Running head: ADAPTIVE INTERFACES IN MOBILE ENVIRONMENTS

Adaptive Interfaces in Mobile Environments - An Approach Based on Mobile Agents

Nikola Mitrovic
IIS Department
University of Zaragoza
Maria de Luna 1
50018 Zaragoza, Spain
mitrovic@prometeo.cps.unizar.es

Eduardo Mena
IIS Department
University of Zaragoza
Maria de Luna 1
50018 Zaragoza, Spain
emena@unizar.es

Jose Alberto Royo
IEC Department
University of Zaragoza
Maria de Luna 1
50018 Zaragoza, Spain
joalroyo@unizar.es

ABSTRACT

Mobility for graphical user interfaces (GUIs) is a challenging problem, as different GUIs need to be constructed for different device capabilities and changing context, preferences and users' locations. GUI developers frequently create multiple user interface versions for different devices. The solution lies in using a single, abstract, user interface description that is used later to automatically generate user interfaces for different devices.

Various techniques are proposed to adapt GUIs from an abstract specification to a concrete interface. Design-time techniques have the possibility of creating better performing GUIs but, in contrast to run-time techniques, lack flexibility and mobility. Run-time techniques' mobility and autonomy can be significantly improved by using mobile agent technology and an indirect GUI generation paradigm. Using indirect generation enables analysis of computer-human interaction and application of artificial intelligence techniques to be made at run-time, increasing GUIs' performance and usability.

INTRODUCTION

Mobile computing is an increasingly important topic in today's computational environment, because the demand for ubiquitous access to information is constantly increasing. Furthermore, users want to increase their efficiency and process information when using mobile equipment. To support this demand, software applications face a number of challenges. One of the important challenges in mobile computation is user interaction.

The importance of user interfaces (UI) comes from the fact that UIs represent the first line of interaction between a user and a computer. A user's ability to execute a required task and his efficiency are directly impacted by the user interface.

In the past, user interfaces have been developed mostly for a specific device - e.g., a specific PDA variant or a work station. Such an interface was usually designed for a single platform; this was done in conjunction with specialized user interface libraries that were defined for a specific platform or programming language. For example, if we assume an application that is developed for Windows and UNIX platforms, the user interface for the windows platform would be designed and developed separately from the UNIX user interface. The cost and effort required for such a development are obviously high; such an approach frequently leads to other problems, e.g., GUI implementations on one (or more) platform(s) being at different levels of development due to the lack of resources required to maintain the same GUI version on multiple platforms.

In the mobile environment GUIs face additional challenges: a user could be using an application on a mobile phone and could require the same application on his PDA or WebTV. In addition, the user could be moving and requiring an application to move with him. For example: while in the car, a user could read his e-mail using car's on-board computer; when he steps out of the car he could prefer to continue working on his PDA until he gets to the office, where a desktop PC could be his preferred equipment to continue working.

Mobile devices have different capabilities and requirements: different processing power, screen size, supported colors, sound functionalities, keyboard, and so forth. In addition, mobile devices use an ever-increasing number of different hardware and OS solutions, and frequently rely on batteries for operation. Mobile applications use wireless networks; wireless networks are not stable, have limited capacity and performance, and are expensive (e.g., 3G networks).

In addition to this, application interface and functionality may change depending on a user's context. For example, a music player application should mute if the user is indoors and should turn on when outdoors. Furthermore, a user could prefer the speaker to be on a louder setting when in the car. These requirements could be either a user's preferences or rules associated with a particular location where the user is.

To meet such challenges researchers in the user interface area have adopted a common approach – user interface abstraction. To be presented on a concrete platform, abstracted user interfaces are transformed and rendered to meet a concrete platform's requirements. This approach provides a single user interface definition that is later transformed to the target device's user interface. The abstraction level in such an abstract user interface definition varies. Some abstract user interface notations offer very abstracted descriptions of user interfaces, while others are more linked to specific user interface concepts – e.g., window-based user interfaces. An abstract user interface definition is usually delivered in XML (W3C, 2000) notation, which enables efficient processing and data exchange between multiple platforms. Some notations describe the user interface at a high level, for example, a button is required; others allow sophisticated definitions of constraints and additional parameters, such as requiring the button for some (specific) device(s) only.

This chapter presents different approaches to adapting user interfaces to devices with specific interest on enabling architectures that adapt to users' preferences and context. We discuss difficulties with mobile user interface generation for wireless devices. Finally, we present an approach for user interface adaptation based on mobile agents and examine sample usage of such an approach. With this example we show a flexible and mobile generation of user interfaces in a wireless environment that allows monitoring of user interaction and application of knowledge.

BACKGROUND

User Interface Abstraction

Abstraction of user interfaces adds flexibility when generating user interfaces. It provides a single and comprehensive description of the user interface. It is a rich layer of information that describes a user's interaction with the computer. Such abstraction is formalized by using an abstract user interface definition language (Stottner, 2001) or task models (Limbourg & Vanderdonckt, 2003). This information can then be used to generate a user interface that meets a concrete platform's limitations and requirements. Such an approach develops a single set of information to support all variants of the user interface that should be created for different devices.

The user interface is typically abstracted through the use of design models (e.g., task models) or by using an XML-based abstract user definition language (Molina, Belenguer & Pastor, 2003; Luyten & Coninx, 2001; Mitrovic & Mena, 2002; Stottner, 2001). Task models (Limbourg & Vanderdonckt, 2003) provide information that is focused on tasks. Some of the task models can describe multi-modal tasks for different types of devices (Paterno & Santoro, 2002). However, task models do not necessarily specify the exact presentation of a user interface. On the other side, XML-based abstract user interface descriptions describe the user interface's presentation and constraints. Many different versions of such abstract user interface definition languages exist.

Such approaches' abstraction levels vary: some approaches include information specific to a device type (e.g., a mobile phone, or a specific mobile phone model), some are more generic and do not consider the specifics of any device type. In addition, user interface abstraction can differ conceptually – some models can define any kind of interaction (e.g., via voice or specialized interfaces), while some are more linked to specific concepts (e.g., window-based user interfaces). Examples of such abstraction languages include XUL, UIML, XIML and XForms (Stottner, 2001).

The ability to effectively adapt such user interface definitions to a concrete platform is a key factor in achieving mobile and efficient user interfaces. The resulting, concrete user interface must meet the specification and be functional on the target device. This requires that device capabilities and limitations be successfully addressed. In addition, user preferences and context frequently impact this adaptation.

Abstract User Interface Adaptation

User interface adaptation is a complex task, and includes not only adaptation to the specific device's capabilities, but also to the user. Mobile devices have different capabilities such as screen size, keyboard and support for particular user interface widgets, hardware platform or operating system. Adaptation to a user includes adaptation to the user’s preferences and changing contexts, but sometimes includes factors such as previous knowledge or location.

Platforms may have exceptionally different user interface capabilities and requirements (see Figure 1). In many cases adapting a user interface simply as a per user interface specification is not sufficient. For example, a combo-box widget as specified in the user interface description may be available on a particular platform, or not; there could be a similar widget or this widget should be transformed into a set of different widgets. To address this and to maintain the user interfaces' plasticity (Thevenin & Coutaz, 1999) additional adaptation effort is required. User interface plasticity is a user interface's capacity to preserve usability regardless of variations in the system hardware specification or operating environment.

Adapting to users includes a wide range of considerations: users’ preferences, context, location, ambient environment, etc. This is a more complex transformation than adaptation to a device as it requires knowledge about the user. The key to this adaptation is the ability to understand the user’s actions, location and other parameters of interest, and to apply different artificial intelligence techniques to such information. To achieve this goal it is very important to gather information surrounding the user, but also to gather information coming from the user himself. This information coming from the user could be in the form of predefined preferences (e.g., color scheme preference) but, more importantly, the data about interactions between the user and his computer are of great value for our purposes. By observing and analyzing the user's interaction, his corresponding interface could be modified according to the type of interaction (e.g., using stylus or keyboard), tasks previously executed by the user, or tasks executed by other users.

Figure 1. Some of the target platforms for mobile applications (two versions of smart phone and a web TV)

Adapting to device capabilities requires a program capable of processing or interpreting the abstract user interface definition in such a way that it can be rendered to a particular device. Two major approaches are used to achieve this:

- Using a design-time tool to create concrete user interfaces for the required platforms
- Using a run-time tool or process to render a concrete user interface

Design-time tools are closer than run-time techniques to traditional user interface development and frequently encompass initial modeling, analysis and later development of multiple user interfaces. Such approaches include design patterns (Seffah & Forbrig, 2002), task models (Limbourg & Vanderdonckt, 2003; Paterno & Santoro, 2002; Molina et al, 2003), or off-line analysis of user interaction (Pitkow & Pirolli, 1999). This approach provides good facilities for designing static user interfaces, but is challenged by mobility and unanticipated situations.

The design-time tools are usually specialized stand-alone tools used by the user interface or software developer. Run-time tools are usually more generic mechanisms built into the application or programming framework.

Run-time tools offer higher flexibility than design-time tools and reduce effort for user interface developers. However, they are less capable of producing fine-tuned user interfaces, which can be produced by design-time methods on a case-by-case basis. The run-time approach is better suited for applications that are accessed remotely or applications that require execution on mobile devices with varied GUI and processing capabilities.

In the following sections we will present the benefits and drawbacks of both design-time and run-time approaches, with a particular emphasis on techniques used to adapt user interfaces to various devices.

Adaptation to Devices

Adapting user interfaces to devices requires a program capable of processing an abstract user interface definition to a device's capabilities. Such a program must adapt the abstract specification to the target (concrete) user interface specification. The concrete user interface specification must represent user interface design well and remain functional on the target device.

Two major approaches are used for such adaptations:

- Design-time: this encompasses using a design-time tool to transform the abstract definition into a user interface adapted to a concrete user interface's specific capabilities. This approach offers flexibility for the user interface developer to fine-tune the user interface's generation; but requires additional effort and is challenged by mobility, as it is not portable to devices with different capabilities, which is a common situation in mobile computing.
- Run-time: this method uses a run-time tool to render an abstract user interface specification for the device on which such a GUI is needed. This method does not allow user interface developers to fine-tune user interfaces, but gives higher flexibility and mobility to developed user interfaces at a much lower effort than design-time approaches.

Design-time adaptation

The design-time adaptation approach is based on generating user interfaces at design-time, as opposed to the run-time approach. Typically, a user interface definition is created for an application. This definition is then transformed using a tool into a concrete user interface for a specific device make or model. Such transformations usually generate different program source code for different interfaces (e.g., Visual Basic, Java or others). Source code is then compiled for the required platform and then executed on the platform. The most common approach is to define a multi-platform task model (Paterno & Santoro, 2002) and then to generate different user interfaces from this model. User interfaces generated in this way can be executed only on a specific device and platform; in contrast, the run-time approach tends to be much more versatile with respect to device capabilities.

Molina et al investigated an approach to define user interface using models and then automatically generating different programs (program source code) for different platforms (Molina et al, 2003). Code generators in this work are created for a limited number of different programming languages/platforms; however, additional code generators can be added to accommodate additional languages or platforms.

A similar approach, using an abstract user interface definition, was developed by Microsoft Corporation (2005) for its Longhorn/Avalon platform – it defines a user interface using an abstract notation, and then programmers develop code for different devices. This centralizes the user interface design but still requires multiple code implementations for different devices. In addition, Microsoft's approach is supported only on Windows platforms.

In general, a design-time approach offers some flexibility for the user interface developer because the resulting user interface can be manually fine-tuned before it is executed on the platform. However, the effort and expertise required for inspecting and fine-tuning multiple GUI versions before compilation and execution can be time-consuming and costly. Some of the design-time approaches still require programmers to develop multiple code implementations to handle user interface rendering and interaction on different platforms.

Run-time adaptation

Run-time adaptation is performed by using a program to adapt user interface definition at the time of program execution. Several approaches are used for run-time adaptation:

- Standalone adaptation: a specialized program adapts an abstract user interface definition to a specific platform.
- Client-server adaptation: a client program communicates with the server program in order to generate and present the user interface.
- Mobile agent adaptation: mobile agents within a mobile agent platform compose a mobile application that generates a mobile user interface.

In this section we will examine standalone and client-server adaptation; mobile agents and approaches based on them are detailed in the next section.

A standalone adaptation is delivered through a specialized program that adapts an abstract user interface definition to a specific platform. An example of such an adaptation is a Windows program that adapts an XML user interface definition to the Windows platform. Multiple implementations of adaptation programs can be developed for different platforms – for Java platform, for Palm PDA's and so forth. Luyten and Coninx (2001) developed a platform that utilizes an abstract user interface definition that is later rendered by multiple middleware software to various platforms. This approach can provide application functionality by a specialized proxy server, rather than by the mobile device itself. In this case data and program functionality are then handled using web services. This could present a limitation in situations where a wireless connection is not available or is not performing well.

A similar approach was adopted by Microsoft Corporation (2002), with "mobile forms" which automatically render to platforms supported by Microsoft's underlying engine, but which offers only a limited set of UI widgets (X Org, 1984). In addition, Microsoft's approach is available only on platforms supporting Windows and Microsoft Mobile Forms. This also restricts the number of available (supported) hardware platforms.

Such an adaptation approach lacks the mobility and flexibility required for mobile applications. The user interface is derived from a single specification, but it is pre-loaded onto devices and each software update triggers a new software set-up. The applications and user interfaces are frequently locked for a single device make and model. Development effort is high as multiple programs should be developed and verified for multiple platforms. In addition, mobile devices may not be capable of executing application functionality and, in such cases, specialized designs have to be developed. This introduces additional complexity and makes software more difficult to develop, maintain and support.

The MobiLife project (IST MobiLife, 2006) investigates the creation of user interfaces for mobile devices. This project studies the use of multi-modal user interfaces defined in an XML-based notation with the use of web services (Baillie et al, 2005). Among other areas, MobiLife focuses on context awareness, sharing and personalization (Kernchen et al, 2006; Salden et al, 2005). The MobiLife approach is based on a client-server architecture (Baillie et al, 2005) and is limited in terms of network mobility and autonomy: platform-specialized program implementations (clients) are used - network mobility is limited only to compatible platforms; autonomy of clients is restricted - autonomy depends on the availability of corresponding server components.

The client-server adaptation is based on the client-server computation model. A server platform works in conjunction with clients and usually provides computation for (less capable) clients. For example, a web application that is capable of transforming abstract UI definitions to both HTML and WML follows a client-server adaptation model. Such an approach requires a specialized client-side program capable of interpreting server-side information. Thus, multiple client programs must be developed for different devices, which introduces difficulties in maintaining client programs and their versions for a variety of devices. In addition, a level of anticipation is required in this approach in order to create constantly functional client programs. For example, if the user interface definition for a particular device is correctly specified, but requires more processing power (e.g., the minimal memory required for processing all UI elements exceeds the available memory for the specific mobile device), then it is anticipated that the client-program would fail unless this has been anticipated by the client-program developer at the time of development.

IBM (2002) developed the WebSphere Transcoding Publisher server, which could transform a user interface specification into HTML, WML and other formats. The Publisher could also handle JavaScript and transformations to WmlScript. However, this tool does not provide true mobility and is designed for web applications. A similar approach, aimed mostly at web development, is present in the Java community under the name Java Server Faces – JSF (Sun Microsystems, 2005). JSF is focused mainly on web development and is server oriented. This framework provides a component-based framework for web user interface development. In addition, this framework allows the specification and development of alternative renders – a rendering engine can potentially render a custom user interface definition to different platforms. For example, a render could transform a JSP or XUL UI definition to WML.

In the server-client model, a client-side program must be developed for all supported platforms, which introduces additional development, support and management complexity, and which could pose limitations when adapting to different platforms. This approach, however, provides limited mobility for the applications, as one application can be used from a number of different platforms. The client-server model relies on a good network connection, which is a limiting factor in costly, unstable or low-performing mobile networks.

Mobile agent adaptation

This adaptation approach is based on mobile agent technology (Milojicic et al, 1998; FIPA, 2002). The mobile agent technology eases automatic system adaptation to its execution environment. A mobile agent is a program that executes autonomously on a set of network hosts on behalf of an individual or organization. Software agents can easily adapt their behavior to different contexts. Mobile agents can bring computation wherever needed and minimize network traffic, especially in wireless networks (expensive, slow and unstable). In addition, mobile agents do not decrease the system's performance, even when communications are based on a wired network, as shown in (Mitrovic, Royo & Mena, 2005). Mobile agents can arrive at the users' device and show their GUIs to the users in order to interact with them. Mobile agents can be hosted by platforms that support different models of user interfaces or that have different processing capabilities. Because mobile agents are autonomous they can handle communication errors (unreachable hosts, etc.) by themselves. Also, they can move to the target device instead of accessing target devices remotely. For instance, agents can be sent to a home computer supporting Java and Swing, or they can play the role of a proxy server for a wireless device, such as mobile telephone or a web terminal; in that case they should produce an adequate GUI -- WML or HTML (Mitrovic & Mena, 2002). Such an adaptation is not limited to web pages or mobile phones; other devices such as PDAs benefit from such architectures (Mitrovic et al, 2005).

Mobile agents' platforms (Grasshopper, 2000; Ilarri, Trillo & Mena, 2006; Bursell & Ugai, 1997) are usually based on Java due to its portability, but are not limited to any particular platform (Wang, Sørensen & Indal, 2003). Mobile agent platforms are being developed for both old and new platforms such as Java Micro Edition (J2ME) (Wang et al, 2003).

Mobile agents can incorporate various learning techniques and learn from past experiences (Mitrovic & Mena, 2003). This is particularly important in the mobile world, where the abilities to learn and adapt to contexts and users are some of the most important requirements.

The mobile agent approach brings higher mobility and flexibility of user interaction while reducing development complexity – only one version of a program is defined.

To summarize: user interface adaptation to different devices is difficult; approaches are challenged by true mobility, require multiple development efforts, and have various degrees of flexibility and transparency. Mobile agents eliminate the majority of these issues and provide a flexible platform for the development of adaptive and portable GUIs.

Adaptation to Users

Different approaches are used to adapt user interfaces to meet users' preferences and contexts. Similar to adaptation to devices, adaptation to users can be done at design-time or at run-time. The design-time techniques include design patterns (Seffah & Forbrig, 2002), analysis of UI usage (Pitkow & Pirolli, 1999) or usability tests followed by UI redesign. Run-time techniques are present mostly in the web arena, where web-page content is rearranged by users' preferences or the website's context (Amazon on-line bookshop, 2006). Run-time techniques are based mostly on statistical and data mining techniques (Zukerman & Albrecht, 2000), which apply various artificial intelligence techniques to interaction data.

An additional challenge for mobile applications and run-time adaptation is users' attention (Vertegaal, 2003). Users' attention is a limited resource that is required by many external stimuli, including software applications. It is important to prioritize and adapt requests coming from software applications toward user in order to most effectively utilize this resource. Therefore, user interaction timing and volume must be considered when designing attentive user interfaces. Monitoring users' attention through physical indicators such as eye movement, geographic location or statistical data mining is crucial for prioritizing, adapting and designing interaction tasks (Vertegaal, 2003).

To be able to effectively adapt UIs to users at the run-time, it is crucial that systems provide facility for collecting information on computer-user interaction. Such facility enables later analysis and exchange of collected information, which could lead to changes in the application logic or user interface. Therefore, another requirement for successful UI adaptation to users is a facility for changing the user interface at run-time. Using this facility, or architecture, the user interface could be easily amended according to the collected information.

The majority of applications providing such facilities are not designed with interoperability in mind, and focus only on the current application's ability to gather and analyze interaction information. Applications frequently cannot exchange interaction information, and applications programmed by different developers cannot use a common set of facilities. This leads to a multiplication of developments designed for just a single application or application vendor. However, some systems based on mobile agents can be used by multiple applications and different (independent) learning modules.

ADUS – AN APPROACH BASED ON MOBILE AGENTS

The following introduces ADUS, our proposal for indirect generation of adaptive and portable GUIs. ADUS – ADaptive User Interface System (Mitrovic, Royo & Mena, 2004) is a system based on mobile agent technology that helps with user interface generation and allows monitoring of user-computer interaction. ADUS is part of the ANTARCTICA system (Goñi, Illarramendi, Mena, Villate & Rodriguez, 2001) – a multi-agent system that provides users with different wireless data services to enhance their mobile devices' capabilities.

ADUS has three main functions: 1) transparently adapting an abstract user interface definition to a concrete platform, 2) monitoring user interaction and communicating this information to other agents, and 3) communicating and collaborating with other agents in the ANTARCTICA platform. ADUS also performs a number of agent-based functions such as optimized network operation and collaboration with other agents. The ADUS system uses XUL (XUL Tutorial, 2002), an XML-based abstract user interface definition language.

Adaptive User Interface Generation in ADUS

In this section we present and discuss several approaches to generate adaptive user interface allowing the monitoring of the user behaviour.

Option 1: The Visitor Agent Generates the GUI

The first approach is that, when the visitor agent arrives at the user's device, it requests from the user agent the available resources, the user's preferences and the device's capabilities. Then the visitor agent creates the GUI by itself and interacts with the user directly.

This approach solves the generation of customized GUIs; however, it still has several problems:

- The user agent cannot monitor the user's behaviour because the data provided to the GUI flow directly to the visitor agent.
- The user agent must trust the visitor agent to render a GUI according to the user's preferences and the device's capabilities. Visitor agents could ignore the user agent descriptions and try to show their own GUI (in that case, the type of GUI created by the visitor agent could not be executed on that device).
- All the visitor agents have to know how to process and apply the knowledge provided by the user agent (which implies that they all must know how to generate any kind of GUI).

Option 2: The User Agent Generates the GUI and Delegates Event Handling to the Visitor Agent

In this approach the visitor agent, after arriving at the user's device, provides the user agent with a specification of the needed GUI. Then the user agent generates a GUI according to the user's preferences, the device's capabilities and the visitor agent's requirements, and it delegates the GUI event handling to the visitor agent.

The user interface specification can be made in the Extensible User-interface Language - XUL (XUL Tutorial, 2002). This interface definition can be later adapted by the user agent using XSL transformations to the required GUI representation language (HTML, WML, etc.). The XUL interpretation on Java-enabled platforms is interpreted by a Java XUL platform that renders XUL using standard AWT and Swing widgets.

The advantages of this approach are:

- The user agent guarantees that the visitor agents' GUIs will be generated correctly (according to the user's preferences and the device's capabilities) if they are specified in XUL.
- Visitor agents do not need to know how to generate GUIs in different devices.
- The user agent can deny permission to generate GUIs to all visitor agents (Mitrovic & Arronategui, 2002) in order to avoid direct GUI generation.

However, following this approach, the user agent cannot monitor the user's behaviour because GUI events are handled directly by visitor agents. Therefore the user agent must trust the visitor agent to get the information about its interaction with the user.

Option 3: An Intermediate Agent Generates the GUI and Handles the Events

In this approach, first the visitor agent sends its XUL specification of the GUI to the user agent, then the user agent generates the GUI and handles all the events (it receives data from the user), and finally, it sends the user data back to the visitor agent.

This approach has all the advantages of the approaches presented above. Furthermore, it allows the user agent to monitor the user's behaviour easily and efficiently as it handles the GUI events. Although this approach is interesting, its implementation faces a problem: the user agent must attend the different services executed on the user's device, and some tasks, such as GUI generation, could overload it. Therefore, a better approach is for the user agent to delegate the generation of adaptive GUIs to a specialized agent (the ADUS agent). Thus, the distribution of the service execution across the three agents (the ADUS agent, the user agent and the visitor agent) allows us to balance the system's load.

Indirect Generation of GUIs

This section describes in more detail the architecture needed to efficiently generate adaptive GUIs. To illustrate this process we use an example application. As shown in Figure 2, the system contains the following agents:

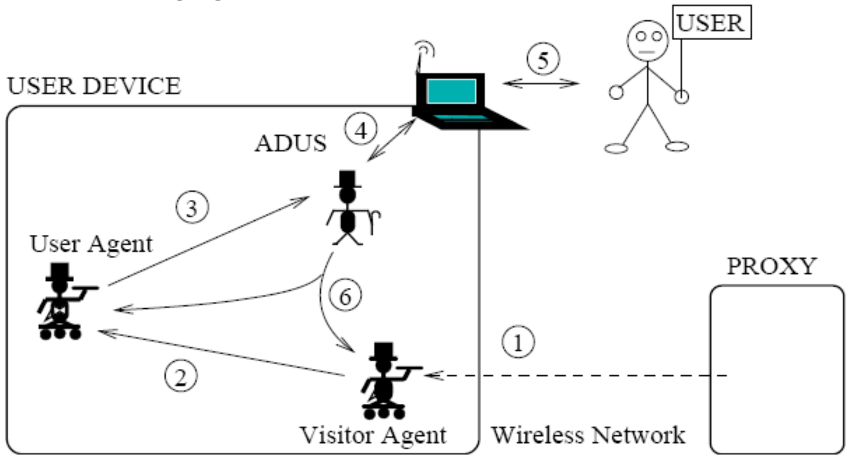


Figure 2. Indirect generation of GUIs

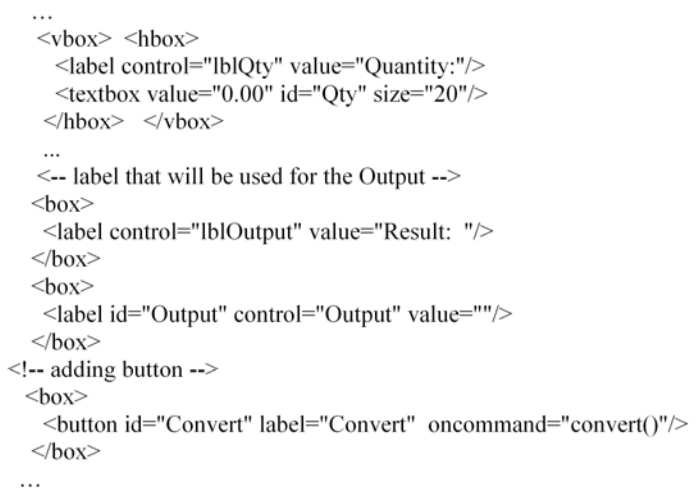
- *The visitor agent*: This is a mobile agent that brings a service requested by the user to the user's device. This agent can generate an XUL specification of the GUIs that it needs to interact with the user. Such XUL specifications are sent to the user agent on the user's device.

- *The user agent*: This is a highly specialized personalization agent that is in charge of storing as much information as possible about the user and his computer. For example, it knows the user's look and feel preferences and the GUI preferred by the user or imposed by the user's device or the operating system. This agent's main goals are: 1) to proxy the generation of user interfaces,
  2) to help the user to use the visitor agent's services, 3) to modify the visitor agent's GUI specification according to the user's preferences, 4) to create an ADUS agent initialized with the static GUI features such as the user's device capabilities, and 5) to monitor user interactions by receiving such information from the ADUS agent.
- *The ADUS agent*: This agent's main activities are: 1) to adapt the user interface to the user's preferences and device capabilities, following the user agent's suggestions; 2) to generate GUIs for different devices according to the XUL specification; and 3) to handle GUI events and communicate them to the visitor and user agents (allowing the user agent to monitor the user's interaction). There is one ADUS agent per visitor agent.

The following describes the synchronization of the above agents. As an example we use a simple currency converter application that converts between currencies per a user's requests, and displays the results of the conversion. This application is executed by mobile agents that travel to the user's device when requested by the user. The main steps are (see Figure 2):

1. The visitor agent travels to the user's device. This step is only for approaches that are based on mobile agents. For example, it is equivalent to the call of a local application (in client-server architecture).

The visitor agent requests the generation of its GUI. In this step the visitor agent sends the XUL description of its GUI to the user agent. In Figure 3 we show the XUL definition of the GUI for the currency converter service.



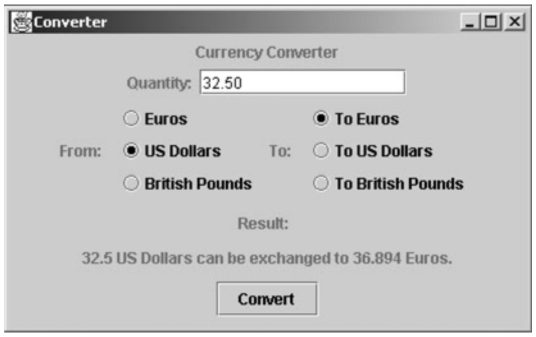
```
   …
   <vbox>  <hbox>
     <label control="lblQty" value="Quantity:"/>
     <textbox value="0.00" id="Qty" size="20"/>
    </hbox>   </vbox>
    ...
   <-- label that will be used for the Output -->
  <box>
    <label control="lblOutput" value="Result:  "/>
  </box>
  <box>
    <label id="Output" control="Output" value=""/>
  </box>
<!-- adding button -->
  <box>
    <button id="Convert" label="Convert"  oncommand="convert()"/>
  </box>
 …
```

Figure 3. XUL definition for the currency converter service

2. The user agent processes the GUI specification, transforms the GUI description to adapt it to the user's preferences, and creates the corresponding ADUS agent initialized with: 1) the transformed XUL description of the GUI to generate, and 2) the static information for the GUI such as the device's capabilities, screen resolution, representation language of the user's device (WML, HTML, Java Swing, etc.), among other information.
3. The ADUS agent generates the GUI according to the information provided by the user agent (static GUI information and specific information for this service). The ADUS agent can map any XUL description into GUIs for devices with different features; for example, a WAP device or a laptop with a Java GUI.

In the example, if the converter application were executed on a device with Java Swing capabilities (e.g., a home PC or laptop) the ADUS agent would generate a Swing GUI (see Figure 4a). When the converter application is executed on a web terminal without applet support, the generated GUI would be based on HTML, as shown in Figure 4b. Finally, if it is executed on a WAP mobile phone, then the GUI is based on WML, as shown in Figure 4c. We point out that this functionality of the ADUS agent works for any XUL specification. The ADUS agent could be extended with mappings to other kinds of GUI languages, such as Macromedia Flash or J2ME.

4. The user interacts with the GUI by looking at the information presented on the screen device and by using the device's peripherals (keyboard, mouse, buttons, etc.) to enter data or select among different options.
5. The ADUS agent handles and propagates the GUI events. User actions trigger GUI events that are captured by the ADUS agent. This information is sent to: 1) the visitor agent, which reacts to user actions according to the service that it executes, perhaps by generating a new GUI (step 2), and 2) the user agent, which can store and analyze the information provided by the user in order to reuse it in future service executions. One of the advantages of the presented architecture is that both messages can be sent concurrently, so the load is balanced.



Quantity: 32.50

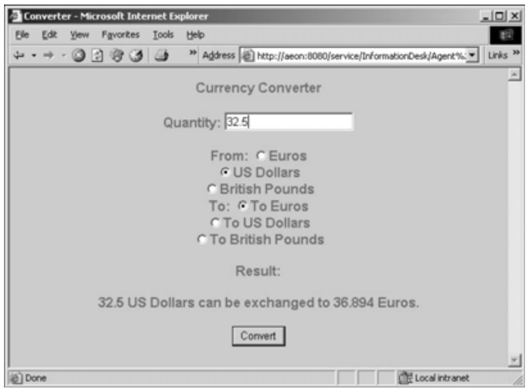
From: ○ Euros [● US Dollars] ○ British Pounds

To: [● To Euros] ○ To US Dollars ○ To British Pounds

Result:

32.5 US Dollars can be exchanged to 36.894 Euros.

(a)


Converter - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Links
Currency Converter
Convert
Done
Local intranet
http://aeon:8080/service/InformationDesk/Agent%

Quantity: 32.5

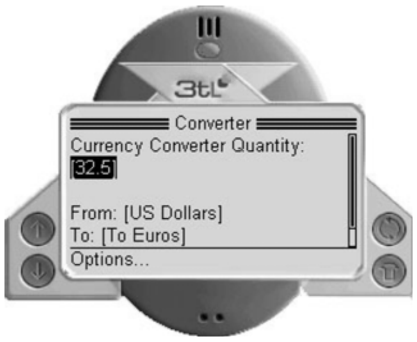
From: ○ Euros [● US Dollars] ○ British Pounds

To: [● To Euros] ○ To US Dollars ○ To British Pounds

Result:

32.5 US Dollars can be exchanged to 36.894 Euros.

(b)



Converter
Currency Converter Quantity:
32.5

From: [US Dollars]
To: [To Euros]
Options...

(c)

Figure 4. Currency Converter Service (Swing, HTML and WAP transformation)

Finally, we would like to stress the relevance to the user agent of monitoring the user's interaction with visitor agents. By knowing the user's reactions and data entered on those services, the user agent can store such data locally and apply different artificial intelligence techniques to extract knowledge about the user's behaviour. In the previous example, the next time that the currency converter service executes on the user's device, the user agent could select (in the visitor agent's XUL specification) US Dollars and Euros as the initial and target currencies, respectively, because that was the user's selection during the previous execution of that service. Even if the user now selects another configuration, the user agent could learn and improve its behaviour for the next time. In addition, the user interface could be rearranged to meet the user's preferences, e.g., the "convert" button could be in a different part of the screen. Thus, the customization of GUIs can become really useful for the user, as the user agent can monitor, store and analyze his interactions with all the GUIs/applications. In this case, mobile agents would learn from interaction data of other applications and from interaction data collected by other users (Mitrovic & Mena, 2003).

FUTURE TRENDS

In our increasingly mobile world, human-computer interaction has become one of the most important topics. Mobile user interface systems are more and more required to be context and location-aware, and we should improve interfaces by applying knowledge-based techniques. The mobility of systems providing a mobile user interface is an important factor, and mobile agents are one of the promising technologies in this field. Significant improvement in mobile agent platforms' scalability and performance is underway (Ilarri et al, 2006).

From the adaptation perspective, trends are likely to be the further standardisation of UI definition languages. Currently many different variants of XML-based user interface languages exist, which makes collective, community efforts less efficient than they would be if only one such language existed. Industry bodies such as the WWW Consortium have already begun proposing standard approaches, such as XForms (WWW Consortium, XForms, 2003). Improvements in mobile user interfaces' plasticity are a challenge yet to be fully addressed by researchers.

Consolidating mobile devices' capabilities and constantly increasing the performance of devices and infrastructure are two of the forces driving mobile systems and mobile user interface systems. Because processing power and different capabilities are two of the key issues for mobile user interfaces, this will positively affect both research efforts and developments. Although mobile devices are constantly being improved, certain device properties such as screen size will require methods that allow adaptation to users' devices.

In the agent-based user interfaces area, future trends are in increased interoperability across different multi-agent platforms. This will improve agent-based user interface systems' cross-platform mobility.

CONCLUSIONS

Generation of adaptive user interfaces in mobile environments is a complex task that faces many challenges. Mobile devices have different capabilities, work in different contexts, and have different kinds of users. Users require adaptation to their context and preferences. Mobile networks remain an expensive and unreliable medium, which presents additional difficulties. Many different approaches are used to solve user interfaces' problems in a mobile environment, from design-time considerations to various run-time solutions.

Design-time solutions are inclined more toward traditional approaches and are based more on multi-modal user interface modelling and code developments for different mobile devices than are run-time solutions. These approaches do not offer mobility and could suffer from mobile devices' low processing capability. Design-time approaches usually allow fine tuning of the user interface code (before deployment), which requires an additional development effort and expertise in the specific mobile system.

Run-time solutions offer a degree of mobility and higher flexibility than design-time solutions. The quality of the automatically generated interfaces is lower than in design-time approaches because of the lack of fine-tuning the user interface code before deployment. Run-time systems based on a client-server model give a degree of mobility but are limited by the requirement for mobile networks' constant availability. In addition, some kind of client software is required, which represents an additional effort.

However, mobile agents as a run-time approach provide good mobility for user interfaces and increased flexibility over other approaches. Mobile user interfaces based on mobile agents can transparently (to developers and users) adapt user interfaces to devices "on the fly." In addition, agents can distribute the processing load appropriately so that devices with low processing capabilities can still execute complex applications. Systems based on mobile agents can monitor and analyse computer-user interaction, and share such information between different program instances, users or systems. Such information can then be used to improve application logic and to create additional user interface adaptations. As mobile agents are fully mobile software entities, they enable mobile-agent based software and user interfaces to follow users wherever required.

Topics yet to be fully explored by researchers in the agent-based user interfaces area include inter-platform interoperability. Providing higher plasticity than is currently seen in mobile user interfaces remains a challenging task for all mobile user interfaces.

REFERENCES

Retrieved from Grasshopper: www.grasshopper.de

Amazon on-line bookshop. (2006). Retrieved from Amazon: www.amazon.com

Baillie, L., Schatz, R., Simon, R., Anegg, H., Wegscheider, F., Niklfeld, G., et al. (2005). Designing Mona: User Interactions with Multimodal Mobile Applications. *HCI International 2005, 11th International Conference on Human-Computer Interaction* (pp. 22-27). HCI International, 11th International Conference on Human-Computer Interaction.

Bursell, M., & Ugai, T. (1997). *Comparison of autonomous mobile agent technologies.* Retrieved from APM: www.ansa.co.uk/ANSATech/97/Primary/198901.pdf

Foundation for Intelligent Physical Agents - FIPA. (2002). Retrieved from www.fipa.org:

Goñi, A., Illarramendi, A., Mena, E., Villate, Y., & Rodriguez, J. (2001). ANTARCTICA: A Multiagent System for Internet Data Services in a Wireless Computing Framework. *NSF Workshop on an Infrastructure for Mobile and Wireless Systems* (ISBN 3-540-00289-8, Vol. 2538, pp. 119-135). Germany: Springer Verlag Lecture Notes in Computer Science LNCS.

IBM. (2002). Retrieved from IBM WebSphere Transcoding Publisher: http://www-3.ibm.com/software/webservers/transcoding/

Ilarri, S., Trillo, R., & Mena, E. (2006, June). SPRINGS: A Scalable Platform for Highly Mobile Agents in Distributed Computing Environments [4th International WoWMoM 2006 workshop on Mobile Distributed Computing (MDC'06)]. (pp. 633-637). New York: IEEE Computer Society.

IST MobiLife. (2006). Retrieved from http://www.ist-mobilife.org/: IST MobiLife

Kernchen, R., Bonnefoy, D., Battestini, A., Mrohs, B., Wagner, M., & Klemettinen, M. (2006). Context-Awareness in MobiLife. *15th IST Mobile Summit, Mykonos, Greece, June 2006.*

Limbourg, Q., & Vanderdonckt, J. (2003). Comparing Task Models for User Interface Design. *The Handbook of Task Analysis for Human-Computer Interaction.* Lawrence Erlbaum Associates.

Luyten, K., & Coninx, K. (2001). An XML Runtime User Interface DescriptionLanguage for Mobile Computing Devices. In Johnson, C. (Ed.) *Design, Specification and Verification of Interactive Systems - DSV-IS 2001.* (pp 1-15), Germany: Springer Verlag Lecture Notes in Computer Science LNCS vol 2220.

Microsoft Corporation. (2002). *Creating Mobile Web Applications with Mobile Web Forms in Visual Studio.NET.* Retrieved from Creating Mobile Web Applications with Mobile Web Forms in Visual Studio.NET: http://msdn.microsoft.com/vstudio/technical/articles/mobilewebforms.asp

Microsoft Corporation. (2005). *Longhorn Developer Center Home: Avalon.* Retrieved from Avalon: http://msdn.microsoft.com/Longhorn/understanding/pillars/avalon/default.aspx

Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., et al. (1998). MASIF, The OMG Mobile Agent System Interoperability Facility. *Mobile Agents* (Vol. 1477, pp 50-68). Germany: Springer Verlag Lecture Notes in Artificial Intelligence - LNAI.

Mitrovic, N., & Arronategui, U. (2002). Mobile Agent Security Using Proxy-agents and Trusted Domains. *Second International Workshop on Security of Mobile Multiagent Systems (SEMAS 2002, pp 81-83)* (ISSN 0946-008X). Germany: German AI Research Center DFKI.

Mitrovic, N., & Mena, E. (2002, June). Adaptive User Interface for Mobile Devices. In P. Forbrig, Q. Limbourg, B. Urban & J. Vanderdonckt (Eds.), *Interactive Systems. Design, Specification, and Verification. 9th International Workshop DSV-IS 2002* (ISBN 3-540-00266-9, pp. 47--61). Germany: Springer Verlag Lecture Notes in Computer Science LNCS vol. 2545.

Mitrovic, N., & Mena, E. (2003). Improving User Interface Usability Using Mobile Agents. *Interactive Systems. Design, Specification, and Verification. 10th International Workshop DSV-IS 2003* (ISBN 3-540-20159-9, pp. 273-288). Madeira, Portugal: Springer Verlag Lecture Notes in Computer Science LNCS vol. 2844.

Mitrovic, N., Royo, J. A., & Mena, E. (2004). ADUS: Indirect Generation of User interfaces on Wireless Devices. *Fifteenth International Workshop on Database and Expert Systems Applications (DEXA)* (ISBN 0-7695-2195-9, pp 662-666). IEEE Computer Society.

Mitrovic, N., Royo, JA., & Mena, E. (2005, September). Adaptive User Interfaces Based on Mobile Agents: Monitoring the Behavior of Users in a Wireless Environment. *Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI 2005)* (pp. 371-378). Spain: Thomson-Parninfo ISBN 84-9732-442-0.

Molina, P. J., Belenguer, J., & Pastor, O. (2003, July). Describing Just-UI Concepts Using a Task Notation. In J. Joaquim, N. Jardim Nunes & J. Falcao e Cunha (Eds.), *Interactive Systems. Design, Specification, and Verification. 10th International Workshop DSV-IS 200, pp 218-230.* Germany: Springer Verlag Lecture Notes in Computer Science LNCS vol. 2844.

Paterno, F., & Santoro, C. (2002). One model, many interfaces. In C. Kolski & J. Vanderdonckt (Eds.), *Fourth International Conference on Computer-Aided Design of User Interfaces* (pp. 143-154). Dordrecht: Kluwer Academics.

Pitkow, J., & Pirolli, P. (1999). Mining Longest Repeatable Subsequences to Predict World Wide Web surfing. *2nd Usenix Symposium on Internet Technologies and Systems (USITS, pp 139-150).* USITS.

Salden, A., Poortinga, R., Bouzid, M., Picault, J., Droegehorn, O., Sutterer, M., et al. (2005). Contextual personalization of a mobile multimodal application. *International Conference on Internet Computing (ICOMP, pp. 650-665).* Las Vegas, USA: International Conference on Internet Computing (ICOMP).

Seffah, A., & Forbrig, P. (2002). Multiple User Interfaces: Towards a Task-Driven and Patterns-Oriented Design Model. *9th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS)* (Vol. 2545, pp. 118-133). Germany: Springer Verlag Lecture Notes in Computer Science LNCS.

Stottner, H. (2001). *A Platform-Independent User Interface Description Language* (16). Linz, Austria: Institute for Practical Computer Science, Johannes Kepler University.

Sun Microsystems. (2005). *Java Server Faces.* Retrieved from JSF: http://java.sun.com/j2ee/javaserverfaces/

Thevenin, D., & Coutaz, J. (1999). Plasticity of User Interfaces: Framework and Research Agenda. *Proc of IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'99, (Vol. 1, pp. 110-117)*,. IOS Press.

Vertegaal, R. (2003). Attentive User Interfaces. *Communications of the ACM* (3, Vol. 46, pp. 30-33). USA: ACM Press.

W3C. (2000). Retrieved from Extensible Markup Notation: http://www.w3.org/TR/REC-xml

Wang, A.I., Sørensen, C.F., & Indal, E. (2003). A Mobile Agent Architecture for Heterogeneous Devices. 3rd IASTED International Conference on *Wireless and Optical Communications (WOC 2003, ISBN 0-88986-374-1, pp. 383-386).* Norway: ACTAPress.

WWW Consortium, XForms. (2003). Retrieved from X Forms: www.xforms.org

X Org. (1984). *X Windows.* Retrieved from X Org: www.x.org

XUL Tutorial. (2002). Retrieved from XUL: http://www.xulplanet.com/tutorials/xultu/

Zukerman, I., & Albrecht, D. (2000). Predictive Statistical Models for User Modeling. In A. Kobsa (Ed.), *User Modeling and User Adapted Interaction (UMUAI, Vol. 11, pp. 5-18) - The Journal of Personalization Research. Tenth Aniversary Special Issue.* Netherlands: Kluwer Academic Publishers.

KEY TERMS

Mobile Agent: A program that executes autonomously on a set of network hosts on behalf of an individual or organization. One of the key features of such agents is mobility.

Multi-Agent System: A system that allows concurrent operation and communication of multiple (mobile) agents.

User Interface Plasticity: A user interface's capacity to preserve usability regardless of variations in systems' hardware specification or operating environment.

Indirect User Interface Generation: A method in which the mobile agent requiring user interaction does not create a user interface directly, but passes the user interface definition to another agent (specialised for the user and his mobile device) that creates the above mentioned user interface and acts as an intermediary between the user and the agent that requires interaction with the user.

Run-time User Interface Adaptation: Automatic adaptation of the user interface by a program during its execution.

Design-time User Interface Adaptation: Manual adaptation of the user interface by a designer, analyst or software developer.

Abstract User Interface Definition: Platform-independent and technology-neutral description of the user interface.

Transparency: Automatic adaptation to specific conditions or circumstances without implicit or explicit intervention from the user, user interface designer or software developer.

BIOGRAPHICAL SKETCH

Nikola Mitrovic earned an MPhil degree in Computer Science from the University of Zaragoza, Spain. He is currently working toward a PhD degree in the area of intelligent and adaptive user interfaces. Nikola's research interest areas include interoperable, heterogeneous and distributed information systems and mobile computing. He has published several papers in international conferences and workshops and has served as a reviewer for several international conferences.

Eduardo Mena earned a BS degree in Computer Science from the University of the Basque Country and a PhD degree in Computer Science from the University of Zaragoza. He is an associate professor in the Department of Computer Science and Systems Engineering at the University of Zaragoza, Spain. For a year he was a visiting researcher in the Large Scale Distributed Information Systems Laboratory at the University of Georgia. He leads the Distributed Information Systems research group at his university and is a member of the Interoperable Database group at the University of the Basque Country. His research interest areas include interoperable, heterogeneous and distributed information systems, semantic Web and mobile computing. His main contribution is the OBSERVER project. He has published several papers in international journals, conferences and workshops, including a book on ontology-based query processing. He also has served as a referee for several international journals and as a program committee member for several international conferences and workshops.

Jose A. Royo earned a BS degree in computer science from the University of Zaragoza, Spain, in 2001. In his dissertation he will present the advantages of mobile knowledge driven agents in distributed and heterogeneous systems. He is currently the system analyst of the Electronic Engineering and Communications Department at the University of Zaragoza, Spain. His research interests include mobile agents; mobile computing; interoperable, heterogeneous and distributed information systems and semantic Web. He has published several papers in international conferences and workshops, and also has served as a reviewer for some conferences, workshops and journals.

AUTHOR NOTE

This work is supported by the CYCYT project TIN2004-07999-C02-02.