An XML-based runtime user interface description language for mobile computing devices

Kris Luyten and Karin Coninx

http://www.springer.de/comp/lncs/index.html

Expertise Centre for Digital Media
Limburgs Universitair Centrum
Wetenschapspark 2
B-3590 Diepenbeek-Belgium
{kris.luyten, karin.coninx}@luc.ac.be

Abstract. In a time where mobile computing devices and embedded systems gain importance, too much time is spent to reinventing user interfaces for each new device. To enhance future extensibility and reusability of systems and their user interfaces we propose a runtime user interface description language, which can cope with constraints found in embedded systems and mobile computing devices. XML seems to be a suitable tool to do this, when combined with Java. Following the evolution of Java towards XML, it is logical to introduce the concept applied to mobile computing devices and embedded systems.

1 Introduction

There is a clear evolution from desktop computing towards embedded and mobile computing devices. Users of these devices are not always experts in their usage, and these users must be taken into account as much as possible. This is not an easy task for user interface designers: while facing diverse environments they have to make user interfaces in the same family of products as consistent as possible. A second challenge is to delegate the user interface design for embedded systems to people who have expertise in designing a user interface as opposed to the people who know how to program an embedded system. Most of the time programming for embedded systems requires very specific technical knowledge, e.g. real-time systems. Finally we will have to take into account very heterogeneous user groups who want to use their device in their own specific way. We will explore the possibility of profiling the users to solve this problem.

Consider the following scenario to give an outline of the problem. A university decides to install projectors on the ceiling of every classroom. They will be using the system described in this paper to make the projectors accessible to different members of the teaching staff and maintenance personnel. All of the employees of the university have a PDA device (e.g. a Palm device), which can receive

data over an infrared connection. Alternatively, a radio-based Bluetooth link between the PDA and the projector can be used. The PDA will be used as a remote control for the projector. During the first class of Thursday, professor Wasaname will have to use the projector for her lectures in HCI. She walks into the classroom and transmits her slides, stored on the PDA, to the projector using the infrared connection. Let us assume the projector knows the slide format and can store and project these. After she has finished transmitting the slides she indicates on her PDA she wants to control the projector. Because her user profile is stored on the PDA, it infers she only wants to cruise through the slides, maybe zoom in on some details and make some annotations, but nothing more. She is not interested in configuring the projector settings like changing the resolution or the color settings. Using this knowledge the PDA "asks" the projector to transmit only those parts of its user interface professor Wasaname is interested in. The projector serializes that part of the user interface and passes it to the PDA device, using the infrared connection. The professor can now project the slides using the PDA as a remote control.

During her first class, professor Wasaname notices the bad resolution and brightness of the projector. After her lesson, a member of the maintenance personnel is asked to fix the problem. The diligent responsible man gets right to the classroom and indicates he wants to use the projector. Looking at his profile, the PDA notices this person is mainly interested in the configuration possibilities of the projector, and asks the projector to only transmit that part of the user interface dedicated to that task. Using his PDA the maintenance man adjusts the brightness and resolution of the projector to a satisfying level.

The previous scenario emphasizes how a user interface should be downloadable and adapted to the users' preferences. We can consider this also as a person entering in an environment in which he or she can use several services offered by a networked computing environment, and use it according to the personal interests of this user. This requires a dynamically downloadable interface for interacting with the desired interface. This can be obtained by serializing a user interface for interacting with a particular service, and migrating the parts of interest of that serialized user interface to the user. For serialization of the user interface an appropriate user interface description language can be used. XML is proposed as a user interface description language for the described problem.

Up to now we are not aware of any project combining mobile computing devices and embedded systems with XML and Java in a similar way. Especially the runtime user interface migration provides the user with powerful means to dynamically take advantage from available services. [19] introduced XWeb, for cross-modal interaction. This work proposes a similar approach, but also includes more interactivity possibilities in the user interfaces that are migrated. Other related work on highlighted aspects will be mentioned in the appropriate sections.

The next section takes a look into what a user interface description language containing constraints should support. Then we choose a description language to develop our example. Next, we consider our proposal for converting a user

interface *at runtime* to its description in the chosen description language. We continue our description by investigating the possibilities of the opposite conversion: an automatic conversion of the user interface description to a user interface for a particular device taking the defined constraints into account. Finally we propose the usage of profiling to reduce the complexity of the problem.

2 Describing User Interfaces Subject to Constraints

As mentioned above, an embedded system or mobile computing device is subject to several constraints, which we generally are not facing with desktop computers. We can divide these constraints in two categories:

static constraints : these are constraints which are not likely to change over time, mostly dependent on our bodily functions. E.g. a very small screen may have a very high resolution, but our eyes will only allow a certain maximal visual capacity. Another example is the number of buttons a device contains: a certain minimal area for the buttons is required if we want to be able to manipulate them with our fingers.

evolving constraints : these are constraints on memory usage, bandwidth, processor speed, etc. Constraints of this kind are likely to change over time and may possibly disappear. To come to a practical proposition for the envisioned systems they should also be included in this discussion.

Our goal is to use a user interface description language which is suitable for a wide range of embedded and mobile computing devices, working in heterogeneous environments, and utilize it at runtime. The example in the introduction shows what kind of problems can be solved using such a description language. Certainly in the case of extensibility of a networked system, integrating new kinds of devices, it will be easier to control the evolution of the system. This can be done without having to cope with inconsistencies in the user interface and interaction methods. We find it also important for the user interface description language to be suitable for design-time usage, providing the designer with a powerful tool to build platform independent and even device independent user interfaces taking constraints in account. Designing a user interface, while taking into account different user groups, is already an extensively explored research domain [24]. Unfortunately, current approaches are either not general enough for embedded systems, do not take the resource limitations of embedded systems into account or do not care about changing environments. Without providing details, we mention some approaches in UIDLs (User Interface Description Languages) [4,20,24]: language-based, grammar-based e.g. BNF, based on state transition diagrams, declarative (e.g. declarative descriptions of application data types), event-based, constraint-based, UAN (User Action Notation, in particular for direct manipulation) and widget-based. Because of the evolving market towards mobile computing devices and embedded systems, a more general approach for describing a user interface for an embedded system or mobile computing device is necessary. In search of a notation for describing such a user interface it should satisfy the following requirements:

Platform independent : because of the heterogeneity of embedded systems, a user interface designer should be allowed to design without having to worry about the system on which the interface will be used. Of course there are certain restrictions to this, which we will discuss further on in this text

Declarative : describing a user interface asks for a certain level of expressiveness for describing human-computer interaction

Consistent : the notation should offer consistency of the user interface among different environments and systems [22]

Unconventional I/O : embedded and mobile computing devices are less conservative in input and output devices. For example: while "normal" desktop computers have a mouse and a keyboard, this is not a requirement for a mobile device, which could very well have a touch-screen and speech recognition

Rapid prototyping : in a highly competitive market, such as mobile devices, developers and designers want to tailor the software towards the users or user groups. A user interface notation should allow rapid prototyping to get the users involved in the development process sooner

Constraint sensitive : because of the constraints embedded systems are coping with, the designers must be able to specify the constraints, or have the system automatically generate them

Easily extensible : we want to extend our user interface with extra functionality, without starting from scratch

Reusability : when a family of products is evolving, we want to reuse the design for the old devices in an optimal way

Notice these are not style guidelines, but rather structure guidelines. It is the designer's responsibility to follow style guidelines as defined in [17] for example.

The demand for a notation enabling the designer to describe constraints for a computing system is an important part of our user interface description language. A constraint can be described as a cost function in which the cost must be limited or kept within an interval for example. Existing specification languages designed for embedded systems use finite-state machines or message sequence charts to represent constraints like this.

3 Choosing a Descriptive Language for UI Design

The previous section listed several properties the user interface description language should have. Instead of creating a new kind of description language, we propose the usage of the eXtensible Markup Language (XML) for describing a user interface. This description language can offer us the properties we want:

Platform independent : XML is platform independent in the same sense that Java is platform independent: if there is an XML parser available on the system, the XML description can be used. If there is no suitable XML parser available for your target platform, XML is so simple that writing your own parser is fairly easy

Declarative : through the usage of XSL[1] [7] XML can be declarative. XSL specifies a stylesheet which is applied to an XML document, and offers powerful transformations on the XML document

Consistent : through the usage of DTD[2] XML can be consistent. A DTD specifies a set of rules for the XML file to follow

Unconventional I/O : XML can describe unconventional I/O: there are plenty of examples to provide evidence: WML [7], SpeechML [5], VoxML [6], etc.

Rapid prototyping : using a stylesheet you can see the results immediately in a browser

Constraint definitions : XML can contain constraint definitions; as well for the form of the XML itself, as for external resources we can add constraint definitions

Easily extensible : because XML is a metalanguage it is by nature an extensible language

Reusability : it is relatively easy to fit an existing piece of XML into another

There is another advantage not addressed in the previous paragraph: because of the simple grammar and structure of XML it is an intuitive markup language. User interface designers do not need a firm technical background to work with XML. Also, it is easy to convert an XML description to different kinds of output presentation using XSLT[3]. Using XSLT, XML can be converted into HTML+CSS[4] for desktop browsers, VoxML for speech driven input or into WML for mobile phones [16]. However, we have to realize that XSLT is no silver bullet for transforming an XML user interface description into different shapes.

This is not the first time XML is proposed to be used as a user interface description language [1,9,11,15]. [9] is aimed for mobile computing devices, while [1,11] are implementations for the desktop computer using Java user interfaces. While most of these description languages only work at design time, we would like to propose an architecture for *runtime* serialization of Java user interfaces into an XML description, inspired by migratory applications [3] and remote user interface software environments [12]. This would enable us to "download" a user interface together with constraints and necessary transformations, which will be discussed in section 4.2. Our description language should serve two purposes: adaptation and plasticity of user interfaces like introduced in [27]. While enabling us to tailor the user interface for particular devices and particular users (adaptation) it should take the defined constraints into account while preserving usability (plasticity). Our proposition will focus on dynamic plasticity.

Having summarized the benefits of using XML as a user interface description language, it remains an open issue how the user interface will be presented in the XML file, including the constraint definitions. Looking at figure 1, we see that a user interface can be structured as a tree, which is the basic structure of an XML file. We have a main window in which the user interface building blocks like

[1] e*X*tensible *S*tylesheet *L*anguage
[2] *D*ocument *T*ype *D*efinition
[3] eXtensible Stylesheet Language Transformations
[4] *C*ascading *S*tyle*s*heets: a stylesheet for an HTML document

buttons, sliders, etc. are laid out. In the main window we can have other windows containing building blocks, which in turn can be windows. It is advisable to make an abstraction here, like the proposed distinction into abstract interaction objects (AIO) and concrete interaction objects (CIO) [28], presented in figure 1, or to use abstract widgets [10]. An AIO represents an abstract interface widget, independent of any platform. A CIO is a possible "implementation" for such an AIO. Using these concepts allows abstracting the user interface independent of the target platform. Abstract widgets represent practically the same thing: they are abstract platform independent representations of platform dependent widgets. If we want to add runtime layout management taking into account constraints defined by the environment, we will have to dynamically change the presentation of an AIO. This can happen due to screen size limitations for example. [9] tries to solve this problem at design time using an intelligent agent (a mediator) for laying out UI components. For mobile devices this seems too much focused on actual screen-output, because no unconventional output device is taken into account. There might be an embedded system or computing device that has no screen at all, and has only some buttons and speech interaction for example. Then the on-screen data could be converted into a spoken dialog either way. Assuming speech interaction can be stored in XML, we follow the same tree-structure for a speech-enabled dialog as we did for the windows. A full description of conversion to speech interaction is beyond the scope of this paper. We are aware that it is not reasonable to say we can construct complex speech driven dialogs using a runtime conversion technique at this time.

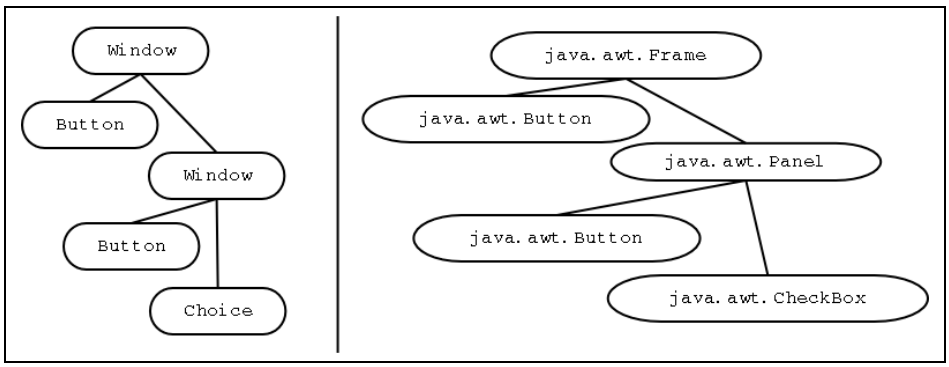

Fig. 1. on the left a contextual representation (AIO), on the right the java.awt classes used to represent the presentation (CIO)

4 Using XML at Runtime

4.1 Runtime User Interface Conversion

As opposed to most modeling approaches, this approach goes one step further: runtime conversion from user interfaces to a user interface description language, XML in this case. In our current approach we have implemented a conversion mechanism for Java user interfaces. There are two well-known user interface toolkits in Java: the oldest one; AWT[5] and a newer and more consistent one: Swing; a subset of the Java Foundation Classes. Because Swing is still too big for most systems with low resources like limited storage capacity and RAM, our implementation is focused on AWT. Using the *Java Reflection Mechanism* all the inspector methods, starting with the prefix `get` are retrieved, and their return values are stored in the XML tree. An advantage of this approach is that not only a user interface can be serialized this way, but all kinds of objects where the state can be retrieved using their inspectors can be serialized. To work with AWT, some ad hoc solutions are required, avoiding unnecessary overhead and circular references. For example, not all inspector methods starting with the prefix `get` are relevant and give useful data. A collection of possible general CIOs is defined, and the matched AWT classes for these CIOs are mapped onto the defined CIO tag names. This process is presented in a simple manner in figure 2. This way the XML description of the user interface preserves the state and stays general enough for converting the XML description into another toolkit like Swing.

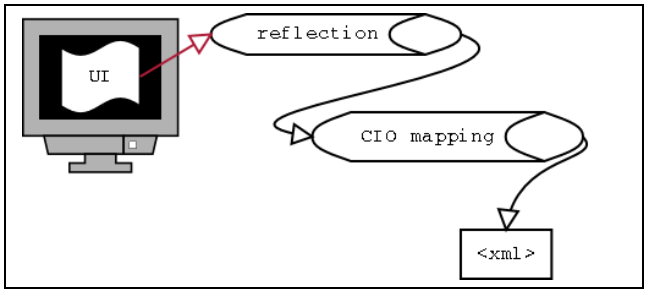

Fig. 2. Serializing a user interface into its XML description

The current implementation supports a serialization of the Java user interface (AWT or Swing) into XML, filtering out the unimportant properties. We presume a user interface can always be hierarchically structured. Our implementation takes advantage of this tree-based structure to recursively descend into the tree, adding widgets to the user interface corresponding to the nodes in the tree and pruning the unwanted branches. A parent node is a widget container

[5] *A*bstract *W*indowing *T*oolkit

for its children this way. For now, we limit the user interface being composed out of JavaBeans, because these have a consistent structure.

Our general approach lacks an important issue: optimizing bandwidth. To optimize the dataflow from source to target, it is a good idea to reduce the size of the XML description. There is a trade-off between several aspects like energy, functionality and time constraints (or performance) [8]. This is why we need to avoid storing redundant or unnecessary data in the user interface description. The method of serializing a Java user interface as presented in the previous paragraph is not smart enough. It needs to make a decision of which data is relevant for the user interface serialization, and which data is not. We have to keep in mind that besides state information and CIOs, constraints must be inserted in the description as well as additional information for layout management, (remote) event handling and type information.

A description of constraints is related to the device it describes. This description should be readable by our system, so it can take decisions how to adapt the transported user interface fulfilling the required constraints. We want the system to be as extensible as possible, so future devices can be easily "plugged in" the existing framework. To accomplish this the description should be as general as possible. It can be described using XML as follows:

```
<Device>
   <Out class="screen">
      <Constraint type="size" data="30*30"/>
      <Constraint type="color" data="8"/>
   </Out>
</Device>
```

This is just a simple example serving the purpose of illustrating the concept; the same can be done for other aspects of IO devices. There are other, more functional and algebraic methods for describing physical devices, but these methods are not further investigated here. At the moment only screen size constraints are taken into account.

Another disadvantage of this approach is the lack of support for other programming languages. In one direction, conversion from XML to an actual user interface, this can be solved easily depending on the maturity of the user interface toolkit, which is targeted. Unlike Java, most other programming languages have no class reflection mechanism, and are much harder to interrogate for their structures at runtime. A conversion from a user interface to its XML description without dedicated data to ease this conversion is rather hard to accomplish. This may imply the need for a supporting framework for XML conversion, inserted into the toolkits built on these programming languages. An example of such a runtime XML conversion of the interface presented in figure 3 is given in the next listing:

```
<Application NAME="test.testUI">
<Property NAME="title">this is an UI2XMLtest</Property>
<Property NAME="name">main-window</Property>
```

Fig. 3. A user interface example

```
<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="TextField">
      <Properties>
        <Property NAME="name">textfield0</Property>
        <Property NAME="selectionEnd">0</Property>
        <Property NAME="columns">20</Property>
        <Property NAME="selectionStart">0</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 listing is kept simple for illustrating purposes. It illustrates how the user interface shown in figure 3 is converted into an XML description. For the buttons, the command to be used when they are pushed is also included in the XML description.

Besides just serializing the properties of the user interface, there must be an indication of which services can be accessed by the user interface, and how these services are connected to the user interface. In fact, we need to connect the services presented by the user interface into actions in the code. There are several ways for doing this: sending the corresponding classes to the remote virtual machine, but this occupies more bandwidth and memory space. A second solution is using a facade that accepts the events and transmits them using a socket connection. Finally, we can use remote method invocation, which connect the events immediately to the remote services. Currently we are investigating the third alternative: using Remote Method Invocation.

4.2 Downloading the User Interface

Now that the user interface is enabled to produce its XML description, we can move the description to another device, where it can be "deserialized" into a user interface for the target device, as presented in figure 4. This deserialization involves mapping the platform independent AIOs on platform specific CIOs. The subsystem responsible for the deserialization has to have sufficient knowledge of the user interface widgets available for the target platform. Notice the scenario in the introduction illustrated this: the persons could use their PDA to retrieve a user interface for operating the projector with. Notice that the structure of the user interface has to stay the same, because the XML file defines it. XML gives us another advantage; it describes the structure but not the look of the program. We can use a kind of stylesheet to adapt the look to a platform. For example, it should be possible to operate the projector by downloading its user interface and using a web browser on a web pad to interact with this user interface.

Once the user interface is set up on the target platform, it has to be fully functional. This requires a kind of remote event handling, like RPC. [12] already proposes a remote user-interface software environment (RUSE) using RPC and a Scheme interpreter. Nowadays there are more flexible architectures possible for this purpose. From a software developer point of view, Java is a perfectly suitable tool to overcome this problem. Java makes it possible to download the code (e.g. applets), make your own Classloaders and use Remote Method Invocation [13,25,26]. The only requirements left are a Java Virtual Machine on the device and network capabilities.

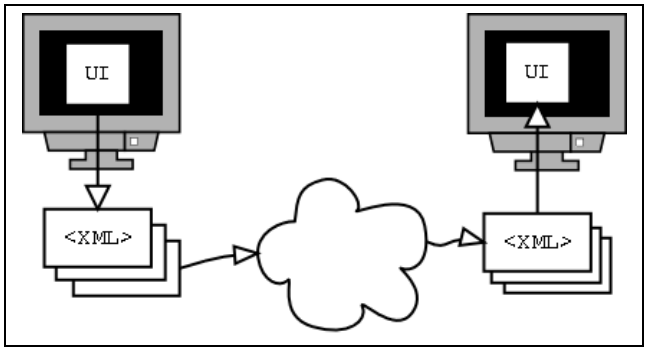

Fig. 4. Downloading a Remote User Interface

5 User Profiling for semi-automatic Layout Management

In the introductory story we have mentioned the use of profiling in function of the part of the user interface, which is downloaded. The scope of this paper is not to investigate profiling of users in depth, only to engage it as a possible tool for a better support for "downloadable" user interfaces and meeting the user's demands. To use profiling, the machine (embedded system, mobile computing device, desktop computer, airplane, cell phone, etc.) has to know who is operating it. This can be very clear, PDAs for example are usually limited to a single user, but in other situations (like a workstation) this requires an identification procedure.

Once the system knows who will be the user of the user interface, it knows which functionality the particular user is most interested in. This is an issue to handle with care: of course the system has to take into account changing preferences of the user. In our scenario, the maintenance man is mostly interested in configuring the projector, but possibly this person wants to test a new configuration by projecting some slides. This is functionality for which there was no indication in the profile, but still must be available to the user.

Making use of a profile can reduce the user interface to be migrated. When using 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. While dealing with certain constraints the profile can help making decisions about omitting a certain subset of the user interface. [9] proposes the generation of an appropriate presentation structure using Logical Windows and Presentation Units where Logical windows group AIOs and Presentation Units group Logical Windows. Using redesign strategies a particular presentation model is composed at design-time. Algorithms for automatic layout management are investigated in [19,14,2]. Combining the user profile with target platform specific constraints could ease the presentation modeling which has to be done at runtime. It certainly can reduce the complexity of possible presentation compositions. Besides this problem, the system should be enabled to switch from interaction modes at runtime.

[10] proposes a designing interface toolkit using abstract widgets for dynamically selecting the interface modality for this purpose.

We propose XSL for describing the user profile. Using the XSLT it is possible to filter out the interested parts for the user. We can make an XSL, which describes the kind of functionality the user is interested in. Letting the XSLT work on a XML structure results in another XML structure only showing the contents of those parts the particular user is interested in. An important part of this conversion is only selecting the relevant paths in the XML user description. This can be done using XPath: an implementation to locate particular branches in the logical tree XML presents. Using an XSL structure we filter out the appropriate section out of the XML description of the user interface. Such an XSL structure can be made by hand, or by monitoring the user's action by an agent, which automatically generates a profile. XSL has declarative properties, which allows us to easily describe what we want, rather than how it has be done. The following simplified example gives an idea of how an XSL structure for the projector user interface introduced in the example in the introduction could look like. It simply specifies it is only interested in properties related to "configuration". When it encounters a configuration property it will call another template making the appropriate actions to preserve that part. We can extend the list of interested parts (by hand or automatically) and create a template for each of them, which will preserve the necessary parts. Notice an XSL structure is also an XML structure [16].

```
<xsl:template match="/">

 <xsl:for-each select="./Property">
    <xsl:if test="contains(Property/@NAME,'configuration')"
      <xsl:apply-template select="./Property"/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>

<xsl:template match="Property">
  ...
</xsl:template>
```

6 Connections

The eight Workshop of *Design, Specification and Verification of Interactive Systems* (DSVIS 2001) provided an excellent opportunity to situate this work in the current research for the specification of user interfaces. [18] indicates the usage of markup languages and provides a sufficient level of abstraction to describe user interfaces in a device independent way. Besides the usage of a model-based approach, it also defines a framework for describing user interfaces in an model-based way. We like to think this works complements the work presented in this paper as it lays out a framework for user interface descriptions using XML.

We also mentioned plasticity in this paper as one of the important properties of a runtime user interface description language. [23] provides a detailed description of how to reach a certain level of plasticity by providing an abstraction of the user interface. Unfortunately the approach taken here is not very suitable for embedded systems and mobile computing devices, but the idea of abstracting the user interface with rich semantic feedback is certainly one of the ideas we also want to introduce in our work for improving the plasticity.

While this work focuses on the runtime serialization and deserialization of user interfaces to make it migratable between different devices and platforms, it does not consider the context in which it will be provided. Whereas there is a distinction between the abstraction of the user interface and the concrete interactors presenting the abstract user interface, the task model is not considered. [21] gives a solution for this: the design of context-sensitive user interfaces. Our work is situated in the last stage; a working user interface, and [21] "extends" the first stage: the creation of a context sensitive model. Our work could be easily combined with the topic presented in [21] because both use a Markup Language (XML) to describe the resulting user interface in a generic way.

7 Conclusions and Future Work

In this paper we have presented a runtime user interface description language, which can cope with constraints found in embedded systems and mobile computing devices. XML seems to be a suitable tool to do this, certainly when it is combined with Java. The usage of XML to generate a user interface description at runtime implies an automatic conversion. Using abstract widgets this can be done in a generic way. To target mobile devices and embedded systems the user interface description has to take the constraints of those particular systems into account. This means the conversion from the user interface description in XML to a system dependent user interface results in a consistent user interface subject to the constraints of the current platform. Users indicate their particular interest in some functionality of a service, by providing a profile (possibly automatically generated).

Future implementations should allow storing more information in the user interface description language (XML in this case). To avoid overhead and waste of bandwidth the description should eliminate redundancy, keeping the user interface description as consistent as possible. Care has to be taken to allow the designer to generate the XML code for the User Interface, without having to build the user interface (as opposed to the runtime method), and choose to generate a specific implementation for it. For getting the actual functionality to migrate with the user interface, a framework for remote event handling has to be provided using RMI[6] or sockets for example.

[6] Remote Method Invocation

Acknowledgments

Our research is partly funded by the Flemish government and EFRO (European Fund for Regional Development). The SEESCOA[7] project IWT 980374 is directly funded by the IWT (Flemish subsidy organization).

References

1. Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. *UIML: An Appliance-Independent XML User Interface Language*. World Wide Web, http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html, 1998.
2. Blaine A. Bell and Steven K. Feiner. Dynamic space management for user interfaces. In *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 239–248, N.Y., November 5–8 2000. ACM Press.
3. K. Bharat and L. Cardelli. Migratory applications. In *Eighth ACM Symposium on User Interface Software and Technology*, pages 133–42, 1995.
4. Keith A. Butler, Robert J.K. Jacob, and Jennifer Preece. *CHI 2000 tutorial notes: HCI: Introduction and overview*. ACM, 2000.
5. Robin Cover. *SpeechML*. World Wide Web, http://www.oasis-open.org/cover/speechML.html, 1999.
6. Robin Cover. *VoxML Markup Language*. World Wide Web, http://www.oasis-open.org/cover/voxML.html, 2001.
7. Robin Cover. *WAP Wireless Markup Language Specification (WML)*. World Wide Web, http://www.oasis-open.org/cover/wap-wml.html, 2001.
8. Maria R. Ebling and M. Satyanarayanan. On the Importance of Translucence for Mobile Computing. In *Proceedings of First Workshop on Human-Computer Interaction for Mobile Devices*, pages 69–72, 1998.
9. 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.
10. Shiro Kawai, Hitoshi Adai, and Tadao Saito. Designing Interface Toolkit with Dynamic Selectable Modality. In *Proceedings of the second annual ACM conference on Assistive technologies*, pages 72–79, 1996.
11. Thierry Kormann. *The Koala User Interface Language*. World Wide Web, http://www-sop.inria.fr/koala/kuil/, 2000.
12. J. Landay and T. Kaufmann. User Interface Issues in Mobile Computing. In *Fourth Workshop on Workstation Operating Systems, Napa, CA*, 1993.
13. 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)*, pages 36–44, 1998.
14. Mark A. Linton, John M. Vlissides, and Paul R. Calder. Composing User Interfaces With InterViews. *IEEE Computer*, 22(2), Febuary 1989.

[7] *S*oftware *E*ngineering for *E*mbedded *S*ystems using a *C*omponent-*O*riented *A*pproach; http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/SEESCOA/

15. 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.
16. Didier Martin, Mark Birbeck, Michael Kay, Brian Loesgen, Jon Pinnock, Steven Livingstone, Peter Stark, Kevin Williams, Richard Anderson, Stephen Mohr, David Baliles, Bruce Peat, and Nikola Ozu. *Professional XML*. Wrox Press, 2000.
17. Sun Microsystems. From desktop to consumer devices; the applet writer's style guide. Technical Report 408-343-1400, Sun Microsystems, JavaSoft, 2550 Garcia Avenue, Mountain View, CA 94043 U.S.A., december 1997.
18. 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.
19. 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.
20. Fabio Paterno. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.
21. Costin Pribeanu, Quentin Limbourg, and Jean Vanderdonckt. Task Modelling for Context-Sensitive User Interfaces. In *Proceedings of the Eight Workshop of Design, Specification and Verification of Interactive Systems*, pages 60–76, June 2001.
22. Pekka Savolainen and Hannu Konttinen. A Framework for management of sophisticated User Interface's Variants in Design Proces. In Jean Vanderdonckt and Angel Puerta, editors, *Computer-Aided Design of User Interfaces II*, pages 205–215. Kluwer Academic Publishers, 1999.
23. Kevin A. Schneider and James R. Cordy. Abstract User Interfaces: A Model and Notation to support plasticity in Interactive Systems. In *Proceedings of the Eight Workshop of Design, Specification and Verification of Interactive Systems*, pages 40–58, June 2001.
24. Ben Schneiderman. *Designing the User Interface, third edition*. Addison Wesley, 1998.
25. 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.
26. Sun Microsystems. *Java Remote Method Invocation*. World Wide Web, http://java.sun.com/products/jdk/rmi/, 1997.
27. David Thevenin and Joelle Coutaz. Adaptation and Plasticity of User Interfaces. In *Workshop on Adaptive Design of Interactive Multimedia Presentations for Mobile Users*, 1999.
28. 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.

