Adaptive User Interface for Mobile Devices*

Nikola Mitrović[1] and Eduardo Mena[2]

[1] IIS Department, University of Zaragoza, Maria de Luna 3,
50018 Zaragoza, Spain
mitrovic@prometeo.cps.unizar.es
[2] IIS Department, University of Zaragoza, Maria de Luna 3,
50018 Zaragoza, Spain
emena@posta.unizar.es
http://www.cps.unizar.es/~mena

Abstract. Adapting a graphical user interface (GUI) to a variety of resources with different capabilities is one of the most interesting questions of today's mobile computation. The GUI constructed for one application should be usable on different interactive devices, e.g. WebTV terminals, WAP phones or Java-enabled devices. In this paper, we discuss existing solutions and present a solution based on mobile agents. Mobile agents construct their GUI using third-party eXtensible User interface Language (XUL), jXUL middleware and XSL transformations. Mobile agents move to host computers and then build their GUI, or act as a proxy to devices without sufficient processing capabilities (e.g., WAP devices). The result is an adaptable GUI platform that can be run on multiple devices without modifications, supporting different resources and architectures. We show the application of this approach by implementing a mobile currency converter and survey.

1 Introduction

Constructing graphical user interfaces (GUIs) in mobile computing area faces many challenges. Main problems are raised from the fact that various target devices have different processing powers, GUI organization and capabilities.

Solutions in this area mainly focus on web applications with client-server architecture, creating specialized and centralized services that transform one type of user interface in another. Some solutions propose creating separate GUI solutions for each device type, that are later dispatched according to the request type (or request origin). Some authors propose XML-described user interfaces that could be later presented as Java AWT [1] or Swing [1], or that can be transformed with XSLT [2].

The idea of this work is to transparently adapt graphical user interface by using mobile agent systems. Agents are highly mobile, and are often hosted by platforms that support different models of user interface or have different processing capabilities. Agents are autonomous, and can handle network errors

* This work was supported by the DGA project P084/2001.

(unreachable hosts, etc.) autonomously; also, they can move to the target device instead of target device requesting service from a server. Agents can be sent to a home computer supporting Java and Swing. On the other side, an agent can play the role of a proxy server for a wireless device, such as mobile telephone or a Web terminal, and in that case it should produce WML [4] or HTML [3], respectively. In contrast, solutions not using mobile agents are client-server systems or use middleware programs that are installed on each user device. Therefore, new updates lead to reinstalling client programs on every user device, which does not happen when using mobile agents (only the mobile agent needs to be updated).

Our prototype adapts user interface using mobile agents [8] that process a user interface definition described in a language called Extensible User-interface Language (XUL) [5], [13]. This interface definition is later adapted using XSL transformations to other notations (HTML, WML, etc). The XUL interpretation on Java-enabled platforms is interpreted by jXUL platform. The jXUL is a third-party middleware that renders XUL using standard Swing interface. Agents automatically adapt the interface definition to the clients' interface, making multiple middleware implementations unnecessary.

This approach gives good results when deployment is needed not only as a web application, or only as a desktop application. This approach combines these approaches, and is truly mobile in its nature; agents can autonomously determine what kind of interface should be presented. One of the advantages of mobile agents approach is that the GUI goes mobile and can be constructed in function of autonomous operation of mobile agent. For example, we could have a user interface that is modified depending on information collected from the agent's trip on the network, and that can be later presented on any type of the device.

The rest of this paper is as follows. Section 2 gives an overview of state of the art and the related work. Section 3 introduces extensible user interface language (XUL) and gives an overview of its possibilities and limitations. Section 4 introduces mobile agent technology. Section 5 we introduce our motivating example and explain bound between mobile agents and GUI. Section 6 describes sample scenario that shows the presented technique. Section 7 concludes the paper and discusses the future work.

2 State of the Art and Related work

Various approaches to adapting user interfaces to various devices are present. Basically the approaches are grouped into two categories: web applications and classic desktop applications. While the first category [11], [14] treats only web content and transformations of web content in order to be usable on other (mostly mobile) devices, the second category treats the problems of universally defining the user interface, so it can be later reproduced by various program implementations [9], [10], [12], [24], [25] (or middlewares) on various platforms.

2.1 Adaptable XML-defined interfaces

Several solutions for defining user interface are present at the moment of writing of this article. Without providing details, we mention some approaches: language-based, grammar-based, e.g., BNF, event-based, constraint-based, UAN (User Action Notation, in particular for direct manipulation) and widget-based. However, the XML-based efforts are most interesting for us, since they provide flexibility and easy manipulation. Some of such efforts include XUL [5], the extensible user interface language, UIML (User Interface Mark-up Language) [26], [10] and XIML [32].

Luyten, et al. [24] investigated the possibility of rendering their own XML-like mark-up languages to Java AWT and Swing, or converting the interface definition to other formats using XSL transformations or XPath. However, this approach is focused on creating different middleware (or transformation) for different platforms (that are not Java compatible), and not on transparent modification of user interface. Also, their prototyped solution do not run in a truly mobile environment (mobile agents), and is focused on rendering the user definition files on multiple platforms by using different middlewares. At the time being, this prototype also lack complete language definition.

Other approaches, such the one from Meller, et al. [25], are more focused on defining the universal XML notation that can be used for platform independent interface generation.

2.2 Web applications and adaptable user interfaces

Application servers are mostly oriented on how to transform web contents to various other formats that can be used on mobile devices (cHTML [30], WML [4], etc.). However, different approaches exist.

Microsoft, one of the industry leaders, in its next-generation technology ".NET" offers Mobile Web Forms [11]. These forms are based on restricted set of components that, to our knowledge, cannot be extended with additional widgets. Each component is intelligent component that transforms its appearance in function of available resources. The controls are highly bound with the .NET family of languages. Unfortunately, Microsoft's solutions are still available only on the Windows platforms, the number of widgets is limited, and the desktop applications are not taken into the consideration.

Other industry leaders, such as IBM, have slightly different approaches. IBM's Transcoding Publisher [14] actually transforms web contents to variety of other formats, giving the user possibility of customisation of the transformation parameters. Some interesting features such as JavaScript [7] transformation and automatic image format transformations are included. Users should be able to customize the transformations in order to maximize the quality the output, which can be a significant plus for complex web applications; another good side is IBM's commitment to Java, therefore multiple platforms are supported. However, the drawback of the approach is ability to transform only web contents, and in a centralized fashion.

Other "traditional" solutions in the web-area also exist, and consist on parsing the request information, and redirecting the petition to the appropriate content [15]. The content is created separately for each device type, and is stored separately. When the user accesses the server with a mobile device, the server will recognize the request type, and will redirect the user to the appropriate content. This solution has a significant overhead, because the content should be created multiple times in order to support different formats. Scalability of this solution can be also questioned.

All these approaches support different level of customisations, but however only web applications. The user interface generation is centralized – on the server.

3 Extensible User-interface Language - XUL

Extensible User interface Language [5], [13] is designed for cross-platform user interface definition. This language is incorporated in Mozilla project [17], acting as a user interface definition language. Being part of Mozilla project, XUL is open and connectable to other Mozilla projects. The format is organized with modern user interface definition in mind, supporting variety of available controls.

XUL lacks the abstraction layer of interface definition, and is restricted to window-based user interface. It is capable of referencing Cascading Style Sheets (CSS) [18] to define the layout of elements. The user actions, property access and functionality can be stored in JavaScript (ECMAscript) [7] files. However, we found XUL as suitable open source solution for our purpose.

A simple XUL window in Fig. 1 could be defined as in Fig. 2.

Fig. 1. Window to be constructed

From this example, we can see that the interface definition is oriented to modern window-based interfaces. We are referencing a StyleSheet, JavaScript library, and using few labels, textbox and a button within the box tag. The box tag is main form of layout in XUL and is similar to Swing JPanel. This model allows you to divide a window into a series of boxes. Elements inside box will orient themselves horizontally or vertically. By combining a series of boxes,

```
<?xml version="1.0"?>
<window align="vertical" class="dialog" height="250" width="370" title="Currency
Converter">
 <link rel="stylesheet" href="html.css" type="text/css"/>
 <script language="JavaScript" src="eventHandlers.js"/>
 <box>
  <label control="lblTitle" value="Currency Converter"/>
 </box>
 <box>
  <label control="lblQty" value="Quantity:"/>
  <textbox value="0.00" id="txtQty"/>
 </box>
 <box>
  <button id="Convert" label="Convert!" oncommand="ccyConvert()"/>
 </box>
</window>
```

Fig. 2. Example XUL document

spacers and elements will flex, and you can control the layout of a window as can be seen in Fig. 1.

4 Mobile Agents and Agent Platforms

A mobile agent [8], [27] is a program that executes autonomously on a set of network hosts on behalf of an individual or organization. The agent visits the network hosts to execute parts of its program and may interact with other agents residing on that host or elsewhere, while working toward a goal. During their lifetime agents travel to different hosts, that can have distinct user interface possibilities. Agents typically posses several (or all) of the following characteristics; they are:

- Goal oriented: they are in charge of achieving a list of goals (*agenda).*
- Autonomous: they are independent entities that pursue certain objectives, and decide how and when to achieve them.
- Communicative/collaborative: to achieve their goal they can cooperate.
- Adaptive/learning: agents "learn" from their experience and modify their behavior respectively.
- Persistent: agent's state (should) persist until all the goals are achieved.
- Reactive: they react to their environment which also could change their behavior.
- They can stop their own execution, travel to another host and resume it once there.

They do not, by themselves, constitute a complete application. Instead, they form one by working in conjunction with an agent host and other agents. Many agents are meant to be used as intelligent electronic gophers – automated errand boys. Tell them what you want them to do – search the Internet for information on a topic, or assemble and order a computer according to your desired specifications – and they will do it and let you know when they have finished. Mobile

Agent Systems (MAS) are the middleware that allows creating and executing mobile agents. For this project, we choose Grasshopper [19] as the most intuitive and stable mobile agent platform, which supports standards such as FIPA [20], CORBA [21] and RMI [22]. In addition, the Grasshopper's feature Webhopper [19] that enables mobile agents for web is a significant plus comparing with other platforms, like Voyager and Aglets [31].

5 Using XUL with Mobile Agents in Multiple Platforms

The idea of this work was to use XUL together with the mobile agent paradigm [8], and to make a prototype that adapts XUL for hosting platform or for remote devices (e.g. a wireless device). By achieving this, one will have a truly mobile user interface that adapts to the platform on the fly.

5.1 A Motivating Example

We present these sample applications, meant to demonstrate the possible every day uses of a mobile agent that adapts its user interface to multiple devices.

The first example is a currency converter application that can be accessed from every point on the network. This application converts among three currencies (Euro, US Dollar, British Pound). We want this application to be accessible from various different devices (Java, WAP phone, web terminal). In all of these cases, the same application should be started, and the same (or equivalent) user interface should be used in order to reduce costs of application development.

The second example is a survey application. It should make a poll of the converter application users, calculate the stats, and return the data to the software company that built the converter application. Similarly to the converter application, we expect users that are taking a survey to have all sorts of devices, different connection types, and possibly problems with network coverage/links. The application ask users to rate the converter application with three possible answers (good, normal, bad), and calculate stats on the answers. All the answers are persisted so the statistics are made on all-times data. In case of loss of network coverage or broken network links, application should re-intent connection or try alternate route to the next host without prompting user.

In Section 6 we describe in detail these sample applications.

5.2 XUL implementation - jXUL

jXUL [6] is a Open Source project that interprets XUL definition and renders it to Java Swing interface, similarly to [16]. Plans for jXUL are very ambitious, aiming to support very complex controls in future releases. Other open source projects that aim to rendering XUL with Java or to DHTML exist [28]; unfortunately, at the time of writing this paper, none of these has any public prototype available.

However, available jXUL implementation lacks basic functionality, such as assigning and getting values from components or ability to connect outer classes to the JavaScript engine that runs within jXUL. Therefore, we put significant effort into redesigning the existing components to support basic functionality, and extended the JavaScript engine functionality by adding connector classes that can be externally connected to jXUL middleware. Unfortunately, jXUL is built to render only to Java Swing, and not to other types of interfaces, so we had to develop for our prototype XSL transformations that transform XUL files to HTML and WML files.

5.3 Putting it all together

The prototype built customizes mobile agents in such manner that programming a system that has adaptable user interface is almost completely transparent; programmers have only to extend the required class, connect the classes and to create interface definition files. This approach combines adaptivity with respect to alocating system functionality and adaptivity with respect to interface layout. However, level of plasticity [33] is basic and will be improved in the future work. The base classes will convert the XUL files to appropriate format and handle the communication.

Thus, we created a simple currency converter application that uses a few basic controls: labels, text boxes, radio buttons and classic buttons. In order to construct this application we need some XUL files (one for each window). The sample XUL file that we created for the sample currency converter is shown in Fig. 3.

After constructing the user interface definition, the worker class should be created. This class should carry all procedures that handle interface events, but the computation is not limited to this class. Because of Grasshopper limitations, we had to create this class as a connector class, to be used from jXUL's JavaScript, and therefore to be accessible from the user interface.

The structure of the sample method that we implemented for currency converter agent is shown in Fig. 4.

Code in Fig. 4 is used for any interpretation of XUL files; no modifications for any platforms should be made. As we can see, this method takes three parameters that are passed from the GUI and then process the request. While processing, the window is closed, and when the result is calculated it is opened again.

What we have created is a mobile agent that transforms itself into three forms: Java Swing, HTML or a WML application, depending on the user device capabilities. Also, the agent is acting both as a server and as an application at the same time - if the originator cannot accept mobile agents (e.g, wireless devices), the agent will act as a content server to that device. However, if mobile agents are accepted, the agent will act as standard application.

6 A sample scenario: mobile calculator and survey

For our prototype we have set up the network consisting of five network nodes:

```
    <?xml version="1.0" encoding="ISO-8859-1"?>
<!-- global window settings and JavaScript link -->
    <window align="vertical" height="255" width="410" title="Converter">
    <script language="JavaScript" src="Handler.js"/>
<!-- title label -->
    <box> <label control="lblAll" value="Currency Converter"/> </box>

<!-- inserting the quantity edit box -->
    <vbox>
      <hbox>
        <label control="lblQty" value="Quantity:"/>
        <textbox value="0.00" id="Qty" size="20"/>
      </hbox>
    </vbox>
 <!-- inserting the From radio group -->
    <box>
       <label control="lblFrom" value="From:    "/>
       <radiogroup orient="vertical" id="From" selected="Usd">
         <radio id="Eur" label="Euros"/>
         <radio id="Usd" label="US Dollars"/>
         <radio id="Gbp" label="British Pounds"/>
       </radiogroup>

       <label control="lblTo" value="To:    "/>

<!-- inserting the To radio group -->
       <radiogroup orient="vertical" id="To" selected="Eur">
         <radio id="Eur" label="To Euros">
         <radio id="Usd" label="To US Dollars"/>
         <radio id="Gbp" label="To British Pounds"/>
       </radiogroup>
    </box>

<-- label that will be used for the Output -->
    <box>
      <label control="lblOutput" value="Result:  "/>
    </box>
    <box>
      <label id="Output" control="Output" value=""/>
    </box>

<!-- adding button -->
   <box>
      <button id="Convert" label="Convert" oncommand="convert()"/>
    </box>
  </window>
```

Fig. 3. XUL definition used in currency converter application

- The DesktopNode is a Java-enabled fixed computer that can render Swing; it is able to host mobile agents.
- The WebNode is a network terminal that can render only HTML. This node cannot host mobile agents.
- The WapNode is a mobile phone with Wap browser that can render WML; this node has a wireless connection to the network and cannot host mobile agents.
- The LaptopNode is a wireless laptop, it is Java-enabled, that can host mobile agents.
- The CorporateNode node is server computer, that hosts agents and can render Java Swing. Provides users with our sample applications implemented

```
public void convert(String From, String To, String Value) {
    // do the initialization

    //close the window
    _agent.closeWindow(_agent.Window);
    try
    {
       //do the computation
    }
    catch (Exception e)
    {
       //handle exception
    }
    //open the new window
    _agent.openWindow(_agent.Window.displayFile, this, "Output", result);
}
```

Fig. 4. Agent code attached to the currency converter interface

as mobile agents. This node also serves as a server for WebNode and WapNode, since these nodes were assumed not to have possibility of running Java. Of course, this is the worst-case scenario, since there is emerging number of mobile devices that run Java.

6.1 Currency converter application

The objective of the currency converter application (described in Section 5.1) was to demonstrate the adaptive interface concept. This application adapts its appearance to the originator of the request. As we can see in the Fig. 6, if DesktopNode or LaptopNode invoke the application, the mobile agent (application) moves there and then it will render the XUL files as Swing.

However, if the WapNode or WebNode invokes the application, a different action will occur. Since these nodes were cannot host agents the CorporateNode acting as server will process their requests.

As we can see in the Fig. 5, we have a mixed architecture. Clients that cannot support mobile agents (WebNode, WapNode) are using CorporateNode as server for their petitions, and therefore client-server architecture is present. However, our currency converter agent travels from CorporateNode to DesktopNode and LaptopNode, using mobile agent architecture.

In Fig. 6, Fig. 7 and Fig. 8, we show how the application looks if invoked from those three platforms. From the Fig. 6 and Fig. 7 we can see that the Swing and HTML outputs are not exactly the same. HTML output for this example could be improved by using tables, but we decided to use simple XSL transformations.

Fig. 8 shows the WML output on the M3Gate WAP browser simulator [29]. As we can see, the output differs significantly from the HTML and Swing outputs as device capabilities and rendering language are different and more limited. For

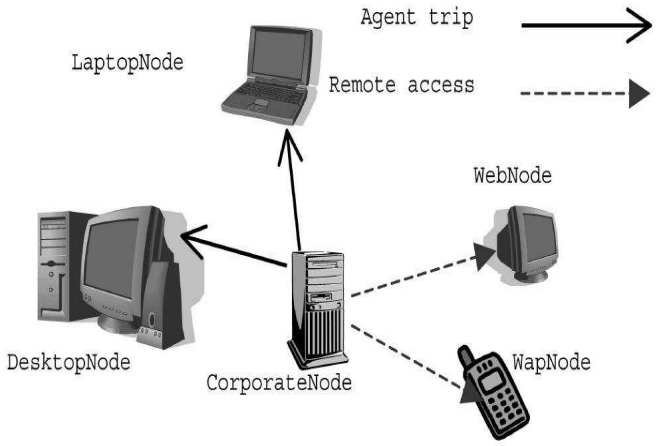

Fig. 5. Network topology and mobile agent trajectories for the currency converter application

Fig. 6. Currency converter agent rendered with Java Swing

example, as seen in Fig. 8, radio groups are initially presented as widgets (Fig. 8, on the left) that are later expanded to a full-screen selection (Fig. 8, on the right).

6.2 Survey application

This application (explained in Section 5.1) shows benefits from mobile agent computing. Agents are autonomous, adaptive, learning, and mobile; survey application demonstrates these properties. The survey agent travels through the network, visiting the hosts that used our currency converter application. When it reaches the destination host, it transforms its appearance in the suitable form

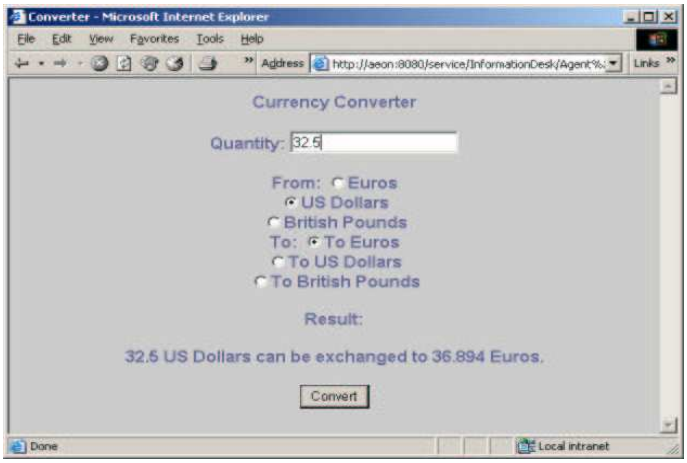

Fig. 7. Currency converter agent rendered as HTML

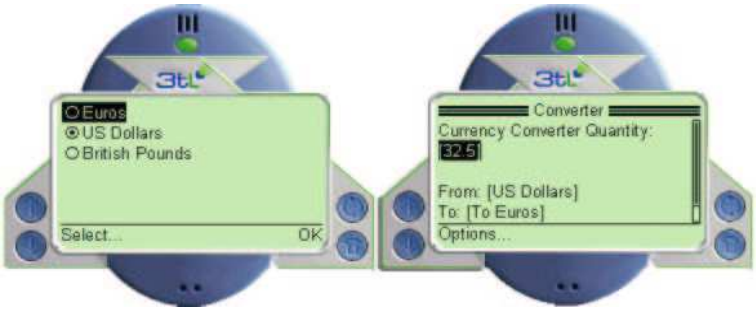

Fig. 8. Currency converter agent rendered as WML and the different output of Radio Group

to ask users for their opinion about the currency converter application. If the network host is unavailable or unreachable, the agent will autonomously decide what to do next. It could wait for host to be available, or continue with the other hosts and return later to the unavailable hosts. Statistics on collected data is calculated.

Fig. 9 shows the network topology that we established for this example. In this Figure we can see that survey application travels from CorporateNode to Desktop-Node and to the LaptopNode. Since the LaptopNode has wireless connection to the network, this link can be broken, and the agent will decide how to reach this node without reporting an error. WebNode and WapNode as we discussed have no processing power, therefore they are served from the CorporateNode. When these hosts complete the survey, the survey application returns to the CorporateNode to deliver results and statistics of the poll.

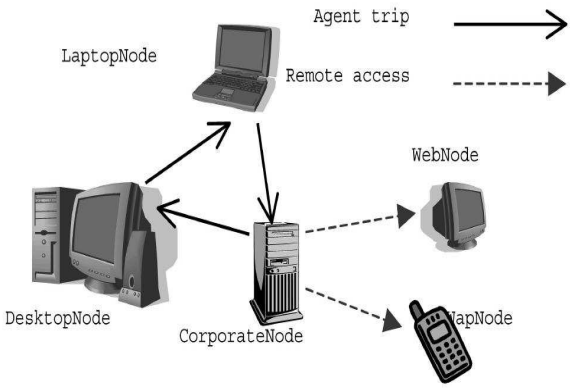

Fig. 9. Network topology and mobile agent trajectories for the survey application

In Fig. 10 and 11 we can see the appearance of the survey application, rendered for Swing and WML clients. The HTML output of this application is very similar to Swing output.

We can see that the agent is not just persisting the survey data, but in fact is calculating statistics based on current data. This distributes the processing among client nodes. There is an open possibility of taking special action depending on survey results. For example, survey agent could return home when it reaches 100 surveys.

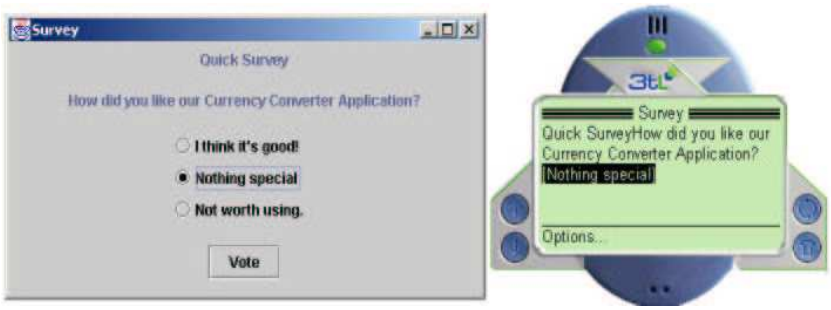

Fig. 10. Survey question, rendered as Swing and WML

Notice that the survey application is a "push" service. The user does not have to request a survey from some central host - agent will visit the client by its own initiative. This kind of service is possible when the user platform supports mobile agents. The users of currency converter application do not need to know about the existence of a survey about the converter application. As we can see from

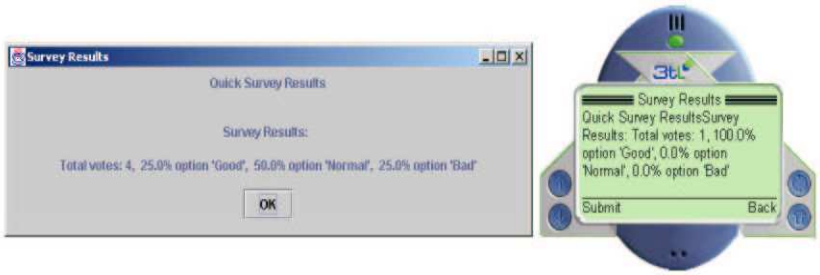

Fig. 11. Survey results, rendered as Swing and WML

the Fig. 9, when the survey is launched, the agent will visit their computer and will pop-up the survey.

Another example application for our approach would be an automated software update agent that could visit hosts, check present software versions and advise users on possible updates.

7 Conclusions and Future work

In this paper we have presented an autonomous and mobile system, based on mobile agents, that transparently adapts GUI to the users. The main features of this approach are:

- We use and extend third-party middleware jXUL to render XUL to Swing.
- We have made simple XSL transformations in order to convert XUL to other mark-up languages, as HTML and WML.
- A Mobile GUI agent was prototyped, that transparently converts XUL-defined user interface to Java, Web and WAP devices.
- Our approach could be used for other mark-up languages similar to XUL.

This system is stable, gives required functionality, and because of it is mobile agent-based, it is very interested for its application to mobile environment.

However, this work has some limitations. Some platforms do not support elements that can occur in the user interface definition, such as image or sound. Design considerations should take place, and this is the area that should be investigated as a continuation of this work. Thus, our future work will be focused on:

- Transformations for more different outputs
- More efficient transformations, with back-end resource definitions
- Direct binding of XUL to Java without using JavaScript
- Data marshalling using XML metadata

References

1. Horstmann, C.S., Cornell, G.: Core Java 2, Volume 1: Fundamentals, Prentice Hall, 2000.
2. XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999, http://www.w3.org/TR/1999/REC-xslt-19991116
3. HTML 4.01 Specification, W3C Recommendation 24 December 1999, http://www.w3.org/TR/html401/
4. WAP-WML Specification Version 1.1, 16 Jun 1999, Wap Forum, http://www.wapforum.org/
5. XUL Tutorial, http://www.xulplanet.com/tutorials/xultu/
6. jXUL, http://jxul.sourceforge.net
7. ECMAScript Language Specification, 3rd Edition, December 1999, ECMA, http://www.ecma.ch/ecma1/stand/ecma-262.htm
8. Distributed Objects & Components: Mobile Agents, http://www.cetus-links.org/oo_mobile_agents.html
9. Mueller, A., Mundt, T., Lindner, W., Cap, C.H.: Platform Independent User Interface Generation with XML, ISAS-SCI (1) 2001: 299-304
10. Stttner, H.: A Platform-Independent User Interface Description Language, Technical Report 16, Institute for Practical Computer Science, Johannes Kepler University Linz, Austria, March 2001.
11. Microsoft Corporation, Creating Mobile Web Applications with Mobile Web Forms in Visual Studio .NET, http://msdn.microsoft.com/vstudio/technical/articles/mobilewebforms.asp
12. XML-MT GUI Definition Language, http://www.mieterra.com/documentation/X-ML-MT.html
13. Cheng, T.: XUL - Creating Localizable XML GUI, Fifteenth Unicode Conference, 1999. http://www.mozilla.org/projects/intl/iuc15/paper/iuc15xul.html
14. IBM WebSphere Transcoding Publisher, http://www-3.ibm.com/software/webservers/transcoding/
15. Hild, S.G., Binding, C., Bourges-Waldegg, D., Steenkeste, C.: Application hosting for pervasive computing, IBM Research, 2000, http://www.research.ibm.com/journal/sj/401/hild.html
16. Olsen, D.R.Jr, Jefferies, S., Nielsen, T., Moyes, W., Fredrickson, P.: Cross-Modal Interaction using Xweb, UIST, 2000.
17. Mozilla project, http://www.mozilla.org
18. Meyer, E. A.:Cascading Style Sheets: The Definitive Guide, O'Reilly and Associates, 2000.
19. Grasshopper, IKV, http://www.grasshopper.de/
20. Foundation for Intelligent Physical Agents, http://www.fipa.org
21. Pope, A.: The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture, Addison-Wesley Pub Co, 1998.
22. Java Remote Method Invocation, http://java.sun.com/products/jdk/rmi/
23. Resource Description Framework (RDF), W3C Specification, http://www.w3.org/RDF/
24. Lyten, K., Coninx, K.: An XML Runtime User Interface Description Language for Mobile Computing Devices, DSVIS, 2001.
25. Mller, A., Forbrig, P., Cap, C.: Model-Based User Interface Design Using Markup Concepts, DSVIS, 2001, 16-27.

26. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E.: UIML: An Appliance-Independent XML User Interface Language. WWW8 / Computer Networks 31(11-16): 1695-1708 (1999)
27. Milojicic, D.S.: Trend Wars: Mobile agent applications. IEEE Concurrency 7(3): 80-90 (1999)
28. SourceForge Network, www.sourceforge.net
29. Numeric Algorithm Laboraties, www.m3gate.com, M3Gate WAP Simulator, 2001.
30. Compact HTML for Small Information Appliances, W3C Note 09-Feb-1998, http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/
31. The Mobile Agent List, University of Stuttgart, http://mole.informatik.uni-stuttgart.de/mal/mal.html
32. XIML (eXtensible Interface Markup Language), http://www.ximl.org/
33. Thevenin, D. and Coutaz, J.: "Plasticity of User Interfaces: Frame-work and Research Agenda", Proc of IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'99, Edinburgh, August 1999, IOS Press, 1999.