ADUS: Indirect Generation of User interfaces on Wireless Devices[*]

N. Mitrović
IIS Depart., Univ. of Zaragoza
Maria de Luna 1
50018 Zaragoza, Spain
Email: mitrovic@prometeo.cps.unizar.es

J. A. Royo[†]
IIS Depart., Univ. of Zaragoza
Maria de Luna 1
50018 Zaragoza, Spain
Email: joalroyo@unizar.es

E. Mena
IIS Depart., Univ. of Zaragoza
Maria de Luna 1
50018 Zaragoza, Spain
Email: emena@unizar.es

Abstract

Nowadays, there exists a great interest in wireless and mobile devices. However, the development of graphical user interfaces (GUIs) for applications in these environments must consider new problems: 1) Different device capabilities and 2) Automatic monitoring of user interfaces.

In this paper, we present an architecture that solves previous problems. We advocate the use of specifications of GUIs and the dynamic generation of the adequate visualization for a specific device without reimplementing each GUI for different devices.

Keywords: *Adaptive user interfaces, Mobile agents, User interface management for pervasive devices*

1. Introduction

Pervasive computing brings many challenges to researchers. Applications built to run anywhere are designed to accommodate even the most restrictive devices and yet to retain the desired functionality. However, different target devices have different processing powers, organization and capabilities. In addition, mobile devices have very restrictive user interfaces and it is very important that they assist users to achieve their goals in the most efficient way.

Solutions in this area mainly focus on web applications with client-server architecture, creating specialised and centralised services that transform one type of user interface into another. Some solutions propose the creation of 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 [13, 20] that could be later presented as Java GUI, or that can be transformed using XSLT [19].

The goal of this paper is to facilitate user interface generation for devices with different capabilities and to monitor the user behavior. Our prototype ADUS (ADvanced User interface System) allows applications to create user interfaces and apply user preferences and context. In order to do so we use indirect user interface generation and specialised intermediary modules. The generation of adaptive GUIs can be performed using a language to specify the user interface, like XUL [9]. This interface definition is later adapted using XSL transformations to any graphical representation of GUIs (HTML, WML [18], etc).

We assume a mobile agents architecture because software agents [4] can easily adapt their behavior to different contexts. Mobile agents are able to arrive at the user device and show their GUIs to the user to interact with her/him. Mobile agents can be hosted by platforms that support different models of user interfaces or have different processing capabilities. Agents are autonomous, and can handle network errors (unreachable hosts, etc.) autonomously. Also, they can move to the target device instead of accessing such a target device remotely. Agents can be sent to a home computer supporting Java and Swing. Also, an agent can play the role of a proxy server for a wireless device, such as mobile telephone or a web terminal; in that case it should produce WML or HTML, respectively.

The rest of this paper is as follows. In Section 2 we introduce our motivating example and explain the bound between mobile agents and GUIs. Section 3 introduces some approaches to solve the problems argued in the previous section. An explanation of the indirect GUI's generation method is detailed in Section 4. Section 5 gives an overview of the state of the art and the related work on this area. Finally, Section 6 concludes the paper and discusses the future work.

[*] Supported by the CICYT project TIC2001-0660, the DGA project P084/2001, and the research institute I3A.

[†] Work supported by the Aragon Government and the European Social Fund (ref B131/2002).

2. Motivating Example

The Adaptive User interface System (ADUS) is part of a more global system called ANTARCTICA [6] whose goal is to provide users with different wireless data services that enhance the capabilities of their mobile devices.

In ANTARCTICA, the intermediary agent Alfred is responsible for personalization, service discovery and generation of GUIs. Alfred is an efficient majordomo[1] that serves the user and is on charge of storing as much information about the user computer, and the user herself/himself, as possible. When another agent (*visitor agent*) wants to show/retrieve data to/from the user it has to communicate with Alfred which should create the appropriate user interface according to the user device capabilities and the user preferences. Let us denote by *user agent* those agents (software components) that, similarly to Alfred in ANTARCTICA, are the only ones that interact with the user because they manage knowledge about his/her preferences and about the device on which they execute.

From the point of view of GUI generation, user interface rendering is a complex task. Different devices have different capabilities: CPU speed, screen size, capability to display images, play sounds or movies, etc. There exist several design approaches that range from a few widgets, for example WML GUIs, to many different widgets, like in Java Swing. According to the device capabilities the designer must select one or another. Therefore, the development of GUIs for applications in a context with heterogeneous devices has the following problems:

- *Adaptation of user interfaces*: The visitor agent must adapt the creation of the user interface to the user preferences and user device capabilities. For example, the user can prefer thumbnails rather than full size images. Visitor agents are not aware of users's context, device capabilities, plasticity [17], etc. In addition, user preferences could change during the execution of the application.
- *Monitoring user interfaces*: If the user behavior is monitored, the user agent could use the data of previous executions to automatically assign initial values to the GUI of future visitor agents that request the same information [10].

Therefore, the goal of this work is to create adaptive user interfaces for *visiting agents* using *user agents* to customise such user interfaces according to user preferences and devices capabilities, and allow the monitoring of user interactions.

1 Alfred is an attempt of reflecting his role similar to a majordomo in the real world.

3. Adaptive User Interface Generation

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

3.1. Option 1: The Visitor Agent Creates the GUI

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

This approach solves the generation of customised GUI's, however, it still has several problems:

1. The user agent cannot monitor the user behavior as the data provided to the GUI flows directly to the visitor agent.
2. The user agent must trust the visitor agent to render a GUI according to user preferences and device capabilities. Visitor agents could ignore the user agent descriptions and show their own GUI directly.
3. 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).

3.2. Option 2: The User Agent Creates the GUI, the Visitor Agent Handles Events

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

- The user agent guarantees that the GUIs of visitor agents will be generated correctly (according to the user preferences and the device 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 the permission to generate GUI's to all visitor agents [12] in order to avoid direct GUI generation.

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

3.3. Option 3: An Intermediate Agent Creates the GUI and Handles the Events

In this approach, first, the visitor agent sends its XUL specification of the GUI to the user agent, second, 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 behavior 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 device and some tasks, like the GUI generation, could overload it. Therefore, a better approach is that the user agent delegates the generation of adaptive GUIs to a specialised agent (*ADUS*). Thus, the distribution of the service execution across three agents (the ADUS agent, the user agent, and visitor agent) allows us to balance and distribute the workload of the system.

4. Indirect Generation of GUI's

In this section we describe in more detail the architecture needed for the efficiently generation of adaptive GUIs. We use an application sample to illustrate such a process.

As shown on Figure 1, our system contains the following agents:

- *The visitor agent*: It is an mobile agent that brings a service requested by the user to the user device. This agent is able to generate a XUL [9] specification of the GUIs that it needs to interact with the user. Such a XUL specification is sent to the user agent on the user device.
- *The user agent*: It is a highly specialised personalization agent that is responsible for storing as much information about the user computer, and the user herself/himself, as possible. For example, it knows: the look and feel preferences of the user, the GUI preferred by the user or imposed by the user device or the operating system. The main goals of this agent is: 1) To proxy the generation of user interfaces, 2) To help the user to use the services of the visitor agent, 3) To modify the GUI specification of the visitor agent according to the user preferences, 4) To create an ADUS agent initialised with the static GUI features, and 5) To monitor user interactions by receiving such an information from the ADUS agent.
- *The ADUS agent*: The main features of this agent are: 1) To adapt the user interface to the user preferences and user device capabilities, following the user agent suggestions, 2) To generate GUIs for different devices according to the XUL specification, and 3) To handle the GUI events and to communicate them to the visitor agent as well as to the user agent. Application developers define user interface using XUL only once. This specification is rendered transparently for developers and users for various platforms. There will be one ADUS agent per visitor agent.

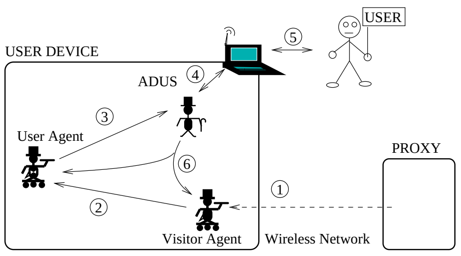
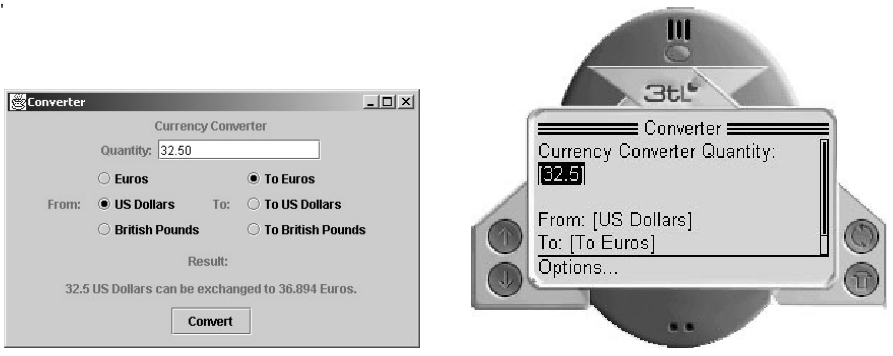


Figure 1. Indirect generation of GUIs

In the following we describe the synchronization of the above agents by using an example. In our previous work [13], we presented a simple currency converter application that converts currencies and displays the result of the conversion. This application is executed by mobile agents that travels to the user device when requested by the user. The main steps are (see Figure 1):

1. *The visitor agent travels to the user 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 a client-server architecture).
2. *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 2.a we show the XUL specification of the GUI for the currency converter service.
3. *The user agent processes the GUI specification*. It transforms the GUI description to adapt it to the user preferences, and creates the corresponding ADUS agent initialised with: 1) the transformed XUL description of the GUI to generate, and 2) the static information for the GUI such as the device capabilities: screen resolution, representation language of the user device (WML, HTML, Java Swing, etc) among other information.
4. *The ADUS agent generates the GUI*: It creates the GUI according to the information provided by the user agent (static GUI information and specific information for this service). The ADUS agent is able to map any XUL description into GUIs for devices with different

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



Converter

Currency Converter

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.

Convert

Converter

Currency Converter Quantity:

[32.5]

From: [US Dollars]

To: [To Euros]

Options...

(a) (b) (c)

Figure 2. Currency converter: a) XUL description, b) Java Swing rendering and c) WML rendering

features, e.g., a WAP device or a laptop with a Java GUI.

In the example, if the converter application is 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 2.b). When it is executed on a WAP mobile phone, then the GUI is based on WML, as shown in Figure 2.c. The ADUS agent could be extended with mappings to other kind of GUI languages, like Macromedia Flash.

5. *User interaction*: The user interacts with the GUI by looking at the information presented on the device screen and using the device peripherals (keyboard, mouse, buttons, etc) to enter data or select among different options.

6. *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 with the service that it executes, perhaps by generating a new GUI (step 2), and 2) the user agent, which can store and analyse 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 send concurrently, so a load balancing is performed.

Finally, we would like to stress the relevance for the user agent to monitor the user interaction with visitor agents. By knowing the user reactions and data entered to those services, the user agent can store such data locally and apply different artificial intelligence techniques to extract knowledge about the user behavior [14]. In the previous example, the next time that the currency converter service executes on the user device, the user agent could select US Dollars and Euros as the initial and target currencies, respectively, because that was the selection of the user during the last execution of that service. If the user selects now another configuration, the user agent could learn and improve its behavior for the next time. Thus, the customization of GUIs can become really useful for the user.

5. State of the Art and Related Work

In this section we present several approaches that allows to generate adaptive GUI and the related works to our knowledge about the monitoring of the user behavior.

5.1. Adaptive user interfaces

Various approaches to adapt user interfaces to various devices are present. Basically the approaches are grouped into two main categories: web applications and classic desktop applications. While the first category [11, 7] 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 universal definition of the user interfaces, so it can be later reproduced by various program implementations [15, 16, 20] (or middleware) on various platforms.

Without providing details, we mention some middleware approaches: language-based, grammar-based, e.g., event-based, constraint-based, User Action Notation and widget-based. However, the XML-based efforts [9, 20, 1] are most interesting for us, since they provide flexibility and easy manipulation. A similar approach is adopted by us, the XML-based user interface definition language is transformed into a concrete GUI, but this transformation is done transparently to the user and on the fly.

5.2. Predicting User Behavior

There are some approaches such as: 1) WebTango[8], that collects user interface metrics for a web site; 2) Predictions of the user behavior based on Markov chains [3]; and 3) Longest Repeating Subsequence (LRS) [14, 5] and Information Scent [2] that perform data mining seeking to analyse navigation path based on server logs, similarity of pages, linking structure and user goals.

In our approach the user agent monitors the user behavior and could make use of all of the above approaches to predict the user behavior.

6. Conclusions and Future Work

We have presented an architecture for the adaptive generation of GUIs in mobile devices, following an indirect approach that allows the monitoring of user interactions. The solution presented in this paper is useful for the majority of the systems that can be executed in different devices with different capabilities. However, we have applied it to an architecture based on mobile agents because they make development of applications in a wireless environment easy.

The main advantages of our approach are the following:

- GUIs of visitor agents will be generated correctly (according to the user preferences and the device capabilities) if they are specified in XUL.
- Visitor agents do not need to know how to generate GUIs in different devices.
- The generation of GUIs by visitor agents can be easily rejected to avoid direct GUI generation.
- Any user interaction can be monitored by the system in order to help the user to interact with future invocations of services.

As future work, we consider to increase the number of transformations managed by the system to be able to generate GUIs for new devices with different languages for GUI generation. We are also studying different techniques to improve the exploitation of the data stored by the user agent about the user interactions.

References

[1] M. Abrams, C. Phanouriou, A. Batongbacal, S. William, and J. Shuster. Uiml: An appliance-independent xml user interface language. In *WWW8 / Computer Networks 31(11-16): 1695-1708*, 1999.

[2] E. Chi, P. Pirolli, , and J. Pitkow. The scent of a site: A system for analyzing and predicting information scent, usage, and usability of a web site. In *ACM CHI 00 Conference on Human Factors in Computing Systems*, 2000.

[3] M. Deshpande and G. Karypis. Selective markov models for predicting web-page accesses. Technical report, University of Minnesota Technical Report 00-056, 2000.

[4] D. M. et al. MASIF, the OMG mobile agent system interoperability facility. In *Proceedings of Mobile Agents'98*, 1998.

[5] J. P. et al. Mining longest repeatable subsequences to predict world wide web surfing. In *2nd Usenix Symposium on Internet Technologies and Systems (USITS)*, 1999.

[6] A. Goñi, A. Illarramendi, E. Mena, Y. Villate, and J. Rodriguez. ANTARCTICA: A multiagent system for internet data services in a wireless computing framework. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Scottsdale, Arizona (USA)*, pages 119–135. LNCS 2538 2002, October 2001.

[7] IBM. Ibm websphere transcoding publisher, 2001. http://www-3.ibm.com/software/webservers/transcoding/.

[8] M. Ivory, R. Sinha, , and M. Hearst. Empirically validated web page design metrics. In *SIGCHI*, 2001.

[9] jXUL, 2002. http://jxul.sourceforge.net.

[10] E. Mena, A. Illarramendi, and A. Goñi. A Software Retrieval Service based on Knowledge-driven Agents. In *Fith IFCIS International Conference on Cooperative Information Systems (CoopIS'2000)*, pages 174–185, September 2000.

[11] Microsoft Corp. Creating Mobile Web Applications with Mobile Web Forms in Visual Studio .NET, 2001. http://msdn.microsoft.com/vstudio/technical/articles/mobilewebforms.asp.

[12] N. Mitrović and U. Arronategui. Mobile agent security using proxy-agents and trusted domains. In *Second International Workshop on Security of Mobile Multiagent Systems, German AI Research Center Research Report*, July 2002.

[13] N. Mitrović and E. Mena. Adaptive user interface for mobile devices. In *Interactive Systems. Design, Specification, and Verification. 9th International Workshop DSV-IS 2002, Rostock (Germany)*, pages 47–61. Springer Verlag Lecture Notes in Computer Science LNCS, June 2002.

[14] N. Mitrović and E. Mena. Improving user interface usability using mobile agents. In *Interactive Systems. Design, Specification, and Verification. 10th DSV-IS Workshop, Funchal, Madeira Island (Portugal)*, pages 273–287. Springer Verlag Lecture Notes in Computer Science LNCS 2844, June 2003.

[15] P. Molina, J. Belenguer, and O. Pastor. Describing justui concepts using a task notation. In *Interactive Systems. Design, Specification, and Verification. 10th International Workshop DSV-IS 2003, Funchal, (Portugal), Springer Verlag Lecture Notes in Computer Science LNCS, I*, July 2003.

[16] H. Stottner. A platform-independent user interface description language. Technical report, Technical Report 16, Institute for Practical Computer Science, Johannes Kepler University Linz,, 2001.

[17] D. Thevenin and J. Coutaz. Plasticity of user interfaces: Frame-work and research agenda. In *Proc of IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'99, Edinburgh*, August 1999.

[18] W. F. WAP-WML Specification Version 1.1, 16 Jun 1999, 1999. http://www.wapforum.org/.

[19] World Wide Web Consortium. XSL Transformations Version 1.0, W3C Recommendation 16, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[20] XIML (eXtensible Interface Markup Language), November 1999. http://www.ximl.org/.