SPECIFYING USER INTERFACES FOR RUNTIME MODAL INDEPENDENT MIGRATION

Kris Luyten, Tom Van Laerhoven, Karin Coninx, Frank Van Reeth
{kris.luyten, tom.vanlaerhoven, karin.coninx, frank.vanreeth}@luc.ac.be
Expertisecentrum Digitale Media
Limburgs Universitair Centrum
Wetenschapspark 2
B-3590 Diepenbeek-Belgium

Abstract The usage of computing systems has evolved dramatically over the last years. Starting from a low level procedural usage, in which a process for executing one or several tasks is carried out, computers now tend to be used in a problem oriented way. Future computer usage will be more centered around particular services, and will not be focused on platforms or applications. These services should be independent of the technology used to interact with them. In this paper an approach will be presented which provides a uniform interface to such services, without any dependence on modality, platform or programming language. Through the usage of general user interface descriptions, presented in XML, and converted using XSLT, a uniform framework is presented for runtime migration of user interfaces. As a consequence, future services will become easily extensible for all kinds of devices and modalities. An implementation serving as a proof of concept, a runtime conversion of a joystick in a 3D virtual environment into a 2D dialog-based user interface, is developed.

1. Introduction

Nowadays new computing devices are introduced into the market at a very high rate. A lot of these devices with their own properties and constraints are capable of delivering specialized services. As the technology evolves, more devices will become available with a more general purpose (e.g. Palm) and less constraints. This equipment is capable of presenting different kinds of services which are available in their environment. One possible service is the usage of a mobile computing device as a remote control.

A transition from an application-oriented view on computing towards a service-oriented view is expected. The last couple of years it has been observed that this approach is not feasible without a reliable framework to support it. This framework should be able to offer support for device-independent migration of services and the user interfaces for these services. It should enable us to create services in a device-independent way, offering the supporting devices the choice of how to present these services. Also, the framework should consider future extensions with new kinds of interaction methods and new possibilities in network communication, memory capacity, processing speed, etc.

This paper presents our ongoing research on the possibility of creating such a framework that will allow for runtime migratable user interfaces, which are independent of the target software platform, the target device and the interaction modalities. These user interfaces are merely considered as a presentation of a single service or of more functionally grouped services. We try to extend the work presented in [5, 18, 14] which all focus on how to abstract a user interface for a platform- and device-independent usage. At the same time we try to use the work presented in [13, 7, 1] using markup languages to describe the user interface with the significant difference with our approach that we want the given markup language to describe the user interface to be generated at runtime.

The next section takes a look at how a user interface can be serialized in a user interface description language at runtime. Then we convert this description in a high level description of the user interface, which complies with some rules. In a following paragraph, we consider a possibility for serializing the services and the context these services operate in. An implementation serving as a proof of concept is presented, and finally conclusions with regard to the current work and possible extensions are formulated.

2. Runtime Generated Specification of the Interface

The concept of a runtime generated user interface description has been introduced in [10]. At runtime a user interface description is generated by means of XML (e*X*tensible *M*arkup *L*anguage) [2] as a description language. The XML description provides an abstraction of the user interface using *Abstract Interaction Objects* (AIO) which can be mapped onto *Concrete Interaction Objects* (CIO) [18]. A more practical view is shown in figure 1. Notice the similarities with Model-based design [15, 16]; the most high-level abstraction of the user interface is a group

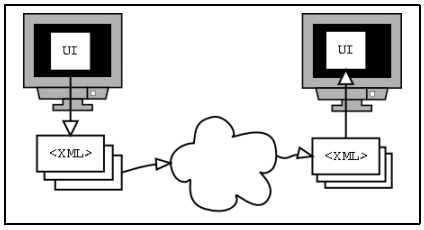

Figure 1. A migrating User Interface

of tasks. Our work does not aim to extract a Task-based description of the User Interface but merely tries to abstract the "contents" of the user interface, not the presentation. Although we acknowledge the idea of splitting the interface model into a User-task model, a domain model, a user model, a presentation model and a dialog model [16] for getting a better mapping of AIO's on CIO's, we do not consider it to be useful when focusing on runtime migration of *existing* user interfaces.

A working user interface is "serialized" into an XML description at runtime and this description can be transported to another system using for example the Internet or an infrared communication protocol. Once arrived at the target system the XML document can be parsed and converted in a working user interface ("deserialized"). An advantage of this approach can be found in the abstraction of the original user interface that the XML document provides. While parsing the XML document that contains an abstraction of the user interface, the renderer of the target platform is free to choose other ways to present the same functionality in the user interface. Much current research is being conducted on how XML is best structured and used to describe such interfaces. More information about other approaches and techniques can be found in [13, 11, 1, 4, 14]. The most advantageous properties of XML for describing a user interface are enumerated in [10].

The feasibility of this approach depends on how easily an XML description can be generated for a user interface of a working service. For example: when a user interface is written in the programming language Java, it is fairly easy to generate an XML description for this user interface at runtime, even if this possibility was not considered while programming the user interface. Using the reflection mechanism, which allows us to interrogate a Java program about its properties at runtime, an XML description of this structure, and more specific of its user interface, can be generated. Another advantage of this approach is the reusability of existing Java user interfaces which can be incorporated in new systems using new technologies. This should reduce the time-to-

market for new versions of systems and reduce the need for reinventing the same user interface over and over again while making use of new technologies.

Unfortunately it is hard or even impossible to add a description of the *context* the user interface is working in at runtime (see section 4 for a more comprehensive explanation). A description of the services it offers should be provided. Those programming languages which do not support a reflection mechanism should provide a framework for the serialization of the user interface in XML at runtime. Section 5.1 shows such a framework currently under development for virtual environments.

3. Abstracting the User Interface Description

To allow the user interface to adapt to the constraints of the target system it is important to provide an abstraction of the original user interface. When the user interface description is generated at runtime by the service it is likely to contain system specific elements. To make sure the XML description is sufficiently general the user interface should be abstracted so it becomes independent of how the interaction takes place with the interface. This allows for changes in the interaction modality without a large effort. A great deal of plasticity [17] is gained, allowing us to use system specific data to adapt the user interface to the new constraints.

As an abstraction of the user interface we have defined a DTD [2]. This DTD restricts the possible elements and ordering used in the XML document. Our current DTD is inspired by [13] and enriched by a subset of interactors like the ones presented in [18]. We are planning to replace it with an equivalent XML Schema [3] as soon as there are more mature tools available to use this. The DTD we use to describe the high level user interface description can be found in figure 3. The transformation of the original user interface description into a higher level description is realized using the XSLT. Every system or user interface toolkit has to define a conversion providing the appropriate XSLT. Once the higher level description is produced a system specific XML document for the target system has to be produced. For every system a XSLT is defined which maps the AIOs defined in the abstract user interface description to CIOs for that particular system. An overview of this process is given in figure 2. Closely related work can be found in AUIML of IBM [12]. However the following difference is to be noticed: in contrast with AUIML our work is mainly focused on runtime migration and adaptation of the user interfaces.

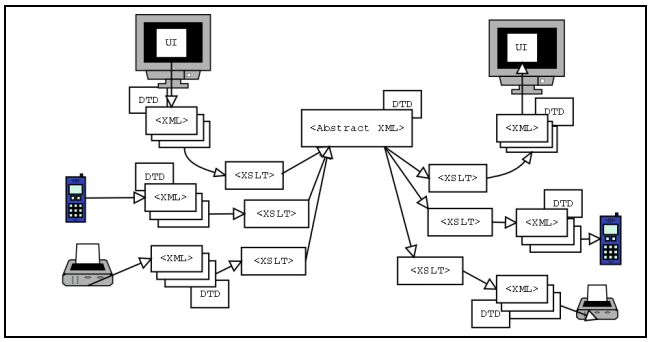

Figure 2. User interface migration

```
<?xml version="1.0"?>
<!ELEMENT interface (aio*)>
<!ELEMENT aio (aio | (input?,output?))+>
<!ATTLIST aio name CDATA #IMPLIED
              type CDATA #REQUIRED>
<!ELEMENT input ( (input_trigger | input_1-n |
                   input_string | input_date | ... ), Icontext)>
<!ATTLIST input name CDATA #REQUIRED>
<!ELEMENT input_trigger (EMPTY)>
<!ATTLIST input_trigger service_name CDATA #REQUIRED>
<!ELEMENT input_1-n (EMPTY)>
<!ATTLIST input_1-n range_min CDATA #IMPLIED
                    range_max CDATA #IMPLIED
                    range_interval CDATA #IMPLIED
                    list_n CDATA #IMPLIED>
...
<!ELEMENT output ( ( output_1-n | output_string | ...), Ocontext)>
<!ATTLIST output name CDATA #REQUIRED>
```

Figure 3. The DTD for the abstract user interface description

4. Runtime Specification of the Services

In the previous sections we did not consider how a user interface can be serialized at runtime, taking into account only the *structure* of the user interface but not the *services* it represents. When serializing a user interface at runtime for migration the services these interfaces offer should be accessible through the migrated user interface. On a lower level two different methods give access to the services. Allowing the services (the actual functionality), which are accessed through the interface, to migrate together with the user interface is a first possibility. A second possibility is accessing the services remotely. In this context the user interface actually serves as a remote control to interact with the service it presents. The former can be a reliable approach if the functionality of the service is limited or does not involve a large quantity of data. The latter can be better if the service implies a large and complex application. Another important aspect to take into account is the possible delay, depending on the communication medium. Unlike the user interface, which is entirely migrated, this is not always the case with the services the user interface presents.

To make more reasonable decisions on how the user interface can be represented on different systems we feel a need to include the context the interface objects are working in. A context can be defined at several levels: on the lowest level, where some interaction objects are grouped together because they do not have a full meaning on their own. On higher levels, where a context can be hierarchically composed of several other contexts. The context can be divided roughly in two parts: a context for retrieving input and a context for expressing output through the user interface. This enables us to group several AIOs together which have a specific task in common together. For example; consider a 2D user interface for filling in the date composed out of three drop-down boxes. One interactor for the day, another one for the month and one for the year. These three interactors are only meaningful when used in group, otherwise they can not accomplish their original goal: retrieving a date from the operator. When dealing with runtime serialization of the user interface it is very important to remember the context a group of AIOs is working in, so they can be grouped in the same context when they are deserialized. A part of the DTD for describing the context can be found in figure 4.

```
...
<!ELEMENT Icontext (Icontext_value)+>
<!ATTLIST Icontext name CDATA #IMPLIED>
<!ELEMENT Ocontext (Ocontext_value)+>
<!ATTLIST Ocontext name CDATA #IMPLIED>
<!ELEMENT Icontext_value (#PCDATA | aio | Icontext_value+ | Ocontext)>
<!ELEMENT Ocontext_value (#PCDATA | aio | Ocontext_value+ | Icontext)>
...
```

Figure 4. DTD rules for describing a context

5. Transforming Modalities

5.1 Description of the Case Study

As a proof of concept we will show how the control for a surveillance camera, presented by a joystick in a virtual environment, can be serialized in XML, processed with XSLT [8], and then deserialized into a 2D control using a Palm device. An operator can change the orientation of the camera in two degrees of freedom: up/down and left/right. The operator must confirm the new orientation with some kind of trigger so the surveillance camera becomes locked in that orientation. This means the input is a position in 2D space with on one axis the vertical orientation and on the other axis the horizontal orientation.

5.2 Description of the Original System

A highly extensible framework for the creation of simulations in a 3D environment has been developed by one of the authors. This framework is used to create a virtual world containing a representation of a control to manipulate a surveillance camera. This control is modeled as a virtual joystick with two buttons providing the user the ability to manipulate the camera. A representation of this joystick is given in figure 5. Objects in the environment consist of a collection of modules describing their properties such as shape, state and interactive abilities. The implementation of each of these properties exists in separate modules, which are plugged into the framework. The same modules also provide XML serialization and deserialization.

The virtual joystick is divided into separate components which in turn can be divided into other components describing its properties. In this case, the lower button consists of a polygon mesh describing its shape, a state module describing its position and orientation, and an interaction module that ensures it the interactive ability to be triggered. Communication between the modules is realised by means of commands, which can also be serialized and deserialized. The button on top of the

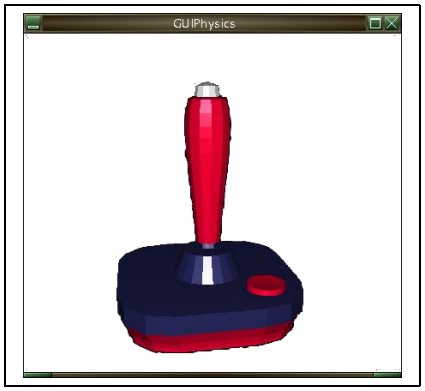

Figure 5. A joystick in a virtual environment

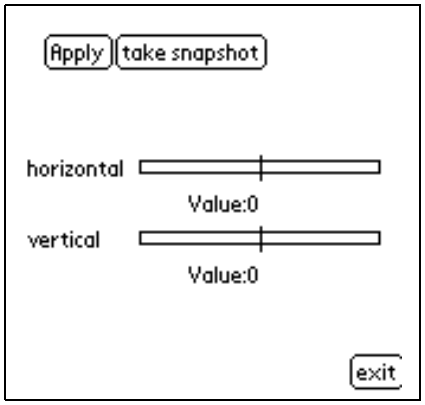

Figure 6. 2D user interface on a Palm device

joystick is modeled in the same way. The handle however requires a different approach. It needs an interaction component that expresses the fact that its container, the handle itself, can give two-dimensional input.

Our framework can serialize this interface in accordance with the XGL [6] specification, a file format designed to capture all 3D information from a system that uses the SGI's OpenGL rendering library. Unfortunately, a DTD for this specification does not exist. The XGL specifications describe their content models by mentioning which elements are required and which are optional, leaving the sequence to the programmer's choice. Defining a content model allowing this kind of freedom, and at the same time expressing the frequency of the elements, is not possible using a DTD. Fortunately, the very few XGL generating applications available seem to agree to some kind of sequence. This is the sequence found in our DTD which is publicly available at `http://lumumba.luc.ac.be/tom/xgl/`.

The resulting XML document that describes a joystick in a 3D world is shown in figure 7. We have limited the example by only depicting the interesting parts of this description.

5.3 Description of the Target System

Our target system is a 2D Java user interface for a Palm device. The user interface operates as a remote control for changing the orientation of a surveillance camera. The description consists of the hierarchy of Java AWT objects which represent the user interface. A subset of Java AWT is used because this Java user interface toolkit is supported by the

```
<component type="object" name="button">
  <component type="shape" subtype="tripolymesh">
     <mesh><!-- XGL specific code here --></mesh>
  </component>
  <component type="interaction" subtype="trigger"/>
  <component type="state" subtype="rigidbody">
     <transform><!-- XGL specific code here --></transform>
  </component>
</component>
```

Figure 7. XML document describing part of the joystick

```
<Component CLASS="VideoControl">
  <Properties>
    <Property NAME="Apply">
      <Component CLASS="java.awt.Button">
        <Property NAME="label">Apply</Property>
         ...
      </Component>
    </Property>
<Property Name="HorizontalSlider">
    ...
</Component>
```

Figure 8. XML document describing a Java user interface for the Palm

Palm device. The XML document describing the user interface (figure 6) is presented in figure 8. This description is obtained by applying the XSLT in figure 9 (only the interesting parts are shown). The XSLT document knows about the constraints the Palm device is subject to, and which the available interaction methods are. The parts of the user interface that cannot be converted, or where too little information is available, could be adapted by a platform specific agent which works on the target XML document for further optimization of the presentation of the AIOs.

5.4 A General View on the Migration Process

From the original serialized XML document (figure 7) an XML document compliant with the DTD in figure 3 should be generated. Each system provides its own XSLT to initiate the conversion. The XSLT document "generalizes" the concrete user interface which was described in the XML document of the original user interface.

Once the high level user interface description is obtained the conversion for the target platform can be initiated. Applying the XSLT document on the high level XML document results in a platform specific user interface description. The XSLT makes sure the AIOs will be

```
...
<xsl:template match="interface">
   <xsl:element name="component">
      <xsl:attribute name="CLASS">java.awt.Frame</xsl:attribute>
      <xsl:attribute name="NAME"><xsl:value-of select="@name"/></xsl:attribute>
      <xsl:apply-templates/>
   </xsl:element>
</xsl:template>

<xsl:template match="aio">
   <xsl:choose>
     <xsl:when test="@type='group'">
       <xsl:call-template name="grouppanel"/>
     </xsl:when>
   ...
  </xsl:choose>
</xsl:template>

<xsl:template name="push_interact">
   <xsl:element name="Component">
      <xsl:attribute name="CLASS">java.awt.Button</xsl:attribute>
        <xsl:attribute name="NAME"><xsl:value-of select="@name"/></xsl:attribute>
        <xsl:element name="Properties">
         <xsl:element name="Property">
            <xsl:attribute name="NAME">label</xsl:attribute>
            <xsl:value-of select="@name"/>
   ...
```

Figure 9. XSLT document for obtaining a Palm user interface

mapped on platform specific CIOs, grouping them in a specific context for interaction. Whether the services this user interface represents are invoked remotely or locally is not taken into account here. This should remain transparent for the user whenever possible.

6. Conclusions and Future Work

In this paper we presented the use of a runtime generated user interface description for presenting services on different kind of systems. The user interface description is expressed in XML and we use XSLT to process these descriptions. In a first stage the user interface description is abstracted while the second stage requires the high level user interface to be mapped on a specific GUI toolkit.

We believe the ability to generate a high level user interface description of a service at runtime is a step closer to offering services independent of the system which is used. When the user interface description provides enough information and an abstraction of a sufficiently high level, a user interface becomes almost modeless and switching between modalities can be done automatically. Since a fully automatic conver-

sion of all kinds of user interfaces is unlikely to be successful, a semi-automatic approach is preferred. In the case we have examined here we have shown this approach is feasible and that it provides many opportunities to add extreme extensibility to future user interfaces. One of the consequences is the possibility to adapt a user interface to the target device's constraints. This can be done without forcing the user interface designer to make separate designs for each system. It is even possible to make a user interface for a particular system without considering other, alternative systems with other constraints. This user interface will still be able to migrate to the other systems, adapting to the device's constraints. The context description, explained in section 4, can contribute to a partial solution to overcome this problem.

Future work includes a further generalization of the conversion mechanisms and adding user profiling and device constraints to the conversion process. Also, there is a lack of support to describe how several interactor objects are grouped and need each others data to use certain functionality. A framework which enables the remote communication or even the migration of services, represented by the migrated user interfaces, is necessary. To make sure that the migrated user interface stays usable when deserialized and that it has the same usability characteristics as the original user interface usability testing would be recommended.

Acknowledgments

Our research is partly funded by the Flemish government and EFRO (European Fund for Regional Development). The SEESCOA[1] project IWT 980374 is directly funded by the IWT (Flemish subsidy organization).

Notes

1. Software Engineering for Embedded Systems using a Component-Oriented Approach, `http://www.cs.kuleven.ac.be/cwis/research/distrinet/projects/SEESCOA/`

References

[1] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. UIML: An appliance-independent XML user interface language. *WWW8 / Computer Networks*, 31(11-16):1695–1708, 1999.

[2] World Wide Web consortium. *Extensible Markup Language (XML)*. World Wide Web, `http://www.w3.org/XML/`, 2001.

[3] World Wide Web consortium. *XML Schema*. World Wide Web, `http://www.w3.org/XML/Schema`, 2001.

[4] danm@netscape.com. *Introduction to a XUL Document.* World Wide Web, `http://www.mozilla.org/xpfe/index.html`, october 1999.

[5] Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta. Applying Model-Based Techniques to the Development of UIs for Mobile Computers. In *IUI 2001 International Conference on Intelligent User Interfaces*, pages 69–76, 2001.

[6] XGL Working Group. *XGL File Format Specification.* World Wide Web, `http://www.xglspec.org/`, 2001.

[7] IBM Corporation. *MoDAL (Mobile Document Application Language).* World Wide Web, `http://www.almaden.ibm.com/cs/TSpaces/MoDAL/`.

[8] Michael Kay. *XSLT Programmer's Reference, 2nd Edition.* Wrox Press, 2001.

[9] Thierry Kormann. *The Koala User Interface Language.* World Wide Web, `http://www-sop.inria.fr/koala/kuil/`, 2000.

[10] Kris Luyten and Karin Coninx. An XML-based runtime user interface description language for mobile computing devices. In Chris Johnson, editor, *DSV-IS*, volume 2220 of *Lecture Notes in Computer Science*, pages 1–15. Springer, June 2001.

[11] Kris Luyten, Karin Coninx, Jan Van den Bergh, and Jos Segers. Software engineering for embedded systems using a component oriented approach; deliverable 4.2: Implementation of a component based user interface, seescoa confidential. Technical report, Expertisecentrum Digitale Media; Limburgs Universitair Centrum, 2001.

[12] Roland A. Merrick. Device Independent User Interfaces in XML. BelCHI Kick-off meeting, 2001. `http://www.belchi.be/event.htm`.

[13] Andreas Mülller, Peter Forbrig, and Clemens Cap. Model-Based User Interface Design Using Markup Concepts. In *Proceedings of the Eight Workshop of Design, Specification and Verification of Interactive Systems*, pages 30–39, June 2001.

[14] Dan R. Olsen, Sean Jefferies, Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-modal interaction using XWeb. In *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 191–200, N.Y., November 5–8 2000. ACM Press.

[15] Fabio Paterno. *Model-Based Design and Evaluation of Interactive Applications.* Springer, 2000.

[16] Angel Puerta and Jacob Eisenstein. Towards a general computational framework for model-based interface development systems. In Mark Maybury, editor, *Proceedings of the 1999 International Conference on Intelligent User Interfaces (IUI-99)*, pages 171–180, N.Y., January 5–8 1999. M Press.

[17] David Thevenin and Joelle Coutaz. Adaptation and Plasticity of User Interfaces. In *Workshop on Adaptive Design of Interactive Multimedia Presentations for Mobile Users*, 1999.

[18] J. Vanderdonckt and F. Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In *ACM Conference on Human Aspects in Computing Systems InterCHI'93*, pages 424–429. Addison Wesley, 1993.