behaviour. Some other component can access this behaviour by asynchronously sending messages to the component. To do so, both components need a port.

[44] Kris Luyten and Karin Coninx, *Het SEESCOA project; jouw user interface, altijd en overall,* 17 januari 2002

[45] D. Urting, Y. Berbers, S. Van Baelen, T. Holvoet, Y. Vandewoude and P. Rigole, *A Tool for Component Based Design of Embedded Software,* Department of Computer Science, Catholic University of Leuven, Belgium.

[46] D. Urting, Y. Berbers, S. Van Baelen, T. Holvoet, Y. Vandewoude and P. Rigole, *A Tool for Component Based Design of Embedded Software,* Department of Computer Science, Catholic University of Leuven, Belgium.

One of the results of the SEESCOA project is a common software platform, using components on a Java Virtual Machine[47].

Seescoa XML objectives

Seescoa XML ==> runtime conversion from the user interfaces to user interface description language, XML.

Kris Luyten and Karin Coninx propose a runtime user interface description language, which can cope with constraints found in embedded systems and mobile computing devices. According to these authors, XML seems to be a suitable tool to do this, when combined with Java.

The goal is to use a user interface description language suitable for a wide range of embedded and mobile computing devices, working in heterogeneous environments, and utilize it at runtime.

Serializing a user interface allows it to interact with a particular service, and migrating the parts of interest of that serialized user interface allows it to interact with the user.

For the serialization of the user interface, XML is used as a user interface description language.

The authors have chosen XML for describing a user interface, because this language offers the following properties:

[47] Kris Luyten, Chris Vandervelpen, Karin Coninx, *Adaptable User Interfaces in Component Based*

Development for Embedded Systems, Expertisecentrum Digitale Media – Limburgs Universitair Centrum, Diepenbeek -Belgium

Platform independent: XML is platform independent in the same way that Java is platform independent. If the system contains an XML parser then the XML description can be used. Otherwise, one can write is own parser.

Declarative: By using XSL[48], XML can be declarative. XSL specifies a stylesheet which is applied to an XML document, and offers powerful transformations on the XML document[49].

Consistent: XML can be consistent by using DTD[50], which specifies a set of rules for the XML file to follow.

Unconventional I/O: XML can easily describe unconventional I/O such as WML, SpeechML, VoxML,…

Rapid prototyping: The use of stylesheets allows seeing the results immediately in a browser.

Constraint definition: XML can contain constraint definitions for the form of the XML itself but also for external resources.

Easily extensible: XML is a metalanguage. So, it is naturally extensible.

Reusability: An existing piece of XML fits easily another one.

Because XML is a markup language, XML has a really simple grammar and structure.

Moreover, an XML description is easily convertible into different kinds of output presentation using XSLT[51] that allows XML to be converted into HTML+CSS[52]

[48] eXtensible Stylesheet Language

[49] Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

[50] Document Type Definitions.

[51] eXtensible Stylesheet Language Transformation

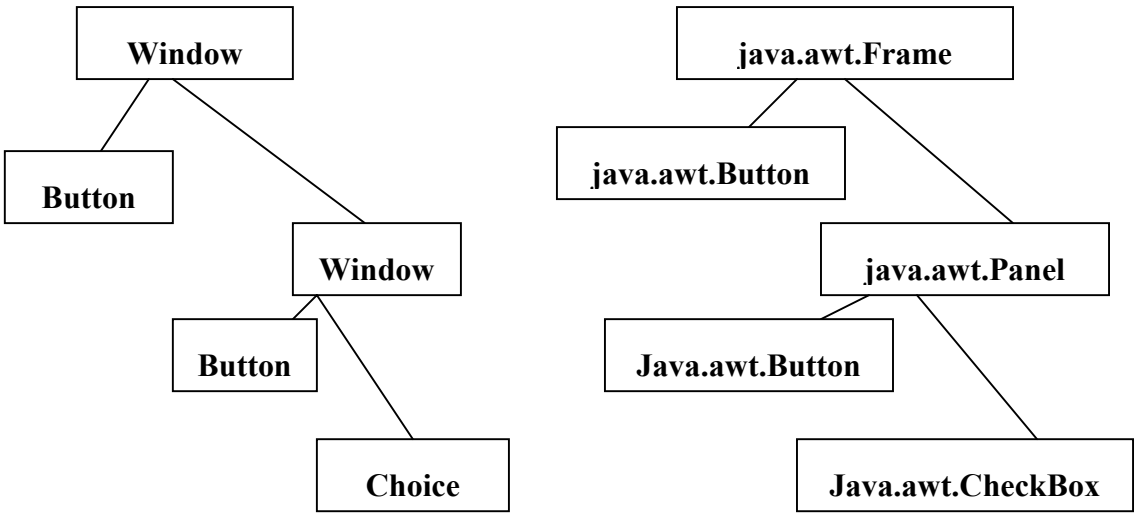
[52] Cascading Stylesheets : a stylesheet for an HTML document.

for desktop browsers, VoxML for speech driven input, or into WML for mobile phones[53].

Seescoa XML environment

Kris Luyten and Karin Coninx want to propose an architecture for runtime serialization of Java user interfaces into an XML description, inspired by migratory applications[54] and remote user interface software environments[55]. This way, it allows "downloading" a user interface together with constraints and necessary transformations.

How the user interface will be presented in the XML file, including the constraint definitions?



[53] Didier Martin, Mark Birbeck, Michael Kay, Brian Loesgen, Jon Pinnock, Steven Livingstone, Peter Strak, Kevin Williams, Richard Anderson, Stephen Mohr, David Baliles, Bruce Peat, and Nikola Ozu. *Professional XML,* Wrox Press, 2000.

[54] K. Bharat and L. Cardelli, *Migratory applications. In Eighth ACM Symposium on User Interface Software and Technology,* pages 133-142, 1995.

[55] J. Landay and T. Kaufmann, *User Interface Language.* World Wide Web, http://www:sop.inria.fr/koala/kuil/, 2000.

Fig. 11[56]**:** On the left contextual representation (AIO), on the right the java.awt classes used to represent the presentation (CIO).

Figure 11 shows that a user interface can be structured as a tree, which is the basic structure of an XML file. The main window, sliders, etc. are laid out. In the main window, the user interface may build blocks like buttons but also other windows that contain building blocks, which in turn can be windows.

XX the following sentence must be rephrasedXXIn this stage, it is advisable to make an abstraction to doing this we have different means, namely, abstract interaction objects (AIO)[57] and concrete interaction objects (CIO)[58], which are represented in the figure 11, and also abstract widgets[59]. These concepts allow to abstract the user interface in a form that is independent from the target platform.

If the runtime layout management is added to take into account constraints defined by the environment, the presentation of the AIO has to be changed dynamically.

[56] Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

[57] AIO represents an abstract interface widget, independent of any platform.

[58] CIO is a possible « implementation » for such AIO.

[59] Abstract widgets are abstract platform independent representations of platform dependent widgets.

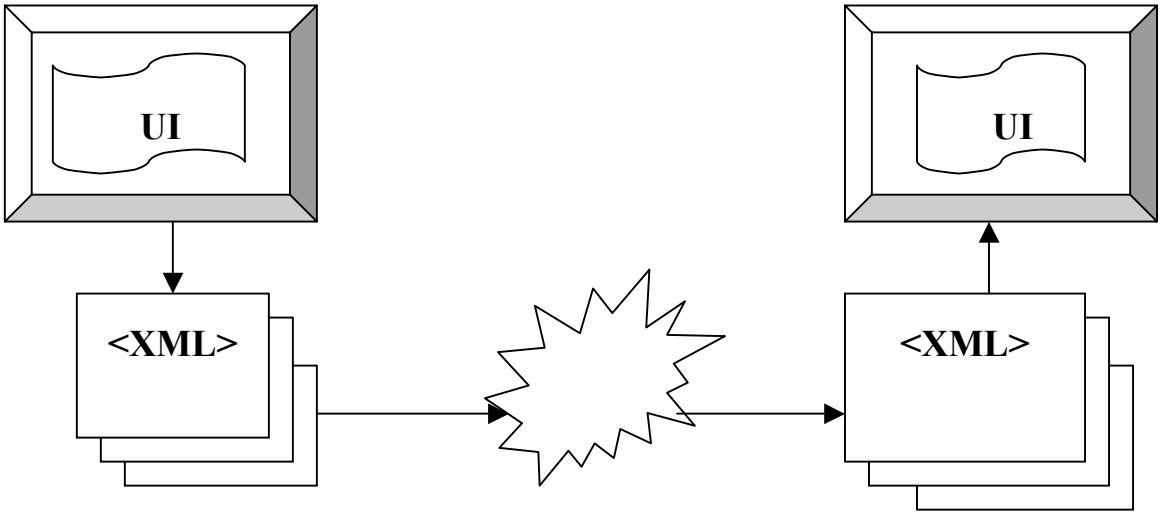


Fig. 13: Downloading a Remote User Interface.

When the user interface has produced its XML description, the description has to move to another device, where it can be "deserialized" into a user interface for the target device. This process is presented in Figure 13.

This deserialization involves mapping the platform independent AIOs onto platform specific CIOs.

When the user interface is set up on the target platform, it has to be fully functional. For this end, we need a kind of remote event handling, like RPC. Java is a suitable remote user-interface software environment using RPC. Java makes it possible to download the code (e.g., applets), make our own Classloaders and use Remote Method Invocation[60]. The last requirements are a Java Virtual Machine on the device and network capabilities.

[60] Sheng Liang and Gilad Bracha, *Dynamic class loading in the Java virtual machine.* In ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'98), pages36-44, 1998 and Frank Sommers, *Object mobility in the Jini environment,* World Wide Web, http://www.javaworld.com/javaworld/jw-01-2001/jw-0105-jiniology.html, January 2001and Sun Microsystems.*Java Remote Method Invocation.* World Wide Web, http://java.sun.com/products/jdk/rmi/, 1997

The profiling use allows the machine (embedded system, mobile computing device, desktop computer, airplane, cell phone, etc.) to know who is operating it.

The advantage of this approach is that when the system knows who is the user of the user interface, it knows which functionality the particular user is interested in. Xxthe next 2 sentences are not understandableXXHowever, profiling can reduce the user interface to be migrated. But the use of an automatic layout agent this information can also be used to present the user interface in a way it is most suitable to the user.

Kris Luyten and Karin Coninx propose XSL for describing the user profile. The XSLT allows filtering out the interested parts of the user. When XSLT processes a XML structure, the result is an another XML structure which shows only the contents of the parts that interest the user. An important part of this conversion is to only select the relevant paths in the XML user description. This is done by XPath: an implementation to locate particular branches in the logical tree XML presents[61].

Seescoa XML possibility to generate user interface

The user interface toolkits in Java are AWT[62] and Swing[63].

Kris Luyten and Karin Coninx have implemented a conversion mechanism for Java interfaces. They have used the AWT toolkit of Java because Swing is still too heavy for most systems with low resources (limited storage capacity and RAM).

[61] Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

[62] Abstract Windowing Toolkit.

[63] Swing is a subset of the Java Foundation Classes.

By using the Java Reflection Mechanism, all of the inspector methods starting with the prefix *get* are retrieved, and their return values are stored in the XML tree[64].

The advantage of this approach is that not only a user interface can be serialized, but also all kinds of objects whose state can be retrieved using their inspectors can be serialized.

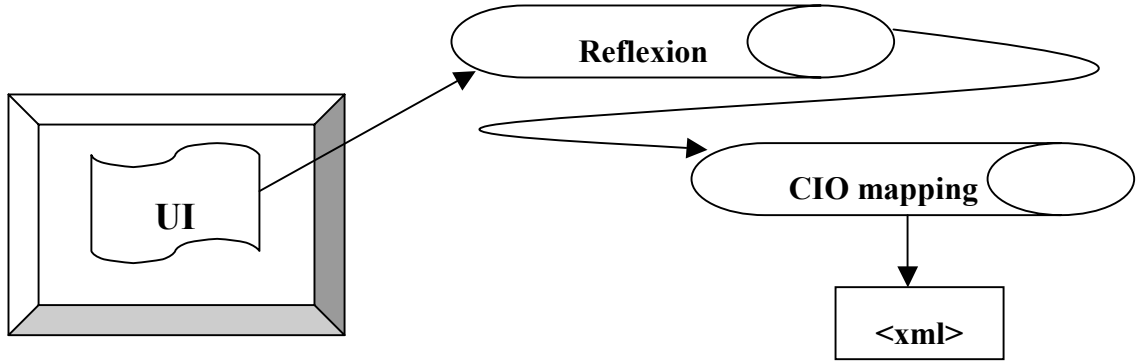


Fig. 12: Serializing a user interface into its XML description.

Figure 12 shows a simple process by which a collection of possible general CIOs is defined, and the matched AWT classes for these CIOs are mapped onto the defined CIO tag names.

By doing so, the XML description preserves the state and stays general enough to convert an XML description into another toolkit.

To allow optimization of the dataflow from source to target, the size of the XML description has to be reduced. So, redundant or unnecessary data have not to be stored in the user interface description. But the method of serializing a Java user

[64] Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

interface presents a disadvantage. It needs to make a decision of which data is relevant for the user interface serialization, and which data is not.

This method presents another disadvantage, the lack of support for other programming languages[65]. XXSentence not understandableXXConversion from XML to an actual user interface, this can be solved easily depending on the maturity of the user toolkit, which is targeted. At the opposite of Java, most other languages have no class reflection mechanism, and are much harder to interrogate for their structures at runtime. It is thereforehard to convert a user interface into its XML description without dedicated data to ease this conversion. This implies the need for a mechanism for XML conversion, inserted into the toolkits built on these programming languages[66].

Example created with Seescoa XML

Here below an example of a runtime XML conversion of the following user interface[67]:

<Application NAME=”test.testUI”>

<Property NAME=”title”>this is an UI2XMLtest</Property>

[65] Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

[66] Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

[67] Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

```
<Property NAME=”name”>main-window</Property>
<Properties>
  <Property NAME=”copyButton”>
    <AIO CLASS=”Button”>
        <Properties>
            <Property NAME=”name”>button0</Property>
            <Property NAME=”actionCommand”>Copy</Property>
                <Property NAME=”label”>Copy</Property>
        </Properties>
    </AIO>
  </Property>
  <Property NAME=”pasteButton”>
    <AIO CLASS=”Button”>
      <Properties>
            <Property NAME=”name”>button1</Property>
            <Property NAME=”actionCommand”>Paste</Property>
            <Property NAME=”label”>Paste</Property>
      </Properties>
    </AIO>
  </Property>
  <Property NAME=”textField”>
   <AIO CLASS=3TextField”>
```

```
    <Properties>
            <Property NAME=”name”>texfield0</Property>
            <Property NAME=”selectionEnd”>0</Property>
            <Property NAME=”columns”>20</Property>
            <Property NAME=”selectedText”></Property>
….
            <Property NAME=”extraLabel”>
   <AIO CLASS=”Label”>
     <Properties>
            <Property NAME=”name”>label0</Property>
            <Property NAME=”alignment”>0</Property>
            <Property NAME=”text”>extra label</Property>
     </Properties>
  </AIO>
 </Property>
</Properties>

</Application>
```

The command used when the buttons are pushed is also included in the XML description.

Seescoa XML advantages and drawbacks

Seesco XML is a runtime user interface language, which can cope with constraints found in embedded systems and mobile computing devices.

Seescoa XML can be defined as a combination of XML and Java.

An automatic conversion is needed when XML is used to generate a user interface description at runtime. This can be done by using abstract interactors.

The conversion from the user interface description in XML into a system dependent user interface results in a consistent user interface, subject to the constraints of the current platform.

A profile, which can be generated automatically, indicates the user's interests in some particular functionality.

Advantages for developers of embedded systems[68] include:

Flexible changing the UI can be done by another renderer component or letting components provide another UI description.

Reusable providing a high level *description* of the UI allows easier reusability of previously designed UIs in contrast with hard coded Uis.

Adaptable by abstracting the UI, device constraints can be taken into account when rendering the concrete UI.

[68] Kris Luyten, Chris Vandervelpen, Karin Coninx, *Adaptable User Interfaces in Component Based*

Development for Embedded Systems, Expertisecentrum Digitale Media – Limburgs Universitair Centrum, Diepenbeek -Belgium

Chapter 6 *The TERESA XML language*

The DTD of TERESA

The TERESA tool has been extensively described in the CAMELEON Deliverable 2.1[69]. In this section we introduce its XML language. It is composed of two parts: the XML description of the ConcurTaskTrees notation and the XML version of a language for describing abstract user interfaces. The XML version of ConcurTaskTrees was introduced in 1999 and accompanied the evolution of the language. It was the first XML language for task models. It is widely known and publicly available. So, its description will not be duplicated here. The XML TERESA description for abstract user interfaces is recent. Here we introduce it through its DTD. The language can be used for specifying how the various Abstract Interaction Objects[70] (AIO) composing the UI are organised, together with the specification of the dialogue of the UI. For clarity, we first consider and comment each part of the DTD and then we provide the whole DTD at the end of this section.

The root of the document is the `interface` object. An `interface` is composed of one or more objects of type `presentation`:

```
<!ELEMENT interface (presentation+)>
```

Each `presentation` has two parts: the first part (`connection`) is related to the dynamic behaviour of the presentation, the second one (`structure`) is related to the static arrangement of the elements (namely, AIOs) composing the presentation itself. For each presentation, we can have zero, one or more objects of type `connection`, with each `connection` mainly identifying the

[69] Mori, G., Paganelli, L., Paternò, F., Santoro, C., *Tools for Model-Based Design of Multi-Context Applications, Deliverable 2.1, CAMELEON Project,* August 2002

[70] Calvary, G., Coutaz, J., Thevenin, D., Bouillon, L., Florins, M., Limbourg, Q., Souchon, N., Vanderdonckt, J., Marucci, L., Paternò, F., Santoro, C., *Reference Framework, Models and Criteria for Usability in Multi-Context Applications, Deliverable 1.1, CAMELEON Project,* August 2002

presentation element whose activation allows the interface to move to a different presentation. The `structure` part mainly describes the static arrangement of the different objects within the presentation itself:

```
<!ELEMENT presentation (connection*, structure)>
<!ATTLIST  presentation
             name ID #REQUIRED>
```

Each `connection` has two attributes: an `interaction_aio_id`, which defines the interaction object whose performance triggers the next presentation which is identified in turn by the `presentation_name` attribute:

```
<!ELEMENT      connection EMPTY>
<!ATTLIST  connection
           interaction_aio_id IDREF #REQUIRED
          presentation_name IDREF #REQUIRED>
```

Each `structure` element can be either an elementary abstract interaction object `(aio)` or a composition of them `(aio_composition)` through the various operators defined in the abstract language:

```
<!ELEMENT      structure (aio | aio_composition)>
```

Each `aio_composition` is the composition of one operator defined in the language (`grouping, ordering, relation, hierarchy`) with a number of expressions which can be, in turn, either elementary interaction objects or complex expression of such elementary objects. Note that the `second_expression` tag is only used when the concerned operator is the relation: in this case we have to model a N:1 relation, so the `second_expression` tag is used to identify precisely the element which the other N elements are in relation with.

```
<!ELEMENT aio_composition (operator, first_expression+,
second_expression?)>
<!ELEMENT      operator EMPTY>
<!ATTLIST      operator
          name     (grouping | ordering | relation |
hierarchy)  #REQUIRED >
<!ELEMENT      first_expression          (aio          |
aio_composition)>
<!ELEMENT      second_expression         (aio          |
aio_composition)>
```

Each `aio` can be either an interaction object or an application object. In any case, it is univocally identified within the presentation by means of the `id` attribute.

```
<!ELEMENT  aio (interaction_aio | application_aio)>
<!ATTLIST  aio
          id ID #REQUIRED>
```

Each `interaction_aio` defines an abstract interaction object which implies an interaction (see the `category` attribute) between the user and the application. It can be of different types depending on the type of task supported: `selection_aio` if the object allows to select between a set of elements; `edit_aio`, if it allows to edit an object, `control_aio` if it allows to trigger an event within the user interface:

```
<!ELEMENT  interaction_aio (selection_aio  | edit_aio |
control_aio )>
<!ATTLIST  interaction_aio
          category  CDATA #FIXED  "interaction">
```

Each selection_aio can be of two different types, depending on the number of elements that will be selected, one (single_choice_aio) or many (multiple_choice_aio). If the element allows a single choice, there are a number of options depending on the cardinality (low/medium/high) of the set which the element will be selected from:

```
<!ELEMENT       selection_aio    (single_choice_aio    |
multiple_choice_aio)>
<!ATTLIST  selection_aio
             type CDATA #FIXED   "selection">

<!ELEMENT  single_choice_aio (singlechoice_low_card_aio
|             singlechoice_medium_card_aio             |
singlechoice_high_card_aio  )>
<!ATTLIST  single_choice_aio
             type CDATA #FIXED   "single choice" >

<!ELEMENT  singlechoice_low_card_aio EMPTY>
<!ATTLIST  singlechoice_low_card_aio
             type CDATA #FIXED    "low card">

<!ELEMENT  singlechoice_medium_card_aio EMPTY >
<!ATTLIST  singlechoice_medium_card_aio
             type CDATA #FIXED    "medium card">

<!ELEMENT  singlechoice_high_card_aio EMPTY>
<!ATTLIST  singlechoice_high_card_aio
             type CDATA #FIXED   "high card">
```

The same holds in case of multiple choice. If the element allows a multiple choice, depending on the cardinality (low/medium/high) of the set which the element will be selected from, various types of objects will be specified:

```
<!ELEMENT                              multiple_choice_aio
(multiplechoice_low_card_aio                              |
```

```
multiplechoice_medium_card_aio                                    |
multiplechoice_high_card_aio  )>
<!ATTLIST  multiple_choice_aio
             type CDATA #FIXED   "multiple choice">

<!ELEMENT  multiplechoice_low_card_aio EMPTY>
<!ATTLIST  multiplechoice_low_card_aio
             type CDATA #FIXED    "low card">

<!ELEMENT  multiplechoice_medium_card_aio EMPTY >
<!ATTLIST  multiplechoice_medium_card_aio
             type CDATA #FIXED    "medium card">

<!ELEMENT  multiplechoice_high_card_aio EMPTY>
<!ATTLIST  multiplechoice_high_card_aio
             type CDATA #FIXED   "high card">
```

Each `edit_aio` can be of two different types, depending on the type of object that will be edited, text (`text_edit_aio`) or graphic (`graphic_edit_aio`):

```
<!ELEMENT  edit_aio (text_edit_aio | graphic_edit_aio)>
<!ATTLIST  edit_aio
             type CDATA #FIXED  "edit">

<!ELEMENT  text_edit_aio EMPTY>
<!ATTLIST  text_edit_aio
             object CDATA #FIXED   "alphanumeric">

<!ELEMENT  graphic_edit_aio EMPTY>
<!ATTLIST  graphic_edit_aio
             object CDATA #FIXED   "graphic">
```

The last type of interaction object is a `control_aio`, which is mainly associated with interaction objects allowing to trigger a particular event within the user interface:

```
<!ELEMENT  control_aio EMPTY>
<!ATTLIST  control_aio
             type CDATA #FIXED  "control">
```

Differently from an `interaction_aio`, an `application_aio` defines an abstract application object which implies an action only from the application (see the `category` attribute):

```
<!ELEMENT   application_aio (text_aio | graphic_aio |
image_aio | feedback_aio)>
<!ATTLIST  application_aio
              category CDATA #FIXED   "application">
```

Each `application_aio` can be of different types (`text_aio`, `graphic_aio`, `image_aio`, `feedback_aio`) depending on the type of output the application provides to the user: a textual output, a graphical output, an image, or a feedback about a particular state of the user interface.

```
<!ELEMENT  text_aio EMPTY>
<!ATTLIST  text_aio
             object CDATA #FIXED   "text">


<!ELEMENT  graphic_aio EMPTY>
<!ATTLIST  graphic_aio
             object CDATA #FIXED   "graphicalObj">
```

```
<!ELEMENT  image_aio EMPTY>
<!ATTLIST  image_aio
             object CDATA #FIXED   "image">


<!ELEMENT  feedback_aio EMPTY>
<!ATTLIST  feedback_aio
             object CDATA #FIXED   "feedback">
```

The whole DTD of TERESA is the following:

```
********************** ************************************


<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT interface (presentation+)>


<!ELEMENT presentation (connection*, structure)>
<!ATTLIST  presentation
             name ID #REQUIRED>


<!ELEMENT      connection EMPTY>
<!ATTLIST  connection
           interaction_aio_id IDREF #REQUIRED
          presentation_name IDREF #REQUIRED>


<!ELEMENT      structure (aio | aio_composition)>

<!ELEMENT aio_composition (operator, first_expression+,
second_expression?)>
```

```
<!ELEMENT      operator EMPTY>
<!ATTLIST      operator
          name     (grouping | ordering | relation |
hierarchy)  #REQUIRED >


<!ELEMENT      first_expression          (aio          |
aio_composition)>

<!ELEMENT      second_expression         (aio          |
aio_composition)>


<!ELEMENT  aio (interaction_aio | application_aio)>
<!ATTLIST  aio
          id ID #REQUIRED>


<!ELEMENT  interaction_aio (selection_aio  | edit_aio |
control_aio )>
<!ATTLIST  interaction_aio
          category  CDATA #FIXED  "interaction">


<!ELEMENT       selection_aio    (single_choice_aio    |
multiple_choice_aio)>
<!ATTLIST  selection_aio
             type CDATA #FIXED   "selection">


<!ELEMENT  single_choice_aio (singlechoice_low_card_aio
|            singlechoice_medium_card_aio            |
singlechoice_high_card_aio  )>
<!ATTLIST  single_choice_aio
             type CDATA #FIXED   "single choice" >
```

```
<!ELEMENT  singlechoice_low_card_aio EMPTY>
<!ATTLIST  singlechoice_low_card_aio
             type CDATA #FIXED    "low card">

<!ELEMENT  singlechoice_medium_card_aio EMPTY >
<!ATTLIST  singlechoice_medium_card_aio
             type CDATA #FIXED    "medium card">

<!ELEMENT  singlechoice_high_card_aio EMPTY>
<!ATTLIST  singlechoice_high_card_aio
             type CDATA #FIXED   "high card">



<!ELEMENT                              multiple_choice_aio
(multiplechoice_low_card_aio                              |
multiplechoice_medium_card_aio                            |
multiplechoice_high_card_aio  )>
<!ATTLIST  multiple_choice_aio
             type CDATA #FIXED   "multiple choice">

<!ELEMENT  multiplechoice_low_card_aio EMPTY>
<!ATTLIST  multiplechoice_low_card_aio
             type CDATA #FIXED    "low card">

<!ELEMENT  multiplechoice_medium_card_aio EMPTY >
<!ATTLIST  multiplechoice_medium_card_aio
             type CDATA #FIXED    "medium card">

<!ELEMENT  multiplechoice_high_card_aio EMPTY>
<!ATTLIST  multiplechoice_high_card_aio
             type CDATA #FIXED   "high card">

<!ELEMENT  edit_aio (text_edit_aio | graphic_edit_aio)>
<!ATTLIST  edit_aio
```

```
              type CDATA #FIXED  "edit">

<!ELEMENT  text_edit_aio EMPTY>
<!ATTLIST  text_edit_aio
              object CDATA #FIXED   "alphanumeric">

<!ELEMENT  graphic_edit_aio EMPTY>
<!ATTLIST  graphic_edit_aio
              object CDATA #FIXED   "graphic">

<!ELEMENT  control_aio EMPTY>
<!ATTLIST  control_aio
              type CDATA #FIXED  "control">


<!ELEMENT   application_aio (text_aio | graphic_aio |
image_aio | feedback_aio)>
<!ATTLIST  application_aio
               category CDATA #FIXED   "application">

<!ELEMENT  text_aio EMPTY>
<!ATTLIST  text_aio
              object CDATA #FIXED   "text">


<!ELEMENT  graphic_aio EMPTY>
<!ATTLIST  graphic_aio
              object CDATA #FIXED   "graphicalObj">


<!ELEMENT  image_aio EMPTY>
<!ATTLIST  image_aio
              object CDATA #FIXED   "image">
```

```
<!ELEMENT  feedback_aio EMPTY>
<!ATTLIST  feedback_aio
             object CDATA #FIXED   "feedback">
```

Chapter 7 *Main features of considered languages: a summary*

User Interface Markup Language

UIML is the result of starting with a clean sheet of paper and creating language for describing user interfaces in a highly platform-independent manner[71]. UIML is a declarative; XML-compliant language intended to describe a user interface into one single language and to produce it into as several platforms as supported. UIML seeks to hide the diversity of the platforms and of the development languages- Swing, Waba, HTML, WML, VoiceXML, etc.- It is a simple language for which there are already compilators for Java/JFC, PalmPilot, HTML, VoiceXML and WML.

But with UIML the specification of the presentation is platform dependent. If the designer wants a user interface for WML and for Java/JFC, he has to produce two distinct specifications: the first one will use the WML tags, and the second one will use AWT/Swing with the Java layout manager.

To solve this problem, Abrams and Phanariou 1999 make double propositions:

An abstract system of the interactors based on the AIO concept. This one is a part of UIML 2.0 specifications with the tags combination <peers> and <presentation> but this notion of AIO remains at the toolkit level as in cross-platform tools[72].

A management presentation system but this system is still under work[73].

[71] Jean Vanderdonckt, *The XIML Files, Towards a Universal Language for User Interface Modeling,* February 8, 2000

[72] Jean Vanderdonckt, *The XIML Files, Towards a Universal Language for User Interface Modeling,* February 8, 2000

UIML covers two particular models only (i.e., presentation and dialogue) without allowing the user to extend them.

Table 2: UIML sum up[74]:

Actors (design phase)	The designer. He specifies the interface by hand. Only the execution of the adaptation is automatic because it is done by each UIML compilator to Java, WML, …
Development support	Specification tool using a language which allows the design of ill-assorted platform. But it is still limited to make multi-platform (need to make in general a UIML specification by target platform.)
Adapted components	System point of view : AIO. User point view : Interface (the presentation)
Target	Platform (available interactors)
Executable user interface	Precalculed user interface (one interface for each target)
Actors (runtime phase)	N/A because the interfaces are precalculated.

[73] Mir Farooq Ali's thesis at Virginia Tech: http://vto-pus.cs.vt.edu/~abrams/uiml/.

[74] David Thevenin, *Adaptation en Interaction Homme-Machine*, 21 septembre 2001

Abstract User Interface Markup Language

AUIML objective is the automatic multi-platform code generation (dynamic HTML, Java Swing, Palm-Pilot). This language solves the main problem of UIML because the specification is independent of the target platform. Indeed, at any time the designer has to use a JButton or a "<href…> in his AUIML description.

AUIML allows the user interface description in terms of:

- manipulated elements (data, structures and complex data such as images or sound)
- interaction element: simple types combining structure and functionalities (choice, group, table, tree);
- actions: allows to describe a micro-dialogue to manage events between the interface end the data.

The other advantage of AUIML, in comparison with UIML is the explicit use navigation manager. This manager builds the structure of the user interface, determines the choice of the navigation systems in function to the data structures. The drawback is that this tool is not yet ended and is little in publicity.

Table 3: AUIML sum up[75]:

Actors (design phase)	The designer : he specifies the interface by hand. System : the execution of the adaptation is

[75] David Thevenin, *Adaptation en Interaction Homme-Machine*, 21 septembre 2001

	automatic because it is done by each AUIML compilator.
Development support	Specification language to design mutli-platform
Adapted components	System point of view : AIO and possibility of CD with the navigation manager. User point of view : Interface (presentation and navigation)
Target	Platform (available interactors, displays)
Executable user interface	User interface precalculated (one interface by target)
Actors (runtime)	N/A because the user interfaces are precalculated.

eXtensible Interface Markup Language

XIML is a description language of graphical interfaces of different platform. The XIML objective is to aid the development of multi-platforms by decreasing the development time and by maximizing the consistence between the platforms. In this purpose, it covers all the conception cycle of an interface by allowing the description:

- of the abstracts aspects of an interface (tasks, domain concept, user);
- of the concretes aspects of an interface (presentation, dialogue);

- of the relations between the described elements (an Y presentation "presents" the X data).

For a better diffusion and application, this language is opened and extensible :

It is in the same type of XML and it doesn't impose any specification tool. It is possible to use the used tools and to make a link between the data described by the tool and the data described in XIML.

It leans on an "matasyntax" to facilitate its extension (it can be considered as an meta-language). So, it will be easy to connect it to tools or to generate new user interfaces.

An XIML description looks like a centralized repository of data. This description is composed of three bases: the component, the attribute, the relation. A component is an element describing the tasks, the domain concepts, the users, the presentations or the dialogue.

An attribute belongs to a component. From a set of attributes, it is possible to describe the features or the proprieties of a component. A relation describes the link between one or several components.

The multi-platform conception cover is assured by the description of a platform presentation within the XIML description. An intermediary description would be possible (platform independent) that will be put in relation by dependent presentation. The tools will assure the description of the intermediary presentation and the generation of the dependent presentations. It is the purpose of tool like MOBI-D.

Finally, XIML should offer resources to the dynamic management of interfaces by facilitating the reorganization of the presentation, the users' preference to the presentation, the presentation distribution (a browser sends to the clients new presentations).

The definition of this language is on way to be finish. Any tool is proposed to manage the different conception levels and the generation of user interfaces.

Contrarily to UIML, XIML allows for the formal specification of abstract information about the user-interface such as task hierarchies, domain information, and user modelling.

Table 4: XIML sum up[76]:

Actors (conception phase)	The designer (will depend on the tools)
Development support	Specification language : modeling (should allow : user, tasks, domain concept, presentation, dialogue). It is a meta-language, so, it is open to several mosels.
Adapted components	System point of view : will depend on the genaration tools. User point of view : will depend on the generation tools.
Target	Initially multi-platform but open (Note: no environment modelling expected at this moment)
Executable user interface	Open : will depend on the tools
Actors (runtime phase)	Target objective (but will depend on the tools): user (by preference) and system.

[76] David Thevenin, *Adaptation en Interaction Homme-Machine*, 21 septembre 2001

Seescoa XML

The Seescoa XML purpose is to serve like a language description not only graphical user interfaces but also its runtime state. For this, it has to be possible to migrate those user interfaces dynamically between platforms (Luyten and Coninx 2001). Doing this Seescoa XML leans on the reflection Java mechanism that allows analyzing dynamically the graphical user interface and generates the corresponding XML description.

This description can be then sent to another platform to be deployed as a graphical user interface equivalent to the original one and on the same state. The user interfaces can be then cloned by this system.

Currently, (Luyten and Coninx 2001) the system generates just a adaptation to the graphical toolkit by the AIO and CIO concepts.

Table 5: Seescoa XML sum up[77]:

Actors (design phase)	N/A
Development support	Mechanisms kit
Adapted components	System point of view: AIO, CIO. User point of view : Interface
Target	Platform
User interface executable	Dynamic
Actors (runtime phase)	System

[77] David Thevenin, *Adaptation en Interaction Homme-Machine*, 21 septembre 2001

Table 6: Comparison of languages for context-sensitive user interfaces.

Language	Models	Output	Design Knowledge	Methodology	XML compliance	Supporting tool	Scope
UIML	Dialog Presentation Domain (parts)	Fully functional interface	None	None	Yes	Code generator	None
AUIML	Dialog Presentation	Fully functional interface	None	None	Yes	Rendering engine	None
XIML	Task Domain User Dialog Presentation Platform Design	Fully functional interface	Design guidelines Heuristics	Model-based approach	Yes	Rendering engine Code generator Editor	None
TERESA XML	Task Abstract User Interface	Interface in XHTML or XHTML Mobile Profile	Design guidelines Heuristics	Model-based approach	yes	Rendering engine Task Model Editor and Analysis	

TERESA XML

It has been introduced to support the transformation performed by the tool. At this time it is possible to transform task models represented in ConcurTaskTrees into abstract user interfaces and abstract interfaces into concrete interfaces in XHTML or XHTML Mobile profile. Xxit would be nice to have a table sum up as for the other languagesXX

Chapter 8. Other User interface specification languages

Other user interface specification languages exist than the ones which have been previously examined. We include some of them for the sake of completeness. However, the analysis is not performed with the same level of details as either the language is only partially relevant to the Cameleon project or is very similar to a previously analyzed one.

8.1 SISL: Several Interfaces, Single Logic

SISL is being used to prototype a new generation of call processing services for a Lucent Technologies switching product. SISL is developed by Microsoft Inc.

8.1.1 SISL objectives

SISL is an architecture and domain-specific language for designing and implementing interactive services with multiple user interfaces.

8.1.2 SISL structure

The key principle that underlines SISL is that all user interfaces to a service can share the same service logic. SISL provides a clean separation between the service logic and the software for a variety of interfaces, including Java applets, HTML pages, speech-based natural language dialogue, and telephone-based voice access[78]. SISL uses an event-based model of services that allows service providers to support interchangeable user interfaces (or add new ones) to a single consistent source of service logic and data. By allowing the implementation of the service logic and user interfaces to proceed independently, SISL modularizes the development process. SISL approach is composed by two parts:

- a standard language independent architecture and

[78] T. Ball, Ch. Colby, P. Danielsen, L. Jategaonkar Jagadeesan, R. Jagadeesan, K. Läufer, P. Mataga, K. Rehor, "*Sisl: Several Interfaces, Single Logic",* Microsoft Research, One Microsoft Way and Bell Laboratories, Lucent Technologies and Dept. of Mathematical and Computer Sciences, Loyola University Chicago, January 6, 2000

- *reactive constraint graphs*, a novel domain-specific language (DSL) for designing and implementing interactive services. Programs written in the DSL are used as components in the architecture. The DSL is based on an analysis of the features required of a service logic that is shared across many different user interfaces, including speech-based natural language interfaces. DSL permits the description of service logics that can accept partial and incomplete information from the user interface, and allows user interfaces to deliver events to the service logic in any order.

SISL is implemented as a library on top of Triveni, a process algebraic API for reactive programming[79]. Because Triveni itself has been implemented as a library in Java, SISL can be used with any (Personal) Java implementation. Moreover, SISL can easily be used by application programmers because SISL has a XML front-end to describe reactive constraint graphs in a markup language. The SISL implementation currently supports applet-based interfaces, HTML interfaces using the Java Servlet API, speech-based natural language interfaces using the Java Speech API, and telephone-based voice access via VoiceXML[80] and the TelePortal platform[81].

[79] Colby, C., Jagadeesan, L. J., Jagadeesan, R., L¨aufer, K., and Puchol, C, Objects and concurrency in Triveni: A telecommunication case study in Java. In Proceedings of the 4th Conference on Object-Oriented Technologies and Systems (COOTS-98), pages 133–148, USENIX Association, Santa Fe, New Mexico, 1998

[80] http://www.voicexml.org/

[81] Atkins, D. L., Ball, T., Benedikt, M., Cox, K., Ladd, D., Mataga, P., Puchol, C., Ramming, J., Rehor, K., and Tuckey, "*Integrated web and telephone service creation"*, Bell Labs Technical Journal, 2(1):19–35, 1997

8.1.3 SISL environment

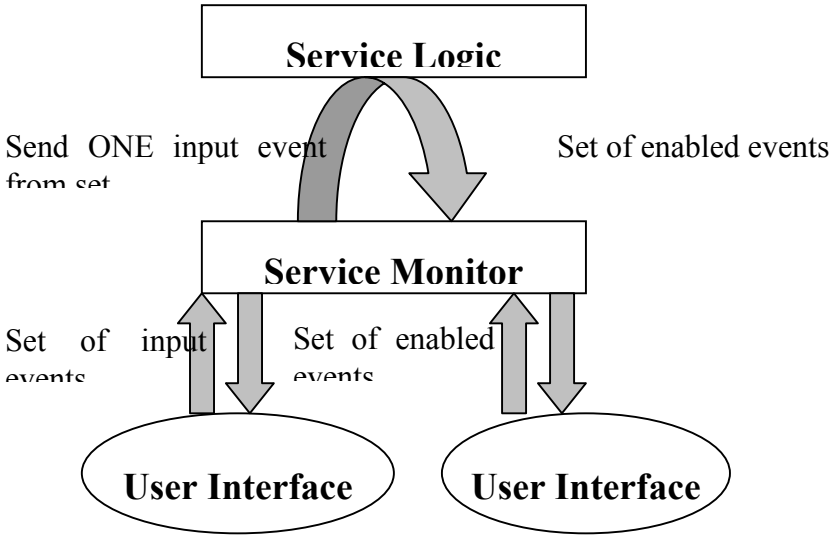


Fig.21[82]**:** The SISL Environment.

The figure 21 shows that all the communications between the service logic and its multiple user interfaces passes through the service monitor, and occurs via events. Note that there is one instance of the service logic for each user interacting with the service. The service monitor has the task of mediation the communication between the service logic and the user interfaces, on an application-specific basis. But more especially, the service monitor has the responsibility to pass events from the user interfaces to the service logic, in a priority order that is tailored to the application. The tasks of the user interface are to prompt and collect events from the user and dispatch these to the service monitor. The interactions between the service monitor and the service logic is made in rounds. In each round, the service

[82] Schema based on the schema in the document of T. Ball, Ch. Colby, P. Danielsen, L. Jategaonkar Jagadeesan, R. Jagadeesan, K. Läufer, P. Mataga, K. Rehor, "*Sisl: Several Interfaces, Single Logic",* Microsoft Research, One Microsoft Way and Bell Laboratories, Lucent Technologies and Dept. of Mathematical and Computer Sciences, Loyola University Chicago, January 6, 2000.

monitor selects an individual event from the input set in priority order and sends it to the service logic. In its turn, the service logic processes the event and performs its associated computations, and reports its new enabled set of events to the service monitor. If events in the input set match the enabled set of events, the service monitor selects one in priority order and the process is repeated. If no event in the input set matches the enabled set of events, the service monitor reports the currently enabled set of events of the service logic to the user interface. Based on these enabled events, the user interface prompts the user and collects the next set of events.

There are three kinds of events: *prompt* events, *up* events, and *notify* events[83]:

- **Prompt events:** indicate to the user interface what information to communicate to the user, and what information the service is ready to accept. There are three kinds of prompt events, namely *prompt choice* events which are disjunctive choices currently enabled in the service logic, *prompt req* events which are the events currently required by the service logic and finally *prompt opt* events which are events enabled in the service logic for which the user may correct previously given information.
- **Up events:** correspond to earlier points in the service logic to which the user may go back. This allows the user to abort any transaction and go back up to the main menu.
- **Notify events:** are simply notifications that the user interface should give the user, for example, that a transaction has completed successfully or that information provided by the user was incorrect or inconsistent.

[83] T. Ball, Ch. Colby, P. Danielsen, L. Jategaonkar Jagadeesan, R. Jagadeesan, K. Läufer, P. Mataga, K. Rehor, "*Sisl: Several Interfaces, Single Logic",* Microsoft Research, One Microsoft Way and Bell Laboratories, Lucent Technologies and Dept. of Mathematical and Computer Sciences, Loyola University Chicago, January 6, 2000

8.1.4 SISL advantages and drawbacks

SISL supports interfaces like including web, automatic speech recognition, and text-based interfaces. But all interfaces have to share the same service logic. Currently, the main limit of SISL is that it doesn't work in collaborative applications, in which users may interact with the system through a variety of devices including personal digital assistants, cellular telephones, and traditional desktop devices.

8.2 XISL: Extensible Interaction-Sheet Language

VoiceXML enables to describe speech grammar, dialogue control, and some other elements that are required for web services by telephone; however, its target modalities are limited only to speech and DTMF. Moreover, when system developers need additional modalities, they can hardly change the language specification. VoiceXML deals with difficulty with multimodalities.

8.2.1 XISL objectives:

To solve the problem mentioned above, a multimodal interaction description language XISL is developed. XISL remodels VoiceXML to meet multimodal interaction requirements. XISL allows[84]:

1. an expansion of modalities available without changing its specification,
2. Reuse of both XML contents and XISL interactions independently.

8.2.2 XISL example

```
<?xml  version="1.0"  encoding="Shift-JIS"?>
<!DOCTYPE xisl  SYSTEM  "xisl.dtd">
<xisl version="1.0">
      <head>.........</head>
      <body>
      <dialogue  id = "WAP_phone">
      <exchange>
```

[84] http://www.ipsj.or.jp/members/SIGNotes/Eng/21/2001/038/article008.html

```
<operation  comb = "alt">
<input  type="dtmf"
event="push" target="/d_gram.gxml"
match="/d_gram/goods_select" return="number">
<param name="mode">digit</param>
</input>
<input  type="speech"  event="recognize"
target="/gram.gxml" match="/gram/goods_select"
return="goods_name">
<param name="mode">data</param>
</input>
< /operation >
< action  comb = "seq" >
<output  type="speech"  event="play"/>
<param name="speaker">nancy</param>
<param name="speech_text">
You selected No. One.
< /output >
< call  dialogue_name="2"/ >
< /action >
```

8.2.3 XISL environment

The figure 23 represents the main components that compose the XISL environment.

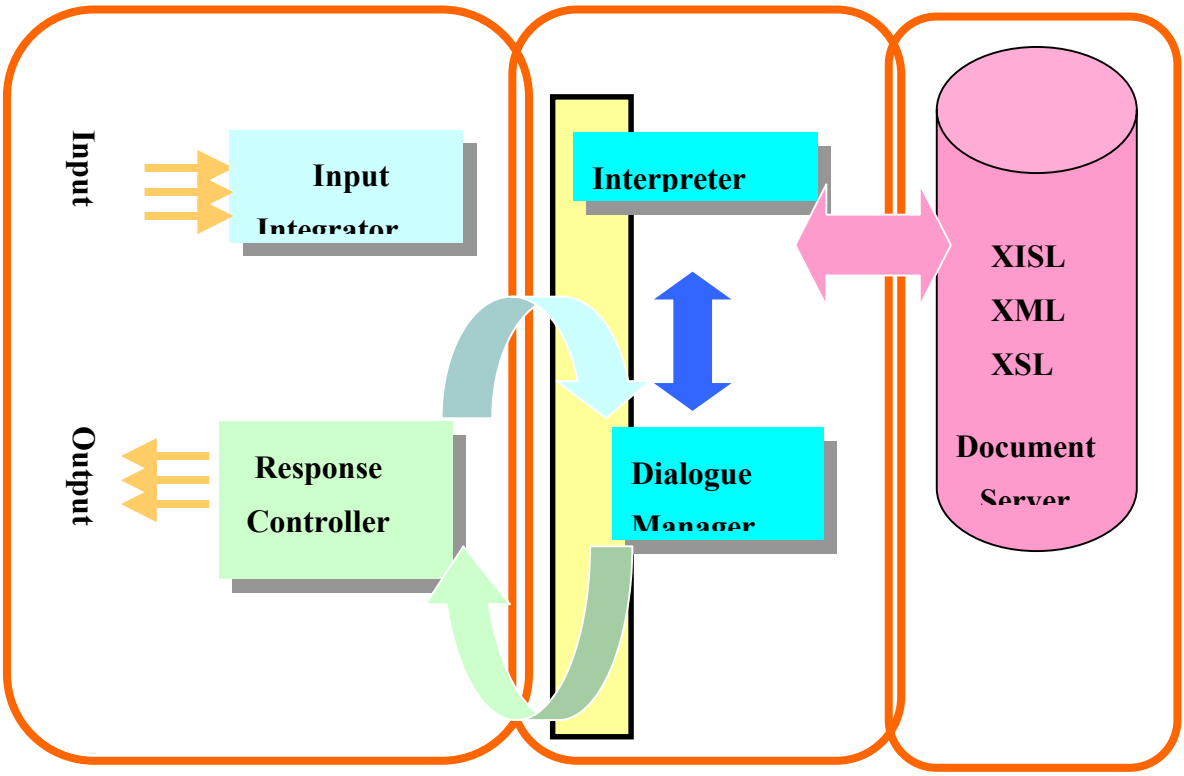


Fig.23[85]**:** XISL environment.

8.2.4 XISL advantage and drawback.

As already mentioned the XISL advantage is that it remodels VoiceXML to meet multimodal interaction requirements. The main drawback of XISL is that it is just applied to VoiceXML and not yet to all kind of user interfaces.

[85] http://www.vox.tutkie.tut.ac.jp/XISL/W3C-Mar01-02.ppt

8.2.5 XISL structure

The figure 22 shows the fact that multimodal interaction description language has no modality-specific tags that enable XISL device-independent.

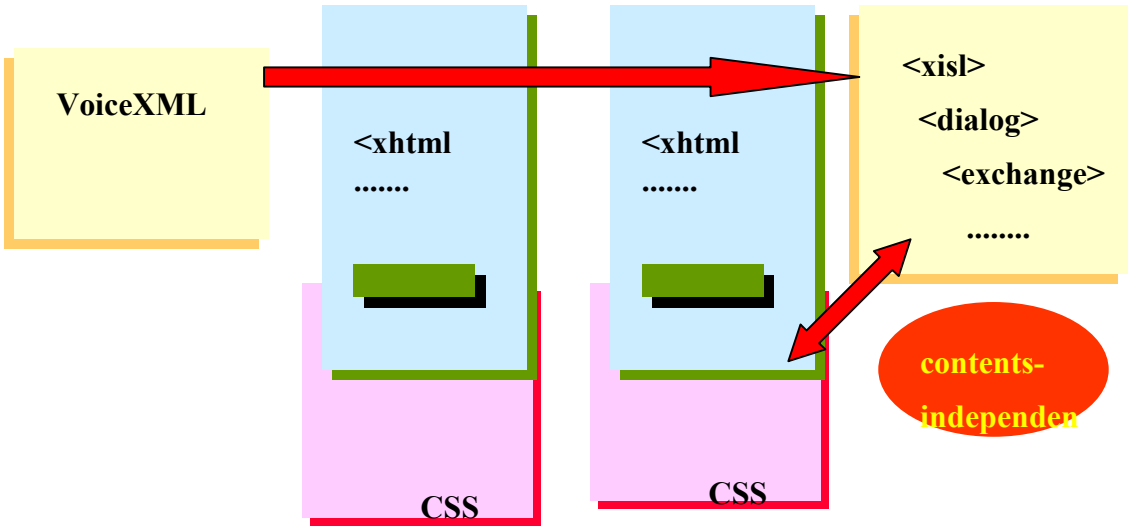


Fig.22[86]**:** eXtensible Interaction Sheet Language structure.

```
<input type="speech" event="recog" target="grammar.gxml"
       match="/grammar/goods" return="goods_name">
       <param name="mode">data</param>
</input>
<input type="dtmf" event="push" target="grammar.gxml"
       match="/grammar/goods_select"return="goods_name">
       <param name="mode">digit</param>
</input>
```

8.3 XUL = XML-based User Interface Language

XUL pronounced "zuul" and is the acronym for "XML-based User Interface Language". XUL is an application of XML. Actually, it is just XML with specific meaning defined for a few element types, and into which HTML can be scattered.

[86] http://www.vox.tutkie.tut.ac.jp/XISL/W3C-Mar01-02.ppt

8.3.1 XUL objectives

Mozilla[87] has configurable, downloadable chrome[88], meaning that the arrangement and even presence or absence of controls in the main window is not hardwired into the application, but loaded from a separate UI description. In fact, most of Mozilla's windows (and dialogs) will be described using this mechanism. These user interface descriptions are built in XUL. Window chrome is displayed and managed by the same layout engine that manages HTML content in the browser. Mozilla has then developed the XUL to use it to describe the layout of most windows in the Mozilla browser, including and especially the main, browser window. The Mozilla's intention is to build cross-platform applications like browsers and mail clients from a set of tools designed for that purpose. So, their intention is not to implement a generic cross-platform application framework.

8.3.2 XUL environment

The XUL file can work with a number of additional files to create a single window. Collectively, these files are referred to as a package, and include such things as the style sheet for the XUL, a DTD[89] that contains entity declarations for

---

[87] Mozilla is an open-source web browser, designed for standards compliance, performance and portability. Mozilla coordinates the development and testing of the browser by providing discussion forums, software engineering tools, releases and bug tracking.

[88] The chrome directory is structured as follows:

- content XUL files describing the UI, and JS files
- skin CSS files
- locale directories containing localized strings
- Chrome:// URLs have the following ``magic'' properties...
- chrome://content/chatzilla resolves to <chrome-directory>/packages/chatzilla/chatzilla/content/chatzilla.xul
- chrome://skin/chatzilla resolves to <chrome-directory>/packages/chatzilla/chatzilla/skin/chatzilla.css
- chrome://locale/chatzilla/chatzilla.dtd resolves to chrome-directory>/packages/chatzilla/chatzilla/locale/<current-locale>/chatzilla.dtd
- Chrome needs to be installed before chrome:// urls will work

[89] Document Type Declaration

localizable strings in the interface, and additional XUL that may overlay the basic XUL file. Menu items and other commands issued in one XUL file can display other XUL files[90].

8.3.3 Example created by XUL[91]

Below is a complete sample XUL document that describes a window with a menu bar and an HTML content area. The menu bar has a menu, File, with a single menu item that dumps "Hello world!" to the debug consoles when selected.

```
<?xml version="1.0"?>
      <?xml-stylesheet href="chrome://global/skin/xul.css"
      type="text/css"?>
      <!DOCTYPE window>
      <window id="main-window"
      xmlns:html="http://www.w3.org/1999/xhtml
      "xmlns="http://www.mozilla.org/keymaster/
      gatekeeper/there.is.only.xul">
      <menubar>
      <menu name="File">
      <menuitem name="Hello World!"
      onclick="dump('Hello world!\n');"/>
      </menu>
      </menubar>
      <html:iframeid="content-frame"
      src="contentframe.html" flex="100%"/>
      </window>
```

8.3.4 XUL advantages and drawbacks

Some of the advantages of writing in XUL are

- XUL does not require C++ skills.
- A complete XUL user interface can be served over the web.

XUL has not been built to be a language that generates generic cross-platform application framework. This constitutes its main limit compared to the four previous languages analyzed in detail in chapter 2 to 5. XUL has its focus on

[90] http://www.mozilla.org/xpfe/xulref/

[91] http://www.mozilla.org/xpfe/xulref/

window-based graphical user interfaces by XML. Its disadvantage is the limitation to such graphical user interfaces. It is not applicable to interfaces of small mobile devices. There is no abstraction of interaction functionality available. The device-independent specification of user interfaces is not planned.

8.3.5 XUL structure

XUL is a language for describing window layout. The task of writing a XUL window description is basically the same as the task of writing an HTML content description, with these exceptions: the syntax is XML (not that different from HTML 4), and there are some elements unique to XUL. These elements are widgets and certain infrastructure associated with the behaviour of the window[92]. Most of the details of writing a XUL document are identical to those for writing an XML document. XUL is an XML-based interface definition language. XUL adheres to the W3C XML 1.0 specification and follows the syntax rules defined there. The four cardinal XUL syntax rules are as follows[93]:

- All events and attributes must be written in lowercase
- All strings must be double quoted
- Every XUL widget must use close tags (either <tag></tag> or <tag/>) to be well-formed
- All attributes must have a value

Generally if one or another element has been improperly marked up an XUL file will not display at all. <Window> is the root element in a XUL file and it defines also the top level of the interface. There must be one and only one <window> element in a XUL file. The entire XUL interface is defined within this root element[94]. Generally, some combination of menu bars, toolbars and content area

[92] http://www.mozilla.org/xpfe/xulref/

[93] http://www.mozilla.org/xpfe/xulref/

[94] http://www.mozilla.org/xpfe/xulref/

composes a XUL interface. A typical XUL file might define a very simple window with the following basic structure[95]:

```
<window>
      <menubar>
            <menu label="File" />
            <menu label="Edit" />
      </menubar>
      <box>
            <browser />
      </box>
    </window>
```

Each of the children of <window> (i.e., menubar and box) may themselves contain child elements. XUL provides for the absolute positioning and nesting of elements. In this way the interface is built up from the basic window[96]. A list of XUL widget groups follow[97]:

- **Window:** holds all other elements
- **Box:** describes positioning of contained elements
- **Menus and menu bars:** are used to build menus. On the Mac, XUL menus are hoisted into the system menu bar. Linux and Windows display menus in the window.
- **Toolboxes** contain other elements in a collapsible box and **toolbars** are used to build button bars
- **Tab widget**
- **Checkbox**

[95] http://www.mozilla.org/xpfe/xulref/

[96] http://www.mozilla.org/xpfe/xulref/

[97] The reader who is interested in a more detailed description of XUL widget can find it on http://www.mozilla.org/xpfe/xulref/

- **Titled buttons**
- **Scrollbar**
- **Splitte**r creates a drag-able splitter between two UI elements
- **Progressmeter**

8.4 TADEUS-XML

TADEUS-XML is an approach to integrate task and object knowledge into the development process and its underlying representations with a special focus on mobile devices.

8.4.1 TADEUS-XML objectives

TADEUS-XML presents a concept of device independent user interface design based on the XML-technology. TADEUS-XML is based on user, task and business-object models[98]. It allows the computer-based development of interactive systems.

8.4.2 TASEUS-XML structure

The three models (user model, task model and business-object model) are the basis for the development of the application logic and the user interface design. Below, the figure 24 represents the main relations between different models.

[98] Müller Andreas, Forbrig Peter, Cap Clemens, "Model-Based User Interface Design Using Markup Concepts", University of Rostock, Department of Computer Science, Germany

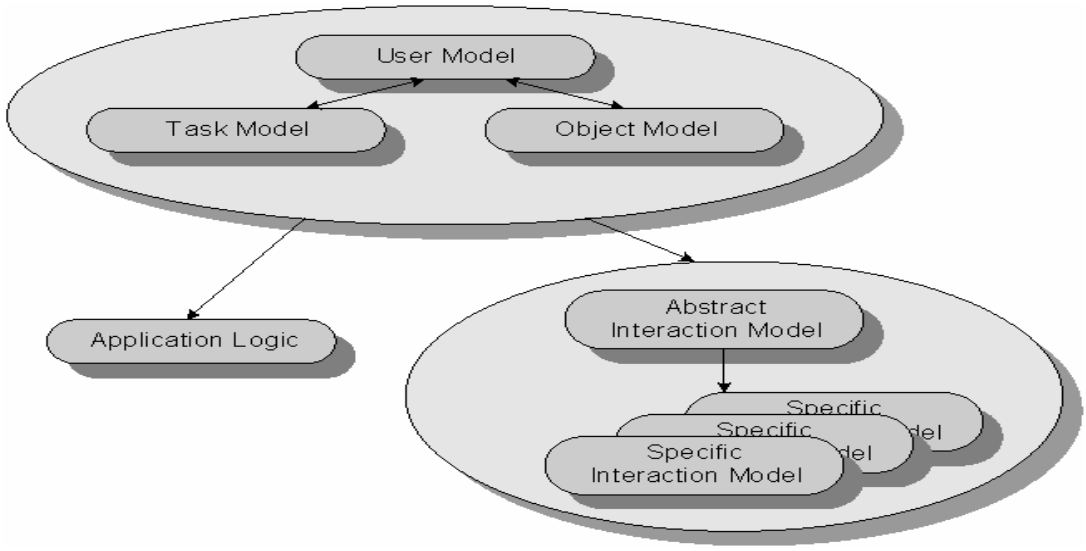


Fig.24[99]**:** Model-Based User Interface Design

TADEUS-XML approach allows a reuse of the designed models apart from final representations. To have a flexible development process from an abstract interaction model to a specific representation, the XML technology[100] is used for representation and transformation features.

8.4.3 TADEUS-XML possibility to create user interface

A user interface is separated into a model component which describes the feature of the user interface on an abstract level (it is also called abstract interaction model) and a presentation component which specifies the user-interface objects with their representations. The abstract interaction model is transformed to be mapped to the specific interaction model during the development. TADEUS-XML uses a term representation of both models. The transformation process is described by attribute grammars, which are different for the different platforms. The

[99] Müller Andreas, Forbrig Peter, Cap Clemens, "Model-Based User Interface Design Using Markup Concepts", University of Rostock, Department of Computer Science, Germany

[100] Goldfarb, C.F., Prescod, P.: XML-Handbook, Prentice Hall, 1999

development of such grammars for different mobile devices represents a real time consuming. It would be good to have more support for the transformation process[101]. The XML technology[102] offers a general support. It allows the description of the abstract interaction model, the description of specific characteristics of different devices and the specification of the transformation process. On the next page the figure 25 gives a description of a model of the transformation process forming an abstract interaction model to a specific user interface.

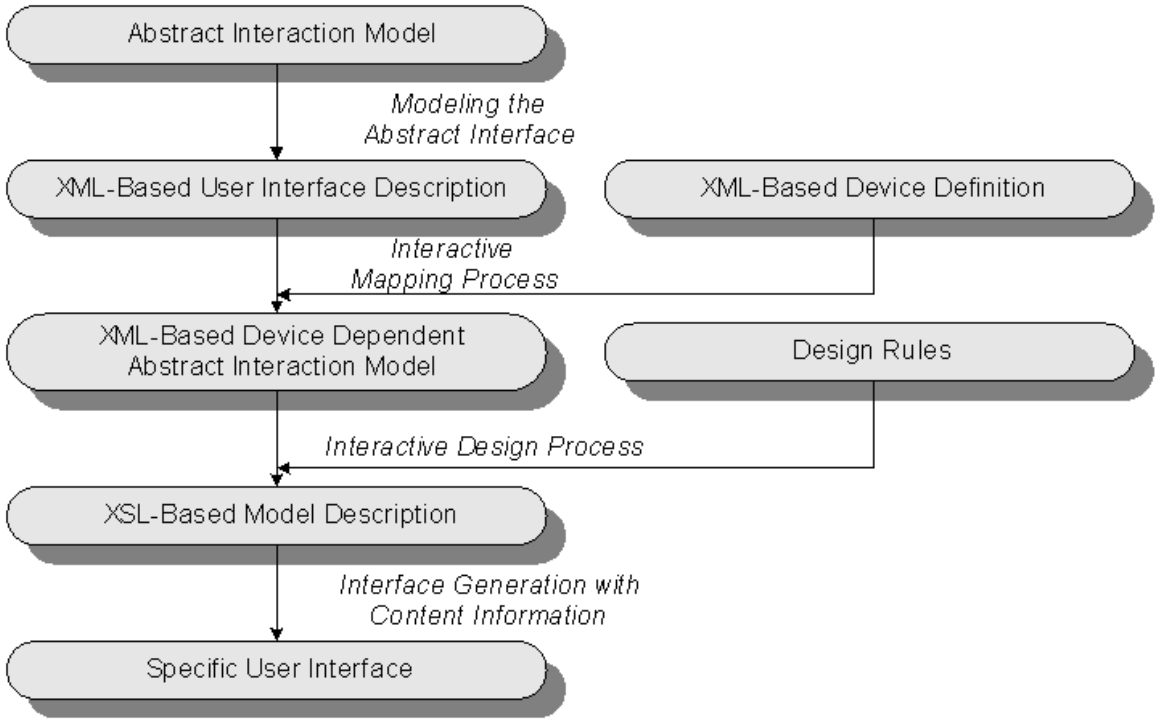


Fig.25: Creation of a User Interface

[101] Müller Andreas, Forbrig Peter, Cap Clemens, "Model-Based User Interface Design Using Markup Concepts", University of Rostock, Department of Computer Science, Germany

[102] Goldfarb, C.F., Prescod, P.: XML-Handbook, Prentice Hall, 1999

8.4.4 Example created by using TADEUS-XML

Appendices 2 show a specific way of using XML in the proposed manner. The example of an e-shop is used to show the principles of this approach. The transformation from an abstract specification to a concrete user interfaces is also shown. Note that requirements that should be fulfilled by a markup language specifying user interfaces are:

- device independence,
- flexibility in representing user interfaces depending on the contexts,
- ability to specify virtual and physical[103] user interfaces,
- localization,
- extensibility of abstractions (abstract User Interface Object's describing interaction behaviour),
- extensibility of concrete User Interface Object's (User Interface Object's of specific representations),

8.4.5 TADEUS-XML advantages and drawbacks

Currently TADEUS-XML approach includes the specification of XML-based languages describing different abstraction of a user interface for mobile devices. A concept that allows mapping process from abstract user interface objects to concrete user interface objects is already developed. A XML language was developed which allows the specification of the features of the user interface of mobile devices. The transformation process of the user interface is based on specifications written in this language. In an interactive design process based on an abstract device dependent abstract interaction model and some design rules a

[103] Müller, A.: Spezifikation von Benutzerschnittstellen durch eine XML-basierte Markup-Sprache. Rostocker Informatik-Berichte 25, 2001

more refined xsl-description of the user interfaces is developed, which allows the generation of specific user interfaces.

The main drawback of TADEUS-XML is that it is just on development stage. The tool supporting the mapping and design process is not yet development. Moreover the abstract interaction model has to be edited manually. Furthermore, it is not sure that whether dialogue sequences can be integrated into the design process.

8.5 WSXL= Web Service Experience Language

WSXL consists of a suite of web services, XML vocabularies, and user facing processing models that build on existing and emerging Web services and XML standards. WSXL provides a Web services layer on top of the standards stack that is aimed at user-facing application development and deployment[104]. WSXL is developed by IBM Inc.

8.5.1 WSXL objectives

WSXL has two main goals, firstly enable businesses to deliver interactive Web applications through multiple distribution channels and secondly enable new services or applications to be created by leveraging other interactive applications across the Web.

8.5.2 WSXL environment

User experiences that are implemented using WSXL can be delivered to end users through a diversity of distribution channels. Moreover, WSXL user experiences can easily be modified, adapted, aggregated, coordinated, synchronized or integrated, often by simple declarative means. WSXL is built on widely accepted

[104] http://www-106.ibm.com/developerworks/webservices/library/ws-wsxl/#2.2.3

established and emerging open standards, and is designed to be independent of execution platform, browser, and presentation markup[105].

8.5.3 WSXL advantages and drawbacks

With WSXL, applications can be built out of separate presentation, data, and control components. This separation helps the developers to insolate design issues that if left intermixed in more monolithic objects would make integration for re-use in multiple channels more difficult. To ensure a fit with existing web-based application architectures, WSXL services generate markup that can be used by conventional browsers and devices through existing formats and protocols. By enabling the assembly of applications from a variety of components, WSXL reduces the cost of producing multiple variations of the same application.

The main drawback of WSXL is that currently it is not yet developed to mobile user interfaces. WSXL is just developed to be the next piece of the set of web services. WSXL provides a web services standards based approach for web application development, deployment and maintenance.

8.5.4 WSXL structure

All WSXL component services implement a set of base operations for life cycle management, accepting user input, and producing presentation markup. WSXL also introduces a new description language to guide the adaptation of user experience to new distribution channels. WSXL includes an extensible Adaptation Description Language where explicit locations of adaptation points, the permissible operations on adaptation points (e.g. insert, delete, modify...), and the constraints on the contents of adaptation (e.g. via an XML Schema) can be specified. The Adaptation Description Language can be used during a post-

[105] http://www-106.ibm.com/developerworks/webservices/library/ws-wsxl/#2.2.3

processing step where the output of a WSXL component can be adapted independently without invoking the component[106].

8.6 xCA.Suite

xCA.Suite is a software created by xCA NV Belgium.

8.6.1 xCA.Suite objectives

xAC.Suite is a software platform for setting up and maintaining multi-channel[107] and multi-device e-business projects, taking into account the context of the various devices and their users[108].

8.6.2 xCA.Suite environment

In fact, xCA.Suite is designed for covering all frequently available devices, like mobile phones, PDA's, iTV[109] ... xCA.Suite doesn't replace the current Internet software but it complements existing technology. xCA.Suite lay-out and the structure of static and dynamic data are really easy to manage and it is also quiet easy to disseminate data to the various channels. To make the platform suitable for current as well as future devices and channels, new channels can be added.

8.6.3 xCA.Suite structure

xCA.Suite is built as an open platform, using standards such as XML and Java and it is also simple to connect it with existing environments[110]. xCA.Suite is composed by different components, a part of them are designed for business users,

[106] http://www-106.ibm.com/developerworks/webservices/library/ws-wsxl/#2.2.3

[107] A channel is the virtual touch point that is made up of a device combined with specific criteria like user groups, time of usage, marketing segmentations, …

[108] xCa NV, *« Technical White Paper »,* Belgium, available at http://www.x-ca.com

[109] xCa NV, *« Business Paper »,* Belgium, September 2001, available at http://www.x-ca.com

[110] xCa NV, *« Technical White Paper »,* Belgium, available at http://www.x-ca.com

and the other part for technical users. The figure 26 gives the general overview of the different components of xCA.Suite that are briefly described follow:

- xCA.project has as function to define the channels and the navigation rules per channel. It is a client tool[111].
- xCa.template, its function is to define the templates per channel. It is a client tool.
- xCa.content serves to create static content. It is a client tool.
- xCa.engine is the server of xCa.Suite. It detects the device doing a request. It selects the correct corresponding channel. It renders the content into the right format and sends it backs to the requestor (visitor).
- xCa.admin is the administrator of xCa.engine. It is a technical tool[112].
- xCa.developer is the backend and the third part integration. It is also a technical tool.[113]

[111] The client tools are all written in C++ and visual basic. The client tools work in a Client/Server model. They are connected to Xca.engine via HTTP or HTTPS.

[112] xCA.admin is written in Java (Swing) and also it is also connected to xCA.engine via HTTP or HTTPS.

[113] xCa NV, *« Technical White Paper »,* Belgium, available at http://www.x-ca.com

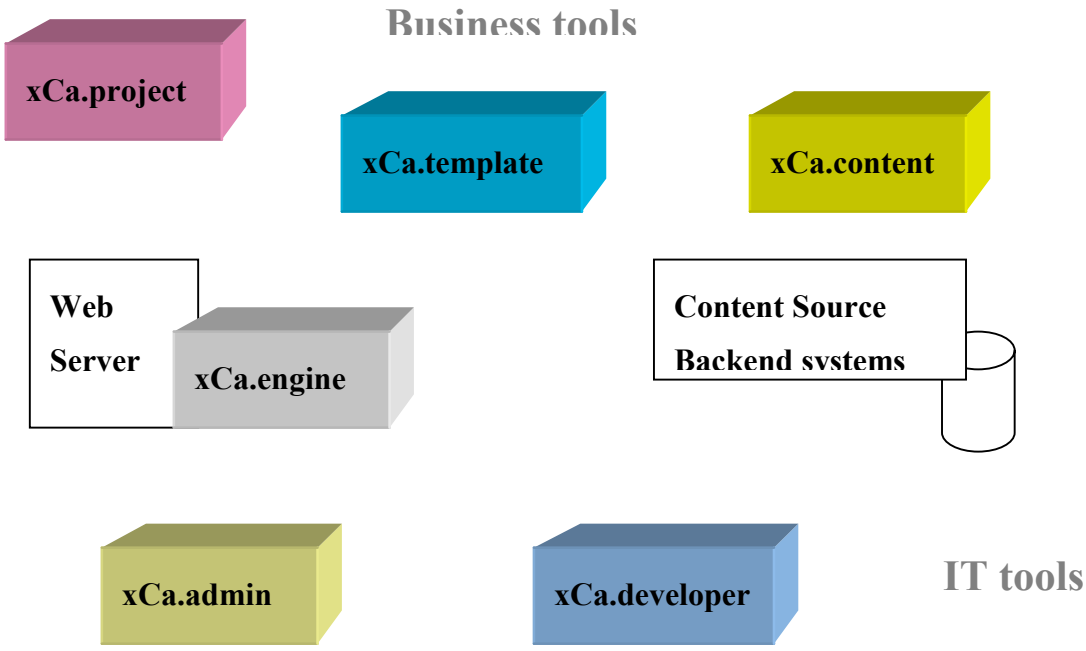


Fig.26: General overview of the different components of xCA.Suite

Chapter 9 *Which XML language for the CAMELEON project*

In the previous chapters, we already provided the reader with a preliminary analysis of some existing user interface specification languages that are XML-based. To address the question of reusing a previously existing XML-based language in the CAMELEON project or building a new one, it turns out that a small discussion of possible candidates may shed some light.

In this chapter we first point out the main motivations that lead the consortium to consider the XIML language as a potential candidate notation for the CAMELEON project. Then, we compare XIML with respect to a hypothetical CameleonXML language to provide a framework for the decision to make in the second year of the project.

Analysis of the external language candidates for the purposes of CAMELEON

XUL (XML-based User interface specification language – see http://www.mozilla.org/projects/l10n/xul-styleguide.html) is a "cross platform way of describing user interface. It could contain XUL specific element types for UI controls, HTML4 markups for content data, and even JavaScript for user event handling. To make XUL files easy to read and maintain, localization friendly, and portable across user agents, we need to have a set of coding style guidelines […]". XUL is an official initiative launched by the Mozilla group. Therefore, it has received some 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. It is mainly intended for browsing, navigating and displaying contents expressed in HTML 4 on different computing platforms. Per se, it does address some requirements for multiple contexts of use (here, restricted to only some computing platforms). But it is not intended to be a genuine and complete UI specification language that can serve in multiple contexts of use. XUL is only very partially supported by existing browsers that equip some web appliances.

UIML (User Interface Markup Language – see http://www.uiml.org/index.php and http://www.harmonia.com) is another XML-based language to express user interfaces for different computing platforms: basically, Java Runtime environments, HTML, WML, and Voice XML. The main idea of UIML is to express any UI in a way that is independent enough of any computing platform to

be abstract from them and to render the specification thanks to a specific rendering engine. Such an engine should be developed for every platform to be supported. In the domain of XML-based languages that support some level of abstraction from the physical independence, UIML has received some interesting attention since it has been launched in 2000. In particular, several software have been developed so far that support UIML. A UIML-based specification basically covers the presentation model and some parts of the dialog model (through the use of events). The set of tags provided by UIML is very compact, yet inflexible and impossible to redefine or to expand. Therefore, other models required to express context-sensitivity, like task model, user model, environment model, cannot be supported. Although UIML is probably the language that is the best supported by tools today, its expressiveness and the power of the underlying languages are too narrow to warrant acceptation in the frame of the CAMELEON project.

AUIML (Abstract User Interface Markup Language – see http://www.belchi.be/download/merrick.pdf, http://math.uma.pt/njn/NunoPhDPrint.pdf, http://math.uma.pt/tupis00/submissions/azevedoroberts/azevedoroberts.html) is an initiative promoted by IBM to express user interfaces for cross-platform computing. AUIML emphasizes more the methodological aspects of expressing UIs for multiple computing platforms at the same time than the software tool support. In particular, some work has been produced to integrate the OVID methodology to develop interactive applications with AUIML. We are not aware of any existing tool. However, some XSLT transformations are possible to apply on the AUIML expression of a UI. Other transcoding techniques, partially supported by the software IBM WebSphere, are possible to produce a UI expressed in another XML-based language that is directly recognized by a specific computing platform. Therefore, rather that rendering the AUIML specification, the file is transformed until it is machine-readable or machine-interpretable (e.g., DHTML, Java, XForms). Like UIML, AUIML only supports some predefined features of the presentation and the dialogue models. It does not support other models that are manipulated in context-sensitivity. Moreover, it is impossible to expand the language. In addition, recent information gained from IBM representatives reveal that IBM no longer supports AUIML per se, but its usage through tools like WebSphere. IBM today focuses more on the development of Web services, through the WSUI and the WSXL languages, than on UI specification language. Both languages only partially support presentation, dialogue, and many features in the application model or in communications aspects, which is beyond the scope of the CAMELEON project.

Xforms (see http://www.w3.org/MarkUp/Forms/) is a W3C initiative to express forms-based UIs at a level that is more abstract than supposed-to be physical HTML description. In some sense, this initiative addresses the question of

context-sensitivity (here, in the support of multiple computing platforms). From a further analysis, it turns out that, although Xforms is promoted by the W3C, thus giving it the widest potential audience abroad, no true implementation already exists neither in existing Web browser or in web appliances. Moreover, Xforms is basically aimed at expressing forms-based user interfaces (with presentation and some dialogue aspects), but does not necessarily support other types of UI involving other interaction styles.

Other similar initiatives exist that address the same problem, like SISL from Bell Labs (see http://www.bell-labs.com/user/lalita/sisl-external.html), XISL (see http://www.vox.tutkie.tut.ac.jp/XISL/W3C-Mar01-02.ppt), SeescoaXML (see http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/SEESCOA/ or http://lumumba.luc.ac.be/kris/publications/dsvis2001/luyten_coninx-dsvis2001/index.html), or TadeusXML (see http://www.dcs.gla.ac.uk/~johnson/papers/dsvis_2001/muller). The points that have been raised above are still valid for these candidates: they mainly focus on computing platform independence (and not on context independence), they are more specific to a predefined set of constraints imposed by the originators of the languages, the set of tags is predefined, yet inflexible. The models supported are mainly presentation, dialogue, and application. Tool support is limited by the project and the potential audience is directly proportional with the size and importance of the organization which promoted the language.

XIML (eXtensible user Interface Markup Language – see http://www.ximl.org and http://www.redhwale.com, http://www.iuiconf.org/02pdf/2002-002-0043.pdf, http://www-scf.usc.edu/~jeisenst/papers/Puerta-IUI02.pdf) is intended to be a universal user interface specification language that can be used as an exchange format between any partner, inside or outside the XIML Consortium. XIML basically describes any type of UI with a user interface model that is composed of one or several component models, of one type or of different types. For example, XIML can hold multiple presentation models at the same time for one context of use or for multiple contexts of use. Canonical component models are: task, domain, user, presentation, and dialogue. A ‘general’ component model may serve as a general-purpose model for specifying features that are not necessarily supported in canonical models. By this way, a general model can serve as an application model, an environment model, a computing platform model, a device model, etc. XIML is equipped with a rich expression of relationships within and across component models. Any model or relationship can hold features, attributes. Relationships can be defined globally or locally to a model, or even for some part of a model. The XIML language is heavily based on solving the mapping problem between models. It is not the main goal of XIML to be used like UIML with rendering engines. However, some software tools are announced like HTML2XIML, XIML2HTML, XIML2WML. Any XIML specification can be

used in a traditional compilation process (a XIML description is transformed into a native code, and then compiled to obtain a final runnable UI) or in an interpretation process (a XIML description is left on the client workstation and interpreted at run-time to produce a final runnable UI). The XIML language has a predefined set of tags, but these tags can describe any new model, feature, attribute or relationship, which is fundamental for the CAMELEON project. From the above qualities, we can reasonably retain XIML as a potential candidate for the CAMELEON project. In the next section, we thus compare XIML with respect to a hypothetical CameleonXML language to provide a framework to decide the future usage of XIML or the definition of a proprietary CameleonXML language.

Comparison between XIML and CameleonXML

We have seen above that XIML provides a rich set of capabilities that are of high interest for the CAMELEON project. However, XIML is protected by copyright by the XIML Consortium. It is free of use for any academic partner, but it is not so explicit for industrial partners. Moreover, any software that is XIML compliant can only be distributed if the future user of this software already possesses a XIML license. Although this license can be freely obtained by filling out the forms and returning them to the XIML consortium, this registration process may be interpreted as a limitation and a potential reduction of the audience or the interest in downloading Cameleon software. On the other hand, XIML is probably today the most advanced UI Specification language. It can serve for context-sensitivity and many other objectives. It might be argued that reinventing the wheel is not a very efficient and effective process. We below attempt to provide the reader with a comparison table.

If a CameleonXML language will be developed, it will take into account the formal work described in the document entitled “The Cameleon notation” (by Quentin Limbourg, UCL). XIML and this approach roughly have the same level of expressiveness, but XIML mainly focusses on syntactical aspects rather than semantic aspects, although both are required to define a full specification language. Indeed, the XIML definition is very precise in the definition of model, feature, relationship, but the interpretation of this information is left to the responsibility of the designer or the software support tool. This means that a XIML description can be syntactically correct, but semantically imprecise. This should be avoided in the Cameleon notation as all concepts are defined in a 4 layers-architecture (ranging from the instance level up to the meta-meta-model level). This does not occur in XIML. Limbourg’s notation is heavily based on graph theory and graph transformations, while XIML is mainly based on solving the mapping problem. In contrast, in the Limbourg’s notation, relationships or component models are rigorously defined on the basis of concepts that are defined at the previous layer, which is not the case in XIML.

The following table provides a first attempt at characterizing the shortcomings and advantages of keeping XIML as an exchange format versus a CameleonXML representation. It should be noted that the CAMELEON notation, as defined in the document, does not necessarily preclude any XML-based representation. It is possible to represent the instance and the model layers thanks to XIML, but not the meta-model level and the meta-meta-model level.

Property	**XIML**	**CameleonXML**
Expressiveness	Excellent	Excellent.
Effectiveness	Good	Good
Compactness	Medium	Good (due to the definition of concepts at the upper layer, concepts can be represented more concisely)
Support for context-sensitivity	Excellent	Excellent
Model coverage	Optimal	Optimal
Relationship coverage	Optimal, but restrained to syntactical aspects	Optimal, but explicitly defined for semantics (especially generic relationships)
Features coverage	Optimal	Optimal
Genericity	Excellent	Excellent, but need to provide partial orders for nodes and edges for every model to be supported
Tool support	Minimal today, but promising (XIML2HTML generator, XIML2WML are announced, graphical editor exists)	Tools exist but are not integrated because of the lack of the CAMELEON project, it can be enhanced in the future.
Expandability	Excellent	Excellent
Openness	Excellent	Excellent
Registration	Need to register for XIML license – any tool, method or model using XIML should be delivered with a free license. Although this is free, this can be considered as a hindrance	To be decided
Distribution	Can be distributed with conditions – is	Can be distributed without any condition – maybe open source

	copyrighted, no open source	
Level of support	Consortium of companies and universities (mainly Stanford University)	To be defined, but should be based on the CAMELEON Consortium and partners.
Origin	United States of America	Europe

It should be finally noted that XIML is exclusively composed of American partners in the Steering Committee of the Consortium. However, any industrial or academic partner may register for free, provided that the conditions are met and non-disclosure agreement is signed. On particular, it is forbidden to publish XIML code without the consent of the XIML consortium, thus taking some time to get. In contrast, publishing code based on the CameleonXML should remain free.

On the one hand, it might be a pity not to reuse efforts already produced in the XIML Consortium, but on the other hand, if the Cameleon tools are only committed to XIML, the distribution of their efforts may be constrained and/or governed by the XIML Consortium, even if the license is free.

If we decide to promote a CameleonXML markup language, there is already a body of work done by various partners that provides a useful basis: Limbourg's formal notation, TERESA XML, the reference framework. Unfortunately, in the first year there has not been sufficient time to discuss these contributions and how to integrate them in order to produce a CAMELEON language. Such language could have some overlapping with the XIML language given that in both cases the promoters are researchers from the model-based community but this effort would enable CAMELEON to remain autonomous from external decisions.

References

- Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S., Shuster, J.: UIML: An Appliance-Independent XML User Interface Language. In: Mendelzon, A. (ed.): Proceedings of 8th Int. World-Wide Web Conference WWW'8 (Toronto, May 11-14, 1999).Elsevier Science Publishers, Amsterdam (1999). Accessible at http://www8.org/w8-papers/5b-hyper text-media/uiml/uiml.html

- Abrams, M., Phanouriou, C.: UIML: An XML Language for Building Device Independent User Interfaces. In: Proceedings of XML'99 (Philadelphia, December 1999). Elsevier Science Pub., Amsterdam (1999). Accessible at http://www.harmonia.com/resources/xml99Fi nal.pdf

- Azevedo, P., Merrick, R., Roberts, D.: OVID to AUIML - User-Oriented Interface Modelling. In: Nunes, N. (ed.): Proceedings of 1st Workshop "Towards a UML Profile for Interactive Systems Development" TUPIS'00 (York, October 2-3, 2000). Accessible at http://math. uma.pt/tupis00/submissions/azevedoroberts/azevedoroberts.html

- K. Bharat and L. Cardelli, *Migratory applications. In Eighth ACM Symposium on User Interface Software and Technology,* pages 133-142, 1995.

- Calvary, G., Coutaz, J., Thevenin, D., Bouillon, L., Florins, M., Limbourg, Q., Souchon, N., Vanderdonckt, J., Marucci, L., Paternò, F., Santoro, C.,

Reference Framework, Models and Criteria for Usability in Multi-Context Applications, Deliverable 1.1, CAMELEON Project, August 2002

- Costin Pribeanu, Quentin Limbourg, Jean Vanderdonckt, *Task Modelling for Context-Sensitive User Interfaces.*

- Jacob Eisenstein, *The XIML Files, Paper 1 XIML: The eXtensible user Interface Markup Language,* November 18, 1999

- Mir Farooq Ali's thesis at Virginia Tech: http://vto-pus.cs.vt.edu/~abrams/uiml/.

- J. Landay and T. Kaufmann, *User Interface Language.* World Wide Web, http://www:sop.inria.fr/koala/kuil/, 2000.

- Kris Luyten and Karin Coninx, *Het SEESCOA project; jouw user interface, altijd en overall,* 17 januari 2002

- Kris Luyten and Karin Coninx, *An XML-based runtime user interface description language for mobile computing devices,* Limburgs Universitair Centrum, Diepenbeek, Belgium.

- Kris Luyten, Chris Vandervelpen, Karin Coninx, *Adaptable User Interfaces in Component Based Development for Embedded Systems,* Expertisecentrum Digitale Media – Limburgs Universitair Centrum, Diepenbeek -Belgium

- Didier Martin, Mark Birbeck, Michael Kay, Brian Loesgen, Jon Pinnock, Steven Livingstone, Peter Strak, Kevin Williams, Richard Anderson, Stephen Mohr, David Baliles, Bruce Peat, and Nikola Ozu. *Professional XML,* Wrox Press, 2000.

- Mori, G., Paganelli, L., Paternò, F., Santoro, C., *Tools for Model-Based Design of Multi-Context Applications, Deliverable 2.1, CAMELEON Project,* August 2002

- Puerta, A.R. "A Model-Based Interface Development Environment". In *IEEE Softwar*e. 1997. pp. 40-47.

- Angel Puerta and Jacob Eisenstein, XIML: *A Universal Language for User Interfaces*.

- Puerta A. and Eisenstein, J. "Towards a General Computational Framework for Model-Based Interface Development Systems". In *Knowledge-Based System*s, Vol. 12, 1999, pp. 433-442

- Sheng Liang and Gilad Bracha, *Dynamic class loading in the Java virtual machine.* In ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'98), pages36-44, 1998

- Frank Sommers, *Object mobility in the Jini environment,* World Wide Web, http://www.javaworld.com/javaworld/jw-01-2001/jw-0105-jiniology.html, January 2001

- Sun Microsystems.*Java Remote Method Invocation.* World Wide Web, http://java.sun.com/products/jdk/rmi/, 1997

- Szekely, P. and al. "Declarative Interface Models for UserInterface Construction Tools: the MASTERMIND Approach". In *Engineering for Human-Computer Interactio*n, L.J. Bass and C. Unger (eds), Chapman & Hall, London, 1995, pp 120-150.

- Tam, R.C.-M., Maulsby D., and Puerta, A. "U-TEL: A Tool for Eliciting User Task Models from Domain Experts". In Proc. *IUI98: 1998 International Conference on Intelligent User Interface*s. San Francisco, CA. ACM Press.

- David Thevenin, *Adaptation en Interaction Homme-Machine*, 21 septembre 2001

- D. Urting, Y. Berbers, S. Van Baelen, T. Holvoet, Y. Vandewoude and P. Rigole, *A Tool for Component Based Design of Embedded Software,* Department of Computer Science, Catholic University of Leuven, Belgium

- Vanderdonckt, J., Bouillon, L., and Souchon, N., "Flexible Reverse Engineering of Web Pages with Vaquita". In Proc. *WCRE'200: IEEE 8th Working Conference on Reverse Engineering.* Stuttgart, October 2001. IEEE Press.

- Jean Vanderdonckt, *The XIML Files, Towards a Universal Language for User Interface Modeling,* February 8, 2000

- http://www.w3.org/2001/di/

- http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html

- http://www.google.be/search?q=cache:5XmjtqsCto0C:www.belchi.be/download/merrick.pdf+AUIML&hl=nl&ie=UTF-8 document done by Roland A. Merrick 21- 24 IBM Copyright IBM 2001

http://math.uma.pt/tupis00/submissions/azevedoroberts/azevedoroberts.html